Chapter 1

Basic Signals and Systems

January 17, 2007

Overview

MATLAB is an ideal software tool for studying Digital Signal Processing. Its language has many functions that are commonly needed to create and process signals. The plotting capability of MATLAB makes it possible to view the results of processing and gain understanding into complicated operations.

In this first chapter, we present some of the basics of DSP in the context of MATLAB. At this point, some of the exercises are extremely simple so that familiarity with the MATLAB environment can be acquired. Generating and plotting signals is treated first, followed by the operation of difference equations as the basic linear time-invariant system. An important part of this chapter is understanding the role of the numerical computation of the Fourier transform (DTFT). Since MATLAB is a numerical environment we must manipulate samples of the Fourier transform, rather than formulas. We also examine the signal property called group delay. Finally, the sampling process is studied to show the effects of aliasing, and different reconstruction schemes.

There many excellent textbooks that provide background reading for the projects in this chapter. We mention as examples the books by Jackson (1986), McGillem and Cooper (1974), Oppenheim and Schafer (1989), Oppenheim and Willsky (1983), and Proakis and Manolakis (1988). In each of these sections we have indicated specific background reading from Oppenheim and Schafer (1989) but similar background reading can be found in most other texts on Digital Signal Processing.

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{Signals}}$

Signals

Overview

The basic signals used often in digital signal processing are the unit impulse signal $\delta[n]$, exponentials of the form $a^n u[n]$, sine waves, and their generalization to complex exponentials. The following projects are directed at the generation and representation of these signals in MATLAB. Since the only data type in MATLAB is the $M \times N$ matrix, signals must be represented as vectors: either $M \times 1$ matrices if column vectors, or $1 \times N$ matrices if row vectors. In MATLAB all signals must be finite in length. This contrasts sharply with analytical problem solving where a mathematical formula can be used to represent an infinite-length signal, e.g., a decaying exponential, $a^n u[n]$.

A second issue is the indexing domain associated with a signal vector. MATLAB assumes by default that a vector is indexed from 1 to N, the vector length. In contrast, a signal vector is often the result of sampling a signal over some domain where the indexing runs from 0 to $N-1$; or, perhaps, the sampling starts at some arbitrary index that is negative, say from $-N$. The information about the sampling domain cannot be attached to the signal vector containing the signal values. Instead the user is forced to keep track of this information separately. Usually this is no big problem until it comes time to plot the signal, in which case the horizontal axis must be labelled properly.

A final point is the useage of MATLAB's vector notation to generate signals. A signficant power of the MATLAB environment is its high-level notation for vector manipulation; `for` loops are almost always unnecessary. When creating signals such as a sine wave, it is best to apply the `sin` function to a vector argument, consisting of all the time samples.

In the following projects, we will treat the common signals encountered in digital signal processing: impulses, impulse trains, exponentials, and sinusoids.

Background Reading

Oppenheim and Schafer (1989), Chapter 2, sections 2.0 and 2.1.

Project 1: Basic Signals

Project Description

This project concentrates on the issues involved with generation of some basic discrete-time signals in MATLAB. Much of the work centers on using internal MATLAB vector routines for signal generation. In addition, a sample MATLAB function will be implemented.

Hints

Plotting discrete-time signals is done with the `comb` function in MATLAB. The following MATLAB code will create 31 points of a discrete-time sinusoid.

```
nn   = 0:30;     %-- vector of time indices
sinus = sin(nn/2+1);
```

Notice that the $n = 0$ index must be referred to as `nn(1)`, due to MATLAB's indexing scheme; likewise, `sinus(1)` is the first value in the sinusoid. When plotting the sine wave we would use the `comb` function which produces the discrete-time signal plot commonly seen

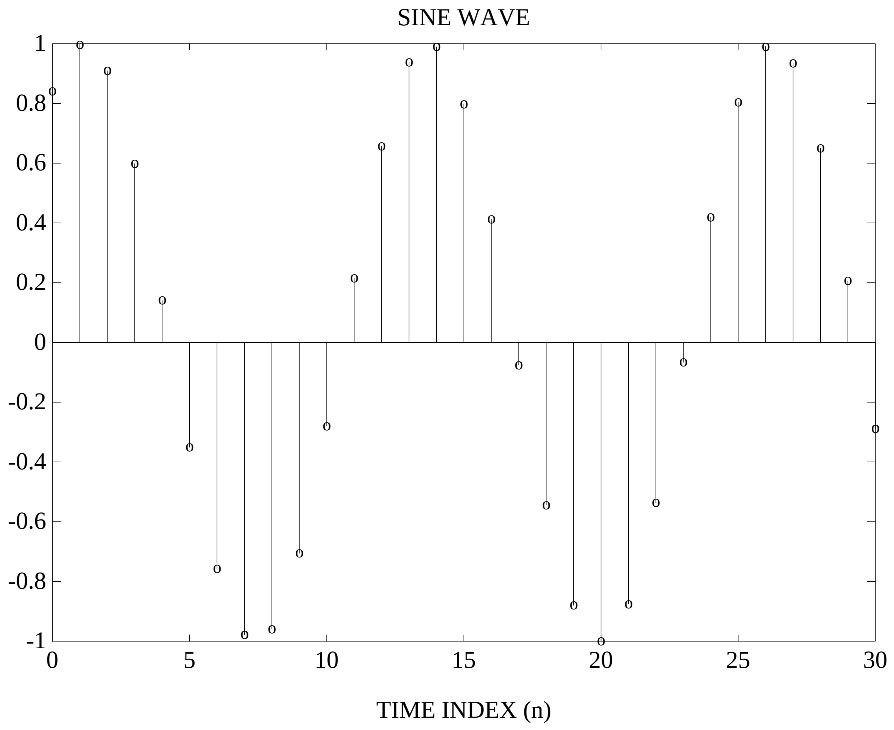


Figure 1: Example of Plotting a discrete-time signal with `comb`.

in DSP books, see Fig. 1:

```
comb( nn, sinus );
```

The first vector argument must be given in order to get the correct n-axis. For comparison, try `comb(sinus)` to see the default labelling.

Exercise 1.1: Basic Signals—Impulses

The simplest signal is the (shifted) unit impulse signal:

$$\delta[n-n_0] = \begin{cases} 1 & n = n_0 \\ 0 & n \neq n_0 \end{cases} \tag{1.1}$$

In order to create an impulse in Matlab, we must decide how much of the signal is of interest. If the impulse $\delta[n]$ is going to be used to drive a causal LTI system, we might want to see the L points from $n = 0$ to $n = L - 1$. If we choose $L = 31$, the following Matlab code will create an "impulse"

```
L = 31;
nn  = 0:(L-1);
imp = zeros(L,1);
imp(1) = 1;
```

Notice that the $n = 0$ index must be referred to as `imp(1)`, due to Matlab's indexing scheme.

(a) Generate and plot the following sequences. In each case, the horizontal (n) axis should extend only over the range indicated and should be labelled accordingly. Each sequence should be displayed as a discrete-time signal using `comb`.

$$\begin{aligned} x_1[n] &= .9\delta[n-5] && 1 \le n \le 20 \\ x_2[n] &= .8\delta[n] && -15 \le n \le 15 \\ x_3[n] &= 1.5\delta[n-333] && 300 \le n \le 350 \\ x_4[n] &= 4.5\delta[n+7] && -10 \le n \le 0 \end{aligned}$$

(b) The shifted impulses, $\delta[n-n_0]$, can be used to build a weighted impulse train, with period P and total length MP:

$$s[n] = \sum_{\ell=0}^{M-1} A_\ell \, \delta[n-\ell P] \tag{1.2}$$

The weights are A_ℓ; if they are all the same, the impulse train is periodic with period P. Generate and plot a periodic impulse train whose period is $P = 5$, and whose length is 50. Start the signal at $n = 0$. Try to use one or two vector operations rather than a `for` loop to set the impulse locations. How many impulses are contained within the finite-length signal?

(c) The following MATLAB code will produce a repetitive signal in the vector `x`:

```
x = [0;1;1;0;0;0] * ones(1,7);
x = x(:);
size(x)    %<--- return the signal length
```

Plot `x` in order to visualize its form; then give a mathematical formula similar to (1.2) to describe this signal.

Exercise 1.2: Basic Signals—Sinusoids

Another very basic signal is the cosine wave. In general, it takes three parameters to completely describe a real sinusoidal signal: amplitude (A), frequency (ω_0), and phase (ϕ).

$$x[n] = A\cos(\omega_0 n + \phi) \tag{1.3}$$

(a) Generate and plot each of the following sequences. Use MATLAB's vector capability to do this with one function call by taking the cosine (or sine) of a vector argument. In each case, the horizontal (n) axis should extend only over the range indicated and should be labelled accordingly. Each sequence should be displayed as a sequence using `comb`.

$$\begin{aligned} x_1[n] &= \sin \tfrac{\pi}{17} n && 0 \le n \le 25 \\ x_2[n] &= \sin \tfrac{\pi}{17} n && -15 \le n \le 25 \\ x_3[n] &= \sin(3\pi n + \tfrac{\pi}{2}) && -10 \le n \le 10 \\ x_4[n] &= \cos\left(\tfrac{\pi}{\sqrt{23}} n\right) && 0 \le n \le 50 \end{aligned}$$

Give a simpler formula for $x_3[n]$ that does not use trignometric functions. Explain why $x_4[n]$ is not a periodic sequence.

(b) Write a MATLAB function that will generate a finite-length sinusoid. The function will need a total of 5 input arguments; 3 for the parameters, and 2 more to specify the first and last n index of the finite-length signal. The function should return a column vector that contains the values of the sinusoid. Test this function by plotting the results for various choices of the input parameters. In particular, show how to generate the signal $2\sin(\pi n/11)$ for $-20 \le n \le 20$.

(c) *Modification:* Write the function to return two arguments: a vector of indices over the range of n, as well as the values of the signal.

Exercise 1.3: Sampled Sinusoids

Often a discrete-time signal is produced by sampling a continuous-time signal such as a constant frequency sine wave. The interrelationship between the continuous-time frequency and the sampling frequency is the main point of the Nyquist-Shannon Sampling Theorem, which requires that the sampling frequency be at least twice the highest frequency in the signal for perfect reconstruction.

In general, a continuous-time sinusoid is given by the following mathematical formula:

$$s(t) = A\cos(2\pi f_0 t + \phi) \tag{1.4}$$

where A is the amplitude, f_0 is the frequency in Hertz, and ϕ is the initial phase. If a discrete-time signal is produced by regular sampling of $s(t)$ at a rate of $f_s = 1/T$, then we get

$$s[n] = s(t)|_{t=nT} = A\cos(2\pi f_0 T n + \phi) = A\cos\left(2\pi \frac{f_0}{f_s} n + \phi\right) \tag{1.5}$$

Comparison with the previous formula (1.3) for a discrete-time sinusoid, $x[n] = A\cos(\omega_0 n + \phi)$, shows that the normalized radian frequency is now a scaled version of f_0, $\omega_0 = 2\pi(f_0 T)$.

(a) From the formula (1.4) for the continuous-time sinusoid, write a function that will generate samples of $s(t)$ to create a finite-length discrete-time signal. This function will need 6 inputs: 3 for the signal parameters, 2 for the start and stop times, and one for the sampling rate (in Hertz). It can call the previously written MATLAB function for the discrete-time sinusoid. To make the MATLAB function correspond to the continuous-time signal definition, make the units of the start and stop times seconds, not index number. Use this function to generate a sampled sinusoid with the following definition:

```
Signal freq = 1200 Hz       Sampling freq = 8 kiloHz
Initial Phase = 45 deg      Starting Time = 0 sec
Amplitude = 50              Ending Time = 7 millisec
```

Make two plots of the resulting signal: one as a function of time t (in millisec), and the other as a function of the sample index n used in $t_n = nT$. Determine the length of the resulting discrete-time signal, and the number of periods included in the vector.

(b) Show by mathematical manipulation that sampling a cosine at the times $t_n = n + \frac{3}{4}T$ will result in a discrete-time signal that appears to be a sine wave, when $f_0 = 1/T$. Use the function from the previous part to generate a discrete-time sine wave by changing the start and stop times for the sampling.

Exercise 1.4: Basic Signals—Exponentials

The decaying exponential is a basic signal in DSP because it occurs as the solution to linear constant coefficient difference equations.

(a) Study the following MATLAB function to see how it generates a discrete-time exponential signal. Then use the function to plot the exponential $x[n] = (0.9)^n$ over the range $n = 0, 1, 2, \ldots, 20$.

```
function  y = genexp( b, n0, L )
%GENEXP   generate an exponential signal: b^n
%   usage: Y = genexp( B, N0, L )
%      B   input scalar giving ratio between terms
%      N0  starting index (integer)
%      L   length of generated signal
%      Y   output signal Y(1:L)
if( L <= 0 )
  error('GENEXP: length not positive')
end
nn = n0 + [1:L]' - 1;  %---vector of indices
y = b .^ nn;
end
```

(b) In many derivations, the exponential sequence $a^n u[n]$ must be summed over a finite range. This sum is known in closed form:

$$\sum_{n=0}^{L-1} a^n = \frac{1 - a^L}{1 - a} \qquad \text{for } \; a \neq 1 \tag{1.6}$$

Use the function from the previous part to generate an exponential and then sum it up; compare the result to the formula above (1.6).

(c) One reason the exponential sequence occurs so often in DSP is that time-shifting does not change the character of the signal. Show that the finite-length exponential signal satisfies the shifting relation:

$$y[n] = ay[n-1] \qquad \text{over the range } \; 1 \leq n \leq L-1 \tag{1.7}$$

by comparing the vectors `y(2:L)` and `a*y(1:L-1)`. When shifting finite-length signals in MATLAB, we must be careful at the end points because there is no automatic zero padding.

(d) Another way to generate an exponential signal is to use a recursive formula given by a difference equation. The signal $y[n] = a^n u[n]$ is the solution to the following difference equation when the input, $x[n]$, is an impulse:

$$y[n] - ay[n-1] = x[n] \qquad \text{initial condition: } y[-1] = 0 \tag{1.8}$$

Since the difference equation is assumed to recurse in a causal manner (i.e., for increasing n), the initial condition at $n = -1$ is necessary.

In MATLAB the function `filter` will implement a difference equation. Use `filter` to generate the same signal as in part (a), i.e., $a = 0.9$.

Project 2: Complex-Valued Signals

Project Description

This project centers on the issues involved with representing and generating complex-valued signals. Although in the "real world" signals must have real values, it is often extremely useful to generate, process, and interpret real-valued signal pairs as complex-valued signals. This is done by combining the signals into a pair, as the real and imaginary parts of a complex number, and processing this pair with other "complex-valued" signals using the rules of complex arithmetic. Use of signal pairs in this way is an important part of many signal processing systems, especially those involving modulation.

Complex exponentials are a class of complex signals that is extremely important because the complex amplitude (phasor notation) provides a concise way to describe sinusoidal signals. Most EE students are familiar with phasors in connection with AC circuits or power systems, but their use in radar, wave propagation and Fourier analysis is just as significant (although the term phasor is not always used).

Hints

In MATLAB, the functions `real` and `imag` will extract the real and imaginary parts of a complex number. When plotting a complex vector, the defaults for `plot` and can be confusing. If `z` is complex, then `plot(z)` will plot the imaginary part versus the real part; and `plot(n, z)` will plot the real part of `z` versus `n`. However, `comb(z)` will just plot the real part. If you want to view simultaneous plots of the real and imaginary parts, the `subplot(211)` and `subplot(212)` commands prior to each `comb` command will force the two plots to be placed on the same screen, one above the other, see Fig. 2 which is created by the code shown below.

```
nn = 0:25;
xx = exp(j*nn/3);    %--- complex exponential
subplot(211)
comb(nn, real(xx))
title('REAL PART'),  xlabel('INDEX (n)')
subplot(212)
comb(nn, imag(xx))
title('IMAG PART'),  xlabel('INDEX (n)')
```

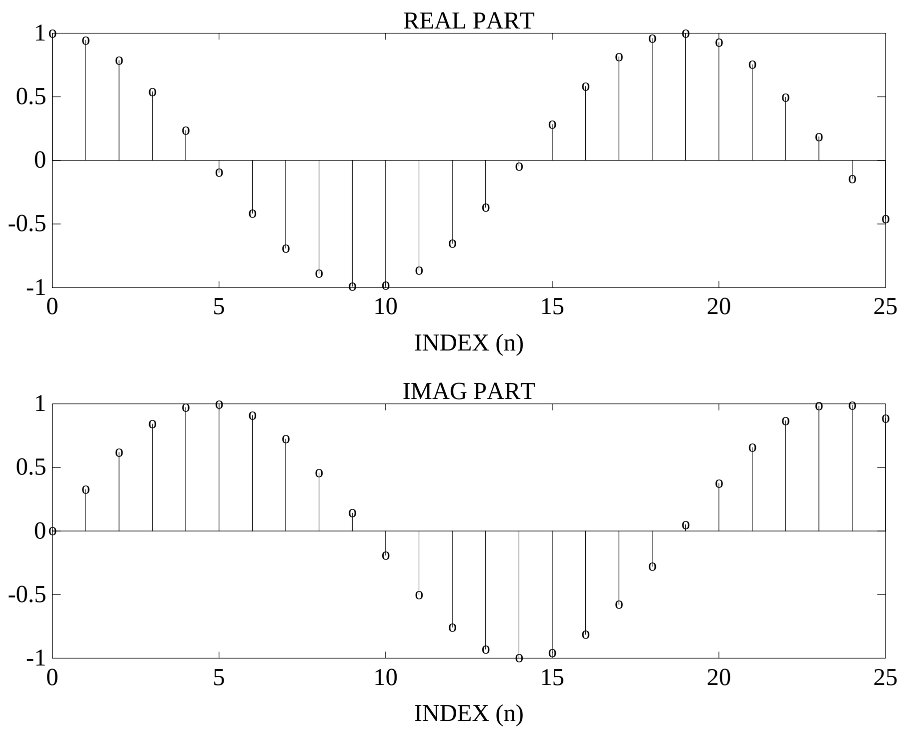


Figure 2: Plotting Real and Imaginary Parts of a discrete-time signal with `subplot`.

Exercise 2.1: Complex Exponentials

The real exponential notation can be extended to complex-valued exponential signals that embody the sine and cosine signals. These signals form the basis of the Fourier transform.

(a) In MATLAB a complex signal is a natural extension of the notation in the previous exercise. Thus the parameter a can be taken as a complex number to generate these signals. Recall Euler's formula for the complex exponential (in a form that gives a signal):

$$x[n] = (z_0)^n = e^{(\ln z_0 + j\angle z_0)n} = r^n e^{j\theta n} = r^n(\cos\theta n + j\sin\theta n) \tag{2.1}$$

where $z_0 = re^{j\theta} = r\angle\theta$. Use this relationship to generate a complex exponential with $z_0 = 0.9\angle 45°$. Plot the real and imaginary parts of $x[n]$ over the range $0 \le n \le 20$. Notice that the angle of z_0 controls the frequency of the sinusoids.

(b) For the signal in part (a) make a plot of the imaginary part versus the real part. The result should be a spiral. Experiment with different angles for θ—a smaller value should produce a better picture of a spiral.

(c) The formula above (2.1) is not general enough to produce all complex exponentials. What is missing is a *complex* constant to scale the amplitude and phase of the sinusoids. This is the so-called *Phasor Notation*:

$$G \cdot z_0^n = A\,e^{j\phi} r^n e^{j\theta n} = Ar^n e^{j(\theta n + \phi)} = Ar^n\left[\cos(\theta n + \phi) + j\sin(\theta n + \phi)\right] \tag{2.2}$$

where $G = A\,e^{j\phi} = A\angle\phi$ is the complex amplitude of the complex exponential.

Generate and plot each of the following sequences. Convert the sinusoids to complex notation; then create the signal vector using `exp`. If the signal is purely real, then it should be generated by taking the real part of a complex signal. In each plot, the horizontal (n) axis should extend only over the range indicated and should be labelled accordingly.

$$\begin{aligned} x_1[n] &= 3\sin(\tfrac{\pi}{7}n) + j4\cos(\tfrac{\pi}{7}n) && 0 \le n \le 20 \\ x_2[n] &= \sin\tfrac{\pi}{17}n && -15 \le n \le 25 \\ x_3[n] &= 1.1^n \cos\left(\tfrac{\pi}{11}n + \tfrac{\pi}{4}\right) && 0 \le n \le 50 \\ x_4[n] &= .9^n \cos\left(\tfrac{\pi}{11}n\right) && -10 \le n \le 20 \end{aligned}$$

For each signal, determine the values of amplitude and phase constants that have to be used in G; also the angle and magnitude of z_0.

(d) These same complex exponentials can be generated by first-order difference equations (using `filter`).

$$y[n] = z_0 y[n-1] + x[n] \tag{2.3}$$

The filter coefficient, $z_0 = re^{j\theta}$, is a complex number. The ratio between successive terms in the sequence is easily seen to be z_0; but the correct amplitude and phase must be set by choosing a complex amplitude for the impulse, that drives the difference equation, i.e., $x[n] = G\delta[n]$, Use `filter` to create the same signals as in the previous part. Verify by plotting both the real and imaginary parts of $y[n]$ and comparing to the signals generated via `exp`.

(e) In the first-order difference equation above (2.3), let $y_R[n]$ and $y_I[n]$ denote the real and imaginary parts of $y[n]$. Write a pair of real-valued difference equations expressing $y_R[n]$ and $y_I[n]$ in terms of $y_R[n-1], y_R[n-1]$, $x[n]$ and r, $\cos\theta$ and $\sin\theta$.

(f) Write a MATLAB program to implement this pair of real equations, and use this program to generate the impulse response of equation (2.3) for $r = \frac{1}{2}$ and $\theta = 0$, and $\theta = \frac{\pi}{4}$. For these two cases, plot the real part of the impulse responses obtained. Compare to the real part of the output from the complex recursion (2.3).

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{Difference Equations}}$

Difference Equations

Overview

Of particular importance in Digital Signal Processing are the class of systems that can be represented by linear constant-coefficient difference equations. This set of projects explores the characteristics of these systems in both the time and frequency domains. Specifically, Project 1 considers the impulse response of IIR difference equations. Project 2 explores the steady state response for step and complex exponential inputs. In Project 3, the frequency response is investigated.

Background Reading

Oppenheim and Schafer (1989) chapter 2, sections 2.2 through 2.5 discuss discrete-time systems, linear time-invariant systems, and linear constant-coefficient difference equations. Sections 2.6 through 2.9 discuss frequency domain representations of discrete-time signals and systems.

Project 1: Time-domain Response of Difference Equations

Project Description

In this project, you will generate the response of an IIR (infinite impulse response) filter, which is an LTI system expressed as a linear constant-coefficient difference equation:

$$\sum_{k=0}^{N_a} a_k \, y[n-k] = \sum_{\ell=0}^{N_b} b_\ell \, x[n-\ell] \tag{1.1}$$

In MATLAB, difference equations are represented by two vectors: one vector containing the feed-forward coefficients, b_ℓ, for the `x` terms, and the other vector containing the feedback coefficients, a_k, for the `y` terms. The coefficient a_0 is usually taken to be 1, so that when $y[n]$ is written in terms of past values it drops out:

$$y[n] = -\frac{1}{a_0}\sum_{k=1}^{N_a} a_k \, y[n-k] + \sum_{\ell=0}^{N_b} b_\ell \, x[n-\ell]$$

In MATLAB the `filter` function will divide out a_0, so it must not be zero.

Hints

The function `y = filter(b,a,x)` implements a digital filter defined by the `a` and `b` coefficient vectors as in (1.1) to filter the data stored in `x`. If `x` is the unit impulse signal, then `y` will be the impulse response $h[n]$. Note that the function `filter` only returns as many samples into `y` as there are in `x`, *i.e.*, the impulse response is truncated to the length of the unit impulse vector, `x`.

Exercise 1.1: A Simple Difference Equation

(a) Create vectors `b` and `a` that contain the coefficients of $x[n]$ and $y[n]$, respectively, in the following difference equation:

$$y[n] + 0.9y[n-2] = 0.3x[n] + 0.6x[n-1] + 0.3x[n-2] \tag{1.2}$$

(b) Calculate $y[n]$ analytically for $x[n] = \delta[n]$.

(c) Now create a unit impulse vector, `imp` of length 128. Generate the first 128 points of impulse response of the filter in (1.2). Use `comb` to plot these values as a discrete-time signal versus time, see `help comb`. It may help to plot just the first 10 or 20 points.

Exercise 1.2: Impulse Response with `filter`

(a) Use the `filter` function to generate and plot the impulse response $h[n]$ of the following difference equation. Plot $h[n]$ in the range of $-10 \leq n \leq 100$.

$$y[n] - 1.8\cos\left(\tfrac{\pi}{16}\right) y[n-1] + 0.81y[n-2] = x[n] + \tfrac{1}{2}x[n-1] \tag{1.3}$$

(b) Also determine the impulse response analytically and confirm your results.

Exercise 1.3: Natural Frequencies

The impulse response of a difference equation such as (1.2) or (1.3) is known to be composed of several natural frequencies. These natural frequencies are determined by the feedback coefficients $\{a_k\}$. Each root (p_k) of the characteristic polynomial gives rise to a term in the output of the form $p_k^n u[n]$.

$$A(z) = 1 + \sum_{k=1}^{N_a} a_k z^{-k} \tag{1.4}$$

(a) For the difference equation (1.3), determine the natural frequencies; see `help roots` for the MATLAB function to extract polynomial roots. If the roots are complex, then the natural frequency response will be a complex exponential. Plot the real and imaginary parts of the signals $p_k^n u[n]$.

(b) For a second-order difference equation, such as (1.3), there are two natural frequencies, and it these are distinct, then the causal impulse response must be of the form:

$$h[n] = \left(\alpha p_1^n + \beta p_2^n\right) u[n] \tag{1.5}$$

where p_1 and p_2 are the natural frequencies. In part (a), these natural frequencies were determined.

For example, suppose that for a second order difference equation, p_1 and p_2 have been obtained using `roots` and two values of $h[n]$ are calculated by direct recursion of the difference equation. Write a pair of simultaneuos equations for α and β. Solve these equations using MATLAB's backslash operator, \, for the difference equation (1.3). Using this result, generate $h[n]$ from (1.5) and verify that it matches the result obtained in Exercise 1.2.

Project 2: Steady-State Response

For certain inputs the output will take a simple form. The most notable of these is the class of complex sinusoidal inputs, in which case we can find the "steady-state" response of the difference equation.

Exercise 2.1: Step response

In most cases, the natural response of the difference equation decays away to zero as n increases, because the roots $\{p_k\}$ are inside the unit circle, i.e., $|p_k| < 1$. Therefore, when the input signal is a constant for all $n \geq 0$, the output signal for large n is due entirely to the input. In fact, the output becomes a constant in this case.

(a) For the system in (1.3), find the response to a step function input of amplitude 3, i.e. $x[n] = 3u[n]$. Use a long enough section of the input signal, so that the output from `filter` is nearly constant. This length can be estimated by considering the size of $|p_k|^n$ versus n. Plot the step response and determine the constant level (G_0) of the output as $n \to \infty$.

(b) The constant level determined in the previous part is the steady-state response. Its precise value can be calculated by observing that both $y[n]$ and $x[n]$ become constants in the limit $n \to \infty$. Thus, $\lim\limits_{n\to\infty} y[n] = G_0$ and $x[n] = 3$. Use these facts in (1.3) to determine G_0.

(c) The variable part of the total response is called the transient response. Determine the transient response $y_t[n] = y[n] - G_0$ and plot it for $0 \leq n \leq 50$.

(d) Since the filter is linear, the response to a step of different amplitude is just a scaled version of the previous result. Verify that steady-state response to $x[n] = 15u[n]$ is five times that found in part (a). Explain why a similar scaling apply to the transient response.

(e) Since the unit impulse signal $\delta[n]$ is just the first difference of the step signal $u[n]$, the linearity property of the filter implies that the impulse response $h[n]$ should be the first difference of the step response $s[n]$. Verify that this property holds. In MATLAB see `help diff` for a first difference operator; be careful, because `diff` reduces the vector size by one.

Exercise 2.2: Steady-State response

The same decomposition into transient and steady-state response will also apply to a wider class of signals, namely the class of complex exponentials, $e^{j\omega_\circ n}u[n]$. In this case, the transient dies away, and the form of the output approaches $Ge^{j\omega_\circ n}u[n]$, where G is a complex constant with respect to n. Note that G does vary with $\omega_\circ$, so we could write $G(\omega_\circ)$. When $\omega_\circ = 0$ we have the case of the step response. If we take the real or imaginary part of the complex exponential we have the response to a sinusoid.

(a) For the system in (1.3), plot the real and imaginary parts of the response to the complex exponential, $x[n] = e^{jn\pi/3}u[n]$. Use a long enough section of the input signal, so that the transient has died out. Determine the limiting value of the complex amplitude $G(\pi/3)$ of the output as $n \to \infty$. Make sure that you account for the complex exponential behavior of the output.

(b) A simple derivation will yield a formula for $G(\omega_\circ)$ that is good for any $\omega_\circ$. By definition, the steady-state response is obtained as $n \to \infty$.

$$\text{INPUT} = e^{j\omega_\circ n}u[n] \qquad \Longrightarrow \qquad \lim_{n\to\infty}\left(y[n] - G(\omega_\circ)\,e^{j\omega_\circ n}\right) = 0$$

For convenience, we now drop the subscript on the frequency, and write ω in place of $\omega_\circ$. Therefore, when $x[n] = e^{j\omega n}$ we can replace $y[n]$ with $G(\omega)e^{j\omega n}$ in the difference equation:

$$G(\omega)e^{j\omega n} - 1.8\cos\left(\tfrac{\pi}{16}\right)\,G(\omega)e^{j\omega(n-1)} + 0.81\,G(\omega)e^{j\omega(n-2)} = e^{j\omega n} + \tfrac{1}{2}e^{j\omega(n-1)}$$

Complete the derivation to obtain the following formula for $G(\omega)$, the complex amplitude of the steady-state response in (1.3) at ω.

$$G(\omega) = \frac{1 + \frac{1}{2}e^{-j\omega}}{1 - 1.8\cos\left(\frac{\pi}{16}\right)\,e^{-j\omega} + 0.81e^{-j2\omega}} \tag{2.1}$$

(c) If $G(\omega)$ is then plotted versus ω, the resulting function is called the *frequency response* of the system. The notation $H(e^{j\omega})$ is used in place of $G(\omega)$, because of its connection with the DTFT and the z-transform. Evaluate the frequency response and plot the magnitude and phase of $H(e^{j\omega})$ versus ω, see `abs` and `angle` for magnitude and phase.

(d) Check the value of $H(e^{j\omega})$ at $\omega = 0$.

(e) Pick off the value of $H(e^{j\omega})$ at another frequency, say $\omega = \pi/4$, and compare to a steady-state response obtained via `filter`.

(f) The total response can again be decomposed into the sum of the steady-state response and the transient response.

$$y[n] = y_{ss}[n] + y_t[n]$$

Determine the transient response and plot it for $0 \le n \le 30$. Note that it differs from the transient response determined for the step input.

(g) Since the filter is linear, the response to a real sinusoid can be determined easily. Take the case of the cosine input. Since the cosine is

$$\cos\omega_\circ n\,u[n] = \tfrac{1}{2}\left(e^{j\omega_\circ n}u[n] + e^{-j\omega_\circ n}u[n]\right)$$

the steady-state response should be one half times the sum of the steady-state responses due to the complex exponentials at $+\omega_\circ$ and $-\omega_\circ$. Verify that this is true, by running the filter and generating the steady-state responses for the three inputs: $\cos\omega_\circ n\,u[n]$, $e^{j\omega_\circ n}u[n]$, and $e^{-j\omega_\circ n}u[n]$. Take $\omega_\circ = \pi/3$. Does the same sort of additive combination applies to the transient responses?

(h) Since the coefficients of the IIR filter (1.3) are all real-valued, an additional property holds. The steady-state response to the exponential at $-\omega_\circ$ is the conjugate of the response due to $+\omega_\circ$. Therefore, a simpler way to express the property in (g) is that the response due to the cosine is the real part of the response due to the exponential at $+\omega_\circ$; likewise, the response due to a sine input is the imaginary part. Verify that these statements are true by applying the `real` and `imag` operators in MATLAB.

Comment: While the steady-state response could be used to generate the frequency response, in most homework problems a formula is evaluated instead. However, when dealing with an unknown system, one experimental approach to measuring the frequency response is to perform steady-state measurements at different frequencies and then plot the result.

Project 3: Frequency Response for Difference Equations

Project Description

In this project, we will investigate a method for directly computing the frequency response of any LTI system that is described by a difference equation.

> Assume that the filter coefficients, a_k and b_ℓ, are known. Then the frequency response will be computed directly by feeding the coefficients of the filter into the `freqz` function.

Consider the same difference equation as in (1.3). Since the transfer function of this system is rational:

$$H(z) = \frac{1 + \frac{1}{2}z^{-1}}{1 - 1.8\cos(\frac{\pi}{16})z^{-1} + 0.81z^{-2}} \tag{3.1}$$

the `freqz` function can be used to find its frequency response, because evaluating the z-transform on the unit circle is equivalent to finding the Fourier transform

Hints

The command `[H,W] = freqz(b,a,N,'whole')` will evaluate the frequency response of a filter at `N` points, equally spaced in radian frequency around the unit circle. If you do not use the `'whole'` option, `freqz` will only use the upper half of the unit circle (from 0 to π in frequency), which is sufficient for filters with real coefficients. The output vectors `H` and `W`, will return `N` frequency response samples (in `H`) and `N` equally-spaced values of ω from 0 to 2π or 0 to π (in `W`).

Exercise 3.1: Frequency Response with `freqz`

For the difference equation (1.3), do the following frequency domain computations:

(a) Make plots of the magnitude and phase reponses, with 512 frequency samples around the entire unit circle. For instance, use `plot(W,abs(H))` or `plot(W,angle(H))`.

(b) Now redo the frequency response using only the upper half of the unit circle (ω ranges from 0 to π). This is sufficient because of the symmetries in the magnitude and phase response, which you should have observed in part (a).

(c) Specify the type of filter defined by this difference equation: highpass, lowpass, bandpass, or bandstop.

Exercise 3.2: Experimentation

You are now encouraged to experiment with other difference equations to see what types of filters you can create. For example, investigate the following third-order difference equation and determine what type of filter it defines.

$$y[n] + .13y[n-1] + .52y[n-2] + .3y[n-3] = .16x[n] - .48x[n-1] + .48x[n-2] - .16x[n-3]$$

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{Fourier Transform: DTFT}}$

Fourier Transform: DTFT

Overview

This set of projects will introduce *basic* properties of the discrete-time Fourier transform (DTFT). Two completely different cases are treated. The first deals with finite-length signals, for which the DTFT can be evaluated exactly. The second case involves infinite-length signals, which must have a special form to be evaluated; namely, exponential signals which have rational z-transforms.

The Fourier representation of a signal via the forward and inverse DTFT is a key part of signal analysis. Equations (0.1) and (0.2) are the analysis and synthesis equations, respectively.

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n} \tag{0.1}$$

$$x[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\omega})e^{j\omega n}\,d\omega \tag{0.2}$$

Likewise, the frequency response, which is the DTFT of the impulse response, provides a concise description of a LTI system when used for filtering. The DTFT $X(e^{j\omega})$ is a periodic complex-valued function of ω. The period is always 2π, and the fundamental period is usually chosen to be the domain $[-\pi, \pi)$. In the context of MATLAB where computability is an issue, the DTFT presents two problems:

(1) its definition is valid for <u>infinitely</u> <u>long</u> signals, and
(2) it is a function of a <u>continuous</u> variable, ω.

The first point is only a problem to the extent that any signal/vector in MATLAB must be finite in length. Thus we have the problem that it is not really possible to use MATLAB to compute the DTFT of an infinitely long signal. One notable exception is when we can derive an analytic form for the transform and just evaluate it, as in the case of $x[n] = a^n/, u[n]$ which has a rational DTFT.

The second issue presents a frequency sampling problem. The best that we can do with MATLAB is evaluate the DTFT on a finite grid of points. We can usually choose enough frequencies, so that our plots will give a smooth approximation to the true DTFT. The choice that is best for computation is a set of evenly-spaced frequencies over the interval $(-\pi, \pi]$, or $[0, \pi]$ for conjugate-symmetric transforms. With such sampling, the forward DTFT formula (0.1) becomes:

$$X(e^{j\omega_k}) = X(e^{j2\pi k/N}) = \sum_{n=0}^{L-1} x[n]e^{-j(2\pi k/N)n} \qquad \text{for} \quad k = 0, 1, \ldots, N-1 \tag{0.3}$$

The periodicity of the DTFT means that the values for $-\pi \le \omega < 0$ are those for $k > N/2$. This formula (0.3) is *computable* because it is a sum over *finite* limits, evaluated at a *finite* number of frequencies, $\omega_k = 2\pi k/N$. Since the signal length must be finite ($0 \le n < L$), a case such as $x[n] = a^n u[n]$ is not covered by this summation form.

When $N = L$ this computable formula (0.3) is just an N-point discrete Fourier transform (DFT), but for the moment we ignore the DFT nature of things and concentrate on computing samples of the DTFT. Details of the DFT will be treated in other packets.

When sampling the DTFT, there is no requirement that N be equal to L, although it is convenient because the computation is usually carried out via the DFT (with the FFT algorithm). Indeed, if $N > L$, then we only have to imagine that $x[n]$ is zero-padded to use an N-point DFT. The case where $N < L$ is much more troublesome. Correct application of the FFT in this case requires *time-aliasing* of $x[n]$ prior to the N-point DFT computation. For now, you should always make sure you evaluate the DTFT at many more frequencies than there are points in the original time sequence, i.e. $N \geq L$.

Background Reading

This basic material about the DTFT can be found in the introductory chapters of any book on DSP.

Project 1: Computing the DTFT: Finite-Length Signals

In this project, we consider the case of finite-length signals. This will take advantage of the `dtft` function defined below. In particular, this project deals with pulses and their DTFTs, because these sorts of signals are the easiest examples of using the DTFT. Most books use the rectangular pulse as the first example of deriving a DTFT.

Hints

We need two functions for computing the DTFT. The MATLAB function `freqz` will suffice for the infinite signal case, but a new function will be needed to compute the DTFT of a finite-length signal. It should be called `dtft(h,N)`, and is essentially a layer that calls `fft(h,N)`.

```
function [H,W] = dtft( h, N )
%DTFT   calculate DTFT at N equally spaced frequencies
%   usage:   H = dtft( h, N )
%      h: finite-length input vector, whose length is L
%      N: number of frequencies for evaluation over [-pi,pi)
%             ==> constraint: N >= L
%
%      H: DTFT values (complex)
%      W: (2nd output) vector of freqs where DTFT is computed
%
L = length(h);
if( N < L )
   error('DTFT: # data samples cannot exceed # freq samples')
end
W = (2*pi/N) * [ 0:(N-1) ];
W = fftshift(W);
W(1:(N/2)) = W(1:(N/2)) - 2*pi;   % <--- [-pi,pi)
H = fftshift( fft( h, N ) );  %<--- move negative freq components
```

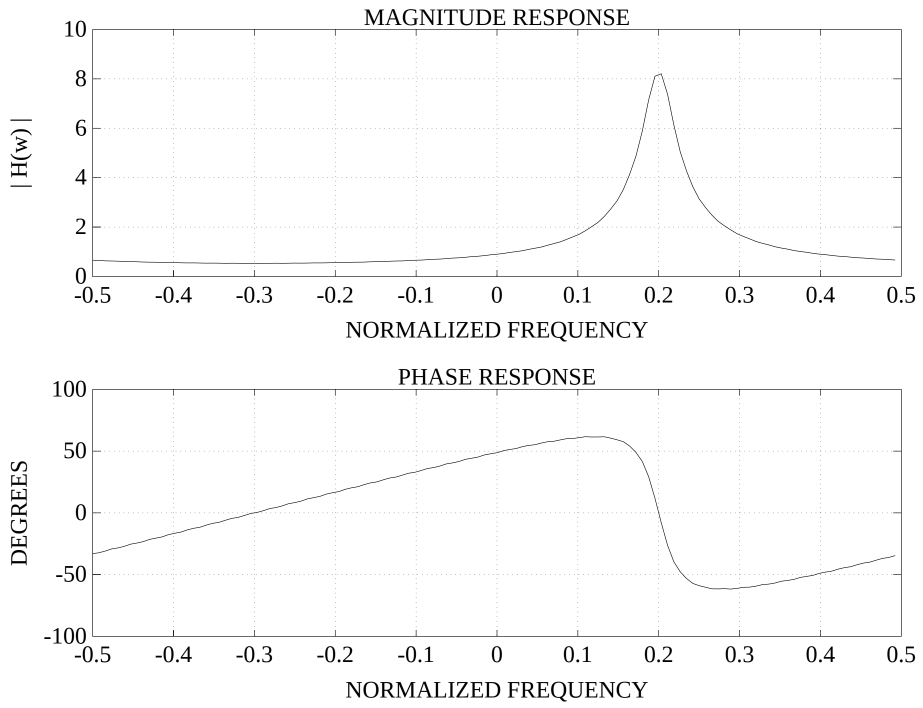


Figure 1: Two-panel frequency domain plot made via `subplot`.

Note that you don't have to input the signal length L, because it can be obtained by finding the length of the vector `h`. Furthermore, since the DTFT is periodic, the region from $\omega = \pi$ to 2π is actually the negative frequency region, so the transform values just need to be re-ordered. This is accomplished with the MATLAB function `fftshift` which exchanges the upper and lower halves of a vector. If `Hrot = fftshift(H)` is applied to the DTFT vector, then the $[-\pi, \pi]$ plot can be produced by noting that `Hrot(1)` is the frequency sample for $\omega = -\pi$.

When plotting in the transform domain it would be best to make a two-panel `subplot` as shown in Fig. 1. The MATLAB program that produces Fig. 1 is given below:

```
%--- example of calculating and plotting a DTFT
%---
format compact, subplot(111)
a = 0.88 * exp(sqrt(-1)*2*pi/5);
nn = 0:40;  xn = a.^nn;
[X,W] = dtft( xn, 128 );
subplot(211), plot( W/2/pi, abs(X) ); grid, title('MAGNITUDE RESPONSE')
   xlabel('NORMALIZED FREQUENCY'), ylabel('| H(w) |')
subplot(212), plot( W/2/pi, 180/pi*angle(X) ); grid
   xlabel('NORMALIZED FREQUENCY'), ylabel('DEGREES')
   title('PHASE RESPONSE')
```

Exercise 1.1: DTFT of a pulse

The finite-length pulse is always used as a prime example of evaluating the DTFT. Suppose that the rectangular pulse $r[n]$ is defined by:

$$r[n] = \begin{cases} 1 & 0 \le n < L \\ 0 & \text{elsewhere} \end{cases} \tag{1.1}$$

(a) Show that the DTFT of $r[n]$ is given by the mathematical formula:

$$R(e^{j\omega}) = \frac{\sin \frac{1}{2}\omega L}{\sin \frac{1}{2}\omega} \cdot e^{-j\omega(L-1)/2} \tag{1.2}$$

The first term in this transform has a special form that occurs quite often in conjunction with the DTFT; it will be called the *aliased sinc* function:[1]

$$\text{asinc}(\omega, L) = \frac{\sin \frac{1}{2}\omega L}{\sin \frac{1}{2}\omega} \tag{1.3}$$

(b) Use the function `dtft` to evaluate the DTFT of a 12-point pulse. Make a plot of the DTFT versus ω over the range $-\pi \le \omega < \pi$. Plot the real and imaginary parts separately, but notice that these plots are not terribly useful. Instead plot the magnitude of the DTFT (see `abs` in MATLAB). To make the plot appear smooth, choose a number of frequency samples that is 5 to 10 times the pulse length. Experiment with different numbers of frequency samples. When plotting, be careful to label the frequency axis correctly for the variable ω.

(c) Notice that the zeros of the asinc function are at regularly spaced locations. Repeat the DTFT calculation and magnitude plot for an odd-length pulse, say $L = 15$. Again, check the zero locations and note the peak height.

(d) Determine a general rule for the regular spacing of the zeros of the asinc function and its DC value.

Exercise 1.2: M-file for asinc

Write a MATLAB function `asinc(w,L)` that will evaluate $\text{asinc}(\omega, L)$ on a frequency grid directly from the formula (1.3). The function should have two inputs: a length `L`, and a vector of frequencies `w`. It must check for division by zero as happens for $\omega = 0$.

Directly evaluate the "aliased sinc" formula (1.2) for the DTFT of a pulse. Plot the magnitude, and save this plot for comparison to the result obtained with `dtft`.

Exercise 1.3: Phase unwrapping in the DTFT

Since the Fourier transform (0.1) is a complex-valued quantity,not only the magnitude, but also the phase is of interest, see `abs` and `angle` in MATLAB.

[1]This is also called the *Dirichlet kernel.*

(a) Make a plot of the phase versus ω. In the formula above (1.2) the phase appears to be linear with a slope of $-\frac{1}{2}(L-1)$ versus ω. Check your plot. If the phase plot appears incorrect, consider the following: the angle is evaluated modulo–2π, so 2π jumps in the phase occur when the phase is "wrapped" into the $[-\pi, \pi]$ interval by the arctangent computed inside `angle`.

    In addition, the asinc term is not always positive, so there are additional jumps of π at the zeros of $R(e^{j\omega})$ which are the zeros of the numerator: $\omega = 2\pi k/L$.

(b) An "unwrapped" version of the phase can be produced. See the MATLAB function `unwrap`. Use this function to compute and plot the unwrapped phase of $R(e^{j\omega})$. Now it should appear linear, except for jumps of π. If not, the DTFT sampling may be too low; unwrapping requires a rather dense sampling of the frequency axis.

Project 2: DTFT Symmetries

Finite-length signals are often "centered" so as to have symmetries that make the DTFT simpler. For example, if the rectangular pulse is shifted to be even about the $n = 0$ point, then the DTFT will be a purely real and even function versus ω. In this project, the six major types of symmetry will be reviewed and illustrated by various examples.

Hints

It will be necessary to modify the `dtft` function so that it can accept an argument that defines the starting point on time axis for the signal $x[n]$. Ordinarily, the default starting time is assumed to be $n = 0$ by virtue of the definition of the DTFT (0.3).

Another skill that will be needed for this project is plotting the DTFT to exhibit symmetries. Ordinarily, the ω_k samples run from $\omega = 0$ to 2π. However, when studying symmetry the DTFT plot should be made from $-\pi$ to $+\pi$, which is the range returned by the `dtft` function, but not by `freqz`.

When checking a transform for the property of purely real or purely imaginary, the part of the signal that is expected to be zero might not be exactly zero. There might be some extremely small values due to roundoff noise in the computation of the DFT. If these values are on the order of 10^{-13} or 10^{-14}, then it is safe to assume that the cause is rounding in the double precision arithmetic of MATLAB (see `help eps`). However, numbers on the order of 10^{-9} are probably not due to rounding, unless a very large number of computations have been performed on the signals.

When checking whether a transform is even or odd, it would be convenient to have an operator that will flip the transform. In other words, we need a MATLAB function that corresponds to the mathematical operation of flipping the frequency axis: $Y(e^{j\omega}) = X(e^{-j\omega})$. The following function will implement this flip on a pair of vectors, `H` for the transform values and `W` for the frequency axis.

```
function  [ G, Wflipped ] = flipDTFT( H, W )
%FLIPDTFT   flip the DTFT:  G(w) = H(-w)
%     usage:
%           [ G, Wflipped ] = flipDTFT( H, W )
%
%            H = DTFT values (complex)
%             W = frequency samples
%             G = DTFT values
%      Wflipped = flipped frequency domain
%                       lies within [-pi,pi)
%
N = length(H);   %<--- works only for vectors !!!
Wflipped = -W(N:-1:1);
G = H(N:-1:1);
%---
%--- now get everything back into the [-pi,pi) interval
%---    assume that W was monotonically increasing
%---    so Wflipped is also increasing !
%---
jkl = find( Wflipped(:)' < -pi );
if( ~isempty(jkl) )
   kk = [ (length(jkl)+1):N  jkl ];
   Wflipped(jkl) = Wflipped(jkl) + 2*pi;
   Wflipped = Wflipped(kk);
   G = G(kk);
end
jkl = find( Wflipped(:)' >= (pi-100*eps) );
if( ~isempty(jkl) )
   kk = [ jkl  1:(jkl(1)-1)  ];
   Wflipped(jkl) = Wflipped(jkl) - 2*pi;
   Wflipped = Wflipped(kk);
   G = G(kk);
end
```

Exercise 2.1: Zero-phase signals

Working with zero-phase signals in MATLAB presents a difficulty—the `dtft` function from project #1 assumes that the signal starts at $n = 0$, but any zero-phase signal must be symmetric around $n = 0$. One way to address this problem is to create a modified form of `dtft` that has an additional input argument to specify the starting index of the signal. This starting index can then be used to modify the DTFT output according to the "shifting property" of the DTFT:

$$x[n - n_0] \quad \overset{\text{DTFT}}{\longleftrightarrow} \quad e^{-j\omega n_0} X(e^{j\omega}) \tag{2.1}$$

In other words, the DTFT must be multiplied (pointwise) by a complex exponential $\exp(-j\omega n_0)$ to undo the effect of the time shift by n_0.

(a) Create a new function `dtft_n0( x, n0, N )` to implement the time shift feature by adding the argument `n0`. This should amount to a minor modification of the `dtft` function.

(b) Test `dtft_n0` by taking the DTFT of a 21-point pulse that starts at $n = -10$. The resulting transform should be a purely real and even function.

(c) Plot the real part of the DTFT and compare to the DTFT magnitude over the domain from $-\pi$ to $+\pi$.

(d) Verify that the imaginary part is zero and that the phase is equal to either 0 or π.

Note that this symmetry only works for odd-length pulses; if the length is even, there will always be a residual phase term corresponding to a half-sample delay.

Exercise 2.2: Triangular pulse

Another simple signal is the symmetric triangular pulse:

$$\Delta[n] = \begin{cases} L - n & 0 \leq n < L \\ L + n & -L < n < 0 \\ 0 & \text{elsewhere} \end{cases}$$

The length of this signal is $2L - 1$, and it can be formed by convolving two L-point rectangular pulses. As a result the DTFT is an asinc-squared function. No phase term is involved, since this is a symmetric signal.

(a) Make a plot of a 21-point triangular pulse over the domain $-20 \leq n \leq 20$. Then compute its DTFT with the function `dtft_n0` and plot the result over $-\pi \leq \omega < \pi$.

(b) For comparison plot the squared magnitude of the DTFT of an 11-point rectangular pulse. It might be easier to make the comparison by plotting both on a log-magnitude (dB) scale.

Exercise 2.3: Symmetries in the DTFT

There are many symmetry properties in the time and frequency domains. One set deals with purely real, or purely imaginary signals, another with even and odd signals. For example, the DTFT of an even signal is even. Each of the following parts concentrates on one type of symmetry. Verification of the symmetry in the frequency domain can be done by plotting the real and imaginary parts of the DTFT (or magnitude and phase), and by using the function `flipDTFT` to examine $X(e^{-j\omega})$.

(a) The DTFT of a real signal is conjugate symmetric, $X^*(e^{j\omega}) = X(e^{-j\omega})$; in other words, conjugating the DTFT is the same as flipping it. For the signal $x[n] = (0.9)^n \cos(2\pi n/\sqrt{31})$, for $0 \leq n < 21$, plot the DTFT (magnitude and phase) and verify that it is indeed conjugate symmetric.

(b) If the signal is purely imaginary, then the DTFT will be conjugate anti-symmetric. Using $x[n]$ from the previous part, define $y[n] = jx[n]$, and display its DTFT $Y(e^{j\omega})$. Use the `flipDTFT` function to compare the flipped version of $Y(e^{j\omega})$ to the conjugated version, and then check the conjugate anti-symmetric property.

(c) An even time signal transforms to an even function of frequency. Prove that the chirp signal $v_e[n] = \exp(j2\pi n^2/25)$ over $-30 < n < 30$ is even. Then compute and disply its DTFT to verify that the transform is also even with respect to ω.

(d) For the odd signal $v_o[n] = n$ over $-20 < n < 20$ verify that the DTFT is also odd.

(e) The properties of real/imaginary and even/odd can be combined. Define a signal that is both purely imaginary and odd. What symmetry properties will its DTFT have? Verify by plotting the transform and by using `flipDTFT`.

Project 3: DTFT of Infinite-Length Signals

It is not usually possible to *compute* the DTFT of an infinitely long signal. However, there is one important class for which the computation is easy. This is the class of exponential signals, where the DTFT is a rational function of $e^{-j\omega}$.

$$H(e^{j\omega}) = \frac{B(e^{j\omega})}{A(e^{j\omega})} = \frac{\sum_{\ell=0}^{Q} b_\ell e^{-j\omega\ell}}{\sum_{k=0}^{P} a_k e^{-j\omega k}}$$

The exponential signal $h[n] = a^n u[n]$ is one member of this class, but cannot be dealt with using the `dtft` function presented above. On the other hand, its DTFT is readily derived as a formula:

$$h[n] = a^n u[n] \qquad \overset{\text{DTFT}}{\longleftrightarrow} \qquad H(e^{j\omega}) = \sum_{n=0}^{\infty} a^n u[n] e^{-j\omega n} = \frac{1}{1 - ae^{-j\omega}} \qquad \text{if } |a| < 1 \quad (3.1)$$

Using the rational form of $H(e^{j\omega})$, it is easy to evaluate this DTFT over a set of frequency samples. The denominator function, $1 - ae^{-j\omega}$, is evaluated at the set of discrete frequencies, and is then divided into the numerator which is the constant 1. This strategy extends to any DTFT that is a rational function of $e^{-j\omega}$. Furthermore, the evaluation of both the numerator and denominator can be done with the FFT, because both are, in effect, finite length signals. Thus, evaluating the rational function amounts to doing two `dtft` calculations. This frequency-domain computation is embodied in the MATLAB `freqz` function.

Hints

The MATLAB function `freqz` is so named because it is applicable to rational z-transforms.

```
[ HH, WW ] = freqz( b, a, N, 'whole' )
```

Like `dtft`, `freqz` has two outputs: the transform values (`HH`), and the frequency grid (`WW`). The fourth input argument is optional, but when set to `'whole'` the output vector `WW` which specifies the frequency grid will range from $\omega = 0$ to $\omega = 2\pi$. If the fourth argument is omitted, the frequency grid consists of `N` equally spaced points over the range $0 \le \omega < \pi$.

Exercise 3.1: Exponential signal

For the signal $x[n] = (0.9)^n u[n]$, compute the DTFT $X(e^{j\omega})$ using `freqz`.

(a) Make a plot of both the magnitude and the phase versus ω over the range $-\pi \le \omega < \pi$. This will require a shift of the `[X, W]` vectors returned from `freqz`. Explain why the magnitude is even and the phase is an odd function of ω.

(b) Derive formulae for the magnitude and phase from equation (3.1) for the first-order system.

(c) Compute the magnitude and phase by a direct evaluation of the formulae, and compare to the results from `freqz`.

Exercise 3.2: Complex exponential

If we take $a = z_0 = re^{j\theta}$ to be a complex number in (3.1), the same transform is applicable. This case is important because we can develop some insight into how the magnitude (r) and phase (θ) of the complex number affect the DTFT.

(a) Take $z_0 = 0.95e^{j3\pi/11}$, and plot $x[n] = z_0^n u[n]$, for $0 \leq n \leq 30$. Plot both the real and imaginary parts versus n together in a two panel `subplot`.

(b) Again with $z_0 = 0.95e^{j3\pi/11}$, compute the DTFT and plot the magnitude versus ω. Note where the peak of the magnitude response lies as a function of ω. Relate the peak location to the polar representation of z_0.

(c) If the angle of z_0 were changed to $\theta = 3\pi/5$, sketch the DTFT magnitude that you would expect. Verify by making a plot from a `freqz` calculation.

(d) Change the magnitude $r = |z_0|$ and redo the DTFT plot. Use four values: $r = 0.975$, 0.95, 0.9 and 0.8. Notice that the bandwidth of the peak changes, as long as $|z_0|$ is close to one. The peak magnitude also changes. Try to develop a simple formula that relates the bandwith and peak height to r.

Exercise 3.3: Decaying Sinusoid

The complex-valued signal $x[n] = z_0^n u[n]$ is quite useful in representing real-valued signals that are decaying sinusoids of the form:

$$y[n] = \Re e\{Gz_0^n u[n]\} = A\, r^n \cos(\theta n + \phi) u[n] \tag{3.2}$$

where $G = Ae^{j\phi}$ and $z_0 = re^{j\theta}$. The resulting DTFT, $Y(e^{j\omega})$, is a rational function with a second order denominator. Its transform can be derived several ways, but it is informative to do so using some properties of the DTFT. In particular, the conjugate property states that the DTFT of a complex-conjugated signal is the flipped and conjugated version of the original transform.

$$x[n] \overset{\text{DTFT}}{\longleftrightarrow} X(e^{j\omega}) \qquad \Longrightarrow \qquad x^*[n] \overset{\text{DTFT}}{\longleftrightarrow} X^*(e^{-j\omega}) \tag{3.3}$$

Thus, if we let $x[n] = Gz_0^n u[n]$, its DTFT is

$$X(e^{j\omega}) = \frac{G}{1 - z_0 e^{-j\omega}} = \frac{Ae^{j\phi}}{1 - re^{-j(\omega-\theta)}} \tag{3.4}$$

so we can apply the conjugate property to $y[n] = \Re e\{x[n]\} = \frac{1}{2}\{x[n] + x^*[n]\}$ to get the DTFT:

$$\begin{aligned} Y(e^{j\omega}) &= \tfrac{1}{2}[X(e^{j\omega}) + X^*(e^{-j\omega})] \qquad (3.5)\\ &= \frac{\frac{1}{2}Ae^{j\phi}}{1 - re^{-j(\omega-\theta)}} + \frac{\frac{1}{2}Ae^{-j\phi}}{1 - re^{+j(-\omega-\theta)}} \\ &= \tfrac{1}{2}A\left(\frac{e^{j\phi}(1 - re^{-j(\omega+\theta)}) + e^{-j\phi}(1 - re^{-j(\omega-\theta)})}{(1 - re^{-j(\omega-\theta)})(1 - re^{-j(\omega+\theta)})}\right) \\ Y(e^{j\omega}) &= A\left(\frac{\cos\phi - r\cos(\theta+\phi)e^{-j\omega}}{1 - 2r\cos\theta\, e^{-j\omega} + r^2 e^{-j2\omega}}\right) \qquad (3.6) \end{aligned}$$

Obviously, these equations start to get very messy, but the simple expedient of using complex arithmetic is easier than plugging (3.2) into the DTFT summation.

In this exercise, we will use MATLAB to calculate the DTFT via this conjugate trick and compare to direct evaluation with `freqz`. The test signal is $y[n] = 3(0.95)^n \cos(2\pi n/7 + \pi/3)\, u[n]$. For both, plot the resulting magnitude and phase of $Y(e^{j\omega})$ versus ω. Verify that the two approaches give the same result.

(a) First, substitute the parameters of the decaying sinusoid $y[n]$ directly into the formula (3.6) for $Y(e^{j\omega})$; then use `freqz(b, a, n)` with the appropriate coefficient vectors `a` and `b` to get the frequency samples.

(b) Express $y[n]$ as the sum of two complex exponentials. Take the DTFT of one complex exponential (via `freqz`), and then apply the conjugate property (3.3) in the transform domain (3.5) to get the second term in the answer.

Project 4: Windowing for the DTFT

In this project, two properties of the DTFT are illustrated: the modulation property and the windowing property. The modulation property is, in effect, a special case of windowing where the frequency domain convolution reduces to a simple frequency shift.

Exercise 4.1: Modulation Property

Many DTFT properties have useful interpretations and applications. One of these is the (complex) modulation property, which finds application in communications and radar. If a signal $x[n]$ is multiplied by a complex sinusoid, $e^{j\omega_0 n}$, then the result in the transform domain is a frequency shift of ω_0; $X(e^{j\omega})$ becomes $X(e^{j[\omega-\omega_0]})$.

(a) Demonstrate this property with the rectangular pulse—take the pulse length to be $L = 21$ and pick $\omega_0 = 2\pi/\sqrt{31}$. Plot the result from `dtft`. Verify that the peak of the DTFT magnitude (an asinc function) has moved to $\omega = \omega_0$. Try a value of ω_0 larger than 2π to exhibit the periodicity of the DTFT.

(b) Repeat the experiment, but multiply the pulse by a cosine signal at the same frequency. This is just double-sideband AM, and it involves only real operations.

Exercise 4.2: Windowing gives Frequency-Domain Convolution

The windowing property of the DTFT states that the DTFT of the (pointwise) product of two time signals (`.*` in MATLAB) is the periodic frequency domain convolution of their Fourier transforms:

$$y[n] = x[n] \cdot w[n] \qquad \overset{\text{DTFT}}{\longleftrightarrow} \qquad Y(e^{j\omega}) = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\theta})W(e^{j[\omega-\theta]})d\theta \tag{4.1}$$

The frequency-domain convolution will "smear" the true DTFT, depending on the exact nature of $W(e^{j\omega})$. Even when there appears to be no windowing, the very fact that the signal is finite means that a rectangular window has been applied. In this case, the window transform is an aliased sinc function.

(a) It should not be possible to compute the frequency domain convolution in (4.1), because it involves an integral. However, there is one case where the frequency domain result can be obtained as a formula—when the unwindowed signal is a complex sinusoid, e.g., $x[n] = e^{j\theta_0 n}$, for all n. Then $X(e^{j\omega})$ is an impulse in frequency, so the convolution evaluates to $Y(e^{j\omega}) = W(e^{j[\omega-\theta_0]})$. Of course, this is just the modulation property. The observed transform takes on the shape of the window's DTFT, shifted in frequency.

Generate a windowed sinusoid with $\theta_0 = 2\pi/\sqrt{31}$:

$$x[n] = r[n] \cdot e^{j\theta_0 n}$$

where $r[n]$ is the rectangular window of length L. The rectangular window $r[n]$ can be created via the function `ones` or `boxcar`. Plot the DTFT, $X(e^{j\omega})$, and note that the peak has been shifted to $\omega = \theta_0$ and that it has the shape of an aliased sinc function.

(b) The following window function is called the von Hann (or `hanning`) window:

$$w[n] = \begin{cases} \frac{1}{2} - \frac{1}{2}\cos\frac{2\pi}{L}n & 0 \le n \le L \\ 0 & \text{elsewhere} \end{cases}$$

Apply a 32-point Hann window to a sinusoid of frequency $\omega_\circ = 2\pi/\sqrt{31}$. Plot the time signal and then calculate its DTFT and plot the magnitude response.

(c) The DTFT of the Hann window can be written in terms of three shifted asinc functions. This is done by viewing $w[n]$ as a rectangular window applied to the signal $\frac{1}{2} - \frac{1}{2}\cos\frac{2\pi}{L}n$, and then using the modulation property.

Make a plot of the magnitude response of the Hann window $W(e^{j\omega})$ for $L = 32$. Explain how $W(e^{j\omega})$ is formed from $R(e^{j\omega})$, the DTFT of the rectangular window. Take into account the fact that both $W(e^{j\omega})$ and $R(e^{j\omega})$ are complex-valued.

(d) Sometimes the following formula is mistakenly used for the von Hann window:

$$v[n] = \tfrac{1}{2} + \tfrac{1}{2}\cos(2\pi n/L) \qquad \text{for} \;\; n = 0, 1, 2, \ldots, L$$

The change from a minus sign to a plus is significant.

Plot $v[n]$ vs. n over the domain $-10 \le n \le L + 10$, with $L = 32$. Notice the discontinuities at the end points, 0 and L. Next, plot its DTFT magnitude; and compare to the result for the Hann window. Explain how the same three asinc terms can be combined to give such a different answer for the DTFT $V(e^{j\omega})$. Take into consideration that the terms being summed are complex, and all have a phase component.

Exercise 4.3: Convergence to true DTFT

There is one situation that has not yet been dealt with: namely, the case of taking the DTFT of a finite portion of an infinite-length signal. This case arises often in practice because we can usually record (and store) only a small portion of a signal for analysis. Nonetheless, we would like to deduce the signal's true frequency content from a DTFT of the finite segment. In this exercise, we will compare a windowed DTFT of an exponential to the true DTFT.

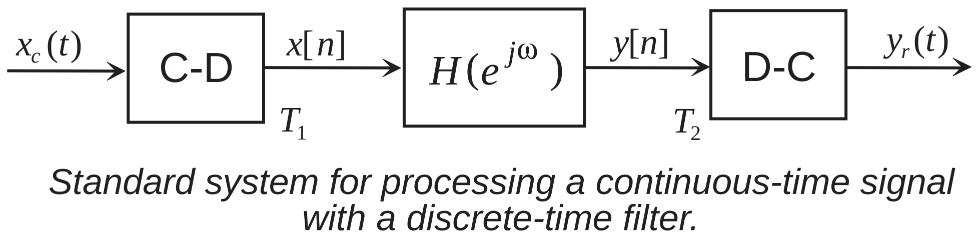

$x_c(t)$
C-D
$x[n]$
T_1
$H(e^{j\omega})$
$y[n]$
T_2
D-C
$y_r(t)$
Standard system for processing a continuous-time signal with a discrete-time filter.

Figure 2: Standard system for implementing an analog filter by means of a cascade of an A/D converter, digital filter, and D/A converter.

The `dtft` function given previously is sufficient to perform the DTFT on any finite-length signal. An easy example of windowing is obtained by taking the first L points of the infinitely long exponential, $x[n] = a^n u[n]$. As the segment length increases, the resulting DTFT should converge to the analytic form given previously for $X(e^{j\omega})$.

(a) Using the choice $a = 0.977$, plot the log magnitude of the DTFT for several different signal lengths: $L = 32$, 64, 128 and 256. Limits on available memory might stop this experiment. Plot all four results together with a four panel subplot. Overlay each with a plot of the "true" DTFT for the infinite-length signal as a dashed line.

(b) Comment on the changes observed as the section length L is increased. Explain the differences for increasing L in terms of a window transform $W(e^{j\omega})$ operating on the true spectrum.

Project 5: Frequency response of a Notch Filter

The DTFT of an impulse response is, in fact, the frequency response of the system. Therefore, the DTFT can be used to describe the filtering nature of a system, and also its steady-state behavior.

Exercise 5.1: Notch Filter Example

A notch filter attempts to remove one particular frequency. Suppose that a bandlimited continuous-time signal is known to contain a 60-Hz interference component, which we want to remove by processing with the standard system (Fig. 2) for filtering a continuous-time signal with a discrete-time filter.

(a) Assume that the value of the sampling period is $T = 1$ millisec. What is the highest frequency that the analog signal can contain if aliasing is to be avoided?

(b) The discrete-time system to be used has frequency response:

$$H(e^{j\omega}) = \frac{[1 - e^{-j(\omega-\omega_0)}][1 - e^{-j(\omega+\omega_0)}]}{[1 - 0.9e^{-j(\omega-\omega_0)}][1 - 0.9e^{-j(\omega+\omega_0)}]} \tag{5.1}$$

Sketch the magnitude and phase of $H(e^{j\omega})$. Pick a trial value of $\omega_0 = 2\pi/5$ and use MATLAB to do the "sketch."

(c) What value should be chosen for ω_0 to eliminate the 60-Hz component? Will the gain at other frequencies be equal to one?

(d) Make a MATLAB plot of the frequency response (magnitude only) using the value of $\omega_\circ$ found in part (c).

(e) Generate 150 samples of a 60-Hz sine wave sampled at $f_s = 1/T_s = 1000$ Hz. Use the function `filter` to process this input signal with the system from part (b) and the value of ω_0 from part (c). Display the output signal to illustrate that the filter actually removes the 60 Hertz sinusoid.

(f) Since the DTFT is a frequency response, it describes the steady-state behavior of the filter. Thus you should observe a "transient" response before the zero of the filter at 60 Hertz completely rejects the input. Measure the duration of this transient (in milliseconds) from the beginning of the signal until a point where the output is less than 1% of the input signal amplitude.

Exercise 5.2: Steady-State response of a filter

In this project, a sinusoidal input will be filtered and the resulting output will be split into two parts: the transient response and the steady-state response. The relationship of the steady-state response to the frequency response function $H(e^{j\omega})$ will be explored.

(a) Use the notch filter (5.1) with $\omega_0 = \pi/5$. Find the `b` and `a` coefficients for the notch filter. Make plots of the magnitude and phase responses of the filter.

(b) Pick a particular frequency, say $\omega = \omega_i = \pi/4$. Compute the numerical values of the magnitude and phase of $H(e^{j\omega_i}) = |H(e^{j\omega_i})| \cdot e^{-j\phi(\omega_i)}$.

(c) Use these numerical values to generate a plot of the "true" steady-state output

$$y_{ss}[n] = |H(e^{j\omega_i})| \cdot \cos(\omega_i n - \phi(\omega_i))$$

by directly evaluating the cosine formula. Let the range of n be $0 \leq n \leq 50$.

(d) Generate the filter output $y[n]$ by using the MATLAB `filter` function when the input signal is the cosine: $v[n] = \cos(\omega_i n)$, $n = 0, 1, 2, \ldots, N$. This output is the sum of both the transient and steady-state responses.

(e) Plot the two signals $y[n]$ and $y_{ss}[n]$, and compare them for the region where n is large. Since the system is stable, these two should be nearly the same, after all transients die out. The steady-state response is, therefore, a reasonable way to measure the frequency response at $\omega = \omega_i$, because the magnitude and phase of the output $y[n]$ will approach the magnitude and phase of $H(e^{j\omega})$, as $n \to \infty$.

(f) Find the transient signal $y_t[n] = y[n] - y_{ss}[n]$ by subtracting the steady-state signal from the total output, and the plot the transient.

(g) The steps above should be repeated for several different frequencies, including one very close to the notch, where the transient response will be much larger than the steady state.

Computer-Based Exercises for Signal Processing

$\boxed{\text{Group Delay}}$

Group Delay

Overview

A convenient measure of the linearity of the phase is the *group delay.* The basic concept of group delay relates to the effect of the phase on a narrow band signal. Specifically, consider the output of a system with frequency response $H(e^{j\omega})$ for a narrowband input $x[n] = s[n]\cos(\omega_0 n)$. The signal $s[n]$ is called the *envelope*, and it must be slowly varying which means that it has a narrow lowpass spectrum. Since it is assumed that $X(e^{j\omega})$ is nonzero only around $\omega = \omega_0$, the effect of the phase of the system can be approximated around $\omega = \omega_0$ by an expansion up to the linear term

$$\angle H(e^{j\omega}) \approx -\phi_0 - \omega n_d. \tag{0.1}$$

With this approximation, it can be shown that the response $y[n]$ to the input $x[n] = s[n]\cos(\omega_0 n)$ is approximately $y[n] = s[n-n_d]\cos(\omega_0 n - \phi_0 - \omega_0 n_d)$. Consequently the time delay of the envelope $s[n]$ of the narrowband signal $x[n]$ with Fourier transform centered around ω_0 is given by the negative of the slope of the phase at ω_0. In considering the linear approximation (0.1) to $\angle H(e^{j\omega})$ around $\omega = \omega_0$, as given above, we must treat the phase response as a continuous function of ω; rather than as a function modulo 2π.

The phase response specified in this way will be denoted as $\arg[H(e^{j\omega})]$ and is referred to as the *unwrapped phase* of $H(e^{j\omega})$.

With the phase specified as a continuous function of ω, the group delay of a system is defined as

$$\tau(\omega) = \mathrm{grd}[H(e^{j\omega})] = -\frac{d}{d\omega}\left\{\arg[H(e^{j\omega})]\right\}. \tag{0.2}$$

An integer constant value for the group delay represents a perfect delay, such as the system $H(z) = z^{-3}$. A non-integer constant value represents a non-integer delay which is typically interpreted in terms of bandlimited interpolation and delay. Any deviation of the group delay from a constant indicates some non-linearity in the phase and corresponding dispersion.

The concept of group delay has its most direct interpretation in relation to the frequency response of a system. More generally it is often useful to characterize the phase associated with the Fourier transform of a signal in terms of the derivative of the unwrapped phase, *i.e.* in terms of Eq. (0.2). Consequently, we often make reference to the group delay of a sequence by which we explicitly mean the negative of the derivative of the unwrapped phase.

Background Reading

Oppenheim and Schafer (1989), Section 5.1.2 and problem 5.3

Project 1: Algorithm For Computing the Group Delay

The group delay associated with a system is defined as the negative derivative of the phase of the frequency response (see Eq. 0.2). For a sequence the group delay is defined as the negative derivative of the phase of the Fourier transform. This derivative cannot be taken directly unless the phase is unwrapped to remove 2π jumps. An alternative algorithm for computing the group delay is based on the Fourier transform property that

$$\text{if} \qquad x[n] \overset{\mathcal{F}}{\longleftrightarrow} X(e^{j\omega})$$

$$\text{then} \qquad n\,x[n] \overset{\mathcal{F}}{\longleftrightarrow} j\frac{dX(e^{j\omega})}{d\omega}$$

This project centers on the implementation of a MATLAB function to calculate the group delay of a rational system function or of a discrete-time sequence.

Hints

Any algorithm implemented on a computer can only evaluate the Fourier transform at a finite number of frequencies. The MATLAB function `fft(x, N)` evaluates the Fourier transform of the sequence `x` at a set of `N` frequency points evenly spaced around the unit circle between 0 and 2π.

Looking for zeros of $H(e^{j\omega})$ is equivalent to finding roots of $H(z)$ on the unit circle. In MATLAB this is accomplished by treating $H(e^{j\omega})$ as a polynomial in $e^{j\omega}$ and using the function `roots`, following by `abs` to extract the magnitude of any complex-valued roots.

Exercise 1.1: Creating a group delay function

The MATLAB signal processing toolbox contains a function `grdelay()` to compute the group delay of a rational system function. However, it is instructive to rewrite this function to have slightly different characteristics.

(a) Express $H(e^{j\omega})$ in polar form as $H(e^{j\omega}) = A(\omega)e^{j\theta(\omega)}$, where $A(\omega)$ is real and prove the following property:

$$-\frac{d\theta(\omega)}{d\omega} = \Re\left\{\frac{j\dfrac{dX(e^{j\omega})}{d\omega}}{X(e^{j\omega})}\right\} \tag{1.1}$$

(b) Using the function `fft` along with any other MATLAB operations (except, of course, the MATLAB function `grdelay`), write a function that computes the group delay of an impulse response `[h]` at `N` points in the interval $0 \le \omega < 2\pi$.

(c) Modify the function so that the signal represented by `[h]` can have a starting index other than zero. Let `nstart` be used as an additional argument to specify this starting index. Internally, the group delay function has to generate the signal $n\,x[n]$, so the starting index is needed in that calculation.

Exercise 1.2: Calculating the group delay

Use the following simple signal to test your group delay function.

(a) For the impulse response $h[n] = \delta[n-4]$, analytically determine the group delay.

(b) Representing $h[n] = \delta[n-4]$ as the row vector `h = [0 0 0 0 1]`, evaluate and plot the group delay of $h[n]$ using the function you wrote in exercise 1.1.

(c) Repeat the computation using the `nstart` argument to the group delay function.

Exercise 1.3: Dealing with zeros

Clearly, the proposed FFT method for computing the group delay might fail if $H(e^{j\omega}) = 0$ for any of the values of ω at which we evaluate the Fourier transform.

(a) For each of the following impulse responses show that $H_i(e^{j\omega}) = 0$ for at least one value of ω between 0 and 2π.

(i) $h_1[n] = \delta[n] + 2\delta[n-2] + 4\delta[n-4] + 4\delta[n-6] + 2\delta[n-8] + \delta[n-10]$

(ii) $h_2[n] = 3\delta[n] + \delta[n-1] - \delta[n-2] + \delta[n-3] - \delta[n-4] - 3\delta[n-5]$

(b) Use your group delay function, or attempt to use it, in order to evaluate and plot the group delay of the impulse responses defined above as well as the following impulse response:

$$h_3[n] = 3\delta[n] + 2\delta[n-1] + 1\delta[n-2] + 2\delta[n-3] + 3\delta[n-4]$$

How could you have predicted this result by looking at the symmetry of $h_3[n]$?

(c) Determine what your computer does when it divides by zero in MATLAB; and what it does when it tries to plot the value `inf` which represent ∞. If necessary, modify your function to deal with the possibility that some samples of $H(e^{j\omega})$ will be zero. The following fragment of code may be helpful. It divides the vector `num` by the vector `den` safely:

```
>> result = zeros(den);
>> result(den ~= 0) = num(den ~= 0)./den(den ~= 0);
>> result(den == 0) = inf * ones(1,sum(den == 0));
```

If there were any sequences whose group delay you couldn't find in part(b), find them and plot them after making your changes.

Project 2: Effect of Group Delay on Signals

For this project we will be looking at the effect of nonlinear phase, or non-constant group delay in the context of a filtering problem. One specific context in which constant group delay is important is in filtering to extract a narrow-time pulse from noise and then estimating, from the output, the time origin of the pulse. This often arises, for example, in radar systems, for which the range to a reflector is obtained by determining the time difference between the transmitted and received pulses. If in the process of filtering out additive noise, the received pulse is dispersed in time, due to nonuniform group delay, the estimation of the arrival time becomes more difficult.

In this project, we examine the effect of group delay on signals. We will look at windowed tone pulses created by multiplying a sinusoid by a smooth window (specifically, a Hamming window, but the details of the window aren't critical for this project).

Hints

This project uses a data file `gdeldata.mat` to define the filters and signals needed for processing. If the data file is loaded via the MATLAB `load` command, the following 256-point signals will have been predefined:

`pulse` is a pulse starting at $n = 0$ and relatively well localized in time.

`noise` is a sample of the noise that will be added.

`pulse_plus_noise1` represents the received signal and consists of a different sample of the noise added to the pulse, delayed by an amount to be estimated.

`pulse_plus_noise2` represents the received signal from a different reflector, i.e., it is the sum of a different noise sample and the pulse with a different delay.

There are two filters defined in the data file: one an FIR filter, the other an IIR filter. The 33-point impulse response of the FIR filter is given in the vector `h`; the coefficients of the IIR filter are contained in the vectors `a` and `b`.

For group delay computations, the MATLAB function `grpdelay(b, a, N)` can be used to evaluate the group delay of a rational filter described by the `b` and `a` coefficients. Alternatively, the file written in exercise 1.1 can be used. Both use the FFT to evaluate the group delay at `N` equally spaced points around the unit circle between 0 and π, or 0 and 2π.

Exercise 2.1: Group delay of the IIR filter

(a) Generate and plot the impulse response of the IIR filter.

(b) Compute and plot its frequency response magnitude and its group delay.

(c) Plot the signals `x1` and `x2` and their Fourier transforms. Use these plots and the plots of the magnitude and group delay from (a) to estimate the output you will get from running each sequence through the IIR system.

(d) Verify your estimate in (c) by explicitly computing the outputs due to `x1` and `x2` using `filter`.

Exercise 2.2: Group delay of the FIR filter

(a) Plot the impulse response of the FIR filter. Then generate and plot its frequency response magnitude and group delay. How could you have anticipated from the impulse response that the group delay would be constant?

(b) For the signals `x1` and `x2`, what output would you expect to get from processing each sequence with the FIR system. Verify your prediction by explicitly computing the outputs due to `x1` and `x2` using `conv` or `filter`.

Exercise 2.3: Filtering a pulse from noise

(a) Filter the signals `pulse_plus_noise1` and `pulse_plus_noise2` with both the IIR and FIR filters.

(b) Plot the output signals and from these estimate, as best you can, the time delay of the pulse in each of the two received signals. Explain your time-delay measurements in terms of the group delay curves plotted earlier.

(c) What you should notice is that the FIR filter has constant group delay and the IIR filter has nonuniform group delay. Describe any differences in pulse shape that you observe. Which filter, FIR or IIR, introduces no distortion?

Exercise 2.4: Pulse distortion

Filter the signal `pulse` with the IIR filter. Note that when you processed the signals `x1` and `x2` with the IIR filter or FIR filter, they were scaled and delayed with little distortion of the pulse shape; but when you processed the signal `pulse` through the IIR filter, its pulse shape was severely distorted and dispersed in time. Explain why this is the case.

Project 3: Negative Group Delay

Project Description

By definition a causal system cannot produce output that anticipates its input. This property might be interpreted as "a causal system always produces delay", and if the group delay had meaning as a true (physical) delay then it should never be negative. However, for many filters the group delay function versus ω will be less than zero over part of the frequency range. In this situation, the negative group delay will act on a suitable narrowband pulse so as to advance the *envelope* of the signal.

Exercise 3.1: Group delay of a minimum-phase system

For the following filter:

$$H(z) = \frac{9.88 - 15.6z^{-1} + 6.26z^{-2}}{1 - 0.571z^{-1} + 0.121z^{-2}}$$

Compute the group delay and plot it versus frequency for $0 \le \omega \le \pi$. Note the segment of the frequency axis where the group delay is *negative.* In fact, for a minimum-phase system, the integral of the group delay is zero, so there must always be a portion of the frequency axis where the group delay goes negative.

Exercise 3.2: Effect of negative group delay

With an input confined to the frequency range in which the group delay is negative, we can illustrate an "advance" of the signal envelope.

(a) Create the following bandlimited input signal:

$$x[n] = \tfrac{1}{2} + \sum_{k=1}^{10} \cos\left(\frac{2\pi}{256}(n-128)k\right) \qquad \text{for } n = 0, 1, 2, \ldots, 255$$

Plot the magnitude of $X(e^{j\omega})$ and note the frequencies occupied by the input signal.

(b) Calculate the output signal, $y[n]$, for this input using `filter` in MATLAB. Plot $x[n]$ and $y[n]$ on the same diagram. Note that $x[n]$ and $y[n]$ both start at the same time ($n = 0$), so the system is causal.

(c) Measure the advance of the envelope (in samples) and compare to the average value of the group delay in the band occupied by the input signal.

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{Basic Sampling Theory}}$

Basic Sampling Theory

Overview

This packet provides exercises designed to illustrate the two basic principles of the sampling process: aliasing and reconstruction. Project #2 explores several different means by which a signal can be recovered from its samples.

Background Reading

1. A.V. Oppenheim and R.W. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall: Englewood Cliffs NJ, 1989.

Project 1: Aliasing Caused by Sampling

It is not easy to illustrate aliasing within a program like MATLAB, because the only type of signals in MATLAB are discrete signals represented as vectors. This project uses visual (and audio) reproductions of a signal to illustrate the nature of aliasing.

Hints

normalized time vs. real time

Exercise 1.1: Aliasing a sinusoid

Consider the formula for a continuous-time sinusoidal signal:

$$x(t) = \sin(2\pi f_\circ t + \phi) \tag{1.1}$$

We can sample $x(t)$ at a rate $f_s = 1/T$ to obtain a discrete-time signal

$$x[n] = x(t)|_{t=nT} = x(t)|_{t=n/f_s} = \sin(2\pi \frac{f_\circ}{f_s} n + \phi) \tag{1.2}$$

If we make plots of $x[n]$ for different combinations of $f_\circ$ and f_s, then the aliasing problem can be illustrated. For the following, take the sampling frequency to be $f_s = 8$ kHz.

(a) First of all, make a single plot of a sampled sine wave. Let the frequency of the sine wave be 300 Hz, and take samples over an interval of 10 millisec. The phase ϕ can be arbitrary. Plot the resulting discrete-time signal using `comb`. It should be easy to see the outline of a sinuoid, because your eyes perform a reconstruction visualizing the envelope.

(b) If necessary, make the plot using `plot`. In this case, the points are connected with straight lines, so the sinusoidal behavior should be obvious. The plot with connected straight lines is a form of "signal reconstruction" that makes a continuous-time signal from the discrete-time samples. It is not the ideal reconstruction specified by the Sampling Theorem, but it is good enough to be useful in most situations.

(c) Now make a series of plots, just like part (a), but vary the sinusoidal frequency from 100 Hz to 475 Hz, in steps of 125 Hz. Note that the apparent frequency of the sinusoid is *increasing* as is expected. It might be better to use `subplot` to put a four plot on one screen.

(d) Make another series of plots, just like the previous part, but vary the sinusoidal frequency from 7525 Hz to 7900 Hz, in steps of 125 Hz. Note that the apparent frequency of the sinusoid is now *decreasing.* Explain.

(e) Again make a similar series of plots, but vary the sinusoidal frequency from 32100 Hz to 32475 Hz, in steps of 125 Hz. Is the apparent frequency increasing or decreasing?

Exercise 1.2: Aliasing a chirp signal

A linear frequency modulated signal makes a good test for aliasing, because the frequency moves over a range. This signal is often called a "chirp" due to the audible sound it makes if played through a speaker. The mathematical definition of the chirp is

$$c(t) = \cos(\pi \mu t^2 + 2\pi f_1 t + \psi) \tag{1.3}$$

The instantaneous frequency of this signal can be found by taking the time derivative of the phase (the argument of the cosine). The result is:

$$f_i(t) = \mu t + f_1$$

which exhibits a linear variation versus time.

(a) Take the parameters of the chirp to be $f_1 = 4$ kHz, $\mu = 600$ kHz per sec, and ψ arbitrary. If the total time duration of the chirp is 50 millisec, determine the frequency range that is covered by the swept frequency of the chirp.

(b) Let the sampling frequency be $f_s = 8$ kHz. Plot the discrete-time samples of the chirp using both `comb` and `plot`. Since the swept bandwidth of the chirp exceeds the sampling frequency, there will be aliasing.

(c) Notice that the chirp signal exhibits intervals in time where the apparent frequency gets very low. In fact, the instantaneous frequency is passing through zero at these points. Determine from the plots, the times where this happens. Verify that these are the correct times by checking where the aliasing of the swept frequency occurs.

Exercise 1.3: Listening to aliasing

If your computer has the capability for sound output from MATLAB through a D-to-A converter and a speaker,[2] it will be interesting to listen to the aliased signals created in the previous exercises. In order to get a reasonable signal, it is necessary to create a much longer signal—perhaps one or two seconds in duration.

(a) For the sampled sinusoid, it makes sense to concatenate several segments, consisting of the sinuoids of slightly different frequency. Each one should be about 200 millisecs in duration, so putting together 5–10 of these will make a signal that can be heard.

[2]For the Macintosh and SUN computers, there are m-files available for playing out vectors from MATLAB.

(b) For the chirp the duration must be much longer than 50 millisec., so the parameter μ must be adjusted to get a swept frequency range that passes through only a few aliases. See if you can pick μ so that a 2 sec. chirp will pass through exactly 5 aliases.

Project 2: Frequency Domain View of Sampling

When a continuous-time signal is sampled, its spectrum shows the aliasing effect because regions of the frequency domain are shifted by an amount equal to the sampling frequency. To show this effect in reality, an oscilloscope is needed. In MATLAB the effect can only be simulated, and that is the goal of this project.

The simulation will consist of a sampling operation, followed by a D-to-A conversion (including a reconstruction filter). This simple system will be driven by sinusoids with different frequencies, and the Fourier transform of the analog signals at the input and output will be compared. The different exercises treat each part of the sampling and reconstruction process. They should be combined into one M-file script that will do the entire simulation.

Hints

In order to simulate the analog signals, a very high sampling rate will be used—at least 5 times the highest frequency that any analog signal will be allowed to have. Thus there will be two "sampling rates" in the problem—one for the actual sampling under study and the other for the continuous-time signals. A second issue is how to display the Fourier transform of the continuous-time signals. Again, this can only be simulated. The following M-file should be used to plot the analog spectra. Notice that one of its inputs is the `dt` for the simulation.

```
function fplot( xa, dt )
%FPLOT
%   fplot( xa, dt )
%
%       xa:   the "ANALOG" signal
%       dt:   the sampling interval for
%              the simulation of xa(t)
%
L = length(xa);
Nfft = round( 2 .^ round(log2(5*L)) );   %<-- next power of 2
Xa = fft(xa,Nfft);
range = 0:(Nfft/4);
ff = range/Nfft/dt;
plot( ff/1000, abs( Xa(1+range) ) )
title('CONT-TIME FOURIER TRANSFORM (MAG)')
xlabel('FREQUENCY (kHz)'), grid
pause
```

Exercise 2.1: Signal Generation

To show the aliasing effect we need a simple analog input signal to run through the system. We will use sinusoids, but after you have the simulation working you may want to try other signals. To get started, you must pick a "simulation sampling frequency"; take this to be $f_{\text{sim}} = 100\text{kHz}$.

(a) Generate a simulated analog signal that is a cosine wave with analog frequency $f_\circ$.

$$x(t) = \cos(2\pi f_\circ t + \phi) \qquad 0 \le t \le T$$

Take the phase to be random. Generate samples (at the rate f_{sim}) over the time interval of length T. Choose T so that you get about 200–300 samples of the input signal.

(b) Plot the signal with `plot` so that the samples are connected. Make sure that you label the time axis with analog time.

(c) Plot the Fourier transform of this signal, see `fplot` above.

Exercise 2.2: A-to-D Conversion

The A/D converter takes samples spaced by Δt. It is simulated by taking a subset of the samples generated for $x(t)$. In order to avoid any complications, the ratio of f_{sim} to the sampling rate of the A/D, f_s, should be an integer ℓ. Then every ℓ^{th} sample of the $x(t)$ vector can be selected to simulate the A/D. Plot the resulting discrete-time signal.

Exercise 2.3: Design a Reconstruction Filter

The D/A section consists of two parts: a spacing of the discrete-time samples by the sampling time interval Δt, followed by an analog reconstruction filter. The reconstruction filter will, of course, have to be a digital filter to simulate the true analog filter. Use the MATLAB filter design function `cheby2` to design this filter: `[b,a] = cheby2(9,60,fcut)`. This will design a 9th order filter with 60-dB of stopband attenuation. The analog cutoff frequency has to be at $\frac{1}{2}f_s$. For MATLAB this has to be scaled to `fcut = 2*(fsamp/2)/fsim`.

Design this filter, and then use `freqz` to plot its frequency response. To get its true analog cutoff frequency on the plot, you must remember that this is a digital filter, where the frequency $\omega = \pi$ is mapped to $\frac{1}{2}f_{\text{sim}}$.

Exercise 2.4: D-to-A Conversion

The actual D/A conversion phase consists of creating an analog signal $x_r(t)$ from the discrete-time signal $x[n]$, and then filtering with the Chebyshev filter. The MATLAB vector simulating the analog signal $x_r(t)$ is constructed from the discrete-time signal vector $x[n]$ by inserting a number of zeros between each sample. The number of zeros depends on ratio f_{sim}/f_s.

Carry out this operation (and then the filtering) on the signal generated in part (a). Plot the resulting continuous-time output signal $y(t)$ and its Fourier transform.

Exercise 2.5: Test for Aliasing

All the steps above should be put into one M-file script. Then tests can be run.

(a) Take the sampling frequency to be $f_s = 8$kHz; and let the input signal frequency be $f_\circ = 2$kHz. Make plots of the input and output Fourier transforms, and compare by plotting them together.

(b) Now try a number of different input signal frequencies: $f_\circ = 6$kHz, 7kHz, 9kHz, 10kHz, and 15kHz. Since $f_{\rm sim}$ is only 100kHz, the input frequency should not be taken large than 20kHz. Make plots of the input and output Fourier transforms, and compare. Notice where the aliasing starts to occur.

(c) To illustrate the aliasing effects on one plot, use `subplot` to put the following four plots together: $x(t)$, $x[n]$, $y(t)$, and $x_r(t)$, the analog signal with zeros inserted. Another interesting multi-plot would show: $x(t)$, $y(t)$, and their Fourier transforms together.

(d) If possible, try some other signals for which you can predict the result. For example, try to simulate the chirp experiment from the previous project.

Project 3: Reconstruction of Signals from Samples

Digital signal processing involves, among many other things, the reconstruction of analog signals from digital samples. This project explores various methods which can be used for this reconstruction. There are many possible analog signals which pass through a given set of time samples. The choice of analog signal depends on assumptions made about that analog signal.

Project Description

Consider the case where you are given three samples of an analog signal, $x(t)$ as follows (see also Fig. 1(a):

$$x(0) = 2, \quad x(1) = 1, \quad x(2) = x(t)|_{t=2} = -1 \tag{3.1}$$

No other information is given. To what analog signal do these samples correspond? It is important to realize that there is no one "right answer" to this problem. It depends on the assumptions you make and the reconstruction methods you employ.

For example, one possible analog waveform that corresponds to the samples indicated in Fig. 1(a) is seen in Fig. 1(b). We have simply drawn an arbitrary curve through the sample points. We do not need to specify where it goes beyond the range shown, and we could clearly draw an arbitrary number of additional arbitrary curves.

In order to be more concrete, we need to state assumptions and reconstruction methods. For example, we see here three equally spaced samples. We could assume that samples have been taken for all possible $n = -\infty$ to $+\infty$, and that only these three were found to be non-zero. On the other hand, we could assume that the three samples are a subset of all possible samples for $n = -\infty$ to $+\infty$, but that we were just not given all the other non-zero sample values—only the three shown.

In choosing a reconstruction methods we might then decide to fit a polynomial, fit a sine wave, use linear interpolation, use an ideal low-pass filter, a non-ideal low-pass filter,

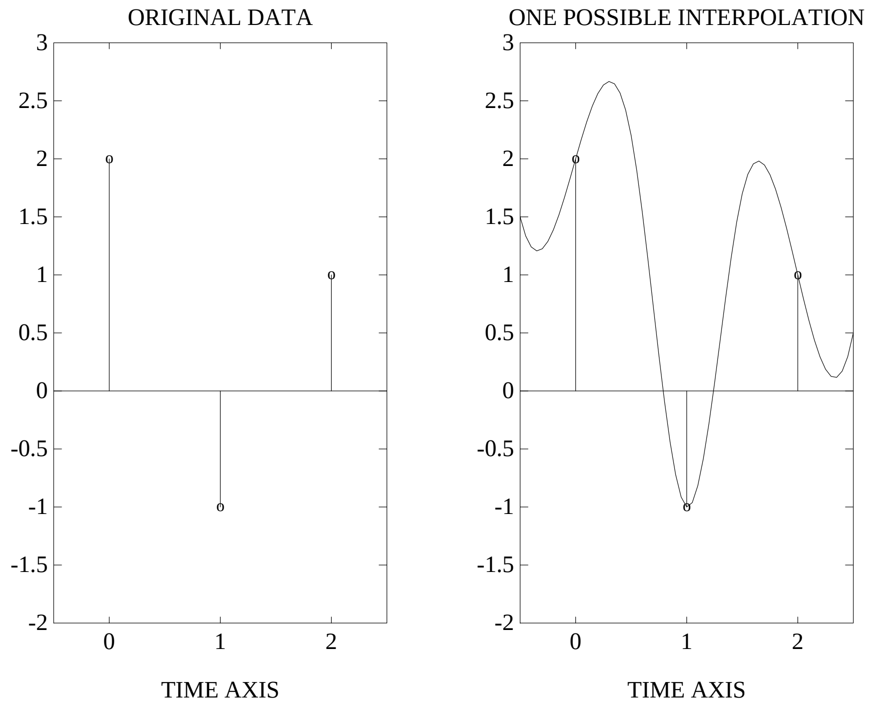


Figure 1: (a) Three Samples (b) One Possible Signal

or any one of a good number of other methods. For this project, we will be trying to fit a sine wave, a polynomial, and then use ideal and non-ideal low-pass filtering.

Exercise 3.1: Fitting a Sine Wave

Assume that the three samples correspond to a sinusoidal waveform of the form:

$$x(t) = A\sin(\omega t + \phi) \tag{3.2}$$

You have $x(0)$, $x(1)$, and $x(2)$. Is this enough information to determine A, ω, and ϕ ? Can you set up the relevant equations? Can you solve these equations? If not, why not?

Can you guess a correct answer? Having found a correct answer, find another answer with a different frequency, ω.

Plot the resulting sinusoids on a very fine grid—use a spacing of less than $\Delta t = 0.01$ secs.

Exercise 3.2: Linear Interpolation

Using MATLAB, connect the samples with straight lines. Plot the result on a fine grid with spacing, $\Delta t = 0.01$ secs. Is this necessary, or will `plot` do this automatically?

Exercise 3.3: Fitting a Polynomial

Using MATLAB, fit a second-degree polynomial to the three data points. Plot the polynomial on a fine grid for $-5 \le t \le 5$. Is this curve realistic in a practical case? Does it do a good job in extending the signal values beyond the known samples?

Exercise 3.4: Ideal Low-Pass Filtering

There are no ideal low-pass filters available in reality. However, we can calculate the waveform that would result from an ideal low-pass filter, as follows: An ideal low-pass operation corresponds to a multiplication of the spectrum of a signal by a rectangular function in the frequency domain. This corresponds to a convolution with the (inverse) Fourier transform (a sinc function) in the time domain. As applied to point samples, this amounts to sinc interpolation.

$$x_r(t) = \sum_{\ell=-\infty}^{\infty} x(t_\ell) \frac{\sin(\pi(t-\ell T)/T)}{\pi(t-\ell T)/T}$$

where the samples $x(t_\ell)$ are taken at $t_\ell = \ell T$.

(a) Write a sinc interpolator or use one that is available in MATLAB. Interpolate a single point sample of value 1 at $t = 0$. Plot the result from about -5 to $+5$. This is the sinc function.

(b) Now interpolate the three-point case. Note that the result is very very different from the sine-wave fitting.

Exercise 3.5: Choice of Assumed Bandwidth

Resolve the following: A signal bandlimited to some frequency f_B can be sampled at $f_S = 2f_B$ and recovered by an ideal low-pass filter with cutoff f_B (as above). The same is true for a second signal that is bandlimited to f_b where f_b is less than f_B, since a signal bandlimited to f_b is also bandlimited to f_B. Also, the signal bandlimited to f_b sampled at f_S can be recovered with an ideal low-pass with cutoff f_b, which has an impulse response (sinc) that is broader than that of the one with cutoff at $f_B = \frac{1}{2}f_S$. Can we interpolate the samples of the signal with bandwidth f_b, sampled at f_S, using the impulse response of the ideal low-pass with cutoff $f_b < \frac{1}{2}f_S$?

Exercise 3.6: Non-Ideal (Digital) Filter

Using MATLAB, convolve the three samples with the impulse response of a digital low-pass "moving-average" filter, which we have seen to be an elementary low-pass filter. Try several different lengths for this filter.

Exercise 3.7: Zero Filling

Convolve the three samples with an impulse response that is triangular (e.g., with 1,2,3,2,1). Next, convolve the same three samples, but first insert four zeros between each of them, and use an impulse response 0.2, 0.4, 0.6, 0.8, 1.0, 0.8, 0.6, 0.4, 0.2. Show that this result is identical to linear interpolation, if we assume that the samples at $t = -1$ and $t = +3$ are zero.

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{Zero-Phase IIR Filtering}}$

Zero-Phase IIR Filtering

Overview

In many filtering problems it is often desirable to design and implement a filter so that the phase response is exactly or approximately linear. If the filter is restricted to be causal, then exactly linear phase can only be realized with an FIR filter. On the other hand, IIR filters are often preferred for bandpass filtering because they can achieve much sharper transition bands for a given filter order. Unfortunately, the phase response of a causal IIR filter is extremely non-linear. However, if the IIR filter is implemented as a non-causal operator, its phase response can be made exactly zero, or exactly linear. In these projects, we will investigate two implementations of non-causal zero-phase IIR filters.

MATLAB has a function in the signal processing toolbox, called `filtfilt`, that will do the zero-phase filtering operation. *It should not be used in these projects.*

Background Reading

Oppenheim and Schafer (1989), problem 5.39

Project 1: Anti-Causal Filtering

Most of the time, we implement recursive difference equations as causal systems for which time runs forward, starting at $n = 0$, then $n = 1, 2, 3, \ldots$ etc. In fact, the MATLAB function `filter` will only do causal filtering. In this project we will need a mechanism to perform filtering backwards, so that the recursive difference equation is applied at $n = 0$, then $n = -1, -2, \ldots$, etc. In this case, the impulse response of the system is left-sided and the system is *anti-causal*.

The bilateral z-transform provides the mathematical framework to describe anti-causal systems. If a causal system has a rational z-transform

$$H_c(z) = \frac{B(z)}{A(z)} \qquad \text{ROC} = \{z : |z| > R_{\max}\} \tag{1.1}$$

then a related anti-causal system is:

$$H_a(z) = \frac{B(1/z)}{A(1/z)} \qquad \text{ROC} = \{z : |z| < 1/R_{\max}\} \tag{1.2}$$

If the radius of the largest root of $A(z)$ satisfies $R_{\max} < 1$, both of these systems are stable, and they will also have exactly the same frequency response magnitude. The region of convergence determines whether the impulse response is right-sided or left-sided. In fact, the two impulse responses are related via a time "flip" (*i.e.*, a time reversal):

$$h_a[n] = h_c[-n]$$

The implementation of the anti-causal system, $H_a(z)$, requires a difference equation that will recurse backwards. In this project, we investigate a method for implementing $H_a(z)$ based on the causal filtering function `filter( )` and time reversals of the signal.

Hints

Since the time base is important when we want to distinguish causal signals from anti-causal signals, it will be convenient to adopt a convention for signal representation that involves two vectors. The first vector contains the signal values; the second, the list of time indices. For example, the following code fragment will define an impulse $\delta[n]$ and a step $u[n]$ over the range $-20 \leq n \leq 30$.

```
        nn = -20:30;
        unit_impulse = (nn==0);
        unit_step = (nn>=0);
```

This works because the logical operators `==` and `>=` return a vector of 1's and 0's, representing TRUE and FALSE, respectively.

Exercise 1.1: A Group Delay M-file

The group delay is defined as the negative derivative of the phase of the frequency response. However, computation of the group delay is best done without explicitly evaluating the derivative with respect to ω. You may want to consider the projects on group delay (Chapter 1) for more details about this computation. However these projects are not required for the following exercises. The M-file below exploits the fact that multiplying by n in the time-domain will generate a derivative in the frequency domain. Furthermore, this function is configured for the case where the signal $x[n]$ starts at an index other than $n = 0$, unlike the function `grpdelay( )` in the MATLAB signal processing toolbox. This is accomplished by passing a vector of time indices `[n]` along with the vector of signal values `[x]`.

```
function [gd, w] = gdel(x, n, Lfft)
%GDEL    compute the group delay of x[n]
%
%    usage:
%           [gd, w] = gdel( x, n, Lfft )
%
%    x:    Signal x[n] at the times (n)
%    n:    Vector of time indices
%    Lfft: Length of the FFT used
%    gd:   Group Delay values on [-pi,pi)
%    w:    List of frequencies over [-pi,pi)
%
% NOTE:  group delay of B(z)/A(z) = gdel(B) - gdel(A)
%
X = fft(x, Lfft);
dXdw = fft(n.*x, Lfft);            %--- transform of nx[n]
gd = fftshift(real( dXdw./X ));   %--- when X==0, gd=infinity
w = (2*pi/Lfft)*[0:(Lfft-1)] - pi;
```

Test the group delay function with a shifted unit impulse signal. Define a unit impulse sequence $\delta[n-n_\circ]$ of length 128, over the range $-64 \le n \le 63$. Pick $n_\circ = \pm 5$, and then make a plot of the signal, with the time axis correctly labeled, to show that the impulse is located at $n = n_\circ$. In addition, compute and plot the group delay to verify that the proper value is obtained.

Another simple test signal would be any finite-length signal that is symmetric. In this case, the group delay should be equal to the value of n at the point of symmetry. Try the signal `x = [1 2 3 4 4 3 2 1]` defined over the range $0 \le n \le 7$.

Exercise 1.2: Causal first-order system

Using the MATLAB function `filter`, generate the impulse response of the *causal* system:

$$H_c(z) = \frac{1}{1 - 0.77z^{-1}} \qquad \text{ROC} = \{z : |z| > 0.77\}$$

Plot the impulse response signal over the range $-64 \le n \le 63$. Also calculate and plot the frequency response magnitude and group delay. This can be done in one of two ways: (1) from $H_c(z)$, by directly evaluating an exact formula based on the numerator and denominator polynomials of $H_c(z)$, or (2), from a finite section of the impulse response, by computing with the FFT. Implement both computations, and plot them together for comparison. Which one is exact, and which one is an approximation?

Repeat for a pole position closer to the unit circle; try 0.95 instead of 0.77. Explain the significant differences between the two cases.

Exercise 1.3: Anti-causal first-order system

For an anticausal filter, the impulse response is zero for $n > 0$. Anticausal filtering can be accomplished in a three-step process: time reverse the input, filter with a causal filter, and then time reverse the output. The signal can be time-reversed using either `fliplr` or `flipud`. Specifically, the two systems shown in Fig. 1 are identical from an input/output point of view. If $H_c(z)$ corresponds to a causal filter, then $H_a(z) = H_c(1/z)$ will correspond to an anticausal filter, and vice versa.

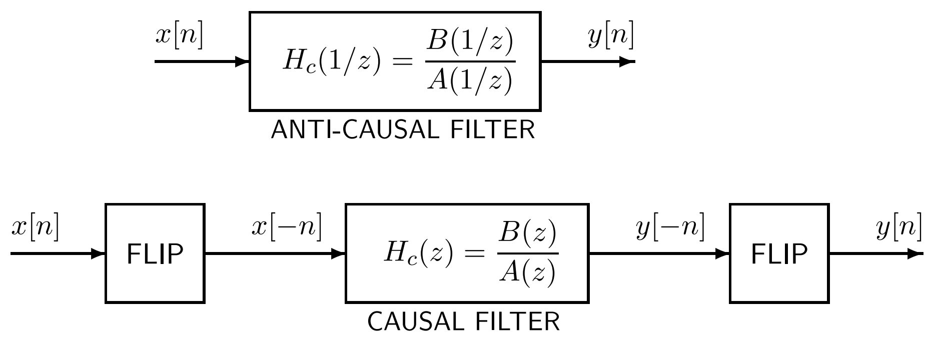


Figure 1: Implementation of anti-causal filter via flips and causal filtering. The two systems above are mathematically equivalent.

For the anticausal filter

$$H_a(z) = \frac{1}{1 - 0.95z} = H_c(1/z) \qquad \text{ROC} = \left\{ z :< \frac{1}{0.95} \right\}$$

generate the impulse response over the range $-64 \leq n \leq 63$ by using the method of Fig. 1, and plot to verify that the response is left-sided. Then calculate and plot the frequency response magnitude and group delay, by explicitly evaluating the Fourier transform of the finite section of the impulse response. Discuss how the impulse response, frequency response magnitude and group delay for this filter relate to those for the causal filter in exercise 1.2. Do the same relationships hold when the pole is at $z = 1/0.77$?

Exercise 1.4: Anti-causal filtering function

Write a general MATLAB function, called `filtrev`, that will implement an anti-causal filter whose numerator is specified by the polynomial $B(z)$ and whose denominator is specified by the polynomial $A(z)$. The arguments to `filtrev` should follow the same convention as in the MATLAB function `filter`, except that the vectors `b` and `a` should specify the coefficients of $B(z)$ and $A(z)$ in *increasing* powers of z.

Project 2: Forward-Backward Filtering

In this project, the causal and anti-causal implementations will be combined to produce a zero-phase IIR filter. In fact, the overall impulse response will only approximate zero phase due to constraints on implementing the non-causal part.

Hints

For this project, it would be best to have a function that will implement an anti-causal filtering operation, such as `filtrev` from 1.4. This can be used as a building block for the zero-phase filter decompositions.

Exercise 2.1: Non-causal filter as a cascade

Since a zero-phase filter must have a symmetric impulse response, an IIR zero-phase filter must be non-causal. The basic approach to implementing any non-causal IIR filter is to decompose it into the combination of causal and anticausal subfilters. This can be done in two different ways: as a cascade of the causal and anticausal parts, or as a parallel combination.

(a) Consider the non-causal filter

$$H_{nc}(z) = \frac{1}{(1 - 0.77z^{-1})(1 - 0.77z)} \qquad \text{ROC} = \left\{ z : 0.77 < |z| < \frac{1}{0.77} \right\}$$

Show analytically that the frequency response of this filter is real-valued and that therefore the phase is zero.

(b) Generate the impulse response of this filter numerically by treating it as a cascade of the filters implemented in the previous project. Plot the impulse response over the range $-64 \leq n \leq 63$ and determine whether or not it is symmetric. (It should not matter in which order the two subfilters are applied.)

(c) Calculate and plot the frequency response magnitude and group delay, by numerically evaluating the Fourier transform of the finite section of the impulse response.

(d) Repeat the implementation, but move the pole location closer to the unit circle—change it from 0.77 to 0.95. Plot the impulse response and the frequency response (magnitude and group delay). In this case, you are likely to find that the group delay is non-zero although it should be zero. What is the most likely explanation for this inconsistency? Does this happen in the passband or stopband of the filter's frequency response?

Exercise 2.2: Parallel form for a non-causal filter

An alternative implementation of the system in exercise 2.1 consists of decomposing it as the *sum* of causal and anticausal subfilters. This gives a *parallel* form implementation, involving two subsystems that are flipped versions of one another.

(a) In order to determine the two subfilters from the zero-phase transfer function $H_{nc}(z)$, it is necessary to perform an expansion similar to a partial fraction expansion. Using this approach, $H_{nc}(z)$ in exercise 2.1 can be expressed as the sum of a causal and anticausal filter.

$$H_{nc}(z) = \frac{1}{(1-0.77z^{-1})(1-0.77z)} = \frac{\beta + \gamma z^{-1}}{1-0.77z^{-1}} + \frac{\beta + \gamma z}{1-0.77z}$$

Determine the constants β and γ in this decomposition.

(b) Generate and plot the corresponding impulse response over the range $-64 \le n \le 63$ by implementing the parallel combination of the causal and anticausal filters.

(c) Compute the magnitude and group delay from the finite section of the impulse response generated in part (b). Compare the magnitude response and group delay to the results from exercise 2.1 and explain any differences.

(d) Repeat parts (a)–(c) for the case when the pole is at 0.95. Is there any significant difference between the parallel and cascade implementations? Compare in both the passband and stopband.

Exercise 2.3: Second-order non-causal filter

Consider the following non-causal filter.

$$H_{nc}(z) = \frac{0.0205z^2 + 0.0034z + 0.0411 + 0.0034z^{-1} + 0.0205z^{-2}}{0.5406z^2 - 1.8583z + 2.7472 - 1.8583z^{-1} + 0.5406z^{-2}}$$

Since $H_{nc}(z)$ is symmetric, i.e., $H_{nc}(z) = H_{nc}(1/z)$, the associated frequency response is purely real, so the filter has zero phase.

(a) $H_{nc}(z)$ can be expressed in factored form

$$H_{nc}(z) = \frac{0.1432 + 0.0117z^{-1} + 0.1432z^{-2}}{1 - 1.2062z^{-1} + 0.5406z^{-2}} \cdot \frac{0.1432 + 0.0117z + 0.1432z^{2}}{1 - 1.2062z + 0.5406z^{2}} = H_c(z)H_c\left(\frac{1}{z}\right)$$

Implement $H_{nc}(z)$ as a cascade of a causal and anticausal filter. Generate and plot the impulse response, frequency response magnitude and group delay using the range $-64 \le n \le 63$.

(b) $H_{nc}(z)$ can also be expressed in parallel form as

$$H_{nc}(z) = \frac{0.1149 + 0.0596z^{-1} - 0.0416z^{-2}}{1 - 1.2062z^{-1} + 0.5406z^{-2}} + \frac{0.1149 + 0.0596z - 0.0416z^{2}}{1 - 1.2062z + 0.5406z^{2}} \tag{2.1}$$

Let $H_1(z)$ denote the causal subfilter and $H_2(z) = H_1(1/z)$ the anti-causal one.

Implement $H_{nc}(z)$ in this additive form in order to generate its impulse response over the range $-64 \le n \le 63$, and plot. Then compute the frequency response magnitude and group delay from this finite section of the impulse response.

(c) Determine the pole and zero locations for this system, and derive the mathematical formulas for the group delay of $H_1(z)$ and $H_2(z)$. Compare these exact formulae to the group delay curves computed for these filters from a finite section of their impulse responses.

(d) Are there any significant differences between the cascade and parallel implementations of the zero-phase filter? What characteristic of $h_{nc}[n]$ guarantees that both implementations will give an excellent approximation to the zero-phase response?

(e) The decomposition of a zero-phase filter into the sum of a causal filter and an anti-causal filter can be done in general. It only requires factorization of the denominator and then the solution of simultaneous linear equations to get the numerator coefficients. Write a MATLAB program that will produce this decomposition in the general N^{th} order case, and verify that the numbers given in Equation (2.1) are correct. Note that polynomial multiplication is convolution, and a function such as `convmtx` can be used to set up convolution in the form of linear equations.

Exercise 2.4: Zero-phase filtering of a square wave

The consequences of non-linear phase versus zero phase can be illustrated in the time domain by processing a pulse-like signal through the two different implementations. For this purpose, construct a signal that is composed of several pulses over the range $-64 \le n \le 63$:

$$x[n] = \begin{cases} -2 & \text{for } -22 \le n \le -8 \\ 2 & \text{for } -7 \le n \le 7 \\ -3 & \text{for } 8 \le n \le 22 \\ 0 & \text{elsewhere} \end{cases}$$

(a) Process $x[n]$ with the non-causal zero-phase filter in exercise 2.3. Try both the cascade and parallel implementations, and note whether or not there is any difference.

(b) For comparison, process the same input signal through the causal filter from exercise 2.3(a). For a fair comparison, the signal should be processed by $H_c^2(z)$, so that it is subjected to the same magnitude response. This can be accomplished by running $x[n]$ through the cascade of two $H_c(z)$ filters. Plot the two outputs on the same scale and compare.

What is the apparent time delay when comparing these outputs to the input signal $x[n]$? Also, note any differences in the distortion of the pulse shape and in the symmetry of the individual pulses. Explain why the zero-phase filter has certain advantages when viewed in terms of these time-domain characteristics.

Chapter 2

The Discrete Fourier Transform

January 17, 2007

Overview

The discrete Fourier transform (DFT) is at the heart of digital signal processing. Although the Fourier, Laplace and z-transforms are the analytical tools of signal processing and many other disciplines, it is the DFT that we can calculate with a computer program. Indeed, it was the development of the fast Fourier transform (FFT) algorithm, which efficiently calculates the DFT, that launched modern DSP.

The DFT and MATLAB are perfectly matched, because we can only do computations on finite-length vectors in MATLAB, which is precisely the case handled by the theory of the DFT. An important goal of the projects in this chapter is to develop an understanding of the properties of the DFT and their use in DSP. The relationship of the DFT to Fourier theory is explored in many cases. The purpose of most of the exercises is to develop insight into the transform, so that it can be used for more than just grinding out numerical results.

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{DFT Properties}}$

DFT Properties

Overview

The properties of the discrete Fourier transform (DFT) are similar to properties of other Fourier transforms. Because the DFT is our primary calculating tool, we must understand its properties and its relation to the other transforms used in DSP. It is the goal of these projects and exercises to develop familiarity with and insight into the use and properties of the DFT. The features unique to the DFT will be emphasized, especially the circular nature of all indexing in both the time and frequency domains.

The DFT is defined as an operation on an N-point time vector $\{\, x[0],\ x[1], \ldots,\ x[N{-}1] \,\}$

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk} \qquad \text{for} \quad k = 0, 1, 2, \ldots, N-1 \tag{0.1}$$

where $W_N^{nk} = e^{-j2\pi/N}$. The operation in (0.1) is a transformation from an N-point vector of time samples $x[n]$ to another N-point vector of frequency-domain samples $X[k]$.

A word about the difference between a DFT and the FFT algorithm needs to be made at the outset. The FFT (fast Fourier transform) is just a fast algorithm for computing the DFT; it is not a separate transform. In MATLAB, the function `fft` is always used to compute the DFT; and there is no `dft` function at all. Therefore, it is usually acceptable to use the terms DFT and FFT interchangeably, when referring to the results of computation.

Background Reading

All DSP textbooks have chapters on the DFT and its properties. Some of the general theory can be found in Chapters 8 and 9 of:

1. Oppenheim & Schafer, *Discrete-Time Signal Processing*, Prentice-Hall: Englewood Cliffs, NJ, 1989.

Project 1: Examples of the DFT of Common Signals

Project Description

In this project, we will develop the DFT of certain basic signals that are commonly used, e.g., pulses, sine waves, aliased sincs, etc. With MATLAB, you can plot out signals and their transforms easily, so the objective is to visualize a number of transform pairs. In the process, you should pay attention to the symmetries that might be present.

Hints

All signals used with the DFT are discrete, so they should be displayed using `comb`; likewise, the transform is a vector of discrete values, so it should also be plotted with `comb`. Since the the DFT is complex-valued, you will have to plot the real and imaginary parts, in general. If you want to view simultaneous plots of the real and imaginary parts of both the time-domain and frequency-domain vectors, use the `subplot(22x)` commands prior to each `comb` command to force the four plots to be placed on the same screen, with real and imaginary parts one above the other, see Fig. 1.

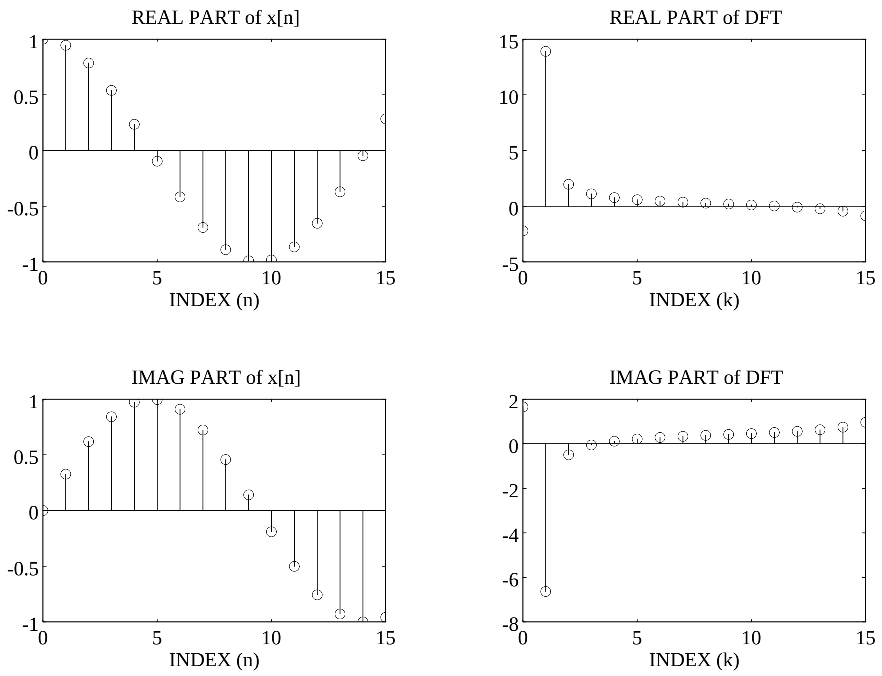


Figure 1: Plotting Real and Imaginary Parts of a discrete-time signal and its DFT with `subplot`.

```
nn = 0:15;
xx = exp(j*nn/3);
XX = fft(xx);
kk = nn;
subplot(221)
comb(kk, real(xx))
title('REAL PART of x[n]'),  xlabel('INDEX (n)')
subplot(223)
comb(kk, imag(xx))
title('IMAG PART of x[n]'),  xlabel('INDEX (n)')
subplot(222)
comb(kk, real(XX))
title('REAL PART of DFT'),  xlabel('INDEX (k)')
subplot(224)
comb(kk, imag(XX))
title('IMAG PART of DFT'),  xlabel('INDEX (k)')
```

Exercise 1.1: Pulses

These are signals containing only ones and zeros. For the following exercises, you can plot the real and imaginary parts of the DFT, but it may be better to plot the magnitude and the phase.

(a) The unit impulse signal: `xi = [1 0 0 0 0 0 0 0]` corresponds to the mathematical definition:

$$\delta[n] = \begin{cases} 1 & n = 0 \\ 0 & n = 1, 2, \ldots, N-1 \end{cases}$$

For this problem compute an 8-point DFT, i.e., $N = 8$. In general, what is the N-point DFT of $\delta[n]$?

(b) Shifted impulse: `xish = [0 0 0 1 0 0 0 0]`. Plot the magnitude of the DFT values. Try other shifts—is there a non-zero shift of `xi` where the DFT is purely real?

(c) All ones: `x1 = [1 1 1 1 1 1 1 1]`. Note that this example together with part (a) illustrates the *duality* principle of the DFT. (The dual for shifted impulses is the complex exponential, see exercise 1.3.)

(d) Boxcar: `xb = [1 1 1 0 0 0 0 0]`. Vary the relative number of ones and zeros.

(e) Symmetric boxcar: `xbsy = [1 1 0 0 0 0 0 1]`. Show that this DFT will be purely real. Compare the DFT magnitudes for `xb` and `xbsy`.

Exercise 1.2: Sine Waves

A real-valued sinusoid is described by three parameters. The mathematical form of the time signal is:

$$s[n] = A\cos(2\pi f_\circ n + \phi) \qquad n = 0, 1, 2, \ldots, N-1$$

where N is the signal length, A is its amplitude, $f_\circ$ its frequency, and ϕ the relative phase at $n = 0$.

(a) Compute the 15-point DFT of a sequence representing exactly one cycle of a cosine wave. Determine the frequency of this sinusoid. Make sure to take exactly one cycle, not one cycle plus one point (i.e., don't repeat the first sample). If done correctly the answer will be extremely simple.

(b) Repeat the previous part for a sine; then for a cosine with a 45° phase shift. Observe carefully the magnitudes of the DFT coefficients (and compare the phases).

(c) Repeat part (a) for 3 cycles of a sinusoid. What is the frequency of this sinusoid (in radians per sample) ?

(d) Try a vector that is 3.1 cycles of a sinusoid. Why is the DFT so different?

(e) Experiment with different frequency sinusoids. Show that choosing the frequency to be $f_\circ = k(1/N)$, when k is an integer, gives an N-point DFT that has only two non-zero values.

Exercise 1.3: Complex exponentials

The complex exponential is defined as

$$c[n] = e^{j\omega_\circ n} \qquad \text{for } n = 0, 1, \ldots, N-1 \tag{1.1}$$

The choice of $\omega_\circ$ gives radically different results for the DFT.

(a) Choose $\omega_\circ = 6\pi/N$, and compute the $N = 16$ point DFT. How many DFT values are non-zero? Since the complex exponentials are the basis signals of the DFT, the orthogonality property renders their transforms in a simple form.

(b) The dual of the complex exponential is the shifted impulse, when $\omega_\circ$ is an integer multiple of $2\pi/N$. Find the 16-point sequence whose DFT is $e^{j6\pi/16}$.

(c) Now try $\omega_\circ = (5.5)\pi/N$, and compute the $N = 16$ point DFT. How many zeros appear in the DFT? Explain.

(d) Now try $\omega_\circ = (5.55)\pi/N$, and compute the $N = 16$ point DFT. Why are there no zeros in the DFT for this case?

(e) Euler's formula, and its inverse, relates the complex exponential to sine and cosine. Let $\omega_\circ = 6\pi/N$. Show how to construct the DFT of $\sin(\omega_\circ n)$ from the result of part (a). Repeat for $\omega_\circ = (5.5)\pi/N$, using the result of part (b).

Project 2: Difficult DFTs

Project Description

The following DFT's are rather difficult to compute using paper-and-pencil manipulations, but the results are not hard to visualize. MATLAB makes it trivial to compute the transform pairs, so the visualization that comes from the duality principle of Fourier analysis can be re-inforced.

Exercise 2.1: Aliased sinc sequence

Once mastered, the duality principle is quite powerful. As a good example of its use in computing, consider the DFT of the "aliased sinc" signal. According to duality, the rectangular pulse and the asinc function are "transform pairs."

(a) The DTFT of a rectangular pulse, is an aliased sinc function in ω; the N-point DFT just samples the asinc at $\omega = (2\pi/N)k$. For an even-symmetric[3] L-point pulse, the result is:

$$R[k] = \operatorname{asinc}(\omega, L)\big|_{\omega = 2\pi k/N} = \frac{\sin(\pi k L/N)}{\sin(\pi k/N)} \tag{2.1}$$

Generate a 7-point pulse, and compute its 16-point DFT. Verify that the correct transform values were obtained. Repeat for a 21-point DFT.

[3]See project on DFT symmetries for more details.

(b) Use MATLAB to find the N-point DFT of the following sampled asinc sequence:

$$a_0[n] = \frac{\sin(9\pi n/N)}{\sin(\pi n/N)} \qquad \text{for } n = 0, 1, \dots, N-1$$

Assume that N is greater than 9, say $N = 16$, or $N = 21$.

(c) Find the N-point DFT of the following shifted asinc sequence:

$$a_1[n] = \frac{\sin(9\pi(n+1)/N)}{\sin(\pi(n+1)/N)} \qquad \text{for } n = 0, 1, \dots, N-1$$

Since the asinc function is periodic, this shift is a circular shift.

(d) NOTE: the factor in the numerator of $a_0[n]$ must be odd! Try replacing the 9 with 10 in either of the previous parts, and compute the DFT. Why is there so much difference between the even and odd cases.

Exercise 2.2: Impulse Train

Perform the following experiment:

(a) Generate an "impulse train" containing 207 samples. In between impulses there should be 22 zeros. The height of each impulse should be a constant. Call this signal $x[n]$.

$$x[n] = \sum_{\ell=0}^{8} \delta[n - \ell M_\circ] \qquad \text{with } M_\circ = 23$$

(b) Compute the DFT of $x[n]$; use a length of 207. Observe that in the DFT domain, $X[k]$ also takes on only the values of zero and a constant.

(c) The period of the input signal, $M_\circ$, is a divisor of the length of the FFT. Use this fact to explain the mathematical form of $X[k]$. Generalize this result. In particular, predict the DFT if the impulses were separated by 9; then verify with MATLAB.

(d) In $x[n]$, there are 9 impulses, spaced by 23 points. When the length of the DFT is changed, there might still be zeros in the transform. First, try a 360-point DFT (length divisible by 9). In this case, $x[n]$ would have to be zero-padded out to a length of 360. Determine which DFT points are exactly equal to zero, and explain why. What would happen for a 460-point DFT (length divisible by 23)? Explain.

(e) Compute the DFT of $x[n]$, using a length of 512; again with zero-padding prior to the FFT. Note that the transform has many peaks, and they seem to be at a regular spacing, at least approximately. Measure the spacing and count the peaks.
State the general relationship between the period of the input signal, $x[n]$, the length of the DFT, and the regular spacing of peaks in the DFT.

Exercise 2.3: A Gaussian

An often quoted result of Fourier theory is that the "Fourier transform of a Gaussian is a Gaussian". This statement is exactly true for the case of the continuous-time Fourier transform, but only approximately true for the DTFT and the DFT.[4]

(a) Generate a (real) Gaussian signal.

$$g[n] = e^{-\alpha n^2} \qquad -L \leq n \leq L$$

Since the signal is symmetric about the origin ($n = 0$), pick the samples from $n = -L$ to $n = +L$. Choose L so that the Gaussian is sampled well out onto its tails; complete coverage is impossible because $g[n]$ is infinitely long. The exact choice of L will depend on α; perhaps the largest exponent αL^2 should be restricted to be less than 100. If we take $\alpha = \frac{1}{2}$, this is a special case where the CTFT yields a transform that is identical.

(b) Form an N-point vector from the samples of $g[n]$. Note that N will be equal to $2L + 1$. Place the samples into the vector so as to preserve the even symmetry, rotate the largest sample $g[0]$ to the beginning of the vector.

(c) Compute the N-point DFT of $g[n]$. Verify that the result is purely real. If it is not, then the time vector was not constructed symmetrically.

(d) Plot the real part of the DFT and compare to a Gaussian. It may be necessary to rotate the DFT vector to see that it looks like a Gaussian, see `fftshift`.

(e) Experiment with different values of α. Keep the transform length constant. Try to make the width of the Gaussian the same in both the time and frequency domains. Notice that when the width of the Gaussian decreases in the time domain, it increases in the DFT domain. This is a demonstration of the "uncertainty principle" of Fourier analysis: "the product of time width and frequency width is always greater than a fixed constant, thus shrinking the time width will necessarily increase the frequency width."

Exercise 2.4: Real Exponential

Another common signal is the real, decaying exponential. The z-transform of this signal consists of a single pole, so it is very simple to evaluate. However, it is wrong to think that the DFT is merely a sampled version of the z-transform.

(a) Generate a finite portion of an exponential signal: $x[n] = (0.9)^n u[n]$, for $0 \leq n < N$. Take a small number of samples, say $N = 32$.

(b) Compute the N-point DFT of $x[n]$, and plot the magnitude of the DFT.

(c) Compare to samples of the magnitude of the DTFT of $y[n] = (0.9)^n u[n]$, the infinitely long exponential.

$$\left|Y(e^{j\omega})\right| = \left|\frac{1}{1 - 0.9e^{-j\omega}}\right| \tag{2.2}$$

Plot the magnitudes on the same graph—explain the difference in terms of windowing.

[4]There are signals for which $\mathcal{DFT}\{v[n]\} \rightarrow v[k]$. These are eigenvectors of the DFT and are treated in the packet *DFT as a Matrix*.

(d) Another related signal is obtained by sampling the DTFT of $a^n u[n]$. Create a DFT by sampling the formula (2.2)

$$V[k] = Y(e^{j\omega})\Big|_{\omega=(2\pi/N)k}$$

Take the N-point IDFT of $V[k]$ to obtain $v[n]$. Experiment with different transform lengths for N, because as $N \to \infty$ the result should get very close to $a^n u[n]$.

Since the DFT was formed by sampling in the frequency domain, there should be aliasing in the time domain. Derive a formula for $y[n]$ in terms of $x[n]$, based on this idea of "time aliasing."

Project 3: Symmetries in the DFT

Project Description

This project explores the different symmetries of even, odd, purely real, purely imaginary, and conjugate symmetry for the DFT. In many cases, these symmetries can be used to simplify computations, especially in different variations of the FFT algorithm.

Hints

The indexing of an N-point vector in Matlab runs from 1 to N. However, many of the symmetries are expressed in terms of flips with respect to the origin, e.g., $x[n] = x[-n \bmod N]$ is an even signal. This presents problems when working with Matlab. A simple solution is to create a new M-file to perform the circular flip—call it `cflip`.[5] Likewise, we will need to write functions for computing the even and odd parts of a signal vector.

Exercise 3.1: Symmetries are circular

All operations with the DFT are done over an indexing domain that is circular. Since all symmetry properties boil down to only *two* basic operations: conjugation and flipping, it is essential have functions for each. Matlab provides a built-in conjugate function: `conj( )`. However, the "flip" operation is tricky. There are built-in Matlab functions called `fliplr` and `flipud`, for flip-left-right on rows, and flip-up-down on columns. Neither of these is what we want for the DFT, because these flips do the following:

$$y_{\text{out}}[n] = x[N-1-n] \qquad \text{for } n = 0, 1, \ldots, N-1$$

Thus $x[0]$ is exchanged with $x[N-1]$; $x[1]$ with $x[N-2]$, etc. On the other hand, the circular flip needed for the DFT would satisfy

$$y_{\text{cir}}[n] = x[-n \bmod N] = x[N-n] \qquad \text{for } n = 0, 1, \ldots, N-1$$

In this case, $x[0]$ *stays put*, while $x[1]$ is exchanged with $x[N-1]$; $x[2]$ with $x[N-2]$, etc.

(a) Write an M-file for the circular-flip operation; call it `cflip( )`. For an N-point row vector `cflip` is simply `[ x(1), x(N:-1:2) ]`, but you should make it work for rows, columns and matrices (c-flip each column).

[5]Need a CASPER function for `cflip()`; do we also need a modulo-N M-file. Both should work for matrices and vectors.

(b) Verify the DFT property: "a c-flip in time gives a c-flip in frequency". Use simple test vectors.

(c) Verify the DFT property: "a conjugate in time gives a conjugate plus a c-flip in frequency". State the dual of this property; then verify it.

Exercise 3.2: Even and odd parts

The primary symmetries are based on evenness and oddness. Of course, these must be defined "circularly".

(a) Generate a real-valued test signal $v[n]$—use `rand`. Pick a relatively short length, say $N = 15$ or $N = 16$. Compute the DFT of $v[n]$, and then try the following for both even and odd lengths.

(b) Calculate the even and odd parts of $v[n]$.

$$\begin{aligned} v_e[n] &= \tfrac{1}{2}\left(v[n] + v[-n \bmod N]\right) \\ v_o[n] &= \tfrac{1}{2}\left(v[n] - v[-n \bmod N]\right) \end{aligned}$$

Write a MATLAB function that will extract the even part of a vector; likewise, for the odd part. These functions should call the `cflip` function written previously.

(c) Using the same signal as in part (a), compute the DFTs of the even and odd parts, and compare the results with the even and odd parts of the DFT computed in part (a).

$$\begin{aligned} V_e[k] = \mathrm{DFT}\{v_e[n]\} \quad &\text{vs.} \quad \Re e\{V[k]\} \\ V_o[k] = \mathrm{DFT}\{v_o[n]\} \quad &\text{vs.} \quad \Im m\{V[k]\} \end{aligned}$$

If $v[n]$ is complex, show that these same realtions do not hold.

(d) The notions of even and odd can be extended to the complex case by defining two attributes called "conjugate symmetric" and "conjugate anti-symmetric". Generate a random complex-valued test signal, $v[n]$. Compute its "conjugate symmetric" and "conjugate anti-symmetric" parts via:

$$\begin{aligned} v_{\text{csy}}[n] &= \tfrac{1}{2}\left(v[n] + v^*[-n \bmod N]\right) \\ v_{\text{cas}}[n] &= \tfrac{1}{2}\left(v[n] - v^*[-n \bmod N]\right) \end{aligned} \tag{3.1}$$

Write MATLAB functions that will extract these parts from a signal vector.

(e) Show that the real part of $v_{\text{csy}}[n]$ is even; and the imaginary part is odd. Verify these in MATLAB for the conjugate symmetric part of a random test sequence. State a similar relation for v_{cas}; verify it.

(f) Verify that the DFT of a conjugate-symmetric signal is purely real. What about the DFT of a conjugate-anti-symmetric signal?

Exercise 3.3: DFT of a real sequence

By duality, the DFT of a purely real sequence should be conjugate symmetric. Start with a real N-point sequence $v[n]$, which is neither even nor odd.

(a) Calculate its DFT: $V[k] = \text{DFT}\{v[n]\}$

(b) Display $\Re e\{V[k]\}$ and $\Im m\{V[k]\}$

(c) Extract its conjugate anti-symmetric part which should be zero.

(d) Show that $\Re e\{V[k]\}$ is even, by computing the odd part of $\Re e\{V[k]\}$ which must be zero. Show that $\Im m\{V[k]\}$ is odd.

Thus we can conclude that the DFT of a real sequence is conjugate symmetric, which is the expected dual property. What would be the result for a purely imaginary input vector?

Exercise 3.4: All possible symmetries

Any complex-valued signal can be decomposed into four sub-signals, each of which exhibits a certain symmetry. The complex signal, $v[n]$, can always be written as:

$$v[n] = v_{\text{r,e}}[n] + v_{\text{r,o}}[n] + j(v_{\text{i,e}}[n] + v_{\text{i,o}}[n]) \tag{3.2}$$

where the subscripts denote real (r), imaginary (i), even (e), and odd (o). Thus $v_{\text{i,o}}[n]$ is the odd part of the imaginary part of $v[n]$. The same sort of decomposition can be done for the DFT, $V[k]$.

$$V[k] = V_{\text{r,e}}[k] + V_{\text{r,o}}[k] + j(V_{\text{i,e}}[k] + V_{\text{i,o}}[k])$$

However, it is wrong to assume that the DFT of one of the sub-signals matches the corresponding sub-sequence of the DFT.

(a) Write a MATLAB function that will decompose any vector into its four parts, defined in (3.2).

(b) The DFT symmetries can be summed up in a diagram that shows the correspondence of the symmetry in the transform domain one of the four sub-signals in the time domain. Complete the following diagram by connecting each time-domain sub-signal • to the appropriate frequency-domain sub-sequence ∘.

DFTSymmetries

$v_{\text{r,e}}[n]$	•	∘	$V_{\text{r,e}}[k]$	real & even
$v_{\text{r,o}}[n]$	•	∘	$V_{\text{r,o}}[k]$	real & odd
$v_{\text{i,e}}[n]$	•	∘	$V_{\text{i,e}}[k]$	imaginary & even
$v_{\text{i,o}}[n]$	•	∘	$V_{\text{i,o}}[k]$	imaginary & odd

(c) Give numerical examples to show that each connection in the diagram is correct.

Project 4: Tricks for the IDFT

Project Description

In practice, special hardware may be built to compute a DFT. In this case, it is advantageous to use the same hardware for the inverse DFT (IDFT). This project shows three different ways that the IDFT can be computed using a forward DFT algorithm. All are based on the fact that the formula for the IDFT is nearly identical to that for the forward DFT.

$$x[n] = \frac{1}{N}\sum_{n=0}^{N-1} X[k]W_N^{-nk} \qquad \text{for} \quad k = 0, 1, 2, \ldots, N-1 \tag{4.1}$$

Hints

Inside MATLAB the `idft` function is actually implemented using one of these tricks. Type out the listing of the function `idft` to see which one.

Exercise 4.1: IDFT via circular rotations

This exercise deals with computing the inverse DFT using a property of the transform that is known as "Duality". This method is interesting because it emphasizes the circular nature of all DFT indexing.

(a) Generate a random sequence $x[n]$ using the `rand( )` function. The length of the sequence can be anything, but it is often chosen as a power of two; if so, select $N = 16$. Use `fft()` to compute the DFT of $x[n]$, and call this $X[k]$. This sequence will be the test sequence for the IDFT. Perform the following steps with MATLAB:

1. Flip the sequence $X[k]$. Since $X[k]$ is just a vector of 16 complex values, it can be treated as though it were a vector of time samples. So together with the flip operation, define a new time vector by $y[n] = X[k]|_{k=(-n) \bmod N}$.
2. Since the sequence $y[n]$ is just a vector of 16 complex values, it can be used as the input to a <u>forward</u> DFT, i.e., apply `fft( )` again. Call the result $Y[k]$.
3. Once again, the sequence $Y[k]$ is just a vector of 16 values, so it can be considered as a time vector, if k is replaced with n.
4. Compare the numerical values of $x[n]$ and $v[n] = Y[k]|_{k=n}$.

(b) Derive the general rule for the relationship between the values of $x[n]$ and $v[n]$, and explain (i.e., prove) why they are related in such a simple fashion.

(c) Program an M-file that implements the IDFT according to this algorithm. Test it on some known DFT pairs.

Exercise 4.2: IDFT via conjugates

This exercise deals with computing the inverse DFT using several well-placed conjugates. Use a test sequence $X[k]$ generated as in the previous exercise.

(a) We want to show that the following three steps are equivalent to the IDFT of $X[k]$.

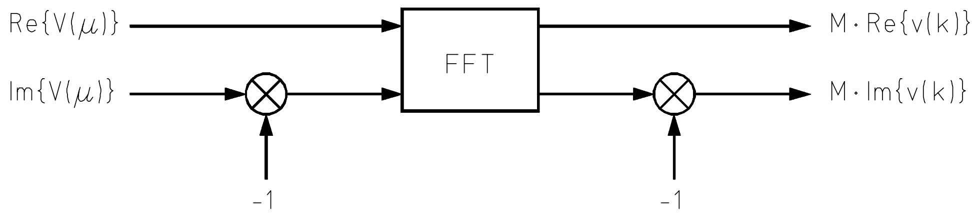


Figure 2: IDFT via conjugate trick.

1. Conjugate $X[k]$.
2. Compute the forward DFT of $X^*[k]$.
   As before, we have an N-point vector so we can do this.
3. Conjugate the resulting DFT output.
   If the result of the previoius DFT were $Y[k]$, we now have $Y^*[k]$.

The cojugate operator merely changes the sign of the imaginary part, as shown in Fig. 2.

(b) Compare the numerical values of $x[n]$ and $v[n] = Y^*[k]|_{k=n}$. Notice that the effect of the conjugate operations has been to change the sign of the complex exponential.

Derive the general rule for the relationship between the values of $x[n]$ and $v[n]$, and explain (i.e., prove) why they are related in such a simple fashion.

(c) Create a Matlab M-file that will implement this algorithm for the IDFT. Compare with the implementation actually used in Matlab; the listing of the Matlab `idft` function can be obtained via `type idft`.

Exercise 4.3: Another IDFT trick

This exercise shows another way to compute the inverse DFT. Its proof also relies on the conjugate property of the DFT. Use the same test sequence as in the previous two exercises.

(a) We want to show that the following three steps are equivalent to the IDFT of $X[k]$, see Fig. 3.

1. Swap the real and imaginary parts of $X[k]$ and define the result as a time vector; i.e., define $v[n]$ so its real part is the imaginary part of $X[k]$ and its imaginary part is the real part of $X[k]$.
2. Compute the forward DFT of $v[n]$, and call this $V[k]$.
3. Swap the real and imaginary parts of $V[k]$ and define it as a time vector; i.e., define $y[n]$ so its real part is the imaginary part of $V[k]$ and its imaginary part is the real part of $V[k]$.

(b) Compare the numerical values of $x[n]$ and $y[n]$. Derive the general rule for the relationship between the values of $x[n]$ and $y[n]$, and prove why they are related in such a simple fashion.

(c) Create a Matlab M-file that will implement this IDFT algorithm.

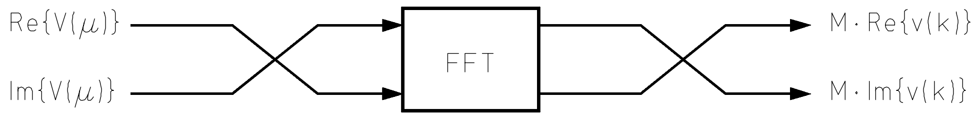


Figure 3: IDFT via swap of real and imaginary parts.

Project 5: The Stretch and Decimate Properties

Two basic operations with the DFT are "stretch" and "decimate". In both cases, new signals are formed that are either longer or shorter than the original. Relating the DFT's for these situations is an important exercise.

Exercise 5.1: Pad with zeros

We can create a long signal by appending zeros to an existing signal. This can be done automatically in the `fft` function by specifying a second argument for the FFT length and making it longer than the signal length.

Create a test signal that is a 16-point sinusoid: $\sin(\omega_0 n)$. Choose $\omega_0 = 2\pi/\sqrt{17}$. Compute the 16-point DFT and call the result $X[k]$. Then compute the DFT for lengths 32, 64, 256, and 1024. Show that the 16 values in $X[k]$ can always be found in each of the longer DFT's.

Exercise 5.2: Zero-Padding in the Middle

One difficulty with zero padding is that is destroys symmetry. In other words, if the original signal were such that its DFT were purely real, then after zero padding the resultant DFT is no longer real. There is a method of zero padding that will preserve such symmetries—this is padding "in the middle."

(a) Create a real and even-symmetric signal for use as a test signal. Since this is easiest to do when the length is odd, take N to be odd. Verify that the DFT of this signal is purely real; and that it is also even-symmetric.

(b) Now take the FFT of the same signal, but with the length three times the original. Verify that the transform is no longer purely real (unless, of course, your test signal were the impulse).

(c) Now create another signal that is also three times longer than the original, but do the padding in the following way:

1. Put the first $\frac{1}{2}(N+1)$ signal points at the beginning of the output vector. (Remember that N is assumed to be odd.)
2. Add $2N$ zeros to the vector.
3. Then tack the last $\frac{1}{2}(N-1)$ signal points at the end of the output

Write an M-file that does this operation, and then verify that it preserves symmetry by testing with simple inputs and their DFTs.

(d) Show how the padding in the middle can be done as the concatenation of three operations: a rotation, zero-padding and then another rotation. Specify the number of samples for each rotation.

(e) How would you do this "padding in the middle" when N is even? Write an M-file that exhibits proper adherence to symmetries for the even length case. The difficulty is $x[N/2]$. If you split it in half, then a symmetric signal can be produced after padding, but will this strategy preserve the interpolation through the original DFT values?

Exercise 5.3: Stretch: Intersperse zeros

We can create a longer signal by putting zeros in between the existing samples. For example, if we start with a length N signal, a length ℓN signal can be created as follows:

$$\tilde{x}[n] = \begin{cases} x[n/\ell] & \text{when } n \bmod \ell = 0 \\ 0 & \text{when } n \bmod \ell = 0 \end{cases} \tag{5.1}$$

Generate a random 10-point signal and compute its DFT. Intersperse 3 zeros, and compute the 40-point DFT, i.e., $\ell = 4$. Verify that the longer DFT is just a repetition of the shorter one. Therefore, the duality property is "periodicity in one domain goes with interspersed zeros as in (5.1) in the other domain."

Exercise 5.4: Decimation

The dual of the stretch property is decimation. In this case, we generate a shorter signal by removing samples. Starting with a length N signal, we can make a length $M = N/\ell$ signal by taking every ℓ^{th} sample:

$$\hat{x}[n] = x[n\ell] \qquad \text{for } n = 0, 1, \ldots, (M-1) \tag{5.2}$$

It is necessary to assume that N contains a factor of ℓ. The M-point DFT of $\hat{x}[n]$ is an "aliased" version of the original DFT, $X[k]$, because sampling is being done in the time domain.

(a) Generate a sinusoidal test signal whose length is 60, i.e., $x[n] = \sin(\omega_0 n)$. Set the frequency at $\omega_0 = 2\pi/5$. Perform a decimation by $\ell = 2$, and then compute the 20-point DFT. Explain the result.

(b) Redo the previous experiment for decimation by $\ell = 3$.

(c) Try the same experiment with a decimation factor of $\ell = 7$, which is not a divisor of 60. This result is much different, and can only be explained by using the "frequency sampling" formula that relates the DFT to the DTFT.

Project 6: The Real Data FFT

There are two cases of interest where the FFT of a real sequence can be computed efficiently. The first is the case of simultaneously transforming two N-point vectors with just one N-point DFT. The second involves the transform of a $2N$-point vector using a single N-point FFT.

Hints

These algorithms are based on the symmetry properties of the DFT. In a previous project on symmetries, two functions were developed for extracting the conjugate-symmetric and conjugate-anti-symmetric parts of a signal vector. These will be needed here.

Exercise 6.1: Computing Two DFTs at once

Given two real N-point signals $v_1[n]$ and $v_2[n]$. Form a complex signal $v[n] = v_1[n] + jv_2[n]$, and compute its DFT; the result is a complex-valued vector $V[k]$. Since the DFT is a linear operator, it follows that

$$V[k] = \mathcal{DFT}\{v_1[n] + jv_2[n]\} = V_1[k] + jV_2[k]$$

Recognize that $V_1[k]$ is not the real part of $V[k]$, nor is $V_2[k]$ the imaginary part, because both $V_1[k]$ and $V_2[k]$ are complex-valued. Instead, use the conjugate symmetry property to show analytically that the following relationships hold:

$$\begin{aligned} V_1[k] &= \textsc{csy}\{V[k]\} \qquad \text{(conjugate symmetric part)} \\ jV_2[k] &= \textsc{cas}\{V[k]\} \qquad \text{(conjugate anti-symmetric part)} \end{aligned}$$

The conjugate symmetric and anti-symmetric operators were defined in (3.1).

(a) Confirm by numerical experiment that these equations hold.

(b) Write a MATLAB function that will compute the DFT of two real vectors, with only one call to `fft`. Test with known DFT pairs.

Exercise 6.2: FFT of Real Data

Given a length-$2N$ real signal $x[n]$, $n = 0, 1, \ldots 2N-1$, we want to develop a method that will compute the $2N$-point DFT of $x[n]$ using just one N-point FFT. This algorithm relies on one fact from the derivation of the FFT, but it is a relatively easy one to derive so we develop it here. If we separate the input sequence into two subsequences, one containing the even-indexed members of the original vector, the other the odd-indexed points

$$x_0[n] = x[2n] \qquad x_1[n] = x[2n+1] \qquad \text{for } n = 0, 1, 2, \ldots N-1$$

then we can write the $2N$-point FFT as

$$\begin{aligned} X[k] &= \sum_{n=2\ell} x[2\ell]W_{2N}^{2\ell k} + \sum_{n=2\ell+1} x[2\ell]W_{2N}^{(2\ell+1)k} \\ &= \sum_{\ell=0}^{N-1} x_0[\ell]W_N^{\ell k} + W_N^k \sum_{\ell=0}^{N-1} x_1[\ell]W_N^{\ell k} \\ &= \mathcal{DFT}\{x_0[\ell]\} + W_N^k \mathcal{DFT}\{x_1[\ell]\} \qquad k = 0, 1, \ldots 2N-1 \qquad (6.1) \end{aligned}$$

The DFTs needed here are N-point DFTs. Thus we can get the $2N$-point DFT from two half-length DFTs or real-only data, which is exactly what was done in the previous exercise.

(a) Form a complex signal $v[n] = x_0[n] + jx_1[n]$, and compute its DFT; the result is a complex-valued vector $V[k]$. Use the conjugate symmetry property to extract the DFTs of each real sequence:

$$\begin{aligned} X_1[k] &= \textsc{csy}\{V[k]\} && \text{(conjugate symmetric part)} \\ jX_2[k] &= \textsc{cas}\{V[k]\} && \text{(conjugate anti-symmetric part)} \end{aligned}$$

(b) Apply the result (6.1) from above to get the entire $2N$ points of the DFT of $x[n]$. Note that $X_i[k]$ has a period of N, so (6.1) can be simplified a bit.

(c) Write a MATLAB function that will compute the DFT of a real vector according to this algorithm—just one call to `fft` with length N.

(d) Test your program versus `fft` and show by numerical example that you get the same answer. If you are interested, use `flops` to compare the number of floating point operations used by both methods.

NOTE: the MATLAB `fft` function uses this trick for the FFT of real data. If you count the number of flops for the FFT of real data and compare to the number of flops for complex data, you will find that the real case takes a little more that 50% of the number for the complex case.

Exercise 6.3: IFFT of Conjugate-Symmetric Data

A procedure similar to the real-data case can be derived for doing the IDFT of a vector that satisfies the conjugate-symmetric property. This technique is *not* implemented in the MATLAB inverse FFT function, so it would be beneficial to add this functionality.

Given a length-$2N$ conjugate-symmetric vector $X[k]$, $k = 0, 1, \ldots 2N-1$, we want to develop a method that will compute the $2N$-point IDFT of $X[k]$ using just one N-point IFFT. Again, we need one fact from the derivation of the FFT, which we develop here. If we separate the input sequence into two subsequences, one containing the even-indexed members of the original vector, the other the odd-indexed points, then we can write the $2N$-point IFFT as

$$\begin{aligned} x[2n] &= \frac{1}{2N}\sum_{k=0}^{2N-1} X[k]W_{2N}^{2nk} = \frac{1}{2N}\sum_{k=0}^{N-1}\left(X[k] + X[k{+}N]\right)W_N^{nk} \\ &= \tfrac{1}{2}\,\mathcal{IDFT}\{X[k] + X[k{+}N]\} \qquad n = 0, 1, \ldots N-1 \\ x[2n+1] &= \frac{1}{2N}\sum_{k=0}^{2N-1} X[k]W_{2N}^{(2n+1)k} = \frac{1}{2N}\sum_{k=0}^{N-1} W_{2N}^{k}\left(X[k] - X[k{+}N]\right)W_N^{nk} \\ &= \tfrac{1}{2}\,\mathcal{IDFT}\{W_{2N}^{k}\left(X[k] - X[k{+}N]\right)\} \qquad n = 0, 1, \ldots N-1 \end{aligned} \tag{6.2}$$

The IDFTs needed here are N-point IDFTs. These two IDFTs can be done simultaneously.

(a) Define the two N-point vectors

$$\left.\begin{aligned} X_0[k] &= X[k] + X[k+N] \\ X_1[k] &= W_{2N}^{k}\left(X[k] - X[k+N]\right) \end{aligned}\right. \qquad \text{for } k = 0, 1, 2, \ldots N-1$$

Since $X[k]$ is conjugate-symmetric (mod-$2N$), you can prove that $X_0[k]$ is a conjugate-symmetric vector (mod-N); and that $jX_1[k]$ is conjugate-anti-symmetric (mod-N). Verify by examples in MATLAB.

(b) Form a complex signal $Q[k] = X_0[k] + jX_1[k]$, and compute its IDFT; the result is a complex-valued vector $q[n]$. Use the conjugate symmetry property to show that:

$$\begin{aligned} x[2n] &= \tfrac{1}{2}\,\Re e\{q[n]\} \\ x[2n+1] &= \tfrac{1}{2}\,\Im m\{q[n]\} \end{aligned} \qquad \text{for } n = 0, 1, \ldots, N-1 \tag{6.3}$$

The factor of $\frac{1}{2}$ comes from the fact that the IDFT computed will be of length N rather than $2N$.

(c) Write a MATLAB function that will compute the IFFT of a conjugate-symmetric vector according to this algorithm—just one call to `ifft` with length N.

(d) One difficulty is detecting whether or not a signal satisfies the conjugate-symmetric property. Write a MATLAB function that will do this by comparing the vector to its circularly flipped version. Allow for round-off error so that the match does not have to exact. (NOTE: If the FFT were computed by a real-data FFT, then the data will be exactly conjugate-symmetric, so the round-off error should not be a problem; otherwise, it will be tricky to set a threshold.

(e) Test your program versus `ifft` and show by numerical example that you get the same answer.

(f) In MATLAB this method will have less flops than the built-in MATLAB function `ifft`. Compare the number of floating point operations used by both methods.

Project 7: Discrete Fourier Series

The DFT can be related directly to the (discrete) Fourier series (DFS) of a periodic signal. If the period of the signal is N, the DFT coefficients are exactly those needed in a DFS representation of the signal. In fact, this periodic nature of the signal (from the Fourier Series) is equivalent to the circular indexing that is always used in the DFT.

Exercise 7.1: Relate DFT to Fourier Series

Given the 16-point sequence `[ 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 ]`, eight 1's followed by eight 0's.

(a) Compute the 16-point DFT.

(b) One interpretation of this vector is that it is one period of a 50% duty cycle square wave. Determine the Fourier series coefficients, i.e., the constants in the following expansion:

$$x(t) = \sum_{k=0}^{N-1} A_k e^{j(2\pi k/N)t}$$

The finite limit on the sum is a truncation of the expansion. It is, in effect, an assumption that the reconstructed signal will be bandlimited.

(c) Relate the Fourier series coefficients to the DFT. Use the fact that the values of $x(t)$ which are known, are those for $t = n$.

(d) The Fourier series expansion can be viewed as an interpolation formula that matches the samples $x[n]$ at the integer-valued points n; and then provides intermediate values between these discrete samples. In order to evaluate the interpolated signal, it is not necessary to directly compute the expansion formula; instead, the IDFT can be used. Suppose the 16-point DFT result is zero padded as follows: 112 zeros are added to the "middle" of the DFT; then the 128-point IDFT is computed. Perform this operation, and then explain why it gives a legitimate interpolated signal—an approximate square wave passing through the original points, but with a sampling density that is 8 times larger.

Exercise 7.2: Pulse train is a sum of complex exponentials

One commonly used periodic signal is the periodic pulse train.

$$p[n] = \delta[n \bmod M] = \begin{cases} 1 & \text{for } n \text{ a multiple of } M \\ 0 & \text{elsewhere} \end{cases}$$

(a) Determine the DFS expansion for this signal.

(b) The signal is real, and even, so its DFS can be written in the following way:

$$p[n] = \sum_k c_k \cos(2\pi k/N)n$$

Determine the values of c_k, and the range on k in the summation.

(c) Extend this to the case where the pulse length is greater than 1, say $L = 5$.

$$p[n] = \begin{cases} 1 & \text{for } n = 0 \bmod M \\ 1 & \text{for } n = \pm 1 \bmod M \\ 1 & \text{for } n = \pm 2 \bmod M \\ 0 & \text{elsewhere} \end{cases}$$

Plot this signal over several periods, and then find its Fourier Series.

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{DFT as a Matrix}}$

DFT as a Matrix

Overview

This set of projects will concentrate on properties of the DFT when viewed as a matrix operator. This viewpoint is becoming more fashionable with the emergence of programs such as MATLAB, which emphasize the matrix-vector nature of computations.

Consider the N-point DFT as a matrix transformation from the complex vector

$$\mathbf{x} = [\, x[0] \quad x[1] \quad \ldots \quad x[N-1] \,]^T \tag{0.1}$$

to another complex vector, in the frequency domain

$$\mathbf{X} = [\, X[0] \quad X[1] \quad \ldots \quad X[N-1] \,]^T \tag{0.2}$$

In matrix-vector notation, the operation is a matrix multiplication:

$$\mathbf{X} = \mathbf{W}\mathbf{x} \tag{0.3}$$

where the DFT matrix $\mathbf{W}$ has entries that are complex exponentials:[6]

$$\mathbf{W} = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & W & W^2 & \ldots & W^{N-1} \\ 1 & W^2 & W^4 & \ldots & W^{2(N-1)} \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 1 & W^{N-1} & W^{N-2} & \ldots & W^1 \end{bmatrix} \qquad \text{where } W = e^{-j2\pi/N} \tag{0.4}$$

Background Reading

All DSP textbooks have chapters on the DFT and its properties. Some of the general theory of discrete Fourier transforms expressed in matrix form can be found in:

1. C. S. Burrus and T. W. Parks, *DFT/FFT and Convolution Algorithms: Theory and Implementation*, John Wiley & Sons: New York, 1985.

Project 1: DFT as an Orthogonal Matrix

The fundamental properties of the DFT rely on the *orthogonality* of the complex exponentials. In terms of the DFT matrix, this property is equivalent to the fact that the column vectors of $\mathbf{W}$ are pairwise orthogonal.

Exercise 1.1: Orthogonality

(a) Generate an instance of the DFT matrix for a small value of N, say $N = 5$. This can be done without `for` loops by raising W to different integer powers; in fact, all these powers of W can be computed at once using the pointwise power operator `W .^ M` in MATLAB, where `W` is the complex exponential and `M = [0:(N-1)]' * [0:(N-1)]` is a matrix containing all the integer powers. Remember to divide by $\sqrt{N}$.

[6]The scaling of $1/\sqrt{N}$ is introduced to make $\mathbf{W}$ a unitary matrix, but this is not the usual definition of the DFT.

(b) Another trick for generating the DFT matrix is `W = fft( eye(N) ) / sqrt(N)`. This method works because the `fft` function applied to a matrix will take the DFT of each column vector in the matrix. Explain why the `fft` function applied to the identity matrix `eye(N)` will generate the complete DFT matrix.

(c) Let $\mathbf{w}_j$ denote the j^{th} column of $\mathbf{W}$. Verify that any two columns of $\mathbf{W}$ are orthogonal, i.e., the inner product $\langle \mathbf{w}_i, \mathbf{w}_j \rangle = \mathbf{w}_i^H \mathbf{w}_j = 0$, when $i \neq j$.

(d) In fact, the columns of $\mathbf{W}$ are *orthonormal*, because a further condition holds, the norm of each column vector is one:

$$\langle \mathbf{w}_i, \mathbf{w}_i \rangle = \mathbf{w}_i^H \mathbf{w}_i = \|\mathbf{w}_i\|^2 = 1$$

(e) The conjugate-transpose of $\mathbf{W}$ is called the *Hermitian* matrix, $\mathbf{W}^H$. The rows of $\mathbf{W}^H$ are the conjugate of the columns of $\mathbf{W}$. Thus, all of the pairwise inner products can be computed simultaneously using the matrix product, $\mathbf{W}^H\mathbf{W}$. Compute this product, and then explain why the result equals the $N \times N$ identity matrix. Since $\mathbf{W}$ satisfies this property, it is called an *unitary* matrix.

Exercise 1.2: The Inverse DFT matrix

The unitary property leads to a trivial definition of the inverse DFT (IDFT).

(a) The IDFT matrix would be computed in MATLAB via: `Winv = ifft( eye(N) )`. This corresponds to the definition usually given in textbooks, but it is not $\mathbf{W}^{-1}$ because it is off by a scale factor. So, we will call this matrix $\mathbf{W}_i$.

(b) Due to the unitary property, the inverse of $\mathbf{W}$ is its Hermitian, $\mathbf{W}^H$. Determine the scale factor relating $\mathbf{W}^{-1}$ and $\mathbf{W}_i$. Verify by computing the largest entry in the difference matrix, $\mathbf{W}^{-1} - \alpha\mathbf{W}_i$, where α is the scale factor.

Project 2: Eigenvalues of the DFT Matrix

The eigenvalues of the DFT matrix follow an amazingly simple pattern. The MATLAB function `eig` makes it easy to compute the eigenvalues and explore these patterns.

Exercise 2.1: Eigenvalues

Use MATLAB to find all the eigenvalues of the DFT matrix; see `help eig`. Note that the repeated eigenvalues will be shown a number of times equal to their multiplicity.

(a) Do this for several consecutive values of N, perhaps $4 \leq N \leq 10$. Make a table of the eigenvalues and their multiplicities.

(b) Propose a *general* rule that gives the multiplicity of each eigenvalue as a function of N. Check your rule by finding the eigenvalues for $N = 3$, $N = 17$ and $N = 24$.

Exercise 2.2: Characteristic polynomial

The eigenvalues are also the roots of the *characteristic polynomial* of the matrix. The MATLAB function `poly` will compute the characteristic polynomial satisfied by the matrix, i.e., $p(\mathbf{W}) = \mathbf{0}$. However, this may not be the minimum order one, which is called the *minimal polynomial.* Since the DFT matrix has repeated eigenvalues, it is necessary to analyze the matrix directly to get the minimal polynomial.

(a) Generate an instance of the DFT matrix for a small value of N, say $N = 5$.

(b) Suppose the matrix is applied twice (i.e., take the DFT of the DFT):

$$\mathbf{y} = \mathbf{W}\mathbf{W}\mathbf{x}$$

Define a new matrix $\mathbf{J} = \mathbf{W}^2$ and observe that many entries of $\mathbf{J}$ turn out to be equal to zero. Then it is possible to write a simple expression for $\mathbf{y}$ in terms of $\mathbf{x}$. Why should $\mathbf{J}$ be called a "flip matrix"?

(c) Suppose the matrix $\mathbf{W}$ is applied 4 times:

$$\mathbf{z} = \mathbf{W}^4\mathbf{x} = \mathbf{J}^2\mathbf{x}$$

What is $\mathbf{z}$ in terms of $\mathbf{x}$?

(d) What is the minimal polynomial of $\mathbf{W}$? Compare to the MATLAB function `poly`. Show that both polynomials have the same roots, even though the multiplicities are different. Use `polyvalm` to apply both polynomials to the matrix $\mathbf{W}$ and show that the matrix does indeed satisfy both.

Exercise 2.3: Eigenvectors

The DFT matrix will have a complete set of orthonormal eigenvectors, because it is a unitary matrix.

(a) Find all the eigenvectors of the matrix $\mathbf{W}$, when $N = 8$. Associate each eigenvector with one of the four eigenvalues. Each of these four subsets constitutes an eigen-subspace. Verify that the eigenvectors from different eigen-subspaces are pairwise orthogonal.

(b) Due to repeated eigenvalues, the set of eigenvectors is not unique. It is possible to normalize and orthogonalize the eigenvector subset belonging to each eigenvalue, and, thereby, produce a new set of eigenvectors where all are pairwise orthonormal. Use the MATLAB function `orth` to perform this task. It is possible to apply `orth` to all the eigenvectors at once, but then the correspondence between eigenvector index and subspace will be lost.

(c) More than likely, a direct application of `orth` still yields complex-valued eigenvectors. However, it is always possible to find eigenvectors that are purely real. To do this, the first step is to note that the eigenvectors possess symmetry. All the eigenvectors belonging to the real eigenvalues display what would be called *even symmetry*; for the imaginary eigenvalues, the symmetry is *odd.* Even symmetry in these vectors means that the 2nd and last entries are equal, the 3rd and second to last are the same, etc. In matrix terms, an even symmetric vector is invariant under the operator $\mathbf{J}$ (defined above)

$$\mathbf{Jx} = \mathbf{x} \qquad \Longleftrightarrow \qquad \mathbf{x} \text{ is even symmetric}$$

Likewise, for odd symmetry we would see a negation due the flip operator $\mathbf{J}$.

$$\mathbf{Jx} = -\mathbf{x} \qquad \Longleftrightarrow \qquad \mathbf{x} \text{ is odd symmetric}$$

Verify these symmetries for the eigenvectors.

(d) A well-know property of the DFT is that a real even-symmetric input is transformed to a real even-symmetric output. Likewise, an imaginary-even input is transformed to an imaginary-even output. This observation can be used to justify the fact that either the real or imaginary part of each eigenvector can be used to construct the orthonormal eigen-subspaces. Justify this procedure and demonstrate that it works by using `orth`, `real`, and `imag` to construct a purely real eigenvector orthonormal basis. (The only complication comes when the real or imaginary part is zero, in which case the other part must be used.)

Exercise 2.4: Orthogonal Expansion

Any matrix that possesses a complete set of eigenvectors, can be expanded in terms of those eigenvectors. The expansion takes the following form when the eigenvectors are orthonormal:

$$\mathbf{W} = \sum_{n=1}^{N} \lambda_n \mathbf{w}_n \mathbf{w}_n^H \tag{2.1}$$

In the case of the DFT, there are only four distinct eigen-spaces, so the sum can be grouped according to the different eigenvalues, $\lambda \in \{1, -1, j, -j\}$:

$$\mathbf{Wx} = \sum_{n\in\mathcal{N}_1} \mathbf{w}_n(\mathbf{w}_n^H\mathbf{x}) - \sum_{n\in\mathcal{N}_2} \mathbf{w}_n(\mathbf{w}_n^H\mathbf{x}) + j\sum_{n\in\mathcal{N}_3} \mathbf{w}_n(\mathbf{w}_n^H\mathbf{x}) - j\sum_{n\in\mathcal{N}_4} \mathbf{w}_n(\mathbf{w}_n^H\mathbf{x}) \tag{2.2}$$

where $\mathcal{N}_1$ is the set of indices for eigenvectors belonging to $\lambda = 1$, $\mathcal{N}_2$ for $\lambda = -1$, and so on. Each term in parentheses is an inner product, requiring N multiplications.

(a) Write a MATLAB function that will compute the DFT via this expansion, specifically for the $N = 16$ case. Verify that the correct DFT will be obtained when the real eigenvectors (determined previously) are used, and compare to the output of `fft`.

(b) *Possible Computation.* Count the total number of operations (real multiplications and additions) needed to compute the DFT via the orthogonal expansion (2.2). Since the eigenvectors can be chosen to be purely real, the computation of the DFT via the orthogonal expansion will simplify when the input vector is purely real. The real part of the transform will depend only on the first two sums; and the imaginary part on the second two.

Exercise 2.5: Gaussian Sum is the Trace

Consider the following sequence which has quadratic phase:

$$x_\lambda[n] = e^{-j2\pi\lambda n^2/N} = W_N^{\lambda n^2} \qquad n = 0, 1, \ldots, N-1$$

This signal is a discrete-time chirp (linear-FM).

(a) When $\lambda = 1$, the sum of $x_\lambda[n]$ from $n = 0$ to $n = N-1$ is called the *Gaussian sum*. It is also equal to the trace of the DFT matrix and is, therefore, the sum of the eigenvalues. State a general rule for the value of the trace as a function of N.

(b) Compute the DFT of $x_\lambda[n]$ when $\lambda = 1$. Try several consecutive values of N. Plot the magnitude and unwrapped phase of the transform.

(c) Compute the N-point DFT of the sequence when $\lambda = \frac{1}{2}$. Show that when the length of the DFT is even, the DFT of the linear-FM signal is another linear-FM signal with a different value of λ.

(d) The derivation of the DFT of a chirp is difficult, but can be attacked by completing the square in the exponents. This works best for the case when $\lambda = \frac{1}{2}$ and N is even. In this case, the magnitude is a constant, so the trace can be used to find the magnitude and phase at DC.

Project 3: DFT Diagonalizes Circulant Matrices

A well known property of the DFT is its convolution-multiplication property. In terms of matrices, this property is equivalent to the fact a whole class of matrices will be diagonalized by the DFT matrix. This is the class of circulant matrices, a special kind of Toeplitz matrix. A square circulant matrix has only N distinct elements—it is completely defined by its first column; see the following example for $N = 5$.

$$\mathbf{C} = \begin{bmatrix} 1 & 5 & 4 & 3 & 2 \\ 2 & 1 & 5 & 4 & 3 \\ 3 & 2 & 1 & 5 & 4 \\ 4 & 3 & 2 & 1 & 5 \\ 5 & 4 & 3 & 2 & 1 \end{bmatrix}$$

The circulant matrix is important because the operation of circular convolution can be expressed as a matrix multiplication by a circulant.

Exercise 3.1: Generating a Circulant Matrix

Write a MATLAB function that will generate a circulant matrix. The function should take one argument: the vector that specifies the first column of the matrix. Use the function `toeplitz` as a model.

Exercise 3.2: Diagonalization of Cyclic Convolution by the DFT

(a) Generate a circulant matrix ($\mathbf{C}$) specified by a single column vector. Compute the eigen-decomposition of the circulant. Scale all the eigenvectors so that their first element is equal to $1/\sqrt{N}$.

(b) Show that the matrix formed from the eigenvectors is just a permuted version of the DFT matrix, i.e., the columns may be out of order.

(c) Show directly that the DFT matrix will diagonalize the circulant. Verify that the similarity transformation $\mathbf{W}^H\mathbf{C}\mathbf{W}$ gives a diagonal matrix. Compare the numbers on the diagonal to the DFT of the first column of the circulant (scale the DFT by $1/\sqrt{N}$).

Project 4: FFT algorithm as a matrix factorization

Project Description

The matrix form of the DFT suggests that the transformation from the n domain to the k domain is a matrix multiply that requires N^2 complex multiplications and $N(N-1)$ complex additions. This is only true when the length N is a prime number, but in many other cases, efficient FFT algorithms have been derived to reduce the number of multiplications and additions. These FFT algorithms can be described in terms of some simple factorizations of the DFT matrix.

Exercise 4.1: The Stretch and Decimate Matrices

The "stretch" operation involves inserting zeros between the elements of a vector; the "decimate" operation applied to a vector removes all the even-indexed elements (assuming that indexing starts at $n = 1$ (as in MATLAB). Both operations can be represented as matrix multiplications, and both have simple consequences in the frequency domain.

(a) In the stretch operation, you start with a length $N/2$ vector, the result is a length N vector, so the stretch matrix must be $N \times N/2$; call this matrix $\mathbf{S}$. Use $\mathbf{S}(N)$ if it is necessary to specify the length. Since the output vector has zero entries in all even-indexed elements, every corresponding row of $\mathbf{S}$ must be zero. Complete the description of $\mathbf{S}$, and give an example for $N = 10$.

(b) Likewise, the decimate operation can be represented by a matrix multiply with an $N/2 \times N$, called $\mathbf{D}(N)$. Describe $\mathbf{D}$, for $N = 10$, by giving all of its entries.

(c) Show that the stretch and decimate matrices are related via $\mathbf{S}(N) = \mathbf{D}^T(N)$.

(d) Prove that $\mathbf{D}(N)\mathbf{S}(N) = \mathbf{I}_{N/2}$; verify with MATLAB. Give an interpretation of this equation. What is the result of $\mathbf{S}(N)\mathbf{D}(N)$?

Exercise 4.2: The Stretch Property

The "stretch" property of the DFT states that interspersing zeros between samples in the n domain will cause a periodic repetition in the k (frequency) domain.

(a) Generate a DFT matrix for an even length, say $N = 6$. When applied to the stretched vector, the DFT becomes:

$$\mathbf{X} = \mathbf{W}\mathbf{x} = \mathbf{W}\mathbf{S}\hat{\mathbf{x}} \tag{4.1}$$

where $\hat{\mathbf{x}}$ is the $N/2$-point vector prior to zero insertion. Therefore, the matrix product $\hat{\mathbf{W}} = \mathbf{WS}$ is a reduced matrix whose size is $N \times N/2$. If we let $\mathbf{W}(N)$ denote the N-point DFT matrix, then this reduced matrix can be expressed solely in terms of $\mathbf{W}(N/2)$. Derive the form of $\hat{\mathbf{W}}$. Verify with a MATLAB example, for $N = 10$

(b) Use the form of $\hat{\mathbf{W}}$ to justify the stretch property; i.e., compare the first half and last half of $\hat{\mathbf{W}}$ to see the repetition in the vector $\mathbf{X}$. Generalize to the case of stretching-by-m where $m-1$ zeros lie between each sample.

Exercise 4.3: The Decimate Property

Repeat the steps in the previous exercise for the "decimate" property: the $N/2$ point DFT of the even-indexed time samples is obtained from the N-point DFT by adding the second half of the DFT vector to the first and dividing by 2 (aliasing). This property is a bit harder to prove.

(a) In this case, we assume that the N-point DFT is already known:

$$\mathbf{X} = \mathbf{W}(N)\mathbf{x} \qquad \Longleftrightarrow \qquad \mathbf{x} = \mathbf{W}^H(N)\mathbf{X}$$

The objective is to derive the $N/2$-point DFT $\mathbf{Y}$ in terms of $\mathbf{X}$

$$\mathbf{Y} = \mathbf{W}(N/2)\mathbf{y}$$

when $\mathbf{y} = \mathbf{D}(n)\mathbf{x}$. Use the relationship between $\mathbf{x}$ and $\mathbf{y}$ to expand the equation for $\mathbf{y}$ in terms of $\mathbf{W}(N/2)$ and $\mathbf{X}$. HINT: you must convert the decimate matrix into a stretch matrix.

(b) Now finish the derivation by writing $\mathbf{Y}$ in terms of the identity matrix $\mathbf{I}_{N/2}$ and $\mathbf{X}$. Interpret this equation as the decimate property.

(c) Verify your derivations with a MATLAB example.

Exercise 4.4: Mixed-radix factorization

The matrix notation can be used to illustrate the decomposition needed for the mixed-radix FFT algorithm. When $N = L \times M$, you can find the L-point DFT matrix, and the M-point DFT matrix, inside the N-point one.

• Try `W(1:L:N,1:1:M)` and compare to the M-point DFT matrix. Can you find other submatrices of $\mathbf{W}(N)$ that are equal to either $\mathbf{W}(M)$ or a scalar multiple of $\mathbf{W}(M)$?

The mixed-radix FFT can be written as a six-step process. Consider the specific case of a 12-point DFT that is factored as a 3×4 DFT:

1. Take the 12-point vector and concatenate it row-wise into a 4×3 matrix. Thus, down one column the entries will consist of every third point from the vector.

2. Compute the 4-point DFT of each column.

3. The "twiddle" multiplications correspond to a *pointwise* multiplication by a 4×3 matrix of complex exponentials.
4. Transpose the result to form a 3×4 matrix.
5. Compute the 3-point DFT of each column.
6. Re-order the result into a vector. How? That is one of the problems for this exercise.

In order to verify that this procedure works, derive the specific matrices needed for the 12-point FFT.

(a) Determine the entries for the matrix of twiddle factors.

(b) Define the re-ordering that is needed to build the k-domain vector from the 3×4 matrix that is the output of the 3-point DFT's.

(c) Demonstrate that the entire process will work by exhibiting the actual matrices involved. This can be done by forming a test vector that is the IDFT of the vector: `[ 0 : 1 : 11 ]`. This test vector will help uncover the re-ordering in the last step.

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{Convolution: Circular \& Block}}$

Convolution: Circular & Block

Overview

This set of projects will concentrate on the circular convolution property of the discrete Fourier transform (DFT). The relation to linear convolution will be studied; as well as the extension to block processing and high-speed convolution. The operation of N-point circular convolution is defined by the equation:

$$y[n] = x[n] \circledN h[n] = \sum_{\ell=0}^{N-1} x[\ell]h[(n-\ell) \bmod N] \tag{0.1}$$

Note that circular convolution combines two N-point vectors to give an answer that is also an N-point vector.

In its own right, circular convolution has little or no use. However, it is a by-product of the DFT and is, therefore, easy to compute via the FFT. The reason that the study of circular convolution is an essential part of DSP, is that it is related in a simple manner to normal convolution, $x[n] * h[n]$, which will be called linear convolution here to distinguish it from $x[n] \circledN h[n]$. Therefore, the FFT can be used to speed up the computation of a linear convolution. Several exercises will study that important connection in detail.

Background Reading

All DSP textbooks have chapters on circular convolution, block convolution, and high-speed convolution via the FFT.

Project 1: Circular Indexing

Circular convolution and the DFT both require that all indexing be done in a circular (or periodic) fashion. At first, these operations seem a bit cumbersome, but Matlab will facilitate the visualization of this sort of indexing.

Hints

Using a simple test signal, such as a ramp, makes it easy to track the circular shifts. Plotting two signals together, via `subplot(21x)` helps visualize the action done by the shift.

Exercise 1.1: Circular shifts and rotations

Indexing for the DFT must always be performed in a "circular" fashion. Thus the expression $x[n-1]$, which usually means "shift right by one sample", must be re-interpreted. Likewise, for the flip operation $x[-n]$.[7]

(a) In order to preserve the Fourier property that says "a shift in one domain is multiplication by a complex exponential in the other", we must define the shift using the

[7]The circular flip operation was also treated in the project on DFT Symmetries, where a function called `cflip` was developed.

modulo operator (from number theory).

$$x[n-\ell] \to x[(n-\ell) \bmod N] = \begin{cases} x[n-\ell] & \text{for } n = \ell, \ell+1, \ldots, N-1 \\ x[n+N-\ell] & \text{for } n = 0, 1, \ldots, \ell-1 \end{cases}$$

This assumes that $0 \le \ell < N$. The operation is referred to as a circular shift because as the sequence is shifted to the right, indices greater than or equal to N are wrapped back into the smaller indices.

The DFT of a circularly shifted sequence is just the original DFT multiplied by a complex exponential vector, $W_N^{+\ell k}$. Verify this property by using simple test inputs such a shifted impulses, shifted pulses, and shifted sinusoids.

For a 16-point DFT, show that a circular shift by 8 is a special case where the transform values will only have their signs modified.

(b) Given the sequence `[ 0 1 2 3 4 5 6 7 ]`, rotate this to the new sequence `[ 4 5 6 7 0 1 2 3 ]`, using only DFT and complex multiply operations. Repeat for a rotation to `[ 1 2 3 4 5 6 7 0 ]`.

(c) Write a function that will compute $n \bmod N$. The `rem` function in MATLAB is not sufficient because it will not handle the case where n is negative. The following simple modification that calls `rem` twice will produce a correct modulo function that always gives an answer in the range $[0, N-1]$. Explain.

```
mod(x,N) = rem( rem(x,N) + N, N);
```

(d) Write a MATLAB function `rotate` that will circularly shift a vector by ℓ places. The function should be general enough to handle rotations when ℓ is negative, and when ℓ is greater than N. Consider how this could be done by an equivalent positive rotation.

Exercise 1.2: Circular Flip

The operation of circular flipping was discussed in conjunction with properties of the DFT. The flip operation $x[-n]$ must also be interpreted in a circular fashion. The index replacement, $n \to -n$, becomes $(-n) \bmod N$, which is a circular reversal of a vector. In this case the index $n = 1$ is exchanged with $n = N-1$, $n = 2$ with $n = N-2$, and so on. The index $n = 0$ does not move. Write a function called `cflip()` that will implement this flip operation.[8] Verify that flipping `[0:1:7]` gives `[ 0 7 6 5 4 3 2 1 ]`.

Exercise 1.3: Flipping and Shifting Signals

In this exercise, you should generate the flipped and rotated vectors as found in circular convolution. All indexing within circular convolution is done within the index range $0 \le n < N$. For example, in (0.1) the difference $(n-\ell) \bmod N$ is needed for the signal $h[\cdot]$.

Consider the signal $h[(n-\ell) \bmod N]$ in the circular convolution sum (as a function of ℓ). Starting with the vector for h, two steps are needed to construct $h[(n-\ell) \bmod N]$ vs. ℓ; a circular flip and a circular shift by n.

[8]Should we provide a CASPER function for `cflip()`; do we also need a modulo-N M-file?

(a) Use the two functions, `cflip` and `rotate` to produce examples of sequences that are flipped and shifted, circularly of course.

1. Start with the 11-point sequence, $x[n] = n + 2$.
2. Plot $x[(\ell - 2) \bmod 11]$ versus $\ell = 0, 1, \ldots, 10$.
3. Plot $x[(\ell + 3) \bmod 11]$ versus ℓ.
4. Plot $x[(4 - \ell) \bmod 11]$. Should you flip first and then shift, or vice versa?
5. Plot $x[(-\ell - 5) \bmod 11]$. If the flip is first, will the shift be by $+5$ or -5?

(b) Generate a exponential signal $x[n] = (0.87)^n$ that is 13 points long; and generate the list of 13 indices `nn = 0:12` for n. Perform a circular shift of $x[n]$ to get $y[n] = x[(n - 4) \bmod 13]$. Do this by shifting the index vector only, and then plotting $x[n]$ versus n-shifted. Which way do you have to rotate `nn`?

(c) Repeat for $z[n] = x[(n + 2) \bmod 13]$. Is this a shift to the "right" or the "left"?

Project 2: Circular Convolution

Combining N-point vectors according to the circular convolution rule (0.1) is not particularly easy to visualize. In this project, we break the circular convolution operation down into its elements: circular shifting (i.e., rotation) and (circular) flipping.

Hints

Should there be a circular convolution function available via CASPER? There is one called `circonv`, that uses time-aliasing plus `conv`.[9]

Exercise 2.1: Function for circular convolution

There are two ways to write a function for circular convolution: (1) in the transform domain, or (2) in the time domain. The MATLAB function would have two inputs, the signal vectors `h` and `x`, and one output signal `y`.

(a) The circular convolution of $x[n]$ and $h[n]$ is equivalent to the multiplication of their DFTs, $X[k] \times H[k]$. Use this idea to write a circular convolution function that requires 3 calls to `fft`. Try the following simple tests for $N = 16$ and $N = 21$.

1. Impulse at $n = a$ convolved with an impulse at $n = b$, where a and b are integers. That is, $x[n] = \delta[(n - a) \bmod N]$ and $h[n] = \delta[(n - b) \bmod N]$.
   • In this case, the output has only one non-zero value; determine its location.
   • Let $b = -a \bmod N$, for a simple case.
2. Two short pulses. Let $x[n]$ be a pulse of length 5, and $h[n]$ a pulse of length 8, starting at $n = 4$.
   • Verify that your function computes the correct output.

[9]Also, see CASPER special called `circulant`.

3. Two long pulses such that the output wraps. Let $x[n]$ be a pulse of length 11, and $h[n]$ a pulse of length 7, starting at $n = 5$.
   • Compute the output and check its correctness. Note that the answer is different for the length 16 and 21 cases.
   • Try different starting positions for both pulses.

(b) Write a circular convolution function directly from the definition (0.1). This can be done with `for` loops, but good MATLAB programming style demands that vector operations be used instead. One way is to regard each output as formed from the inner product of `x` with circularly flipped and shifted versions of `h`. Then only one loop is needed. Write the function based on this idea, and check it on the examples above.

(c) A refinement of the inner-product approach would be to do the operation as a matrix-vector multiply. Use the circularly shifted versions of $h[n]$ to construct a "circulant" matrix—one whose columns are all just rotations of the first column. A square circulant matrix has only N distinct elements—it is completely defined by its first column (or its first row); see the following example for $N = 5$.

$$\mathbf{C} = \begin{bmatrix} 1 & 5 & 4 & 3 & 2 \\ 2 & 1 & 5 & 4 & 3 \\ 3 & 2 & 1 & 5 & 4 \\ 4 & 3 & 2 & 1 & 5 \\ 5 & 4 & 3 & 2 & 1 \end{bmatrix}$$

Write a function to construct a circulant matrix, and then call that function in doing the circular convolution. Again, check the examples from part (a).

Exercise 2.2: More Examples of Circular Convolution

The following signals should be combined by circular convolution. They can be used to check out the different M-files written in the previous exercise. In parts (a–c), try an even and odd length for N, say $N = 16$ and $N = 21$.

(a) $x[n] = 1$ for all n and $h[n] = (-1)^n$.
• Notice the difference in the even and odd cases.
• Given $N = 16$, can you find other inputs for which the output will be zero?
• When N is odd, is it possible to find an $x[n]$ for which the output will be zero?

(b) Let $x[n]$ be a ramp: $x[n] = n$ for $0 \le n < N$, and let $h[n] = \delta[n-3] - \delta[n-4]$.
• Verify that your function computes the correct output.

(c) Two periodic pulse trains. Let $x[n]$ be non-zero only when n is a multiple of 3; and let $h[n]$ be non-zero only for n a multiple of 4.
• Compute the output and check its correctness. Note that the answer is different for the length 16 and 21 cases.

(d) Generate the signal $x[n] = (-1)^n + \cos(\pi n/2)$; create 50 points. Generate another signal $h[n]$ as a finite pulse of length 11. Compute the 50-point circular convolution of these two signals; zero-pad $h[n]$ with 39 zeros. You can also do this analytically (paper and pencil).

Exercise 2.3: Circular Deconvolution

Suppose $y[n] = x[n] \circledN h[n]$, where the convolution is circular. If $y[n]$ and $h[n]$ are known, then recovering $x[n]$ is called the "deconvolution" problem. It is difficult to solve in the time-domain, but rather easy in the DFT domain. Consider the following puzzle:

Nine Tech students are seated around a circular table at dinner. One of the students (a DSP expert) challenges the others to a guessing game to demonstrate her "magic" powers. While she leaves the room, each of the other eight students ask those seated to his/her immediate right and left what their IQ is, adds it to their own and reports the sum of three IQs. When she returns, these eight sums are given to the DSP student to work her magic. The game is to use just these 8 partial sums to determine each student's IQ.

(a) The DSP magician is confident that she will be able to solve the puzzle, because it is "just circular deconvolution." Show how to *model* this puzzle as a circular convolution. Call the sums $s_3[n]$, and the IQs $q[n]$, for $n = 0, 1, 2, \ldots, 7$. State an algorithm (based on the DFT) for determining $q[n]$ from $s_3[n]$ and prove that it will work in all cases. Write a simple MATLAB M-file that will solve the puzzle; it should also generate $s_3[n]$.

(b) In order to further demonstrate her powers, the DSP student challenges another student to play the same guessing game but with a slight modification in the rules. This time, each of the remaining 8 students will ask only the person to his/her left for their age, and these sums will be reported to the group. Show that, in this case, it might not be possible to get the answer. Use MATLAB to generate a specific counter-example to *prove* that a solution is not generally possible for the case where the sums are taken over two people. Let the IQ be denoted as $q[n]$, for $n = 0, 1, 2, \ldots, 7$ and the two-person sums as $s_2[n]$.

Project 3: Relation to Linear Convolution

Circular convolution is most useful because it can be related to linear convolution, $y[n] = x[n] * h[n]$, which is the normal operator that applies to linear time-invariant systems.

Hints

In MATLAB there is a function called `convmtx` which produces a linear "convolution" matrix. There is also a CASPER special `convolm` which makes a convolution matrix function that has some flexibility for zero-padding at both ends.

Exercise 3.1: Study the `conv` function

In MATLAB convolution can be performed by the function `conv`. Any two finite-length signals can be convolved.

(a) Demonstrate the `conv` function by doing the convolution of two rectangular pulses; both of length 7. The expected result is a signal that has a triangular shape. Verify with some examples of signals with randomly chosen lengths. Note: if the signals are zero-padded, then the MATLAB function `length` will count the zeros.

Exercise 3.2: Convolution as a Matrix Operation

The circulant matrix is a special case of a Toeplitz matrix. The case of a Toeplitz matrix is important because there is another special case that corresponds to linear convolution. In other words, the operation of convolving two finite-length signals can be represented as a matrix-vector product. Interpretation of the convolution sum in this way leads to the "convolution matrix", which is a rectangular $(N \times p)$ Toeplitz matrix whose first row is all zero except for the first element, and whose first column has zeros in its last $p-1$ elements.

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \\ 4 & 3 & 2 & 1 \\ 0 & 4 & 3 & 2 \\ 0 & 0 & 4 & 3 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

The convolution of $x[n]$ and $h[n]$ can be done by making one of the signals the non-zero part of the first column, and letting the other signal be a vector that multiplies the convolution matrix.

$$\mathbf{y} = \mathbf{H}\mathbf{x} \tag{3.1}$$

(a) Do the convolution of a 3-point ramp with a 7-point ramp, by constructing the convolution matrix. What are the dimensions of the matrix? Check with the output of `conv`.

(b) Suppose that we wanted to do deconvolution based on the matrix representation of convolution (3.1). Thus we assume that $\mathbf{y}$ and $\mathbf{H}$ are known. Since (3.1) is always under-determined, i.e., more equations that unknowns, $\mathbf{x}$, the answer will not be unique. In MATLAB the backslash operator will still compute a solution for this case.

For the case of the two ramps in the previous part, let the 7-point ramp be $h[n]$. Apply the backslash operator to see how well the inversion of (3.1) can be done, i.e., compare the result to the expected 3-point ramp. This is a noise-free case, so you might want to add a bit of noise to the $\mathbf{y}$ vector to simulate errors.

Exercise 3.3: Circular Convolution via Time Aliasing

We first consider how to modify a linear convolution to obtain a circular convolution. One way to use `conv` to compute a circular convolution is to take the results of `conv` and perform a time-aliasing on the vector.

$$\tilde{y}[n] = \sum_{\ell} y[n+\ell M]$$

The effect of the summation is to produce a value of $\tilde{y}[n]$ by adding up all values of $y[n]$ that are offset by M. The constructed signal $\tilde{y}[n]$ will be periodic with period M. Since all three signals in the circular convolution must have the same length, we would pick M in the time-aliasing to be that length.

(a) Write a MATLAB function that will perform time-aliasing. The parameter M must be one of its inputs.

(b) Return to the example of convolving two length-7 pulses. Perform a 9-point circular convolution with `conv` followed by time-aliasing. This requires that both signals be zero-padded out to the length 9. Check against the results obtained with one of the previously written circular convolution M-files.

(c) Repeat for circular convolution lengths of 11 through 16. When does the time-aliasing cease to have an effect?

Exercise 3.4: Circular Convolution via Periodic Extension

A second way to use `conv` to get a circular convolution result is to perform what amounts to "periodic convolution." In this case, three steps are needed:

1. One of the input signals is extended in a periodic fashion from length N to length N', say $x[n]$. The new signal is called $x'[n]$.
2. The `conv` function is applied to compute $y'[n] = h[n] * x'[n]$. The output length is greater than N.
3. Finally, N points are selected from the output. Which ones, that is a question below.

(a) Determine the minimum length N' needed for the extension.

(b) Determine which points to take from $y'[n]$.

(c) Write an M-file function that will implement this approach to circular convolution. Test it on some of the previous examples.

Exercise 3.5: Zero-Padding

As suggested by the time-aliasing example, zero-padding can be used to make circular convolution give a correct results for linear convolution. All that is needed, is to make the length of the circular convolution long enough that time-aliasing does not come into play.

(a) Generate two random signals; the signal $x[n]$ should be length 50, the other signal $h[n]$ of length 27. What is the length of $x[n] * h[n]$?

(b) Compute the N-point circular convolution of these two signals, where N is the length of the linear convolution. Verify that the answer for circular convolution matches the linear convolution.

(c) Use FFTs to do a circular convolution of length 128. Zero pad both $x[n]$ and $h[n]$. Multiply the DFTs, $X[k]$ and $H[k]$, to get $Y[k]$ (`Y = H .* X;` in MATLAB) and take the inverse DFT to get $\hat{y}[n]$.

(d) Verify that $\hat{y}[n]$ contains the values of the linear convolution of $x[n] * h[n]$.

Therefore, we see that any circular convolution longer than the minimum will work.

Exercise 3.6: Good Outputs vs. Bad Outputs

In the previous examples, the comparison of the circular convolution output to a linear convolution output shows that they are not always the same. Some values may be correct, while others are wrong because of time-aliasing.

In the overlap-save method of block convolution, it will be important to identify these good and bad points. So we consider two cases—one with zero-padding, one without.

(a) Suppose that we convolve (circularly) the signals

$$x[n] = \begin{cases} (0.9)^n & 0 \le n < 13 \\ 0 & \text{elsewhere} \end{cases} \qquad h[n] = \begin{cases} 1 & 0 \le n < 12 \\ 0 & \text{elsewhere} \end{cases}$$

Choose the length of the circular convolution to be 21. Determine which values of $\hat{y}[n] = x[n] \, Ⓝ \, h[n]$ are the same as those in the linear convolution result $y[n] = x[n] * h[n]$. Give the list of output indices where the values are "bad."

(b) Suppose the two signals are defined as:

$$x[n] = \begin{cases} (0.9)^n & 0 \le n < 13 \\ 0 & \text{elsewhere} \end{cases} \qquad h[n] = \begin{cases} 1 & 9 \le n < 21 \\ 0 & \text{elsewhere} \end{cases}$$

Where are the good and band points now that $h[n]$ has zeros at the beginning?

(c) Consider the following example, which relates to the overlap-save situation.

$$x[n] = \begin{cases} 1 & 0 \le n < 17 \\ 0 & \text{elsewhere} \end{cases} \qquad h[n] = \begin{cases} \sin(n\pi/13) & 0 \le n < 100 \\ 0 & \text{elsewhere} \end{cases}$$

Suppose that a 100-point circular convolution is performed. (There is no zero-padding of $h[n]$.) Determine the good and bad points in the output.

Project 4: Block Processing

The case of convolving two short signals is not very useful for filtering. In continuous filtering, the input $x[n]$ would be extremely long, at least with respect to the filter's impulse response $h[n]$. Furthermore, the entire input may not be available at one time so it is not feasible to do the circular convolution with a length that is greater than $L_x + L_h - 1$. Instead, a more reasonable strategy is to chop up the input into blocks, and process each one through a circular convolution. This is the basic idea that leads to "FFT convolution."

Hints

There are two types of block processing algorithms: overlap-add and overlap-save. Within MATLAB the function `fftfilt` implements the overlap-add method to do long convolutions. However, in the following exercises you should develop your own M-files to implement these two methods of block convolution.

Exercise 4.1: Sectioning to form small convolutions

Exercise 4.2: Overlap-add

The overlap-add method works by breaking the long input signal into small non-overlapping sections. If the length of these sections is M, and the length of the impulse response is L_h, then a circular convolution length of $N > M+L_h-1$ will avoid all time-aliasing effects (through the use of zero-padding). However, each piece of the output is now longer than M. To put the output together it is necessary to add together the overlapping contributions from each segment. Therefore, this method could also be called the "overlap-outputs" method. The description here is brief, so further details must be taken from a DSP textbook.

(a) The following code was adapted from the MATLAB function `fftfilt` which implements the overlap-add method. Point out where in this code, the overlap is taken care of. Notice that the section length is not the same as the FFT length.

```
H = fft(h,Nfft);
M = Nfft - length(h) + 1;     %--- Section Length
%******* assume that length(x) is multiple of M *******
%
for ix=1:L:length(x)
     x_seg = x(ix:ix+M-1);         %--- segment x[n]
     X= fft(x_seg, Nfft);          %--- zero pads
     Y = X.*H;
     y_seg = ifft(Y);
     y(ix:ix+Nfft-1) = y(ix:ix+Nfft-1) + y_seg(1:Nfft);
end
%
%------ check for purely REAL case -----------
if ~any(imag(b)) & ~any(imag(x))
     y = real(y);
end
%------------------------------------------
```

(b) Write an MATLAB function that implements the overlap-add method of block convolution. One of the inputs to this function should be either the section length M, or the FFT length. The circular convolution should be done in the DFT domain. Ultimately, this will provide the fastest running time for your program.

(c) Test the function by comparing to a convolution done with `conv`.

Exercise 4.3: Overlap-save

The overlap-save method uses a different strategy to break up the input signal. If the length of the circular convolution is chosen to be N, then input segments of length N are taken. Each segment has a starting location that is offset by a block length amount called M. Thus this method could be called the "overlapped inputs" method.

The filter's impulse response is zero-padded from length L out to length N and an N-point circular convolution is computed (probably using N-point DFTs).

Using the idea of good and bad points, it is possible to identify $M = N-L+1$ good points in the output. These M points inserted into the output stream. No additions are needed to create the output, so this method is often preferred in practice.

(a) Write a MATLAB function to implement the overlap-save method of block convolution. Either the block length M or the circular convolution length N must be specified. It is best to specify N since it should usually be chosen as a power of 2 to exploit the FFT algorithm.

(b) Test this function versus `conv`.

(c) Use MATLAB to count the number of floating-point operations via `flops`, and compare to the overlap-add method, and to `conv`. Make a table for $L_h = 10, 20, 50$ and $L_x = 1000$; use $N = 128$.

Exercise 4.4: Breaking up both sequences

In some rare cases, both sequences, $x[n]$ and $h[n]$, may be too long for the FFT, e.g., if fixed hardware is being used for the FFT. Then the circular convolution length N would be fixed. If both sequences are longer than N, then both must be segmented prior to block processing.

(a) Take the specific case where $N = 32$, and $L_h = 68$ and $L_x = 60$. If an "overlap-add" strategy is used, and the section length is chosen to be 12 for segmenting $x[n]$, determine a section length for $h[n]$, and also how many 32-point circular convolutions must be done.

(b) Implement this method in an M-file, but try to do the program so that the segment length for each signal is a variable. Experiment with different block lengths.

(c) If the performance objective were to minimize the total number of circular convolutions, determine the best choice of section lengths.

One strategy for the algorithm is to use a loop around either the overlap-add or overlap-save method. In this case, the approach is to break off part of $h[n]$ and convolve it with the entire signal $x[n]$; then break off another piece and convolve. Finally, all the convolutions would be added together.

Project 5: High-speed convolution

Since circular convolution can be implemented in the transform domain, and since the FFT algorithm is a very efficient means to compute the DFT, the combination yields an extremely fast method for convolution. The best choice for an FFT length is a power of two, so it is best to choose the block lengths accordingly. However, should one use the next higher power of two, or something larger? When would it be better to use direct convolution? A plot of operation counts versus FFT length or versus filter length will give the answer.

Hints

Use `etime` and `clock` to do timing; or `flops` to count operations.

Exercise 5.1: An FFT `conv` Function

The `conv` function in MATLAB calls `filter` to do its computation. This is efficient when the lengths of the signals to be convolved are small. However, if the signal lengths are long (e.g., > 30), then `filter` is quite slow; and a convolution based on the FFT would be much faster. In fact, such an option is available in MATLAB if the `fftfilt` function is used in a special way. However, for this exercise, you must write an M-file function that could replace the existing `conv` function.

1. The new convolution function should only use the FFT.
2. When calling the FFT, it should use a power of two.
3. The function must return a vector that is the proper length, not one with extra zeros tacked on.

After doing the next exercise, you may want to modify this function so that it only uses the FFT when that would be more efficient; otherwise, it would call `conv`.

Exercise 5.2: Cross-over point

The first experiment to run involves a comparison of FFT convolution versus direct convolution, as implemented with `conv`. For extremely short convolutions, the direct convolution is more efficient, but for longer ones the $\log_2 N$ behavior of the FFT makes it much faster. A generally quoted number for the cross-over point where the FFT has less operations is $N \approx 23$, or $N = 32$ if you stick with powers of two. However, this number is a function of the precise implementation, so we would like to deduce it for MATLAB.

In this exercise we will construct the plot by comparing the running time of the FFT convolver from the previous exercise to `conv`.

(a) Generate two signals $x[n]$ and $h[n]$, both of length L. The length will be varied over the range $10 \leq L \leq 80$. Note: this range might have to be changed if the running time of the FFT on your computer is too slow.

(b) Since the FFT can do complex-valued operations, the generated signals should be complex. This will effect the flops counter in `filter`.

(c) Convolve $x[n]$ with $h[n]$ using the `conv` function. Have MATLAB measure the number of floating point operations. Save these values in a vector for plotting later on.

(d) Now do the convolution by using an FFT that is long enough to contain the entire result of the linear convolution. Use the next higher power of two. The convolve function written in the previous exercise could be used here. Again, measure the flops and save in a vector.

(e) Plot the two vectors of flop counts together. Determine the cross-over point where the FFT method is faster. Note that there is actually not just one cross-over, but it is possible to determine a length beyond which the FFT is almost always better.

(f) If you have time, make a true comparison for real-valued signals. This will involve writing a MATLAB function for a real FFT—one that exploits the real-valued nature of the input.

Exercise 5.3: Compare with FIR filtering

The previous exercise concentrates on an artificial case where the lengths of $x[n]$ and $h[n]$ are identical. A different situation that is more interesting is the case of continous filtering. Here, one of the inputs is indefinitely long—it could be a continuous stream of data. Therefore, the block length for an overlap-add or overlap-save algorithm needs to be chosen. Increasing the block length will make the process more efficient, but only up to a point! As a rule of thumb, it turns out that a block length on the order of 5–10 times the filter length is a good choice. In order to see that there is an optimum choice, we can construct a plot of operation counts versus FFT length.

(a) Take the filter length to be $L_h = 40$. To simulate the very long input, make the length of $x[n]$ as large as possible in your version of MATLAB; greater than 20,000 if possible. The two signals can be random sequences.

(b) Since the FFT can operate on complex data, we must be fair (to the FFT) and do the comparison for a complex-valued convolution. Otherwise, we should use one of the modified FFT algorithms that does the FFT of real data with a half-length FFT.

(c) Use `fftfilt` to do the convolution in sections. One of the inputs to `fftfilt` is the section length, so start at $N = 64$, and try successively higher powers of 2.

(d) Measure the flops and collect in a vector. Convert to operations per output point and plot versus $\log_2 N$. For comparison, do the filtering with `conv`, and count the flops. Convert to operations per output and plot this number as a horizontal line.

(e) If you put this process in a loop, and print out the length of the FFT each time through the loop, you can see roughly how long it takes each time and gauge which FFT length is best.

(f) Repeat this experiment for several different filter lengths, L_h. Do enough cases to verify that the rule of thumb stated above is correct. Notice also that several lengths near the optimum yield about the same performance.

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{Related Transforms}}$

Related Transforms

Overview

This set of projects will introduce two other transforms that are closely related to the DFT. These are the cosine transform and the discrete Hartley transform. There are many other discrete orthogonal transforms that could also be studied with MATLAB. For example, `hadamard` in MATLAB will generate a matrix for the Hadamard transform.

Background Reading

Need a reference

1. P. Yip and K. Ramamohan Rao, "Fast Discrete Transforms," in *Handbook of Digital Signal Processing: Engineering Applications,* ed. by D. F. Elliot, Academic Press: San Diego, CA, 1987.

2. R. N. Bracewell, *The Hartley Transform,* Oxford University Press: New York, 1986.

Project 1: Discrete Cosine transform (DCT)

The DCT has found wide-spread use in coding applications. It is part of the JPEG standard for image coding. Originally, the DCT was developed as an approximation to the optimal Karhunen-Loève transform. Since then, numerous fast algorithms have been developed for its computation.

There are four types of DCTs. Their definition can be written in the form

$$C[k] = \sum_{n=0}^{N-1} x[n]\,\phi_{kn} \tag{1.1}$$

where the basis functions ϕ_{kn} take the following form:

$$\text{TYPE I} \qquad \phi_{kn} = \sqrt{\frac{2}{N}}\left[c_k\, c_n \cos\left(\frac{kn\pi}{N}\right)\right] \tag{1.2}$$

$$\text{TYPE II} \qquad \phi_{kn} = \sqrt{\frac{2}{N}}\left[c_k \cos\left(\frac{k(n+\frac{1}{2})\pi}{N}\right)\right] \tag{1.3}$$

$$\text{TYPE III} \qquad \phi_{kn} = \sqrt{\frac{2}{N}}\left[c_n \cos\left(\frac{(k+\frac{1}{2})n\pi}{N}\right)\right] \tag{1.4}$$

$$\text{TYPE IV} \qquad \phi_{kn} = \sqrt{\frac{2}{N}}\left[\cos\left(\frac{(k+\frac{1}{2})(n+\frac{1}{2})\pi}{N}\right)\right] \tag{1.5}$$

and where the factor c_ℓ is

$$c_\ell = \begin{cases} 1/\sqrt{2} & \text{if } \ell = 0 \bmod N \\ 0 & \text{if } \ell \neq 0 \bmod N \end{cases} \tag{1.6}$$

Exercise 1.1: Basic Properties of DCT

The DCT is a purely real transform, unlike the DFT which requires complex numbers.

(a) Write an M-file to implement DCT-I, the TYPE I DCT. Do the implementation directly from the definition. As usual in MATLAB, do the operation as a matrix-vector multiply by setting up a Φ matrix defined in (1.1). The factors of c_n and c_k can be implemented as pointwise multiplications of either the input or output vectors; the factor of $\sqrt{2/N}$ must also be appllied to the output.

(b) Show that the DCT-I is its own inverse, and that the DCT-I matrix is unitary (its inverse is its transpose). Show the same for DCT-IV.

(c) Write M-files to implement the other types of DCTs.

(d) Show that the inverse DCT for DCT-II is not DCT-II, but rather DCT-III. Also show that the DCT-III matrix is the transpose of the DCT-II matrix.

(e) Determine whether or not the DCT has a circular convolution property, i.e., find the DCT of $x[n] \circledN h[n]$ and see whether or not it has a simple form.

Exercise 1.2: Computing the DCT via an FFT

If we write the kernel of the DCT-I as

$$\cos\left(\frac{kn\pi}{N}\right) = \cos\left(\frac{2\pi kn}{2N}\right) = \Re e\left\{W_{2N}^{nk}\right\}$$

Then it is easy to see that the DCT can be constructed from a $2N$-point DFT. Let's consider the case of the TYPE II DCT in detail. In this case the kernel function is

$$\phi_{kn} = \left[c_k \cos\left(\frac{k(n+\frac{1}{2})\pi}{N}\right)\right] = c_k\, \Re e\left\{W_{2N}^{k(n+\frac{1}{2})}\right\}$$

If we expand the exponential and plug it into the DCT-II definition, we get

$$C_{II}[k] = c_k\, \Re e\left\{W_{2N}^{k/2} \sum_{n=0}^{2N-1} x[n]\, W_{2N}^{nk}\right\} \qquad \text{for } k = 0, 1, \ldots N-1 \tag{1.7}$$

Therefore, the DCT-II can be computed using a $2N$-point FFT using three steps:

1. Zero pad $x[n]$ out to length $2N$, and compute its FFT. Remember that $x[n]$ starts out as a real-valued length-N sequence. Since $x[n]$ is real, the FFT output will be conjugate-symmetric.

2. Take the first N outputs of the FFT and multiply them by $W_{4N}^{k} = W_{2N}^{k/2}$.

3. Take the real part and then multiply it by c_k, and by $\sqrt{2/N}$.

Obviously, the same approach can be used for the other three types of DCTs.

(a) Implement M-files for all types of DCTs based on this fast computation strategy.

(b) Test these against the direct method of computation implemented in the previous exercise.

(c) A second approach to computing via the FFT is to create the length $2N$ input to the FFT so as to eliminate the real part operator. This is done for the DCT-II case by creating

$$\tilde{x}[n] = \begin{cases} x[n] & \text{for } n = 0, 1, \dots N-1 \\ x[2N-1-n] & \text{for } n = N, N+1, \dots 2N-1 \end{cases}$$

Then the $2N$-point FFT output only needs to be multiplied by $\sqrt{2/N}\, c_k\, W_{4N}^{k}$ Implement this method for DCT-II and verify that it gives the same result as the other implementations.

(d) The fact that we must use a length $2N$ FFT is bothersome. A third way to approach the computation reduces the FFT length to N. In this case, the signal $x[n]$ is packed into a new N-point vector as

$$y[n] = \begin{cases} x[2n-1] & \text{for } n = 1, 2, \dots, N/2 \\ x[N-2n] & \text{for } n = N/2+1, \dots N-1 \\ x[0] & \text{for } n = 0 \end{cases}$$

Then we compute the N-point FFT and multiply the result by W_{4N}^{k}; call this result $Y[k]$. Now we must extract the real and imaginary parts to create a vector that is almost the DCT.

$$\tilde{Y}[k] = \begin{cases} \Re e\{Y[k]\} & \text{for } k = 0, 1, \dots N/2 \\ \Im m\{Y[N-k]\} & \text{for } k = N/2+1, \dots N-1 \end{cases}$$

The DCT-II is obtained by multiplying $\tilde{Y}[k]$ by $\sqrt{2/N}\, c_k$.

Implement this method and compare the number of flops to the previous computations with the $2N$-point FFT.

Exercise 1.3: Discrete Sine Transform

Similar to the DCT, we can define a discrete sine transform (DST) which has four forms. In this case, the basis functions ϕ_{kn} take the following form:

$$\text{TYPE I} \qquad \phi_{kn} = \sqrt{\frac{2}{N}} \left[c_k c_n \sin\left(\frac{kn\pi}{N}\right)\right] \tag{1.8}$$

$$\text{TYPE II} \qquad \phi_{kn} = \sqrt{\frac{2}{N}} \left[c_k \sin\left(\frac{k(n-\frac{1}{2})\pi}{N}\right)\right] \tag{1.9}$$

$$\text{TYPE III} \qquad \phi_{kn} = \sqrt{\frac{2}{N}} \left[c_n \sin\left(\frac{(k-\frac{1}{2})n\pi}{N}\right)\right] \tag{1.10}$$

$$\text{TYPE IV} \qquad \phi_{kn} = \sqrt{\frac{2}{N}} \left[\sin\left(\frac{(k-\frac{1}{2})(n-\frac{1}{2})\pi}{N}\right)\right] \tag{1.11}$$

(a) Write M-files for the four types of DSTs. Use an approach based on the $2N$-point FFT to speed up the computation. In this case, the sine terms can be generated by taking the imaginary part.

(b) Determine the inverse transforms for each type. Show that DST-I is its own inverse, and find the inverse of DST-II.

Exercise 1.4: Performance of the DCT

The DCT finds its main application in coding, because its performance approaches that of the KL (Karhunen-Loève) optimal transform. This exercise shows how to construct a simple experiment to illustrate this fact. Suppose that the data to be coded comes from a first-order Markov process. Then its covariance matrix will be $\mathbf{R}$ with entries $r_{mn} = \rho^{|m-n|}$. If the optimal KL method is used, then the matrix $\mathbf{R}$ is diagonalized by an eigenvector deomposition, and the eigenvalues determine which vectors are the best for coding. If ℓ vectors were to be used, the eigenvectors corresponding to the ℓ largest eigenvalues would be chosen. The eigenvalue itself is a measure of how much energy is in that feature. Therefore, one way to compare alternative methods is to calculate $\mathbf{TRT}^{-1}$ for different transforms and compare the diagonal elements (see ref [1]).

(a) Generate the $N \times N$ covariance matrix for $\rho = 0.9$; choose $N = 16$.

(b) Do the KL transform using `eig`. Make a vector of the eigenvalues in decreasing order, and save for comparison to other methods.

(c) Use the DCT-II as the transform matrix $\mathbf{T}$. When $\mathbf{T}$ is applied to $\mathbf{R}$ now, the result is not a diagonal matrix, but the off diagonal terms are quite small. Again take the diagonal entries in decreasing order. Compare to the KL result. Try other values for ρ in the range $0.6 \leq \rho \leq 0.99$.

(d) Now implement the DFT for $\mathbf{T}$. The resulting diagonal elements should all be real. Again order these by decreasing size. Compare to the DCT and KL results.

(e) Implement the DST and compare to the others. Also, try different types of the DCT.

(f) The total energy in the first ℓ coefficients can be obtained by adding up the first ℓ diagonal entries. Do this for $\ell = 1, 2, \ldots N$ and make a plot for each method. Notice that the DFT curve lies below the others until $\ell = N$. Thus the DCT which tracks the KL transform closely is very close to optimal for this particular coding application.

Project 2: Discrete Hartley transform (DHT)

The discrete Hartley transform (DHT) is defined by the equations:

$$H[k] = \frac{1}{N} \sum_{n=0}^{N-1} x[n] \operatorname{cas}(2\pi nk/N) \tag{2.1}$$

$$x[n] = \sum_{n=0}^{N-1} H[k] \operatorname{cas}(2\pi nk/N) \tag{2.2}$$

where $\operatorname{cas}(\cdot) = \cos(\cdot) + \sin(\cdot)$. It has the advantage of being a purely real transform.

Exercise 2.1: Basic Properties of DHT

The first task is to verify the definition of the forward and inverse transforms.

(a) Write two M-files to implement the DHT. Notice that the forward transform has a multiplicative factor, so it would be easier to write the inverse first, and then the forward can call the inverse.

(b) Test these functions by computing the DHT folowed by its inverse to show that an original vector is obtained.

(c) Consider the cas signals. Prove that they are orthogonal:

$$\sum_{n=0}^{N-1} \text{cas}(2\pi nk/N)\text{cas}(2\pi n\ell/N) = \begin{cases} N & \text{for } k = \ell \bmod N \\ 0 & \text{for } k \neq \ell \bmod N \end{cases}$$

Demonstrate this fact in MATLAB by constructing some examples of the cas signal.

(d) Show that a circular reversal of $x[n]$ gives a circular reversal of its DHT.

(e) The shift property is a bit trickier. If this property were analogous to the DFT, the shift would yield a multiplication of the DHT by a cas function. However, the property involves one additional term. In the DHT domain, the result is a combination of both the DHT and its circular reversal. One is multiplied by cosine, the other by sine.

(f) Show that a "Parseval" relation holds, i.e., the energy summed in the frequency domain is N times the energy summed in time domain.

Most of these properties can be proven mathematically, but for these exercises, MATLAB should be used to demonstrate that they are true. In other words, write an M-file to check the property and then test it on many different cases.

Exercise 2.2: Relation of DHT to FFT

The DHT has a simple relationship to the DFT. Since the kernel of the DFT is

$$W_N^{nk} = e^{-j2\pi nk/N} = \cos(2\pi nk/N) - j\sin(2\pi nk/N)$$

it is obvious that the DHT is just the real part minus the imaginary part of the DFT.

(a) Verify this relationship by computing the DHT from the FFT and comparing to the DHT functions already written.

(b) Now try to determine the relationship in the opposite direction. Start with the DHT of a real sequence. What operations have to be performed on $H[k]$ to find the FFT of that same sequence? Hint: the answer will involve circular reversals of $H[k]$.

(c) Once the relation has been determined, test it on some examples.

Exercise 2.3: Circular Convolution Property

The circular convolution property of the DFT states that multiplication of the transforms gives circular convolution in the time domain. It is one of the most useful properties of the DFT.

(a) Demonstrate that the same property does *not* hold for the DHT. Try convolving some shifted impulses.

(b) Show that circular convolution maps to the DHT domain to give the sum and difference of four products: between the DHTs of the two signals and their reversals.

(c) Write a function that does circular convolution in this manner.

Chapter 3

Spectrum Analysis

January 17, 2007

Overview

In this chapter, we present methods for the analysis of signals in the frequency domain. The material here is restricted to the deterministic case.

Computer-Based Exercises for Signal Processing

Spectral Windows

Spectral Windows

In these projects we will study an number of different window types. By plotting their frequency response, we can compare their primary characteristics. In addition, we will introduce several metrics to quantify the performance of the different windows. Finally, we will consider the use of windows in spectral analysis where it is crucial to understand their performance.

The primary goal of this packet is to show that a very large number of windows have been proposed, but there is also a simple way to characterize their performance.

Background Reading

F. J. Harris, "On the use of windows for harmonic analysis with the discrete Fourier transform," *Proc. IEEE*, vol. 66, pp. 51–83, January 1978.

Project 1: Window Types

Many different kinds of windows have been proposed for use in spectral analysis and filter design. In all cases, the window acts in the time domain by truncating the length of a signal

$$x[n] \cdot w[n] \qquad \text{where } w[n] = 0 \quad \text{outside of } 0 \le n \le N-1$$

The important properties of the window are usually described in the frequency domain where the DTFT windowing property states that the windowed signal has a DTFT that is the convolution of the true DTFT with that of the window.

$$y[n] = x[n] \cdot w[n] \qquad \overset{\text{DTFT}}{\longleftrightarrow} \qquad Y(e^{j\omega}) = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(e^{j\theta}) W(e^{j[\omega-\theta]}) d\omega$$

where $W(e^{j\omega})$ is the DTFT of the window.

In this project, we will examine many different classes of windows, and evaluate their frequency response. In later projects, the performance of these windows will be considered.

Hints

The DTFT of the window is computed by sampling the DTFT $W(e^{j\omega})$, i.e., by using a zero-padded FFT.[10] The length of the FFT should be 4–5 times longer than the window length to get adequate sampling in frequency for plotting. Then the plot should be made with `plot` which will connect the frequency samples and draw a continuous looking plot. In addition, the sidelobes of $W(e^{j\omega})$ are best compared on a dB plot, see `log10` and `db`.[11]

Exercise 1.1: Rectangular and Triangular Windows

The simplest window is the rectangular window. Anytime a signal is truncated, there is a window; even if no weighting is used, the window is, in effect, the rectangular window. In MATLAB this window can be generated by `ones` or `boxcar`.

[10] See the CASPER special function `dtft`.
[11] A CASPER special that avoids the log of zero.

(a) Generate a rectangular window of length 31. Compute its DTFT and plot the magnitude on both a linear scale and a dB scale.

(b) Repeat for different window lengths: 21, 41, 50, and 64.

(c) The triangular-shaped window is also called a Bartlett window. MATLAB has two different functions for this window, and they actually generate windows of different length. Use `bartlett` to generate a length-11 window. Plot the window samples with `comb`. Now use `triang(11)` and redo the plot. What is the window length (number of non-zero samples) for this case?

(d) Generate the DTFT for triangular windows of length 31, 41 and 64; and plot them on both linear and dB scales.

(e) The triangular window can be related to the rectangular window in a simple fashion. Show that an odd-length triangular window (length $= L$) is the convolution of a length-$(L+1)/2$ rectangular window with itself. Compare the log magnitude of the L point triangular window with that of the length-$(L+1)/2$ rectangular window. Show that the sidelobes structure is the same, but the height of the rectangular window sidelobes is exactly twice that of the triangular window sidelobes (in dB). Explain.

Exercise 1.2: Window Functions in MATLAB

MATLAB has a number of built-in operators for generating windows. These are contained in the signal processing toolbox, and include `hamming`, `hanning`, `blackman`, `chebwin`, and `kaiser`. These are the Hamming window, the Hann window, the Blackman window, the Dolph-Chebyshev window, and the Kaiser (Bessel) window, respectively. The first four of these find most use in spectrum analysis applications. The Kaiser window is important for FIR filter design via the windowing method.

(a) For the first three, there is only one parameter to specify—the window length. All three are based on the cosine function. You can type out the functions to see the exact formula, e.g., `type blackman`.

(b) For each of the first three windows, generate a window of length $L = 31$, and plot the window coefficients $w[n]$.

(c) Now take the DTFTs of each and make one plot of the magnitude of the Hamming, Hann, and Blackman windows together. Include the rectangular window also (for reference). Make this frequency-domain plot in three ways: first as a dB plot, then blow up the region from $\omega = 0$ to $\omega = 20\pi/L$ and plot on both a linear scale and a dB scale. The choice of $20\pi/L$ is arbitrary—just take a sufficient region to see the first few sidelobes. Use the colon operator to select part of the DTFT, see `help :` in MATLAB.

These different plots will illustrate the different mainlobe widths and sidelobe heights for these windows.

(d) For the Dolph-Chebyshev window, there is an additional parameter that must be specified. This window offers control over the sidelobe height, and this second parameter is the specified sidelobe height. In fact, the sidelobes should all be the same height—called equiripple.[12] Generate three Dolph-Chebyshev windows of length $L = 31$. For the first use a sidelobe height of 30 dB, for the second 40 dB, and 50 dB for the third. Plot the window coefficients versus n, and compare to the Hamming window coefficients.

(e) Compute the DTFTs and plot them all together with the Hamming window DTFT. Make the plots in the same three ways as in part (c) above. The Hamming is included for comparison because its sidelobe structure is nearly equiripple (at approximately -42 dB).

Note the inverse trade-off between sidelobe height and mainlobe width for these cases.

(f) Repeat the experiments above for an even window length, say $L = 64$.

Exercise 1.3: Kaiser Window

The Kaiser window of length L is

$$w[n] = \frac{I_0(\beta\sqrt{1-(n-M)^2/M^2})}{I_0(\beta)} \qquad \text{for } n = 0, 1, \ldots L{-}1$$

The mid-point M is $M = \frac{1}{2}(L–1)$; for an odd-length window M is an integer. The parameter β should be chosen between 0 and 10, for useful windows.

An approximate formula for its frequency response is:

$$W(e^{j\omega}) \approx \frac{2M\sinh(\beta\sqrt{1-(\omega/\omega_\beta)^2})}{\beta I_0(\beta)\sqrt{1-(\omega/\omega_\beta)^2}} \qquad \text{where } \omega_\beta = 2\pi\frac{\beta}{M}$$

The value ω_β is the approximate width of the mainlobe.

For $\omega > \omega_a$, the sinh function becomes a sine, so

$$W(e^{j\omega}) \approx \frac{2M\sin(\beta\sqrt{(\omega/\omega_a)^2-1})}{\beta I_0(\beta)\sqrt{(\omega/\omega_a)^2-1}}$$

This formula predicts that the sidelobes will fall off as $1/\omega$, away from the mainlobe.

(a) For the Kaiser window, the parameter β must be chosen. This window offers a tradeoff between the sidelobe height, and the width of the mainlobe. Generate three Kaiser windows of length $L = 41$. Try different choices of the parameter β; $\beta = 3$, 5 and 8. Plot the window coefficients, and compare to the Hamming window coefficients.

(b) Compute the DTFTs and plot them all together with the Hamming window DTFT. Make the plots in the same three ways as in part (c) of the previous exercise. The Hamming is included for comparison because its sidelobe structure is nearly equiripple (at approximately -42 dB).

Note the inverse trade-off between sidelobe height and mainlobe width for these cases.

[12]These windows could also be designed with the FIR filter design program `remez`.

(c) Plot the approximate formulas and compare to the true DTFT of the Kaiser window. Determine whether the mainlobe and sidelobes follow the approximations. See `sinh` and `besseln` in MATLAB.

(d) Repeat the experiments above for an even window length, say $L = 64$.

The Kaiser window is most useful for filter design, where its frequency domain convolution with the frequency response of an ideal LPF (or BPF) yields bood stopband attenuation. Then the parameter β provides control over the passband and stopband ripples, in a trade-off with the transition width of the filter.

Exercise 1.4: Other Windows

There are many other windows that have been proposed. Consult reference [1] for a rather comprehensive tabulation. In this exercise, a few of these families will be introduced. For each of these, you should write an M-file to generate the window coefficients and plot them for even and odd lengths. Also, the DTFT should be computed and displayed. The Hamming or rectangular window should also be displayed for reference.

(a) All these windows are finite and have a point of symmetry, so their DTFT is also linear phase. It is convenient to give a notation to this point of symmetry: $M = \frac{1}{2}(L-1)$, where L is the window length and M is the point of symmetry (or mid-point).

(b) *Cosine series windows.* The Hamming, Hann and Blackman windows are members of this family, where the window coefficients are given by:

$$w[n] = \sum_{\ell=0}^{M} a_\ell \cos\left[\frac{2\pi\ell}{L-1}(n-M)\right] \qquad \text{for } n = 0, 1, \dots L-1$$

This class of windows has a simple analytic form for the DTFT, because they are based on linear combinations of cosines. Sometimes the formula is written with $2\pi/L$ in the argument of the cosine. The difference in the resulting frequency response is slight, especially for large L.

The Hamming and Hann windows require a_0 and a_1. For the Blackman case, there are three coefficients:

$$a_0 = 0.42 \qquad a_1 = 0.50 \qquad a_2 = 0.08$$

If an optimization of these coefficients is done to minimize the maximum sidelobe level, the resulting windows are the Harris-Nutall windows. For the three-term and four-term cases the coefficients are

Harris-Nutall Windows		
	-67 dB	-94 dB
a_0	0.423 23	0.358 75
a_1	0.497 55	0.488 29
a_2	0.079 22	0.141 28
a_3	—	0.011 68

Implement these windows and verify the sidelobe height of their Fourier transforms. Also show that the mainlobe width is inversely proportional to the window length.

(c) Derive the analytic form of the DTFT for the cosine series windows, as a weighted sum of aliased sinc functions.

(d) *Parabolic (Parzen) windows.* These are based on a simple polynomial formula:

$$w[n] = 1 - \left[\frac{n-M}{M}\right]^2$$

Plot the window coefficients versus n. Determine the mainlobe width and sidelobe height for this window; try several different lengths.

(e) *Cauchy window.*

$$w[n] = \frac{M^2}{M^2 + \alpha^2(n-M)^2}$$

Make plots of both the time-domain and frequency-domain for $\alpha = 4$.

(f) *Gaussian window.*

$$w[n] = \exp\left[-\tfrac{1}{2}\alpha^2\left(\frac{n-M}{M}\right)^2\right]$$

Do the plot for $\alpha = 3$ and $\alpha = 6$.

As you can see, many different functions can be used to create a time window. All have the characteristic taper near the edges, with a peak in the middle. Slight adjustments of the time-domain shape of the window can lead to quite different frequency responses.

Project 2: Window Performance

Plots of the window and its frequency response as done in the previous project indicate that the "performance" of a window can vary dramatically. In this project, we examine some different ways to measure the performance of a window.

Hints

The MATLAB function `find` can be used to locate points on a curve. Suppose that `H` is the frequency response vector, and that it has been normalized so that its maximum value is `1.0000`. Then `indx = find( abs(H) > 0.5 )` will list all the indices where $|H(e^{j\omega})| > \frac{1}{2}$. From this list of indices we can measure the mainlobe width. If we are examining a window transform that has only one peak, the list of indices will be contiguous, so the first will be the minimum and the last the maximum. Thus `indx(length(indx)) - indx(1)` gives the mainlobe width in samples, which can then be converted to a frequency width.

Exercise 2.1: Window Length

The most important parameter under our control for defining a window is its length. It should always be true that increasing the length of the window will increase its performance from the point of view of mainlobe width. A longer window (of the same type) has a narrower mainlobe.

(a) Verify this fact for the rectangular window by measuring the mainlobe width as the 3-dB width (down by a factor of $1/\sqrt{2}$ in amplitude). Take the length to be $L = 10$, 20, 40, 80, and 160; also try some lengths in between. A plot of the mainlobe width versus L should illustrate the inverse relationship.

(b) Do the same measurement for the Hamming window, and the Hann window. Again take several lengths to see the inverse dependence.

Exercise 2.2: Mainlobe Width

As shown in the previous exercise, the mainlobe width is affected by the window length. Among different types of windows, however, there can be a wide variation in the mainlobe width. In order to measure mainlobe width, we must establish a common reference point, so we with take the 3-dB width.

(a) Compare the mainlobe widths of the rectangular and triangular windows.

(b) Repeat to get a comparison of the Hamming and Hann versus the rectangular. Which window has the smallest mainlobe width? Why?

(c) An alternative definition of the mainlobe width uses a normalization to what is called the *bin width.* For the rectangular window, the first zero crossing in ω is always at $\omega = 2\pi/L$, this number is the bin width. Thus we can define a normalized width, by dividing the 3-dB width by the bin width. This removes the dependence on the length L.

(d) Convert the previous 3-dB measurements to a normalized 3-dB width.

(e) Verify that a single number will suffice for describing each window's width. In other words, recompute the normalized width for a wide range of L.

(f) Determine the nomalized width of the Kaiser window as a function of β. Is its dependence on β linear?

Exercise 2.3: Sidelobe Height

Control of the sidelobe height is important for filter design applications, and for minimizing leakage in neighbofing channels of a spectrum analyzer. The rectangular window has the worst sidelobes, and most other windows have been proposed with the objective of providing lower sidelobes. Again we need a measure of sidelobe height. The most common is taking the height of the first sidelobe which is usually the highest. We will make the definition the *maximum sidelobe height.*

(a) Determine the maximum sidelobe height for the rectangular, triangular, Hamming and Hann windows. Do this for $L = 41$.

(b) Determine whether or not the sidelobe height of these windows can be influenced by changing L. Try values of L smaller and greater than 41, i.e., try doubling and halving to see if there is any change.

(c) The Blackman window, and the Harris-Nutall windows were designed to have extremely low sidelobes. Verify their performance.

(d) The advantage of the Kaiser window is that the parameter β can be used to change the sidelobe height. Run enough cases of the Kaiser window to make a plot of sidelobe height vs. β.

(e) Which value of β gives performance that is nearly the same as the Hann window? For this value of β, plot the windows coefficients of the Kaiser and Hann windows to compare.

(f) Usually a reduction in the maximum sidelobe height is accompanied by an increase in the mainlobe width. This effect can be shown with the Dolph-Chebyshev window, because its constant sidelobe level can be specified in the design. Run enough cases to make a plot of mainlobe width versus sidelobe height, and thus show that a decreas in the sidelobe level is accompanied by a broadening of the mainlobe.

Exercise 2.4: Equivalent Noise Bandwidth

Another way to measure the spectral width of the window is to base the measurement on the window's performance as a filter in a spectrum analyzer. If $w[n]e^{j\omega_\circ n}$ is the impulse response of a filter, and the input to the filter consists of a sinusoid at $\omega_\circ$ plus white noise, then the output will have a large peak whose height is

$$\left|\sum_{n=0}^{L-1} w[n]\right|$$

plus a noise background with variance equal to

$$\sum_{n=0}^{L-1} |w[n]|^2$$

The ratio of the noise variance to the peak height squared tells how well the window performs. It also characterizes the noise bandwith in the sense that if the mainlobe were narrow and the sidelobes low the noise contribution would be very low. This quantity is called the *Equivalent Noise Bandwidth.*

$$\text{ENBW} = \frac{\displaystyle\sum_{n=0}^{L-1} |w[n]|^2}{\left|\displaystyle\sum_{n=0}^{L-1} w[n]\right|^2}$$

(a) Determine ENBW for the commonly used windows.

(b) Compare ENBW to the normalized 3-db width for these same windows.

Exercise 2.5: Scallop Loss

Another measure of the peak and mainlobe performance is again motivated by a spectrum analysis application. In a sliding-window DFT, the FFT length is equal to the window length. Thus, samples in frequency lie on a grid at $\omega_k = (2\pi/L)k$. If, on the other hand, the input sinusoid has a frequency that is half way between grid points, then the output from the FFT will be lower than the true amplitude of the sinusoid, due to the shape of the mainlobe. This reduced gain is called *scallop loss,* and it is equal to

$$\text{SCALLOP LOSS} = -20\log_{10}\left[\frac{W(e^{j\pi/L})}{W(e^{j0})}\right]$$

(a) Determine the scallop loss for the commonly used windows.

(b) Now define a composite measure of the scallop loss and the ENBW. The worst case of the ratio of the height of a sinusoid to output noise power is the sum of the scallop loss and the ENBW (in db). This is called the *Worst case processing loss.* Determine this measure for the common windows.

Exercise 2.6: Summary

For all the windows above, make a table that gives all four measures

Project 3: Resolution

The measures of mainlobe width and sidelobe height for different windows are important indicators of how a window will perform in a spectrum analysis situation. One of the often misunderstood aspects of windowing when used with the FFT is that of resolution. The length and type *of the window* control resolution, not the length of the FFT. Sometimes these lengths are the same, but often zero padding is used to interpolate in the frequency domain. In this project, we study an objective measure of resolution.

Exercise 3.1: Definition of Resolution

In spectrum analysis, resolution refers to the abililty of the processing system to distinguish between two separate signals whose frequencies are very nearly the same. Thus take the two signals to be

$$x_1[n] = A_1 e^{j(\omega_1 n + \phi_1)} \qquad \text{and} \qquad x_2[n] = A_2 e^{j(\omega_2 n + \phi_2)}$$

The difference between the frequencies $\Delta\omega = |\omega_1 - \omega_2|$ is the parameter of interest.

In a straightforward FFT, the signal to be analyzed is the sum of $x_1[n]$ and $x_2[n]$. For most of this study we will take the amplitudes to be equal, $A_1 = A_2$.

(a) Generate a finite portion of the signal $y[n] = x_1[n] + x_2[n]$, for $n = 0, 1, \ldots, L-1$. The phases should be random over $0 \le \phi < 2\pi$, and the amplitudes equal. Compute the FFT and plot the magnitude, so take take $L = 64$. Vary the two frequencies and determine how small you can make $\Delta\omega$ and still see two separate peaks in the DTFT.

(b) Repeat the same experiment as in part (a), but use a length 256-point FFT with zero-padding. Verify that the minimimum separation remains the same!

NOTE: in this case, a formula can be written for the DTFT of $y[n]$, because each sinusoid generates a term like $W(e^{j[\omega-\omega_i]})$, which is a frequency shift.

Exercise 3.2: Peak Finding

In order to carry out more extensive testing, it is necessary to have an automatic method for peak finding.

(a) Write a function that will automatically pick the peaks. This will requires a definition of what a peak is. The visual criterion used in the previous exercise will have to be turned into a set of rules for what constitutes a peak.

(b) Try to make the program work for different levels of frequency interpolation—ranging from smooth (lot's of zero padding) to none (no zero padding). if

(c) Test your function on the rectangular window which was studied in the previous exercise.

(d) Now try it on the Hamming window to see if still works.

Exercise 3.3: Measuring Resolution

With the objective peak finding algorithm, we are now ready to study different widnows. The approach will be to generate a large number of test cases with different separations, collect all the data, and then plot a score of successful resolution versus $\Delta\omega$.

(a) Make sure that your peak finding function returns a true/false result.

(b) Write a script that will generate signals, compute their FFTs, and then look for peaks. Vary the separation $\Delta\omega$ over a range of values, but in the neighborhood of $2\pi/L$. Generate 10 instances of $y[n]$ at each separation, for each of these random values of ω_1 should be used.

(c) Collect all the scores (percent correct) and plot versus $\Delta\omega$. Since the expected resolution is inversely proportional to window length, it would be better to make a normalized plot of score versus $\Delta\omega/(2\pi/L)$, i.e., normalized to the bin width.

(d) Apply your algorithm to the Hamming, Hann and Kaiser ($\beta = 2$, 3, 5 and 8) windows.

Compare these values for resolution to the mainlobe width of these windows. Often the 3-dB mainlobe width is quoted as the appropriate value for resolution, do you agree for all windows that you tested?

Computer-Based Exercises for Signal Processing

$\boxed{\text{Sliding Window DFT}}$

Sliding Window DFT

Time-frequency descriptions of changing signals are becoming more common, because the computational and display capability of fast computers make them easy to obtain. One case that has a long history is the speech spectrogram, which shows a gray-scale plot of the energy distribution versus time and frequency for a speech utterance; this is the so-called "voice print". Since speech signals are composed of ever-changing narrowband features, this presentation of the data is extremely useful in understanding the signal. Many other time-varying signals can be understood better if a time-frequency energy distribution is shown. Finally, a considerable amount of DSP research over the last 10 years has been devoted to the study of different types of time-frequency representations. It is not the objective of this set of projects to go into these different distributions, but rather to present the one based on the FFT. This is the so-called "short-time Fourier transform."

Background Reading

A little bit in Oppenheim & Schafer, Chapter 11, section 11.5.

Project 1: Spectrogram

The functionality needed in a general spectrogram program can be provided rather easily in MATLAB. The basic functions exist for windowing and the FFT. The spectrogram only requires a few parameters to describe the time-frequency plot desired.

Exercise 1.1: Program for Spectrogram

Write an M-file to implement a general spectrogram, which is nothing more than a sliding DFT. The program needs to have 4 inputs:

1. An input signal.
2. The window. By giving the window values, its length can be determined by MATLAB.
3. The amount to skip from one section to the next. This determines the time sampling of the method.
4. The frequency resolution. In fact, it would be best if an upper and lower frequency were also given so that a part of the frequency domain could be analyzed.

The window length or the frequency resolution will determine the section length for the FFT analysis, so it would be best to choose one of these as a power of two.

The output from the spectrogram could be just the magnitude of the spectral values, but for later use, it would be better to take the complex-valued outputs from the FFT.

For plotting the spectrogram it would be convenient to have a plot option in the spectrogram program, so that the correct units could be put on the plot. Otherwise, a separate plotting function should be written to generate the correct axis lablelling.

Exercise 1.2: Process LFM

Generate a linear-FM signal and process it thru the sliding DFT function.

$$(\text{LFM CHIRP}) = \cos(2\pi W t^2)$$

Since the linear-FM chirp has a known functional form for the frequency variation versus time, you can relate that form to a ridge in the time-frequency plot.

A more general case is $x(t) = \cos[2\pi f(t)t]$. The sliding DFT can measure the "instantaneous frequency", but the instantaneous frequency of $x(t)$ is not $f(t)$.

Exercise 1.3: Process Speech

Load in a speech file and make its spectrogram. Experiment with the parameters of the sliding DFT versus the parameters of speech to see what sort of display you can get. The plot will have to be done with `contour` or `mesh`.

Exercise 1.4: Wideband vs. Narrowband Spectrogram

Changing the parameters of the spectrogram analysis give two different views of the speech waveform:

(a) A speech signal has a pulsed behavior when a vowel is spoken. This gives rise to a line spectrum in frequency. To see these lines on a spectrogram, it is necessary to do a "narrowband" analysis. Determine the frequency resolution parameters needed and use your spectrogram program to make this display. Determine the spacing between the lines in the spectrum.

(b) Another spectral feature of a vowel is that it is made up of a few formants—resonant peaks in frequency. In order to see these (and not the line spectrum), it is necessary to do a "wideband" analysis. Determine appropriate parameters for this analysis, and plot a spectrogram that shows the formants of a vowel.

Exercise 1.5: Bandpass Filter Bank

The sliding DFT can be viewed as a spectrum analyzer. If the offset between blocks is taken to be one, the processing of a sliding DFT can be interpreted as a filter bank, containing N bandpass filters where N is the FFT length. The frequency response of each filter is determined by the window. The different bandpass filters are just offset from one another due to multiplication by a complex exponential.

(a) Consider a case where the length of the window is $N = 32$, and a Hamming window is used. If the channels are numbered from $\ell =$ to $\ell = 31$, plot the frequency response for channels 0, 3, 6 and 9 on one plot. Then plot channels 12 and 13 together. What do these plots say about the "resolution" of the DFT spectrum analyzer.

(b) The DFT simultaneously computes N outputs, and each one is a singLe point in the output stream of the bandpass filters. Therefore, we can test the response of the sliding DFT to a sinusoidal input. Generate a sine wave input at a fixed frequency. Make the signal long enough that you can process it with a sliding DFT and get about

256 points in each bandpass filter. Take the number of filters to be rather small for this experiment, say $N = 16$. Now compute the energy in the output of each channel and show that the correct channel (relative to the input frequency) is responding with the most energy. Why is there energy in all the channels.
NOTE: make sure that you don't pick the input frequency to be a multiple of $2\pi/N$.

(c) Compute the DTFT of one the channel signals, say $\ell = 7$. Plot its magnitude and explain what you see.

(d) What is the effect of changing the skip factor from one to some higher value?

Project 2: Tone generator

This project deals with the implementation of a sliding window DFT spectrum analysis system. In order to create a scenario that has some element of challenge, imagine a system whose job is to distinguish various tones. An example application might be a touch-tone telephone[13] in which each tone represents a digit.

Hints

Use axis labels that reflect the natural units in the horizontal and vertical directions; otherwise, it is quite difficult to interpret the 2-D plots. Beware that MATLAB plots 2-D signals in what might be called a "matrix format", with the origin of a contour plot in the upper left-hand corner. See `rot90` for a function that will rotate the plot and put the origin in the correct place, see also `fliplr()` or `flipud()`.

It is not possible to make a gray-scale "spectrogram" with MATLAB, but you can use `contour()` to approximate one. The function `contour()` is a bit tricky to control. If you choose the contour levels correctly, the plot will contour the regions of interest nicely. See `help contour` for information on the optional arguments to `contour`. However, if you use the defaults, there may be so many contour lines that you cannot distinguish any features. You may also need to use a log plot rather than a linear one to see certain features, see `db()` or `log10()`. The only alternative to the contour plot is the mesh plot, `help mesh()`.

On a PC with limited memory, this project will tax the available memory if you are not careful with your MATLAB program. You cannot create an array containing more than about 8100 elements. If you use big arrays for temporary operations, `clear` them when they are no longer needed. You can also try `pack` to avoid memory fragmentation.

Exercise 2.1: Frequency Coded Input Signal

The input signal must be analyzed to determine its frequency content versus time. It is known to be composed of three different signals:

1. The desired signal is a succession of short-duration sinusoids of constant frequency. The duration of each sinusoid is not necessarily the same, but the length of each is usually between 15 and 20 milliseconds. The desired signal encodes a sequence of digits by using different frequencies to represent individual digits, according to the table below:

[13]Use of this example is not meant to imply that a present-day touch-tone phone uses a sliding DFT.

DIGIT	0	1	2	3	4	5	6	7	8	9
FREQ(Hz)	250	750	1250	1750	2250	2750	3250	3750	4250	4750

Thus, the signal changes frequency to one of the values in the table roughly every 15-20 millisec. Such a scheme is similar to *Frequency Shift Keying* (FSK) in a communication system.[14] This coded signal has a constant amplitude. For example, the digit sequence 7-3-4-1-9 would consist of a sinusoid of frequency 3750 Hz, followed by one of 1750 Hz, followed by 2250 Hz, then 750 Hz, and finally 4750 Hz. In order to avoid instantaneous changes in frequency, the step changes in frequency are filtered through a low-pass filter to provide some smoothing of the frequency versus time.
Notes:
(1) The coded portion of the signal is not assumed to start at $n = 0$ in the data set, so part of the problem will be to locate the transition points where the frequency changes.
(2) The sampling rate in the system is 10 kHz, and for this problem you can assume that the sampling was carried out perfectly—no aliasing.

2. The second signal component is an interference that is quite a bit stronger than the desired frequency-coded signal. This interference is a narrowband signal (i.e., sinusoid) whose frequency is changing continuously with time. Its peak amplitude is about 10 times stronger than the coded signal. You will have to determine the value of this amplitude, and the frequency variation of this signal versus time. However, you are not required to remove this component prior to the sliding window DFT processor; just estimate its parameters.

3. The third signal component is an additive noise term. This component is zero-mean Gaussian white noise with a variance that is equal to 30% of the amplitude of the desired signal. Thus, there is some noise present, but the signal-to-noise ratio is favorable for processing the coded signal. This noise component has been added to make the problem "realistic", but it should not be a major factor in designing your processing scheme.

The function `projsig(digits)` takes as input a 5-element vector and produces an output signal that contains the frequency coded version of this input vector plus the interference and noise. (Actually, there is a second output, which is the actual 5-element vector used, in case the input was not integers in the range [0,9].) If you supply a second input argument to `projsig(digits,scale)`, the value of `scale` is interpreted as a multiplier that will scale the additive interference plus noise. Therefore, you can experiment with situations in which only the coded signal is present, by using the call `projsig(digits,0)`. The coded signal produced by the function `projsig()` will contain variable length tones. With just a scalar input arguments, `projsig(0)` will generate a set of digits based on the present time and return these in its second output argument.

Exercise 2.2: Sliding DFT

The method of analysis to be implemented is the sliding-window DFT, but you must pick the parameters of the system: window length, window type, FFT length, segment skip, etc.

[14] Technically, the duration should be constant to get FSK.

For each design parameter chosen, consider several possibilities.

The primary task in this project is to design and implement a sliding window DFT system that will extract the coded information. For the purposes of implementation, the total signal length will be about 100 millisec, so that the frequency-coded signal contains 5 digits. The exact starting index of the coded signal is variable, but the duration of the code is less than 100 millisec. Your processing system must be able to identify these 5 digits. This can be done in two ways:

(a) Create a spectrogram, and "read" the spectrogram visually to extract the information.

(b) (AUTOMATIC) Design a peak picking algorithm that will ignore the interference and will automatically extract the information. This will be difficult, but try to make it work for any combination of input digits. After some experimentation, you will find that some cases are easier than others, due to the presence of the strong interference.

(c) Then examine the complexity of your implementation. Is there an easier way to do the same thing? How many blocks are you processing? Is the FFT length a power of two and is it as small as possible?

Exercise 2.3: Demonstration of a Working System

To demonstrate that your system works correctly, you must process the following sets of unscaled data, i.e., create the signal using the call `projsig(digits)`:

(a) `n = [ 9 8 1 2 3 ]`

(b) pp`n = [ 4 3 4 7 6 ]`

(c) `n` = last five digits of your phone number.

(d) OTHERS: if you have implemented a general decoder, show that it will work for a random input.

You should try to determine the amplitude and duration of the tones from your sliding window DFT processor. Comment on how well or poorly you can make such measurements.

Exercise 2.4: Interference Signal

The interference signal must be estimated. Determine the amplitude of the interference relative to the amplitude to the frequency-coded tones. Also determine a functional form for the frequency variation of the interference signal. The sliding DFT is measuring a sort of "instantaneous frequency", so one possibility is $\cos[2\pi f(t)t]$, but be careful how you relate the measurement to $f(t)$.

Review the example of the "chirp" signal, or test that case specifically to establish what the spectral shape would be for a chirp. In any event, you should provide a plot of $f(t)$ for the interference; or a plot of the instantaneous frequency and then relate it to the "cosine" functional form given above.

```
function [y,code] = projsig(digits,scale)
%PROJSIG
%    generate "mystery" signal for ee 4078 project
%               (summer 1989)
%    usage: [Y,C] = projsig(D,S)
%      D are digits of a 5-element code
%      S is a scale factor that multiplies the
%          interference signal
%      Y is output signal
%      C is output code
%
load qwert
if ~exist('yint')
   error(' problem loading interference')
end
tones=[ 0.025+0.05*[0:9]' ];
if nargin == 1
   scale = 1.0;      % add 100% of interference
end
if length(digits) < 5
   digits=mod(fix(clock),10); digits=digits(2:6);
end
code  = mod(fix(digits(1:5)),10);          % just 5 digits, must be integers
tones(code+1);                      % create the tones
rand('uniform')
LL = 50*rand(7,1)-25;     % variation in lengths
LL = fix(LL) + [55;175*ones(5,1);95];
Ltot = sum(LL);
if Ltot > length(yint)
   LL = fix(LL*Ltot/length(yint));
end
ttt = [0.5*rand(1);tones(code+1);0.5*rand(1)];
for j = 1:7;
   f1 = [ f1; ttt(j)*ones(LL(j),1) ];
end
N = length(f1);   Nm1 = N-1;   nn = [0:Nm1]';
%----------
tau = 0.8;
[ttt,f1i] = filter(1-tau,[1 -tau],f1(1)*ones(99,1));   % set init conds.
f1 = filter(1-tau,[1 -tau],f1,0.9*f1i);
rand('normal')
y = cos(2*pi*f1.*nn);
y = y + scale*yint(1:N);
```

Figure 1: Listing of projsig.m

Computer-Based Exercises for Signal Processing

$\boxed{\text{Narrowband Signals}}$

Narrowband Signals

The frequency domain is quite useful in describing and processing signals that occupy only a small interval in frequency. These are narrowband signals, and they include sinusoids, hi-Q filter responses, and pulse trains. The objective in this packet is to introduce some features that are most important for narrowband signals in order to gain more understanding of the Fourier description of signals in general.

Background Reading

1. Oppenheim & Schafer: DFT chapters 8 and 11.

Project 1: Synthesis of a Bandpass Pulse

Whenever computations must be done in both the time and frequency domains, the sampled nature of both domains is an important consideration. This project explores some aspects of the frequency sampling, which is inherent in the use of the DFT. Since the DFT is the primary computational device used for numerical Fourier analysis, the primary issue of frequency sampling is the relationship between the DFT samples and the DTFT, or the continuous-time Fourier transform if the signal was created via time sampling.

The particular scenario of this project is the synthesis of a class of “bandpass” pulses. In the next project we will study the processing of these pulses through narrowband filters. The pulses can be synthesized to have a well-defined frequency content, if specified in the DFT frequency domain. One benefit of this type of synthesis is that the pulses can be endowed with perfect linear phase.

Hints

In order to examine the frequency content of a signal $x[n]$ it is necessary to compute its DTFT $X(e^{j\omega})$. See `freqz` or `fft`.

Throughout this project the pulse specifications are given in the continuous-time frequency domain. Therefore, it is necessary to convert such information into the normalized frequency scale of the DTFT, and then into the correct frequency indices of the DFT. These conversions are simply linear re-scalings of the frequency axes.

Exercise 1.1: Time-Domain Synthesis

In this exercise, a bandpass pulse will be created from the product of a baseband pulse (i.e., centered around DC) and a cosine signal. The baseband pulse must be finite in length, so various window functions will be investigated for the purpose of truncating the signal. The cosine signal will translate the Fourier spectrum to the correct center frequency location.

The pulse should have the following characteristics:

BANDPASS PULSE PARAMETERS		
Parameter	Value	Units
Center Frequency	$\frac{1}{7}\pi$	radians
Pulse Length	25	samples
Essential Bandwidth	$\frac{1}{5}\pi$	radians

Table 1. Desired Parameters for Bandpass Signal

Exercise 1.2: Modulating a window

The simplest possible time signal would be the rectangular window, but other could be used.

(a) Use the `fft` function to compute samples of the DTFT of the rectangular $r[n]$. Use a transform length that is at least 4–5 times the signal length. Make a plot with the correct frequency axis, i.e., labeled with radian frequency. Since the DTFT $R(e^{j\omega})$ is, in general, a complex-valued function of ω, it will be necessary to plot the magnitude and phase separately. Of these, the magnitude is more informative.

(b) Observe that the DTFT magnitude has a bulk of its energy concentrated in a mainlobe. What is the width of this mainlobe? How is it related to the length of the time signal, $r[n]$?
A formula for the true DTFT can be determined quite easily—it is an "aliased sinc" function, asinc( ). Verify that the same numbers (as the DFT) can be obtained by evaluating the asinc function at $\omega = \omega_k$.

(c) Try another window, such as the Hamming window.

(d) *Modulation.* The window signal is a lowpass signal, so it must be frequency-translated to the desired center frequency, by multiplying by a sinusoid.

$$x_1[n] = r[n]\,\cos(\omega_\circ n)$$

Create $x_1[n]$ and plot its DTFT. Show that it more or less occupies the desired band of frequencies.

Exercise 1.3: A Better Pulse

The primary problem with the rectangular pulse for this project is that the bandwidth of the rectangular pulse is always inversely proportional to the pulse length. Thus the desired specs on bandwidth cannot be fulfilled by $x_1[n]$ because the mainlobe is too narrow. A different time signal must be used if the frequency domain bandwidth criterion is to be met. Since the desired DTFT should extend over a band of frequencies, one obvious choice for the time signal would be the inverse DTFT of a pulse in frequency. This is a particularly easy inverse transform to compute analytically, and the result is a "sinc" function:

$$s[n] \;=\; \frac{\Delta\omega}{\pi}\,\mathrm{sinc}(\Delta\omega\, n) \;=\; \frac{\sin(\Delta\omega\, n)}{\pi n}$$

The formula is given for the lowpass case, so cosine modulation will have to be applied to move the transform to the correct center frequency.

One problem with this formula is that it extends over all n, from $-\infty$ to $+\infty$. So, in order to use it for a bandpass pulse, it must be windowed (i.e., truncated). Since the main portion of the sinc function lies symmetrically about zero, the window must be placed symmetrically about the $n = 0$ point. For this exercise, keep the length of time signal the same as before. Make a plot of the DTFT on a dB (log) scale. Verify that the bandwidth is now correct—at least if bandwidth is taken to be the width of the mainlobe.

Exercise 1.4: Windowing

The DTFT of the "sinc" signal has rather high frequency domain sidelobes outside of the desired frequency band. On a log plot these sidelobes are easy to see since their height is about -13 dB relative to the mainlobe peak. One way to improve the sidelobe structure is to multiply the time signal by a window other than the boxcar.

(a) Experiment with the Hamming (and Hann) windows to demonstrate that the sidelobes in the DTFT can be lowered significantly. Make plots of the DTFT, and verify that the bandwidth of the windowed signal is still correct.

(b) Measure effective time length of the signal. State a formula that would give the time length.

Exercise 1.5: Synthesis in the frequency domain via the DFT

The bandpass pulse can also be created from its frequency-domain description. The result is a pulse similar to that obtained from time-domain synthesis. However, the frequency content can be controlled more closely.

For the computer, the characteristics of the bandpass pulse must be specified in terms of its DFT samples. Its DFT samples are set to ones for the in-band frequencies, zeros for out-of-band. The time-domain pulse is synthesized by taking the inverse DFT and then windowing the result. Choice of the DFT length is not crucial, but it must be greater than some minimum and should not be so large as to require unecessary computation.

Use the FFT to create a "bandpass pulse" signal. The desired signal should have the following characteristics:

BANDPASS PULSE PARAMETERS		
Parameter	Value	Units
Center Frequency	6.5	kHz
Pulse Length	???	samples
Two-way Bandwidth	3	kHz
Sampling Period	12.5	μsec

Table 2. Desired Parameters for Second Bandpass Signal

(a) The length of the DFT to be used is at your discretion. For the moment, take it to be $N = 256$. NOTE: the pulse length does not have to equal N, because windowing will be applied in the time domain.

(b) The *Uncertainty Principle* of Fourier analysis states that the product of time-duration and bandwidth is always greater than a constant. (The exact value of the constant depends on the units used for frequency and on the definition of duration, but it serves as a fixed lower bound for all signals.) An example of the uncertainty principle can be seen with the rectangular pulse/aliased sinc function transform pair — when the length of the pulse is L, the width of the mainlobe in frequency (DTFT) is approximately $2\pi/L$. Thus the time-bandwidth product is greater than 2π. A similar result holds true for continuous-time signals (and transforms).

Use the uncertainty principle of Fourier analysis to estimate the "essential" length of the pulse, i.e., which of the 256 points are really needed. Express this answer as a total time in milliseconds and in number of samples. NOTE: your answer will actually be a lower bound, so it will have to be increased somewhat if the signal length (in time) is to be sufficient to represent the bandpass pulse.

(c) Use the `ifft( )` function to make the signal. In other words, create the DTFT of the signal by setting frequency samples to one or zero, and then use `ifft( )` to get the actual time signal.

(d) Make sure that your final signal is *real-valued*. This involves two issues: (1) make sure the DTFT is specified so that its inverse transform will be purely real, and (2) check the output for tiny values in the imaginary part due to rounding error in the inverse FFT. These would be $\approx 10^{-14}$.

(e) If necessary, rotate the signal within the vector of 256 samples so that the pulse appears in the middle of a plot. See `help fftshift`.

(f) *Alternate way to get real-valued signal.* Show that exactly the same real-valued signal will be generated by using the following steps: (1) make a DTFT that is not symmetric, i.e., set the "negative" frequencies equal to zero. (2) take the inverse DTFT to get a complex-valued signal, then (3) take the real part of that signal.

(g) Use `freqz` to verify that your synthetic signal has the correct frequency components. *Make sure that you window the signal to its "essential" length prior to doing the DTFT.* Make the plot with a frequency axis labeled in kHz.
NOTE: the pulse will not be absolutely confined to the specified frequency region, but there should be very little energy outside the desired region. Also the in-band spectral content should be nearly flat.

(h) Can you generate the same signal with a 512-point FFT? A 128-point FFT? Explain how the frequency samples would have to be chosen. Is there any significant difference in the "essential" part of the pulse? What is the minimum length FFT that could be used?

Exercise 1.6: Window the Pulse

If the output of the IFFT is truncated, then, in effect, a rectangular window has already been used to create the finite length bandpass pulse. Better results can be achieved with a Hamming window. In this part, the Hamming window will be studied, see `help` on `hamming`.

(a) Generate an instance of the Hamming window for the appropriate signal length, L. Compute the DTFT of both the Hamming window and the rectangular window (for $L = 31$), and plot the magnitude versus normalized frequency. Use `freqz` so that you get plenty of zero-padding to make a smooth (interpolated) DTFT plot. See `help boxcar` for generating the rectangular window.

(b) Plot the magnitude on a dB scale. Superimpose the two DTFT's to illustrate the difference in sidelobe levels. Use the `hold` command.

(c) Plot just the region where $|\omega| \leq 10\pi/L$ and superimpose to show the different mainlobe widths. Use the colon operator to select part of the DTFT, see `help :` in MATLAB.

(d) Now apply the Hamming window to the main part of the pulse obtain from the inverse DFT. Take the DTFT of this windowed waveform and plot the result. (Use zero padding to get a dense sampling of the frequency axis.) Use both a linear plot and a log plot to show how well the Hamming window works.

(e) Since the Hamming window tapers to near zero at its ends, the "effective" length of the window is somewhat less than its true length. Therefore, the length of the Hamming window should be taken longer than the rectangular window already used for the bandpass pulse. Experiment with different window lengths, and try to find a relatively short window length that will give excellent frequency domain behavior.

Exercise 1.7: Windowing the Pulse

The bandpass pulse has a DTFT with rather high sidelobes — these can be seen on a log plot. It is possible to reduce these sidelobes if a time-domain window is applied to the signal. A "better" window is the Hamming window which has lower sidelobes in the frequency domain.

(a) Generate a Hamming window whose length is approximately equal to the main region of the bandpass pulse signal. This length was estimated in part 1, using Fourier analysis (i.e., based on the bandwidth of the pulse).

(b) Multiply the pulse from part 1 by the Hamming window to create a shortened finite-length pulse. Use the pointwise multiply operator `.*` in MATLAB.

(c) Compute the DTFT of the windowed pulse and plot the log magnitude via `db( )`. Observe that the sidelobes are much lower!

Exercise 1.8: M-file

This method of bandpass pulse synthesis will be used to generate test signals for studying both FIR and IIR filters. Take the commands needed to synthesize the Hamming weighted bandpass pulse and write an m-file to synthesize such pulses for different parameters: sampling frequency, center frequency, and bandwidth.

Project 2: Filtering the Bandpass Pulse

Frequency-selective filters can be used to separate pulses such as those created in the previous two projects. In this project, we will study the action of IIR and FIR filters when processing these pulses. Both types of filters are effective in isolating individual pulses because the magnitude characteristic can be tailored to the frequency band of interest. However, the linear-phase FIR filters introduce no phase distortion to these symmetrical pulses; whereas, the IIR filters are shown to cause noticeable changes in the pulse shape, even though all the energy in the pulse is preserved.

The linear-phase bandpass pulses make ideal test signals for frequency-selective digital filters. The processing can be done in three different ways: IIR filtering with a set of second-order bandpass filters, or FIR filtering with linear-phase filters, or a sliding FFT. The difference between the phase response of FIR and IIR filters can be demonstrated by examining the output of these different processing systems.

Hints

In this set of exercises, an unknown signal file is needed. It is contained in the file `summ91.dat`. The data file can be loaded into MATLAB via the `load summ91.dat` command. This file contains three bandpass pulses: one in the frequency from 5–8 kHz, one between 10.5 and 15.5 kHz, and the third in the 18–20 kHz band. These signals are each repeated at a regular interval. The data record is *noisy.* High-pass noise was added to disguise the signals in a plot of the raw data. The signal-to-noise ratio is poor, but the noise is out of band, so frequency selective filtering will work quite well.

In order to plot very long signals, the function `striplot` can be used.[15] This makes a plot the occupies several rows of a normal plot.

Exercise 2.1: IIR filtering

To find the signals it is necessary to do bandpass filtering. In this part, the bandpass filtering will have to be implemented with three separate IIR filters. Each one can be optimized to the known bands occupied by the pulses.

The simplest possible bandpass filter is a second-order section, where the pole locations are chosen to obtain the desired center frequency and 3-dB bandwidth. If we specify a second-order section with poles at $z = re^{\pm j\theta}$, and zeros at $z = \pm 1$, the transfer function is

$$H(z) = \frac{1 - z^{-2}}{1 - 2r\cos\theta\, z^{-1} + r^2 z^{-2}}$$

The frequency response of this system has zeros at $\omega = 0$ and $\omega = \pi$. The poles create a peak at $\omega = \theta$ with a 3-db bandwidth equal to $2(1-r)/\sqrt{r}$.

(a) Verify the frequency response of the second-order section by choosing θ and r based on the 5–8 kHz bandwidth for pulse #1 above. Make a plot of the frequency response versus frequency (in Hertz) to show that the passband extends from 5 kHz to 8 kHz.

(b) Use the `filter` function in MATLAB to process the noisy signal. Plot the output (in sections) to see if you can detect where the 5–8 kHz bandpass pulse lies. It might be useful to use `striplot` to plot the long signal over several lines in the graph window.

[15] Need this as a CASPER special.

(c) Repeat for the other two bandwidths: 10.5–15.5 kHz and 18–20 kHz.

(d) The bandpass filters can also be designed by standard methods to produce Elliptic filters, or some other common filter type. If you have already studied this sort of filter design, create three of these filters and process the data one more time.

After processing, create a table that lists the repetition periods for each signal. Give your answer in millisec. Show one or two representative plots to explain how you estimated the time between pulses. You should also count the total number of pulses at each frequency.

NOTE: these pulses are very easy to find when you have the correct filters. Sharp cutoff filters, such as Elliptic, are not really necessary.

Exercise 2.2: FIR filtering

A reasonable FIR filter can be created by using a finite-length bandpass pulse as the impulse response of the filter. The passband of the FIR filter will simply be the frequency range specified for the bandpass pulse.

(a) Use the bandpass pulse idea to create the impulse responses for the three bandpass filters needed for processing. Use the method developed in the previous project.

(b) Compute and plot the frequency response of all three bandpass filters. Label the horizontal axis with frequency in kHz. Verify that the passbands are in the correct location. Define the stopbands of the filters and estimate the attenuation in the stop band for each filter. You should be able to get about 40 dB of rejection, if a Hamming window is used. If necessary, make the impulse of the filter longer to get a better frequency response.

Estimate the size of the passband ripples in the frequency response of the FIR filters. Define the passband according the three band given above.

(c) Process the signal through each of the FIR filters and estimate the pulse locations and the interpulse period for each of the three signal components.

(d) Use the function `remez` to design the FIR filters. If you have already studied filter design, this will be straightforward. The `remez` function is an implementation of the Parks-McClellan algorithm for FIR filter design.

(e) Compare the output pulse shape for the FIR processing versus the IIR processing. Show that the IIR filter distorts the pulse shape, while the FIR filter causes no distortion at all. This is due to the non-linear phase characteristic of the IIR filter.

Exercise 2.3: Sliding Window FFT Processing

Since we are searching for events in both frequency and time, the FFT can be used to doing a moving Short-time Fourier analysis. This would be implemented in the following way:

(a) Take a short section of the signal, say 100 points. This length is determined by the resolution needed in the frequency domain—each output of the FFT represents the energy in a frequency band whose width is inversely proportional to the number of signal points taken into the FFT. For the three bands being explored, the bandwidth of the FFT must be less than the smallest of the three. An increase beyond this length is useful for creating smoother plots, but is not needed for detection of the pulses.

(b) Multiply by a Hamming window, whose length is the same as that of the signal.

(c) Compute the FFT (after zero padding).

(d) Display the magnitude of the FFT.

(e) Move over by a fraction of the window length (say 25% or 50% for the Hamming), and repeat the FFT analysis.

(f) Collect all the FFTs together into one large 2-D array — one FFT result per column.

(g) Make a `contour` plot to show where the energy peaks lie in the time-frequency plane. Label the contour plot with the correct units of tiem and frequency. For the time labels, assume that the FFT analysis is done at a time corresponding to the middle of the Hamming window.

The only decision to make in this approach is the choice of window length. This should be determined by the bandpass pulse lengths which can be estimated from their bandwidths.

If you wish to experiment a bit you can also change the amount by which the window is moved over for each FFT. The figure of 25% given above is arbitrary; 50% is another common choice.

Project 3: Resonant Peaks

Another class of narrowband signals consists of the output signals from second-order resonators. These simple signals can be used to model the formant in a speech signal, for example.

Exercise 3.1: Bandwidth of a resonant peak

MATLAB can be used to demonstrate the variation in bandwidth for a second-order digital filter. An interesting 3-D plot can be made to show the relationship between the frequency response and the location of the poles with respect to the unit circle in the z-plane. The idea is to show changes in the frequency response, from which the influence of the pole positions can be understood. In this case, we will study the relationship between the pole radius and bandwidth of the frequency response. An approximate relation is given by the formula:

$$\text{3-dB bandwidth} = 2\,\frac{1-r}{\sqrt{r}} \qquad \text{(radians)} \tag{3.1}$$

For $r \approx 1$, i.e., very close to the unit circle, the dependence is nearly linear.

(a) To see this behavior, a 3-D plot of a set of frequency responses can be constructed using the `mesh` command. Write an M-file that will compute the frequency response of a second-order all-pole filter, for varying r:

$$H(z) = \frac{1}{1 - 2r\cos\theta\, z^{-1} + r^2\, z^{-2}} \tag{3.2}$$

As the pole location moves closer to the unit circle, the frequency response exhibits a sharp peak, and its bandwidth gets very narrow. Collect all the frequency responses together in a matrix, but only include the frequency reange near the peak. In order to make the mesh plot look like a good comparison over the wide range of pole radii being studied, the plot should be constructed on a dB scale.

(b) Relate the location of the resonant peak to r and θ in $H(z)$. Give a formula that describe this dependence.

(c) Another view of the set of frequency responses is provided via the contour plot. This has the advantage of showing the precise location of the 3-dB points in the frequency response. In MATLAB, the contour heights drawn can be specified directly, so 5 contour lines are drawn very close to -3 dB level to emphasize that location. In addition the analytic formula (1) for the bandwidith can be superimposed (solid line) on the contour plot to check the agreement

(d) Derive the formula (3.1) for the relationship between r and the bandwidth.

Thus the mesh plot gives a visual representation of the resonant peak, and the contour plot confirms the approximate formula and shows its range of applicability.

Exercise 3.2: Another pole movement example

Write a function that will synthesize a section of a decaying sinusoid $y[n]$:

$$y[n] = A\,r^n \cos(\omega_o n + \phi) \qquad \text{for } n = 0, 1, \ldots N-1$$

This signal is the output of a second-order resonator.

(a) Determine the z-transform of $y[n]$, and use this to find a system that will generate $y[n]$ as its impulse response.

(b) Use the MATLAB function `filter( )` to generate the signal as the impulse response of a rational system. The point of this exercise is to relate the parameters of the desired signal to the parameters needed by `filter( )`, i.e, $Y(z) = H(z) = \dfrac{B(z)}{A(z)}$.

(c) Demonstrate that your `filter()` function works by generating and plotting the first 50 points of the signal:

$$y[n] = 20\,(0.975)^n \cos\left(\frac{2\pi}{17}n + \frac{\pi}{4}\right) \qquad \text{for } n = 0, 1, \ldots 49$$

(d) In MATLAB, it is possible to plot the true frequency response by using the `freqz( )` function. The arguments to `freqz(b, a, N)` are the vectors containing the polynomial coefficients for $A(z)$ and $B(z)$.

(e) Compute the 256-point DFT of $y[n]$, padded with zeros, and plot the magnitude together with the DTFT from `freqz`. How closely does this *approximate* the true frequency response of the rational system? Pay special attention to the peak width and sidelobes.

(f) For either frequency response, relate the location of the resonant peak to the ω_o parameter of the signal. Give a formula that describe this dependence.

(g) Use the same 50 point signal, and measure the peak width for different values of r in the range 0.8 to 0.99. (This dependence can be studied by generating signals having $r = 0.9$, $r = 0.95$, and $r = 0.975$.) Over what range or r is the formula given in the previous exercise still valid.

(h) Explain why the peak width becomes a constant for r greater than some value $r_\circ$.

(i) Repeat the previous two steps for a longer signal, say 100 points, 200 points, and then 500 points. Also do it for a shorter signal of 25 points. Explain the why the longer signals have peak widths that match the formula (3.1) over a wider range of r.

Project 4: Line Spectra

Periodic signals have line spectra. When the signal length is finite, the lines are not impulses, but rather have a finite spectral width. If such a finite pulse train is then processed by a filter, the output signal will, in some sense, sample the frequency response of the filter. Thus the output signal would exhibit three characteristics: an envelope, spectral lines at a regular spacing, and non-zero width to the spectral lines.

This project is intended to show these effects. Since voiced speech is a signal of this type, it is used as an example in the first exercise, but the second one deals deal with synthetic signals.

Hints

The DTFT is computed with a zero-padded FFT. To load dat from a MATLAB file, see help `load`. If you computer has limited memory, use `clear` to remove arrays that are no longer needed, such as long speech signals.

Exercise 4.1: Fourier Domain

Perform the following experiment with MATLAB:

(a) Load the file `bat.mat` which contains speech data for the word "bat". After the `load` command the signal in the workspace will be called `sa`; it is about 4000 points in length, so you have to chop out a segment for analysis. Define $x[n]$ to be a section of the signal in the range $n = 1400$ to $n = 1800$. Take $x[n]$ to be 256 points somewhere in this region.

(b) Take the 256-point DFT of $x[n]$, and call it $X_1[k]$.

(c) Take the 1024-point DFT of $x[n]$, and call it $X_2[k]$.

(d) Plot the log magnitudes of both on the same plot. One plot should be smoother than the other, but the frequency content should be about the same.

(e) The speech data was sampled at $f_s = 8$ kHz. Make the frequency plots so that the horizontal axes are labelled with frequencies from 0 to 4 kHz (the Nyquist frequency). Recall that the Nyquist frequency is defined as half the sampling frequency frequency.

(f) Blow up a region of the frequency plot. (Use the colon notation in MATLAB to select a section of the FFT.) Note that the transform has many peaks, but they seem to be at a regular spacing. Measure this spacing in Hertz.

(g) The input signal $x[n]$ is nearly periodic—what is its period in milliseconds.

(h) State the relationship between the period of the speech signal and the regular spacing of peaks in the Fourier transform. EXPLAIN this relationship.

(i) Measure the spectral width of the lines in the frequency plot. Relate this to the total length of the signal analyzed.

Exercise 4.2: Filtering a Pulse Train

In this project we investigate the response of a first-order IIR filter to pulse train input. Perform the following experiment:

(a) Generate an "impulse train" containing 207 samples. The spacing between impulses should be 23 samples. The height of each impulse can be one. Call this signal $x[n]$.

$$x[n] = \sum_{\ell=0}^{8} \delta[n - \ell M_\circ] \qquad \text{with } M_\circ = 23$$

(b) Plot the DTFT of $x[n]$; use a length of 512. Since the magnitude of $|X(e^{j\omega})|$ is even w.r.t. ω, only half of the response is needed.

(c) Filter this input signal with a first-order filter:

$$H(z) = \frac{B(z)}{A(z)} = \frac{1-a}{1-az^{-1}} \qquad \text{with } \ a = 0.95$$

Use the MATLAB function `filter()`; but be careful to specify the `a` input correctly. The numerator value $(1-a)$ is needed to normalize the low pass filter gain to be one at DC.

(d) The result of the filtering is an output that is 207 samples long. Call this signal $y[n]$. Plot its DTFT.

(e) Compute the frequency response of the first-order filter $H(z)$, by using the MATLAB function `freqz()`. Specify 256 frequency samples; recall that the default for `freqz()` is frequency samples from 0 to π. Plot this result; it should be a low-pass filter.

(f) Plot the magnitudes and log magnitudes of all DTFTs. One plot should be smooth, the others should contain spectral lines. If you use `db()`, the plots will all be normalized. Then the smooth one should be the envelope of the filtered line spectrum, $Y(e^{j\omega})$. Explain.

(g) Blow up a region of the log magnitude frequency plot of $Y(e^{j\omega})$. (Use the colon notation in MATLAB to select a section of the DTFT.) Note that the transform has many peaks, but they seem to be at a regular spacing, $\Delta\omega$. Measure this spacing in radians per second.

(h) State the relationship between the period of the input signal, $x[n]$, and the regular spacing of peaks in the Fourier transform of $X(e^{j\omega})$ or $Y(e^{j\omega})$.

(i) Measure the width of the spectral lines and relate this value to the total length of the impulse train signal.

(j) Repeat this experiment with a filter that is second-order with a resonant peak as in (3.2) Notice that line spectrum of the input does not necessarily hit the peak of the resonance.

Project 5: Undersampling the DTFT

Exercise 5.1: Less DFT samples than data

For the N-point DFT, the frequency samples are located at

$$\omega_k = \frac{2\pi k}{N} \qquad\qquad k = 0, 1, \ldots N-1$$

If the length of the signal $x[n]$ is finite, then the DFT is applicable. Assume the signal length is L points; when L satisfies $L \leq N$, the DTFT samples are computed by zero-padding the signal $x[n]$ out to a length of N points and then computing the N-point DFT. This can be accomplished in MATLAB by invoking the `fft` function with two arguments. When the second argument is greater than the length of the vector in the first argument, zero-padding is performed automatically. For example, if the length of the vector `x` below were 100 points, then 156 zeros would be appended before the FFT is computed.

```
X = fft( x, 256 );      % ZERO-PAD before FFT
```

CAUTION: it is not an error to make the second argument less than the length of `x`, but, in this case, MATLAB *truncates* the vector `x`. This is *incorrect* from the DSP point of view, because the frequency sampling property of the DFT will be violated. Instead, this case should be handled with the following equation, when $L > N$:

$$X(e^{j\omega_k}) = \sum_{n=0}^{N-1} \left(\sum_{\ell=0}^{r} x[n + \ell N] \right) e^{-j2\pi nk/N}$$

where $r = \left\lceil \dfrac{L}{N} \right\rceil$. To do so, would require a new function written by the user.

Chapter 4

Multi-Rate Processing

January 17, 2007

Overview

In this chapter, techniques related to decimation, interpolation and other topics in multirate processing are covered.

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{Bandlimited Interpolation}}$

Bandlimited Interpolation

Overview

The process of interpolation essentially corresponds to estimating or reconstructing the values of a signal at locations (times) between the sample values.

Figure 1 depicts a system for interpolating a signal by a factor of L, where the output of the first system, referred to as a sampling rate expander is

$$x_e[n] = \begin{cases} x[n/L], & n = 0, \pm L, \pm 2L, \text{etc.}, \\ 0, & \text{otherwise}, \end{cases}$$

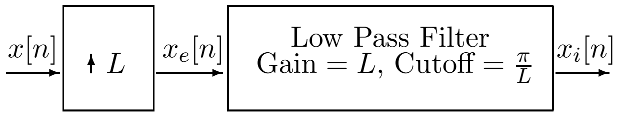


Figure 1: General System for interpolation by L

The lowpass filter interpolates between the non-zero values of $x_e[n]$ to generate the interpolated signal $x_i[n]$. The output $x_i[n]$ essentially corresponds to an upsampled version of $x[n]$. When the lowpass filter is ideal, the interpolation is referred to as bandlimited interpolation.

Accurate bandlimited interpolation requires a carefully designed high-order lowpass filter. Two simple and very approximate procedures which are often used instead are zero-order hold and linear interpolation. For zero-order hold interpolation, each value of $x[n]$ is simply repeated L times, i.e.,

$$x_i[n] = \begin{cases} x_e[0], & n = 0, 1, \ldots, L-1 \\ x_e[L], & n = L, L+1, \ldots, 2L-1 \\ x_e[2L], & n = 2L, 2L+1, \ldots, \\ \vdots \end{cases} \tag{0.1}$$

This can be accomplished by convolving $x_e[n]$ with the impulse response

$$h_{zoh}[n] = \delta[n] + \delta[n-1] + \ldots + \delta[n-(L-1)] \tag{0.2}$$

Zero-order hold interpolation is often used in digital to analog converters, resulting in analog "stairstep" waveforms (e.g., each digital sample is converted to a voltage, and that voltage is "held" for the duration of the sampling period).

Linear interpolation can be accomplished using a system with impulse response

$$h_{lin}[n] = \begin{cases} 1 - |n|/L, & |n| \leq L-1, \\ 0, & \text{otherwise}. \end{cases} \tag{0.3}$$

Unlike the zero-order hold interpolator, the linear interpolator is noncausal, and has zero group delay.

The ideal bandlimited interpolator is given by the noncausal impulse response:

$$h_{ideal}[n] = \begin{cases} \dfrac{\sin(\pi n/L)}{(\pi n/L)} & n \neq 0 \\ 1 & n = 0. \end{cases} \tag{0.4}$$

Background Reading

Oppenheim and Schafer (1989), Crochiere and Rabiner (1983), Oetken, Parks, and Schüßler (1975), and Golomb and Weinberger (1959).

Project 1: Interpolation Filter Performance

In this project the performances of three interpolation filters are evaluated: a zero-order hold, linear interpolator, and a high order, sharp cutoff lowpass filter.

Hints

In order to perform the sampling rate expansion shown in Figure 1 to generate the signal $x_e[n]$, it will be useful to define the function `srexpand(x,L)` that takes a sequence x and an integer L as arguments and returns a sequence which is a zero-filled version of x as follows:

```
%srexpand  y = srexpand(x,L) zero fills a sequence X by placing L-1
%          zeros between each sample of the sequence. The resulting
%          sequence has length equal to length(X)*L.
function y = srexpand(x,L)
N = L*length(x);
y = zeros(1,N);
y(1:L:N) = x;
```

The MATLAB function `filter(b,a,x)` implements a difference equation. The output of `filter` will be exactly as long as the input data sequence–the same as convolving the input data sequence with the impulse response, and truncating the result to the length of the input sequence. The MATLAB function `conv` performs true convolution.

Before continuing, type `load BLIdata` at the MATLAB prompt to load the variables for this project.

Exercise 1.1: Linear and Zero Order Hold Interpolation

(a) Given $x_e[n]$, the output of the sampling rate expander in Figure 1, write out the difference equations that correspond to the zero order hold interpolation filter and the linear interpolation filter for the impulse responses $h_{zoh}[n]$ and $h_{lin}[n]$.

Note that the impulse response in (0.3) and therefore the difference equation for the linear interpolator is non-causal.

(b) Enter the impulse responses for the zero order hold and linear interpolators using the value $L = 5$. The zero order hold interpolator should satisfy equation (0.2). Note that the linear interpolator in equation (0.3) is non-causal—because MATLAB does not recognize negative indices, you can enter a causal version, and then "time advance" the result by relabelling the time axis appropriately. Assign the impulse response sequences to the MATLAB variables `hzoh` and `hlin`, respectively. Plot the magnitude of the frequency responses of the zero order hold and linear interpolation filters on the same graph. Which is a better approximation to the ideal bandlimited interpolator?

(c) Using the MATLAB functions `srexpand` and `filter`, implement the interpolation system of Figure 1 with a value of $L = 5$. Use both the zero order hold and the linear interpolation impulse responses as the interpolation filter. For input to the system use the sequence `data1` that has been provided. Assign the upsampled sequences to the MATLAB variables `xzoh1` and `xlin1`.

(d) Using the MATLAB function `comb` plot the sequences `hzoh`, `hlin`, `xzoh1`, and `xlin1` on a time axis from -10 to +10. In establishing the time axis for the plot be sure to take into account that `hlin` is non-causal. You can check whether or not you have the time axis appropriately lined up for `xzoh1` and `xlin1` by noting that for both the zero order hold and the linear interpolator, all the values of the original data are exactly preserved on the expanded time axis.

Exercise 1.2: Interpolation With A Sharp Cutoff Filter

(a) You have been provided with an interpolation filter which more closely approximates the ideal lowpass filter than either the linear interpolator or the zero order hold interpolator. The impulse response of a causal version of this filter is located in the MATLAB variable `sharpfilt`. The cascade of the causal version and an ideal time advance of 20 samples is the correct (noncausal) implementation.

Analytically express the phase of the frequency response of the *noncausal* filter.

(b) Plot the magnitude of the frequency response of the filter `sharpfilt`.

(c) Interpolate the sequence `data1` using `sharpfilt`. Assign the resulting sequence to the MATLAB variable `xsf1`.

(d) Using the MATLAB function `comb` plot `xsf1` on a time axis from -30 to +50.

Exercise 1.3: Some Bookkeeping

In exercises 1.1 and 1.2 we have implemented the system of Figure 1 with $L = 5$ for three different filters. In exercise 1.4 we will want to compare the results with essentially ideal bandlimited interpolation. To do that we need to be careful about comparing appropriate segments of $x_i[n]$.

The MATLAB function `filter` filters a finite length input data vector with a causal filter and returns an output data vector which is truncated to be the same length as the input data vector. Implementing a noncausal interpolation filter requires filtering with a causal version of the impulse response, then time advancing the output. Consequently after filtering $x_e[n]$ and applying the time advance appropriate to each filter, `xzoh1` `xlin1` and `xsf1` can only be compared on an interval of length less than the length of `data1`.

(a) For each of the three filters, with the appropriate time advance incorporated, specify the interval in n over which $x_i[n]$ is available.

(b) From your answer in (a), what is the largest interval common to all three filtered outputs.

(c) A second bookkeeping issue is that the MATLAB function `filter` assumes that `data1` is zero for $n < 0$ and consequently there is a startup transient in filtering until the filter impulse response totally engages the data. The duration of this transient depends on the length of the FIR filter. The three filtered outputs should only be compared after the transient. Taking this into account together with your answer to (b), `xzoh1`, `xlin1` and `xsf1` should only be compared over a common time interval $n_1 \leq n \leq n_2$. Determine n_1 and n_2.

Exercise 1.4: Performance

In this exercise we evaluate the performance of the different interpolation filters, as compared with what would be obtained by essentially exact bandlimited interpolation. The sequence corresponding to perfect bandlimited interpolation is contained in the MATLAB variable `ideal1`.

A measure of the average interpolation error is

$$e_x = \frac{1}{n_2 - n_1 + 1} \sum_{n=n_1}^{n_2} (x_i[n] - x_{ideal}[n])^2 \tag{1.1}$$

where $x_i[n]$ is the result of using one of the interpolating filters, $x_{ideal}[n]$ is the ideal bandlimited interpolation, and the parameters n_1 and n_2 are those determined in exercise 1.3(c).

Compute the average interpolation error for the three filters, e_{xzoh1}, e_{xlin1}, and e_{xsf1}. Which interpolation filter has the best performance? Is this what you expected?

Exercise 1.5: New Data

Upsample the sequence located in the MATLAB variable `data2` using the interpolators `hlin`, `hzoh`, and `sharpfilt`. Recompute the interpolation errors by comparing the interpolated sequences with the sequence located in the MATLAB variable `ideal2`. Which filter performs best for this set of data? Why are the interpolation errors so much different from those computed above? (Hint: examine the magnitude of the Fourier transforms of the two expanded sequences)

Project 2: Minimum-Norm Interpolation

The theory of optimal estimation of signals known to be in the class of bandlimited, finite energy signals is used to provide a best estimate for missing samples of a bandlimited signal. A bandlimited signal with bandwidth B is found which passes through given, equally spaced samples, and which has minimum energy. Samples of this bandlimited signal are computed by filtering a sequence made by inserting zeros in between each of the original known samples as shown in Figure 2. In the figure, the sampling rate is increased by a factor of $r = 4$ by using a weighted combination of $L = 2$ points on each side of the point to be estimated. The weights on these 4 points are calculated from the requirement that a bandlimited signal of normalized bandwidth $\frac{\alpha}{2r}$, where $0 < \alpha < 1$, fits the 4 known samples and has minimum energy.

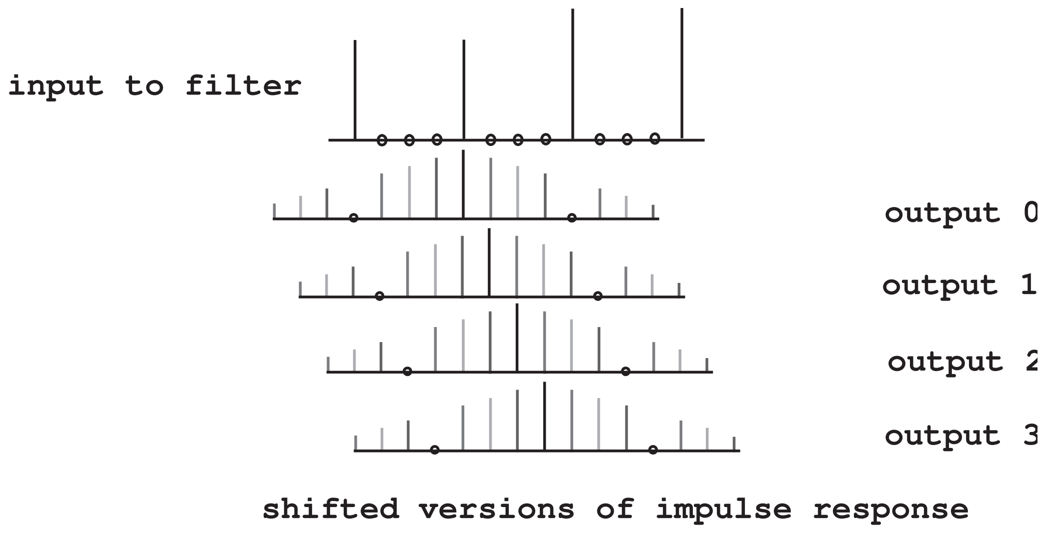


Figure 2: Interpolation as Linear Filtering

Exercise 2.1: Representers

Show that, if $u[n]$ is a bandlimited discrete-time signal with normalized bandwidth B,

$$u[n] = \sum_{m=-\infty}^{\infty} 2B\mathrm{sinc}(2B(n-m))u[m]$$

where

$$\mathrm{sinc}(x) = \frac{sin(\pi x)}{\pi x}.$$

In other words, the representer of the linear functional $F(u) = u[n_0]$ is

$$\phi[n] = 2B\mathrm{sinc}(2B(n-n_0)),$$

when the inner-product is

$$\langle x, y\rangle = \sum_{n=-\infty}^{\infty} x[n]y[n].$$

Exercise 2.2: Interpolation by a factor of 4

Assume that a signal $u[n]$ is bandlimited to a bandwidth $B = \frac{1}{8}\alpha$ where $0 < \alpha < 1$. Show that the best linear estimate for a $u[k]$, for $k = 1, 2, 3$, given $u[-4], u[0], u[4], u[8]$

$$\hat{u}[k] = \sum_{m=-1}^{2} a_{k,m}u[4m],$$

has coefficients $a_{k,m}$ given by solution of the following 4×4 system of linear equations:

$$\sum_{m=1}^{4} \mathrm{sinc}(\alpha(m-n))a_{k,m} = \mathrm{sinc}(\alpha((n-2) - \frac{k}{4})).$$

Exercise 2.3: Bandlimited Interpolation Filter

Find the impulse response of a length-15 filter as shown in Figure 2, which when convolved with the zero-filled bandlimited signal

$$u[n] = 0, \qquad n \bmod 4 \neq 0$$

with bandwidth $\frac{1}{8}\alpha$ will give the optimum bandlimited interpolation of the missing samples. *HINT:* Rearrange the coefficients $a_{k,m}$ found in the previous exercise to form the necessary impulse response.

Examine this filter in the frequency domain by plotting the magnitude of its frequency response. How does it compare with the magnitude responses of the zero-order hold interpolation filter, and the linear interpolation filter?

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{The Zoom Transform}}$

The Zoom Transform

Overview

The N-point DFT of an N-point sequence represents samples of the DTFT and contains all the information about the DTFT of the sequence. However, some characteristics of the DTFT may not be visually apparent from these samples. Consequently, it is often useful to interpolate the DFT in a frequency band of interest. In this set of exercises we examine two different methods for performing this interpolation, often referred to as the "zoom transform."

In both approaches, we consider $X_N[k]$ to be the N point DFT of a finite length sequence $x[n]$, representing N frequency samples separated in frequency by $2\pi/N$. Given $X_N[k]$, we would like to zoom in on the region between $\omega_c - \Delta\omega$ and $\omega_c + \Delta\omega$. We assume that in this region we want to calculate L equally spaced frequency samples, *i.e.* the result of the zoom transform will be L equally spaced frequency samples in the interval $\omega_c - \Delta\omega$ to $\omega_c + \Delta\omega$, specifically, the frequency samples

$$\omega_k = (\omega_c - \Delta\omega) + \frac{2\Delta\omega}{L}k, \qquad k = 0, 1, \ldots, L-1$$

Background Reading

Oppenheim and Schafer (1989) Chapter 11 and Problem 11.4, and Proakis and Manolakis (1992) Chapter 10 and Problem 10.20.

Project 1: The Zoom Transform

Exercise 1.1:

The first method that we consider is shown below in Figure 1. Starting from the N-point DFT $X_N[k]$, $x[n]$ is computed, modulated and lowpass filtered to form $x_1[n]$, then compressed to form $x_z[n]$. The P-point DFT of $x_z[n]$ contains the L desired zoom transform samples ($P \geq L$). Assume that $h[n]$ is the impulse response of an ideal lowpass filter with frequency response

$$H(e^{j\omega}) = \begin{cases} 0 & -\pi \leq \omega < -\Delta\omega \\ 1 & -\Delta\omega \leq \omega < \Delta\omega \\ 0 & \Delta\omega \leq \omega \leq \pi \end{cases}$$

and that $f[n]$ is the complex exponential sequence

$$f[n] = e^{-j\omega_c n}$$

Depending on the choice of M, the sequence $x_z[n]$ may need be extended with zero values (zero–padded) prior to computing the P-point DFT. Find appropriate values for $\Delta\omega$ and M, such that the value of P in Figure 1 can be equal to L, assuming that $\pi/\Delta\omega$ is an integer. With $P = L$, we don't compute more DFT samples than we desire.

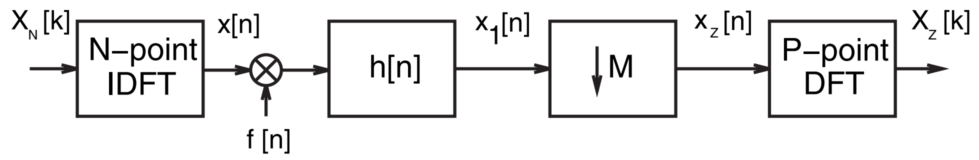


Figure 1: Zoom Transform, Method 1

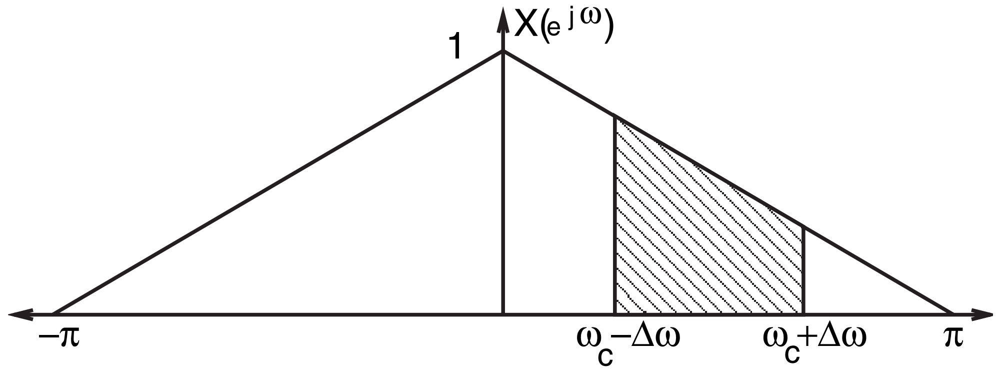


Figure 2: $X(e^{j\omega})$

Exercise 1.2:

Consider $x[n]$ with Fourier transform $X(e^{j\omega})$ shown in Figure 2. Sketch the Fourier transforms of the intermediate signals $x_1[n]$ and $x_z[n]$ when $\omega_c = \dfrac{\pi}{3}$ and $\Delta\omega = \dfrac{\pi}{4}$, and M is chosen as in Exercise 1.1.

Exercise 1.3:

In the system in Figure 1, $h[n]$ is a lowpass filter. If the filter is ideal, $\Delta\omega$ and M can be chosen as in Exercise 1.1. However, since the transition bands of any $H(e^{j\omega})$ we can implement has nonzero width, we will have to choose a smaller value of M to avoid aliasing. For $M = 3$ and $\Delta\omega = \dfrac{\pi}{4}$ an appropriate set of specifications is:

- passband edge frequency $\omega_p = .25\pi$
- stopband edge frequency $\omega_s = .31\pi$
- passband tolerance $\delta_1 = .01$ (passband varies from $(1 + \delta_1)$ to $(1 - \delta_1)$)
- stopband tolerance $\delta_2 = .01$

Use any appropriate design method to obtain the impulse response of a linear phase FIR filter meeting these specifications. Be sure to document the method used, and demonstrate that your filter meets the specifications given.

Exercise 1.4:

Implement the system in Figure 1 with $N = 512$, $L = 384$, $M = 3$, $\Delta\omega = \frac{\pi}{4}$ and $\omega_c = \frac{\pi}{2}$. Choose an appropriate value for P. Test your system on the sequence

$$x[n] = \sin(0.495\pi n) + \sin(0.5\pi n) + \sin(0.505\pi n) \qquad 0 \leq n \leq 511.$$

Turn in a plot of the magnitude of the DFT of $x[n]$ and the magnitude of the zoomed DFT for $\omega_c = \frac{\pi}{2}$ and $\Delta\omega = \frac{\pi}{4}$. Specifically note which points on the zoomed DFT correspond to $\omega_c - \Delta\omega$, ω_c and $\omega_c + \Delta\omega$.

Exercise 1.5:

In this exercise we consider the problem of zooming in (in the time domain) on a portion of a discrete-time periodic sequence $\tilde{g}[n]$. In exercise 1.6 we will then use a somewhat similar approach to obtain an expanded view of the spectrum of a signal, *i.e.* , as an alternative method for implementing the zoom transform.

Consider a periodic, bandlimited continuous-time signal $\tilde{g}_c(t)$ with period $T = 2\pi$, sampled with a sampling period of $T_s = \frac{2\pi}{N}$ (where N is sufficiently large to avoid aliasing). The resulting discrete-time signal $\tilde{g}[n]$ will be periodic with period N. Given only the samples $\tilde{g}[n]$, we would like to "zoom" in on a portion of $\tilde{g}_c(t)$. This can be accomplished by interpolating $\tilde{g}[n]$ to obtain L equally spaced time samples in the region between $t_c - \Delta t$ and $t_c + \Delta t$. The time samples t_k should satisfy

$$t_k = (t_c - \Delta t) + \frac{2\Delta t}{L}k, \qquad k = 0, 1, \ldots, L-1$$

The basic approach is shown in Figure 3:

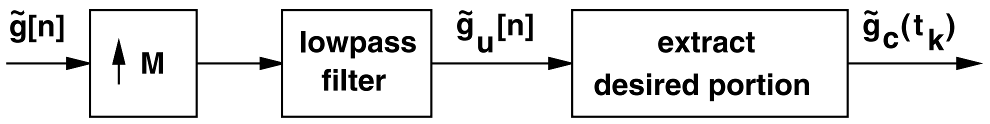


Figure 3: Interpolate $\tilde{g}[n]$

We assume that

$$\frac{2\Delta t}{L} = \frac{2\pi}{MN}$$

If the lowpass filter is a linear phase filter with integer group delay, the upsampled version of $\tilde{g}[n]$ ($\tilde{g}_u[n]$) will correspond to $\tilde{g}_c(t)$ sampled at integer multiples of T_s/M. If the group delay is not an integer, the samples $\tilde{g}_u[n]$ can be at noninteger multiples of T_s/M.

(a) For $M = 3$, determine $H(e^{j\omega})$, the Fourier transform of the lowpass filter in Figure 3, so that the desired output will be obtained.

(b) Suppose you used the following signal as the input to the system:

$$g[n] = \begin{cases} \tilde{g}[n] & 0 \leq n < N \\ 0 & \text{otherwise} \end{cases}$$

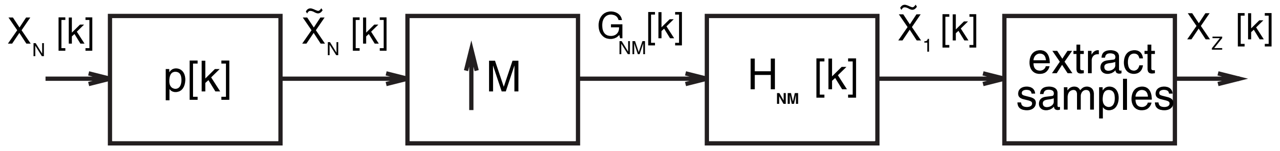


Figure 4: Zoom Transform, Method 2

and the lowpass filter implemented a circular convolution rather than a linear convolution, to obtain a finite-length output $g_u[n]$. Would $g_u[n]$, be equivalent to one period of $\tilde{g}_u[n]$? Explain why or why not.

Exercise 1.6:

We now want to use an interpolation strategy similar to that in exercise 1.5 to interpolate the DFT and extract the desired frequency samples. Essentially, the strategy is to upsample $X_N[k]$ as indicated in Figure 4. As before, $X_N[k]$ is the N-point DFT of a sequence $x[n]$, $G_{NM}[k]$ is the NM-point DFT of a sequence $g[n]$ and $H_{NM}[k]$ is the NM-point DFT of a sequence $h[n]$. $X_1[k]$, the output of the "low-time lifter", is the NM-point circular convolution of $G_{NM}[k]$ and $H_{NM}[k]$. The sequence $X_z[k]$ then corresponds to the desired frequency samples.

(a) In terms of N and M, specify $h[n]$ so that $X_1[k]$ will exactly correspond to the NM-point DFT of $x[n]$.

(b) One way of approximately implementing the circular convolution in Figure 4 is to replicate $G_{NM}[k]$ several times and implement the linear convolution of this replicated sequence with the finite-length sequence $H_{NM}[k]$. Implement this strategy with an appropriately chosen $H_{NM}[k]$ for the parameters and test signal used in exercise 1.4. Hand in a plot of the magnitude of the resulting 384 "zoomed" frequency samples. Also, explicitly indicate how many replications of $G_{NM}[k]$ were used to approximate the circular convolution.

(c) The desired circular convolution can also be implemented through the use of the DFT, *i.e.* by multiplying together the IDFT's of the two sequences to be circularly convolved. Implement the system of Figure 4 using this strategy for the circular convolution. Again, hand in a plot of the magnitude of the resulting 384 "zoomed" frequency samples.

Computer-Based Exercises for Signal Processing

$\boxed{\text{Rate Changing}}$

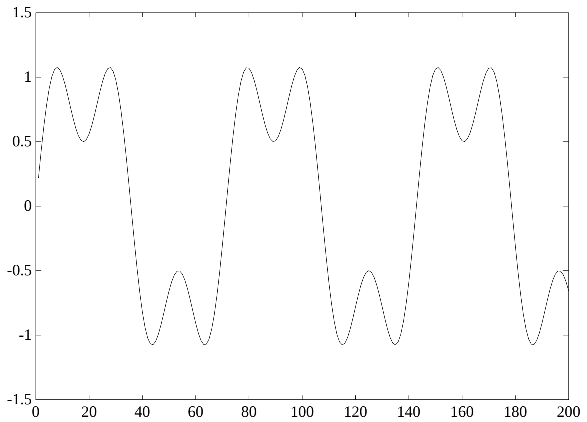


Figure 1: Continuous Two-Component Signal

Rate Changing

Overview

In this packet, we will be looking at the ideas of decimation and rate changing. In particular, we will need to learn how to recognize aliasing in a multi-component signal, how decimation may cause aliasing, and how a pre-decimation filter may prevent aliasing (at a cost of some loss of information, however).

Background Reading

1. A.V. Oppenheim, R.W. Schafer: *Discrete-Time Signal Processing*, Prentice-Hall, Englewood Cliffs N.J., 1989.

Project 1: Rate Reduction, Decimation

This project uses a graphical approach to see the effect of decimation, or rate reduction on a waveform.

In order to recognize aliasing in the time domain (or as plotted by Matlab), we need to have some type of frequency analyzer available. That is, we need first to recognize the individual sinusoidal components of a particular signal, and to then identify the frequencies of these components. Only then can we address the question as to whether or not these components have been aliased down from higher component frequencies. In the case of a single frequency component we only need to look for a single output (sinusoidal) component. This analysis process becomes more difficult when there are more components to identify (and much more difficult with real audio signals such as speech and music).

The eye is quite good at identifying apparent patterns in graphical data. For example, Fig. 1 shows a waveform for a signal that consists of a fundamental frequency and a third harmonic.

As seen in this manner, as a continuous function, we feel confident in identifying the relative frequencies of the two components. Similarly, presented as a dense set of discrete

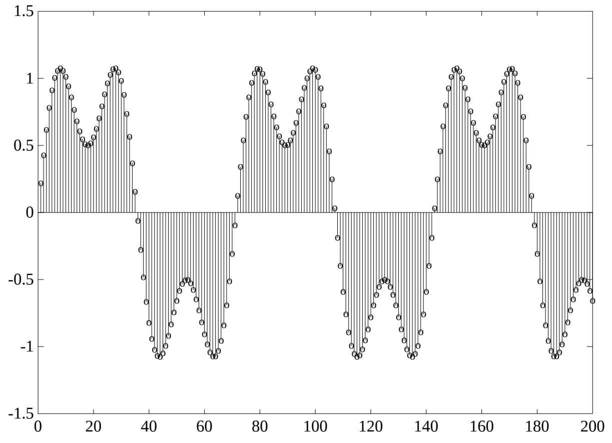


Figure 2: Discrete Two-Component Signal

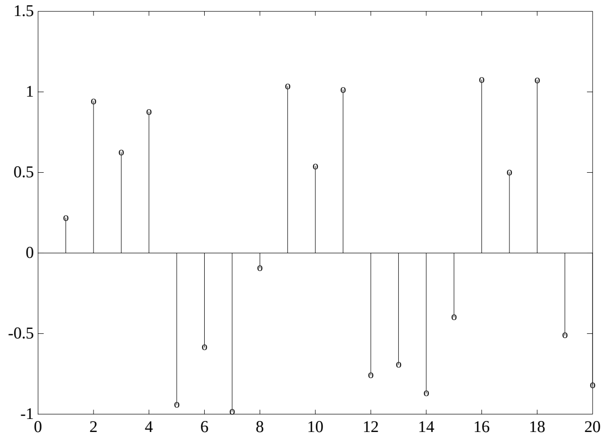


Figure 3: Fig. 2 at 1/10 the Sample Rate

samples (Fig. 2), we will usually draw the same conclusions (assuming no aliasing).

What we have seen here is that if we have many samples per cycle at all frequencies of interest, we get a good idea what components are present.

Fig. 3 shows a case that has 10 times fewer samples than the case of Fig. 2, and its frequency content is less obvious overall.

While the eye still detects a certain periodicity at the lower of the two components, it is not at all clear that there is a higher frequency present (the deviations from the waveform of the lower frequency component might have been noise, for example). If there is a higher frequency, it is not obvious what its frequency might be.

However, we can interpolate the signal of Fig. 3 by a factor of 4, using bandlimited interpolation, arriving at Fig. 4, which again shows strong evidence of the third harmonic.

These interpolation methods make the assumption that the signals are bandlimited.

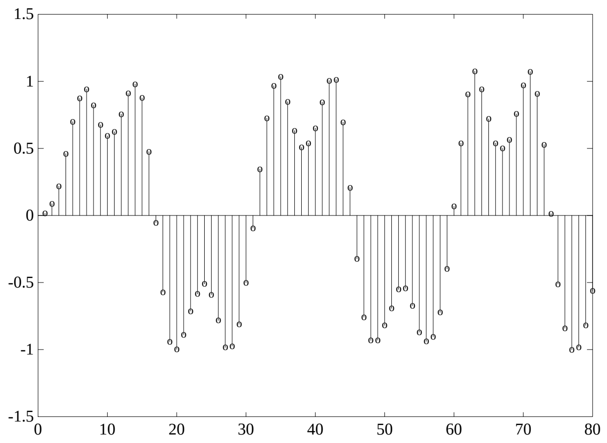


Figure 4: Fig. 3 Interpolated by factor of 4

That is, we are attempting to recover samples that would have been obtained if we had simply sampled faster in the first place. This is possible if the sampling theorem was obeyed at the lower (uninterpolated) rate.

Exercise 1.1: Decimation of a Two Component Signal

Obtain a signal consisting of two sinusoidal components, one of which has a frequency of 0.02 times the sampling frequency and the other which has a frequency of 0.09 times the sampling frequency. Plot about two cycles (at the lower frequency) worth of data. Plot this as a continuous function (Matlab `plot`) and as discrete samples (Matlab `comb`) in the manner of Fig. 1 and Fig. 2, respectively. Note that you have over 10 samples per cycle at the higher frequency. Identify the cycles by eye.

At this point, we want to decimate the signal. We will look at this in two ways—with and without an appropriate "pre-decimation" filter. Whether or not you need such a pre-decimation filter depends on the original signal's bandwidth, and the decimation factor. This overall problem can be simply evaluated by considering that if you take samples, and then throw them away, it is the same as not taking them in the first place. Thus, if it is the case that aliasing would have occurred if the original signal had been sampled at the final rate, then aliasing will occur if we do not use an appropriate pre-decimation filter.

Exercise 1.2: Decimation Without Pre-Decimation Filter—Case 1

Decimate the signal obtained above by a 4:1 factor. Note that this decimation makes it difficult to see what frequencies result. Consequently, we need to interpolate back additional samples. [The fact that in order to approach or obtain aliasing we need to decimate by a factor that leaves relatively few samples per cycle at the highest frequency component, so that we can't see it well, is a sort of "Catch 22" for this lab. We use interpolation to get around this problem.] Use a favorite method of discrete time interpolation to iterpolate back up and identify the frequency components.

Exercise 1.3: Decimation Without Pre-Decimation Filtering—Case 2

Repeat Exercise 1.1, but use a 6:1 decimation factor.

Exercise 1.4: Use of Pre-Decimation Filter

Decimate as in the previous exercise, but before you decimate, low-pass filter the sequence at slightly below 1/2 the final sampling rate.

Exercise 1.5: Post-Decimation Filtering

Apply the low-pass filter of the previous exercise to the decimated signal of exercise 1.2. Can you use the pre-decimation filter as a post-decimation filter with the same results?

Project 2: Rate Changing

This project asks you to analyze the result of the combination of a rate reduction (decimation) and a rate increase (interpolation) to get non-integer changes in rate.

In this project you are expected to design appropriate filters, implement the suggested rate change, and answer the specific questions in the exercises.

Exercise 2.1: Rate Increase by a Factor of 3/2.

Using a signal of your choice, implement a 3/2 sample rate increase. Does the order of interpolation and decimation matter for this case? Consider both the maximum sampling rate involved here, and any possible problems with aliasing. Do you need a decimation filter for this case? What would be the consequences of choosing other signals as your starting point?

Exercise 2.2: Rate Decrease by a Factor of 2/3

Using a signal of your choice, implement a 2/3 sample rate decrease. Does the order of interpolation and decimation matter for this case? Do you need a decimation filter for this case? If not, why not? Do you need a separate decimation filter, or does the interpolation filter do the job of both? Consider these questions in detail as they apply to your particular case.

Chapter 5

Systems and Structures

January 17, 2007

Overview

This chapter deals with linear systems, its possible different descriptions, its division into classes with different properties and some of the numerous structures for their implementation.

Of special importance are those descriptions, which characterize the system completely. MATLAB provides some programs for the transformation of one set of parameters, belonging to one type of description into another. Further programs of this type will be developed in the exercises. It turns out that some of these transformations are simple while others require rather involved calculations, thus yielding an impression of the practicability of certain descriptions. Obviously there is a relation to the identification problem: Starting with an existing system with unknown parameters, measuring procedures are of interest, which provide results to be used for the calculations of a complete description of the system under consideration.

Other characterizations show specific properties of a system only, but they fail, if completeness is required. It is one purpose of the corresponding project to show the difference between a complete and an incomplete description of a system.

Based on its properties, expressed either in the time or in the frequency domain, systems can be separated into classes. That includes basic features such as stability and causality, but as well distinctions based on the length of the impulse response, which can be of finite or infinite length. Other separations are related to the phase response, yielding the classes of linear-, nonminimum- and minimum phase systems. Furthermore allpasses have to be considered. Beside different several practical applications they are of theoretical interest, since a nonminimum phase system can always be described as a cascade of a minimum phase system and an appropriately chosen allpass. Finally a distinction of systems based on energy considerations is of interest. Especially lossless systems found practical applications, which can be implemented with allpasses as building blocs.

Tests are to be developed yielding a decision about the type of system in terms of these classes, while in other projects the properties of the different groups are considered. Furthermore it turns out that the required number of information for a complete description of a system can be reduced, if the class is known, the system belongs to.

Considering structures of systems yields a bridge to its actual implementation. There are numerous possibilities for building a system, if e.g. its transfer function is given. Under ideal conditions, i.e. if all wordlength effects are neglected, all different implementations yield the same behavior, if the design has been done properly. But they might differ in terms of the required number of arithmetic operations and delay elements. So the few structures, considered in this chapter, will be compared in terms of their arithmetic complexity, while their differences due to the limited wordlength are one topic in chapter 7.

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{Systems and Structures}}$

Systems and Structures

Overview

Linear systems with constant coefficients can be described differently. In the time domain a difference equation, which might be given in state space notation, or the impulse response yield all information. Correspondingly the transfer function $H(z)$ in its different representations provide complete descriptions in the z-domain. Its specialization on the unit circle yields the frequency response $H(e^{j\omega})$ and its components. This chapter deals in its first two projects with these possibilities. As we shall see most of these descriptions characterize the system completely, some only partially. If they provide a complete description all the others can be found by appropriate transformations.

Systems can be divided into classes, characterized by their behavior. These distinct types yield corresponding properties of the transfer function $H(z)$, the frequency response $H(e^{j\omega})$ or its components and the impulse response. They are considered in project 3.

Concerning structures project 4 deals with a few of them. Beside the basic ones, usually known as the direct-, the cascade- and the parallel structure, a system consisting of a combination of allpasses as well as implementations in Lattice form are considered.

We refer to a large extent to chapter 5 in Oppenheim and Schafer, [1], furthermore to chapter 8 in Mullis and Roberts [2], chapter 8 in Lim and Oppenheim [3] and section 11.6 in Reid [4].

1. A.V. Oppenheim, R.W. Schafer, Discrete Time Signal Processing, Prentice Hall: Englewood Cliffs, N.J. 1988

2. R.A. Roberts, C.T. Mullis, Digital Signal Processing, Addision-Wesley Publishing Company, Reading, MA. 1987

3. J.S. Lim, A.V. Oppenheim, Advanced Topics in Signal Processing, Prentice Hall: Englewood Cliffs, N.J. 1988

4. S.G. Reid: Linear Systems Fundamentals, Continuous and Discrete, Classic and Modern. McGraw-Hill Company New York 1983.

Project 1: Description of Systems

In this project seven different descriptions of systems are introduced, their relation will be considered.

Project Description

Linear systems are described primarily in the time domain either by a difference equation of certain order or by state equations, containing additional information about the internal structure of the system. The difference equation is

$$\sum_{k=0}^{N} a_k y[n-k] = \sum_{\ell=0}^{M} b_\ell v[n-\ell], \quad a_0 = 1\,. \tag{1.1}$$

The corresponding state equations are

$$\begin{aligned} \mathbf{x}[n+1] &= \mathbf{A}\mathbf{x}[n] + \mathbf{b}v[n] \\ y[n] &= \mathbf{c}^T\mathbf{x}[n] + dv[n]\,, \end{aligned} \tag{1.2}$$

where $\mathbf{x}[n] = \Big[x_1[n],\ x_2[n], \ldots, x_N[n]\Big]^T$ is the vector of state variables. If $M = N$, one state representation equivalent to (1.1) is

$$\mathbf{A} = \begin{bmatrix} -a_1 & 1 & 0 & \ldots & 0 \\ -a_2 & 0 & \ddots & & \vdots \\ \vdots & \vdots & \ddots & \ddots & 1 \\ -a_N & 0 & \ldots & \ddots & 0 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} b_1 & - & b_0a_1 \\ b_2 & - & b_0a_2 \\ \vdots & & \vdots \\ b_N & - & b_0a_N \end{bmatrix} \tag{1.3}$$

$$\mathbf{c}^T = [1, 0, \ldots, 0]\,; \quad d = b_0$$

Descriptions of a system either by the coefficients a_k and b_ℓ in (1.1) or by the state equations (1.2) are complete, where the particular form of $\mathbf{A}, \mathbf{b}, \mathbf{c}$ and d in (1.2) provides further information about the particular structure used for the implementation of the system (see project 4).

Another description of a system in the time domain is given by

$$y[n] = \sum_{k=0}^{n} h[k]v[n-k]\,, \tag{1.4}$$

where $h[n]$ is its impulse response, in general a sequence of infinite length. A first closed form expression can be found as solution of the state equation (1.2) for $v[n]$ being the impulse $\delta[n]$. It is

$$h[n] = \begin{cases} d\,, & n = 0 \\ \mathbf{c}^T\mathbf{A}^{n-1}\mathbf{b}\,, & n \geq 1\,. \end{cases} \tag{1.5}$$

Particular values of $h[n]$ for $n = 0(1)\ldots$ can be calculated in MATLAB using the m-file `filter`, with $\delta[n]$ as the input signal, where the system has to be described by the coefficients a_k and b_ℓ. It provides the stepwise solution of (1.1) as

$$h[n] = -\sum_{k=1}^{N} a_k h[n-k] + \sum_{\ell=0}^{M} b_\ell \delta[n-\ell]\,, \tag{1.6}$$

yielding for $n > M$

$$h[n] = -\sum_{k=1}^{N} a_k h[n-k]\,. \tag{1.7}$$

It turns out that only the values $h[n]$, $n = 0(1)(N+M)$ are required for a complete characterization of the system.

A description in the z-domain is obtained after applying the z-transform to the difference equation or the state equations. We get the following two equivalent versions of the transfer function:

$$H(z) = \frac{\sum\limits_{\ell=0}^{M} b_\ell z^{-\ell}}{1 + \sum\limits_{k=1}^{N} a_k z^{-k}}, \tag{1.8}$$

$$H(z) = \mathbf{c}^T (z\mathbf{E} - \mathbf{A})^{-1}\mathbf{b} + d. \tag{1.9}$$

There are other equivalent representations of $H(z)$; besides the form given in (1.8) as a quotient of two polynomials, we can write a product form in terms of poles and zeros:

$$H(z) = b_0 \cdot \frac{\prod\limits_{\ell=1}^{M} (1 - z_\ell z^{-1})}{\prod\limits_{k=1}^{N} (1 - p_k z^{-1})}. \tag{1.10}$$

Furthermore, the partial fraction expansion is useful. In case of distinct poles and $M = N$ it is

$$H(z) = B_0 + \sum_{k=1}^{N} \frac{B_k}{1 - p_k z^{-1}}. \tag{1.11}$$

The relation to the impulse response $h[n]$ is given by

$$H(z) = Z\{h[n]\} = \sum_{n=0}^{\infty} h[n] z^{-n}, \tag{1.12}$$

yielding with the partial fraction expansion (1.11) a further closed form expression for $h[n]$

$$h[n] = Z^{-1}\{H(z)\} = B_0 \delta[n] + \sum_{k=1}^{N} B_k p_k^n. \tag{1.13}$$

Considering $H(z)$ especially on the unit circle $z = \exp(j\omega)$ yields the frequency response

$$H(e^{j\omega}) = \frac{\sum\limits_{\ell=0}^{M} b_\ell e^{-j\ell\omega}}{1 + \sum\limits_{k=1}^{N} a_k e^{-jk\omega}}. \tag{1.14}$$

$H(e^{j\omega})$ is a periodic function in ω with period 2π. Its components and related functions are

$$\begin{aligned} P(e^{j\omega}) &= \operatorname{Re}\{H(e^{j\omega})\}, \\ Q(e^{j\omega}) &= \operatorname{Im}\{H(e^{j\omega})\}, \end{aligned} \tag{1.15}$$

$$\text{the log-magnitude } \ln|H(e^{j\omega})| = \operatorname{Re}\{\ln[H(e^{j\omega})]\}, \tag{1.16}$$

$$\text{the phase } \varphi(\omega) = \text{phase}\{H(e^{j\omega})\} = \operatorname{Im}\{\ln[H(e^{j\omega})]\}, \tag{1.17}$$

$$\text{the group delay } \tau_g(\omega) = -\frac{d\varphi}{d\omega}. \tag{1.18}$$

The frequency response $H(e^{j\omega})$ as described by (1.14) can be calculated for distinct points ω_k using `freqz` yielding after minor manipulations the related functions, as given in (1.16) and (1.17). The m-file `grpdelay` calculates the group delay not as the derivative of the phase as given by (1.18) but according to a description based on the coefficients a_k and b_ℓ (see packet 4 in chapter 1).

Note that in (1.1), (1.8) and (1.14) the same coefficients are used. This is called the "transfer function" representation. Obviously, the sets of coefficients a_k and b_ℓ provide a complete description of the system. Other equally sufficient characterizations are given by $[\mathbf{A}, \mathbf{b}, \mathbf{c}^T, d]$, the parameters of the state space case (1.2) and the representations (1.10) and (1.11), given by the parameters $[p_k, z_\ell, b_0]$ and $[p_k, B_k, B_0]$ respectively. If one of these sets of parameters is given, the others can be calculated.

As has been mentioned, the same holds for a sufficient large number of values $h[n]$, which can be used for the calculation of the coefficients a_k and b_ℓ. In MATLAB this can be done by the m-file `prony`. Finally samples $H(e^{j\omega_k})$, $\omega_k = k \cdot \pi/K$, $k = 0 : K-1$, where $K \geq n$, corresponding to $2K+1$ real numbers, are sufficient for the description of the system. The m-file `invfreqz` calculates the coefficients a_k and b_ℓ, using these values. In both cases the degrees M and N of the numerator and denominator polynomial must be known (see chapter 8, packet 4).

For some investigations the autocorrelation sequence $\rho[m]$ of the impulse response is needed (see Exercise 3.2 in this chapter or chapter 6, section on FFT Spectrum Estimation). It is defined as

$$\rho[m] = \sum_{n=0}^{\infty} h[n]h[n+m] = h[m] * h[-m], \ \forall m \in \mathbb{Z} \tag{1.19}$$

$$= Z^{-1}\{H(z)H(z^{-1})\} = \frac{1}{2\pi j} \oint H(z)H(z^{-1})z^{m-1}dz\,. \tag{1.20}$$

If the partial fraction expansion of $H(z)$ is given as in (1.11), the autocorrelation sequence $\rho[m]$ can be calculated for $m \geq 0$ as

$$\rho[m] = Z^{-1}\{B_0 H(\infty) + \sum_{k=1}^{N} \frac{B_k}{1 - p_k z^{-1}} H(p_k^{-1})\}\,, \tag{1.21}$$

which yields for $m \in \mathbb{Z}$

$$\rho[m] = B_0 H(\infty)\delta[m] + \sum_{k=1}^{N} B_k H(p_k^{-1}) p_k^{|m|}\,. \tag{1.22}$$

In general the sequence $\rho[m]$ does not give all information about the system. But if it is of minimum phase (see project 3), the values $\rho[m]$, $m = 0(1)2n$ are sufficient for the calculation of a system, the impulse response $h[n]$ of which belongs to this autocorrelation sequence (see exercise 3.3). More generally speaking: given $\rho[m]$ of any system, it is possible to calculate the coefficients of a minimum phase system, related to the original one in the sense that both frequency responses have the same magnitude.

Fig. 1 shows a diagram of all the representations and the relations between them, together with the m-files that transform one representation into another. Most of these m-files are already provided by MATLAB or the Signal Processing Toolbox (`residuez`, `ss2tf`, `tf2ss`, `ss2zp`, `zp2ss`, `filter`, `freqz`, `prony`, `invfreqz`); the others are to be developed in the following exercises.

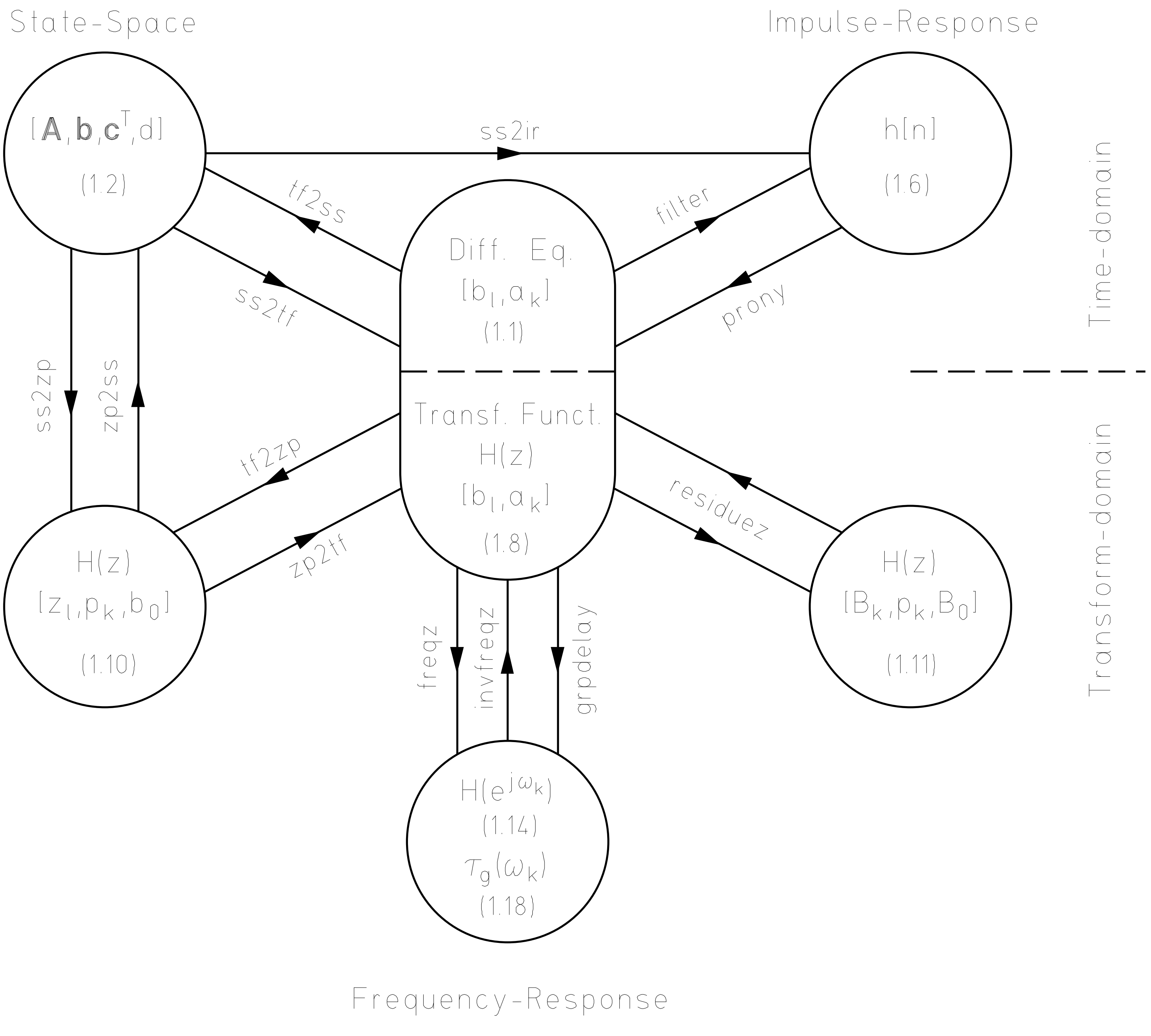


Figure 1: Representations of a system; m-files for transforming one into another

Exercise 1.1: Given a Difference equation

You are given the coefficients a_k and b_ℓ of a difference equation as[16]

$$\begin{aligned}\mathbf{a} &= [1 \quad -0.8741 \quad 0.9217 \quad -0.2672]^T \\ \mathbf{b} &= [0.1866 \quad 0.2036 \quad 0.2036 \quad 0.1866]^T\end{aligned} \tag{1.23}$$

(a) Calculate the parameters of the following descriptions of the system

1. The state-space representation using `tf2ss`
2. The partial fraction expansion (1.11) using `residuez`

(b) Write a function `tf2zp` for calculating the pole-zero representation (1.10) out of the transfer function. Then apply this function to the coefficients given in (1.23).

Hint: The factorization of a polynomial can be done with the m-file `roots`.

(c) Calculate 512 samples of the complex frequency response $H(e^{j\omega})$ for $0 \le \omega < 2\pi$ using `freqz`. Plot $|H(e^{j\omega})|$ for $0 \le \omega < 2\pi$ as well as an amplified version $10H(e^{j\omega})$ for $\frac{\pi}{2} \le \omega \le \frac{3\pi}{2}$ using `axis('square')`. Note the symmetry of $H(e^{j\omega})$ with respect to the real axis. Plot as well $|H(e^{j\omega})|$ and the phase $\varphi(\omega)$ for $0 \le \omega \le \pi$ with `axis('normal')`.

(d) Calculate and plot 100 values of the impulse response $h[n]$, first by using `filter` and then based on the closed form (1.13) using the partial fraction coefficients, found in part a2.

Exercise 1.2: Given samples $H(e^{j\omega_k})$ of the frequency response

You are given K samples $H(e^{j\omega_k})$ at $\omega_k = k\cdot\pi/K$, $k = 0(1)K-1$, calculated using `freqz` as in exercise 1.1, part c. Determine the coefficients of the transfer function by using `invfreqz` with $M = N = 3$. See `help invfreqz` for more information. Compare your result with the values given in (1.23), you started with. Find experimentally the smallest number of samples $H(e^{j\omega_k})$, i.e. the value $\min(K)$, required for obtaining accurate results.

Exercise 1.3: Given values $h[n]$ of the impulse response

Your are given L values $h[n],\ n = 0(1)L$, calculated by using `filter` as in exercise 1.1, part d. In order to find the coefficients with `prony` we need the degree N of the denominator of $H(z)$. If it is not known as in our example, it can be found by making use of the linear dependencies of the $h[n]$, as expressed in (1.7). As is shown in [•], the rank of a Hankel matrix, built out of $h[n],\ n \ge 1$ is the desired degree N. So proceed as follows:

- Let $h1 = h[n],\ n = 1 : L/2$,
  $h2 = h[n],\ n = L/2 : L$
- Generate a Hankel-matrix by

```
S = hankel(h1,h2)
```
(1.24)

[16]Elliptic filter found by `ellip(3,1,20,0.4)`

- It is

$$\texttt{N = rank(S)}. \tag{1.25}$$

Now proceed with `prony`, using N and $M = N$ for the degrees of the transfer function. Verify your result by comparing it with the values given in (1.23). Find experimentally the smallest number L of values $h[n]$, required for obtaining accurate results.

Hint: it is not necessary to repeat in all cases the degree-finding procedure, explained above.

Exercise 1.4: Given poles and zeros

You are given a transfer function $H(z)$ as in (1.10) specified by its poles p_k and zeros z_ℓ and the constant b_0.

$$\begin{aligned} &p_1 = 0.9\,, \quad p_{2,3} = 0.6718 \pm j0.6718 \\ &z_1 = -1\,, \quad z_{2,3} = \pm j \\ &b_0 = 1/77 \end{aligned}$$

(a) Write a function `zp2tf` for converting the pole zero description into the description by the difference equation (1.1) or the transfer function (1.8).

Hints: While `roots` yields a factorization of a polynomial the inverse operation of building a polynomial form out of its roots is accomplished with the function `poly`.

(b) Apply your program to the example given above. Transfer your result into the partial fraction expansion. Then compute the impulse response according to (1.13). Compare your result with that you obtain with `filter`, using the transfer function representation, found before.

(c) Calculate the output sequence $y[n]$ for the following input sequences of length 100:

$$\begin{aligned} v_1[n] &= \texttt{ones(1,100)} \quad \text{(yielding the step response)} \\ v_2[n] &= [1, -1, 1, -1, 1, -1, \ldots] \\ v_3[n] &= [1, 0, -1, 0, 1, 0, -1, \ldots] \end{aligned}$$

Compare and explain, why the outputs approach constant values for increasing n. Calculate these values using the transfer function representation of the system and the properties of the input sequences.

(d) Find an input sequence $v[n]$ of length 3 such that the corresponding output sequence is proportional to $(0.9)^n$ for $n \geq 3$.

Exercise 1.5: Given a state-space representation

You are given the description of a system by

$$\begin{aligned}
\mathbf{A} &= \begin{bmatrix} 0.3629 & 0 & 0 \\ 1.3629 & 0.5111 & -0.8580 \\ 0 & 0.8580 & 0 \end{bmatrix} \\
\mathbf{b} &= \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \\
\mathbf{c}^T &= [1.3629 \quad 0.6019 \quad 0.3074] \\
d &= 1
\end{aligned}$$

(a) Calculate the transfer function $H(z)$ as expressed by (1.8) using `ss2tf`. Find the poles p_k of the transfer function. Verify that they are equal to the eigenvalues of $\mathbf{A}$.

(b) Write a function `ss2ir` for the calculation of the impulse response $h[n]$ by solving the state equations

$$\begin{aligned}
\mathbf{x}[n+1] &= \mathbf{A}\mathbf{x}[n] + \mathbf{b}v[n] \\
y[n] &= \mathbf{c}^T\mathbf{x}[n] + dv[n]
\end{aligned}$$

such that $\mathbf{x}[n]$ as well as $y[n] = h[n]$ are obtained. Apply your program for $[\mathbf{A}, \mathbf{b}, \mathbf{c}^T, d]$ as given above for $n = 0 : 50$. Plot the components of $\mathbf{x}[n]$ and $h[n]$ together using `subplot`.

(c) Compare the result you obtained for $h[n]$ with the one you get with `filter`, using the transfer function, found in part (a).

(d) Draw a signal flow graph of the system, described by $\mathbf{A}$, $\mathbf{b}$, $\mathbf{c}^T$, d.

Exercise 1.6: Autocorrelation sequence of an impulse response

(a) Write a function `acimp` for the calculation of the autocorrelation sequence $\rho[m]$ of an impulse response $h[n]$ according to the closed form expression in (1.22) starting with the transfer function $H(z)$ given either as in (1.8) by the polynomials a and b or by its poles and zeros and a constant factor as in (1.10). Use `residuez` for the calculation of the required partial fraction expansion. Apply your program for the system described by (1.23) by calculating $\rho[m]$ for $m = 0 : 50$. Check your result by calculating $\rho[m]$ approximately either by convolving a truncated version of $h[m]$ with $h[-m]$ using `conv` or as $\mathrm{DFT}^{-1}\{|H(e^{j\omega})|^2\}$.

(b) As an alternative $\rho[m]$ can be calculated for $m \geq 0$ as the impulse response of a system, the transfer function of which is given in partial fraction expansion form as

$$H_{ac}(z) = B_0 H(\infty) + \sum_{k=1}^{N} \frac{B_k}{1 - p_k z^{-1}} H(p_k^{-1})\,. \tag{1.26}$$

(see (1.20)). So instead of determining $\rho[m]$ by using (1.22) it can be found with `filter` and the transfer function $H_{ac}(z)$ as the quotient of two polynomials, to be calculated out of (1.26) with `residuez`. Write a correspondingly modified version of your program `acimp`. Check your result by applying it for the same example given by (1.23). Compare the resulting sequence $\rho[m]$ with that obtained in part a.

(c) The coefficients of $H_{ac}(z)$ can be calculated as well using samples of $\rho[m]$ by applying `prony`, including the method for determing the degree of the system, as outlined in exercise 1.3. Use this procedure with the values $\rho[m]$, found in part a for the example and compare your resulting $H_{ac}(z)$ with that of part b.

Project 2: Measuring the frequency response for $\omega = \omega_k$.

Project Description

Samples of the frequency response $H(e^{j\omega})$ at $\omega = \omega_k$ can be found either by calculation according to (1.14), if the coefficients a_k and b_ℓ are known or by measurement either of the steady-state response for an excitation by $e^{j\omega_k n}$ in the lab or by using the measured impulse response, if the system is unknown. In this exercise the accuracy of results found by different measurement schemes and the reason for possible errors will be investigated.

In order to determine the error we use a test system, whose parameters a_k and b_ℓ are known. Thus the exact values $H(e^{j\omega_k})$ can be calculated with `freqz` and used for comparison. The first exercise serves as an introduction into the problem.

Exercise 2.1: Approaching the steady-state.

(a) As is well known, a particular value of $H(e^{j\omega_k})$ basically describes the steady-state response of the stable system, if excited by $v_k[n] = e^{j\omega_k n},\ n \geq 0$. In general this excitation yields an output sequence consisting of two parts

$$y_k[n] = y_{tk}[n] + H(e^{j\omega_k})e^{j\omega_k n} = y_{tk}[n] + y_{sk}[n]\,,$$

where the transient sequence $y_t[n]$ is decreasing exponentially due to the assumed stability. We want to separate the two parts for four values of ω_k, using the system described by (1.23).

1. Calculate $H(e^{j\omega_k})$ for $\omega_k = 0,\ \pi/4,\ 1.6164,\ 2\pi/3$ using `freqz`,
2. Calculate $y_k[n]$ using `filter` with $v_k[n] = e^{j\omega_k n},\ n = 0:100$.
3. Calculate the steady-state signals

$$y_{sk}[n] = H(e^{j\omega_k}) \cdot v_k[n] = H(e^{j\omega_k})e^{j\omega_k n}\,.$$

4. Calculate the transient signals by subtracting the steady state.

$$y_{tk}[n] = y_k[n] - y_{sk}[n]\,.$$

5. Plot the real parts of $v_k[n],\ y_k[n],\ y_{tk}[n]$ and $y_{sk}[n]$ together for each k using `subplot(22x)`

Determine by inspection how long it takes till the transient part can be ignored such that

$$y_k[n] \approx H(e^{j\omega_k}) \cdot e^{j\omega_k n}$$

with an error, the magnitude of which is $\leq 1\,\%$. Does the answer depend on $|H(e^{j\omega_k})|$?

(b) Show that the measured value for $H(e^{j\omega_k})$ can be obtained as

$$H(e^{j\omega_k}) = u_k[\infty] = \lim_{n\to\infty} y_k[n] \cdot v_k^*[n] = \lim_{n\to\infty} y_k[n] \cdot e^{-j\omega_k n}\,.$$

Calculate $u_k[n] = y_k[n] \cdot v_k^*[n]$ for the given four values ω_k and $n = 0 : 100$. Plot $\mathrm{Re}\,\{u_k[n]\}$ and $\mathrm{Im}\,\{u_k[n]\}$. Check the differences between $u_k[100]$ and the true values $u_k(\infty) = H(e^{j\omega_k})$ by calculating $|u_k[100] - H(e^{j\omega_k})|$.

Exercise 2.2: Frequency Response determined with Multitone Periodic Excitation

The operations outlined in exercise 2.1 basically describe a measurement procedure for use in the lab, where the complex frequency response $H(e^{j\omega})$ is measured point by point. We want to show that this measurement can be done simultaneously at $N/2$ points $\omega_k = = k \cdot 2\pi/N,\ k = 0 : N/2-1$ as follows:

(a) Generate the periodic input sequence

$$v[n] = \frac{1}{2} + \sum_{k=1}^{N/2-1} \cos nk2\pi/N + \frac{1}{2}(-1)^n\,, \quad n = 0 : \ell N - 1 \tag{2.1}$$

where N is chosen to be a power of 2 (e.g. $N = 32$) and ℓ is an integer > 1.

Hint: Look for a fast method to generate $v[n]$, taking into account that $v[n]$ is really a Fourier series.
Plot $v[n]$ using `comb`.

(b) Then calculate the output sequence $y[n]$ for the example given in (1.23) using `filter`. Note that for sufficiently large n the sequence $y[n]$ is approximately periodic.

(c) Select the ℓ-th part of $y[n]$ as $y_\ell[n] = y[n = (\ell-1)N : \ell N - 1]$ and calculate its N point DFT

$$Y_\ell[k] = \mathrm{DFT}\,\{y_\ell[n]\}\,.$$

We claim that the following result holds with high accuracy

$$H(e^{j\omega_k}) \approx \frac{2}{N} Y_\ell[k], \quad \text{for } \omega_k = k2\pi/N,\ k = 0 : N/2-1\,. \tag{2.2}$$

Calculate and plot the magnitude of the error

$$\epsilon_\ell[k] = \left| \frac{2}{N} Y_\ell[k] - H(e^{j\omega_k}) \right|$$

for $\ell = 1 : 4$.

(d) Express $y[n]$ in terms of the impulse response $h[n]$, regarding that $v[n]$ as given by (2.1) can be expressed as

$$v[n] = \begin{cases} N/2\,, & n = \lambda N\,,\ \lambda = 0 : \ell - 1 \\ 0\,, & n \neq \lambda N \end{cases} \tag{2.3}$$

What is the reason for a possible error? How does it depend on ℓ and N?
Under which condition is the above statement (2.2) precisely correct?

(e) A more general multitone periodic input signal can be used as well. Modify the above described procedure such that it works with

$$v[n] = \sum_{k=0}^{N-1} V[k] e^{jnk2\pi/N},\ n = 0 : \ell N - 1\,. \tag{2.4}$$

The complex values $V[k]$ can be choosen arbitrarily, except that the two conditions $V[k] \neq 0$ and $V[k] = V^*[N-k]$, such that $v[n]$ is real, must be satisfied.

Exercise 2.3: Frequency Response determined with a measured Impulse Reponse

According to (1.12) samples of the frequency response can be determined as

$$H(e^{j\omega_k}) = \sum_{n=0}^{\infty} h[n] e^{-jnk2\pi/N}\,. \tag{2.5}$$

Using the truncated version $h[n],\ n = 0 : L-1$ with $L \leq N$ we get

$$H(e^{j\omega_k}) \approx H_i(e^{j\omega_k}) = \sum_{n=0}^{L-1} h[n] e^{-jnk2\pi/N}\,, \tag{2.6}$$

to be executed in MATLAB as `fft(h,N)`.

Use this method to determine the frequency response of the system described by (1.23) again e.g. with $N = 32$. Calculate and plot the magnitude of the error

$$\epsilon_{iL}[k] = |H_{iL}(e^{j\omega_k}) - H(e^{j\omega_k})|$$

for different values of the length L. Compare your results with those obtained in exercise 2.2, part d.

Project 3: Types of systems

Based on possible properties of a system, several classes can be defined, leading to different characteristics of $H(z)$:

- A system is called *real*, if its response to any real input signal is real. The consequences for $H(z)$ are: The coefficients in the transfer function (1.8) are real; its poles and zeros are symmetric with respect to the real axis of the z-plane. Furthermore, the frequency response is conjugate symmetric: $H(e^{j\omega}) = H^*(e^{-j\omega})$.

  Remark: All systems considered in the exercises of this project are real.

- A system is called *stable*, if its impulse response is absolutely summable. That means for $H(z)$ that all its poles p_k are strictly inside the unit circle. The stability can be determined either by calculating the roots of the denonimator polynomial or by using a stability test procedure such as the Schur-Cohn-test.

- A system is called *causal*, if its impulse response $h[n]$ is zero for $n < 0$. That yields a transfer function of z with a numerator polynomial, whose degree is at most equal to the degree of the denominator polynomial, i.e. $M \leq N$. The components $P(e^{j\omega})$ and $Q(e^{j\omega})$ are then related by the Hilbert-Transform. In case of a real system the following relations hold:

$$\begin{aligned} P(e^{j\omega}) &= \mathrm{Re}\,\{H(e^{j\omega})\} &&= \sum_{n=0}^{\infty} h[n]\cos\omega n &&= h[0] + \mathcal{H}\{Q(e^{j\omega})\}\,, \\ Q(e^{j\omega}) &= \mathrm{Im}\,\{H(e^{j\omega})\} &&= -\sum_{n=1}^{\infty} h[n]\sin n\omega &&= -\mathcal{H}\{P(e^{j\omega})\}\ . \end{aligned} \tag{3.1}$$

- A stable system is called *FIR*, if its impulse response $h[n]$ is of finite length. In this case we have in (1.8) $a_k = 0$, $k > 0$, i.e. $N = 0$ and $b_n = h[n]$, $n = 0 : M$. Obviously all poles p_k in (1.10) are zero. The $M+1$ values of $h[n]$ describe the system completely and immediately. Note that this statement is a special case of the corresponding one for an IIR-system with an impulse response of infinite length, as given by (1.6), where $M+N+1$ values are necessary as a basis for a complete description. But as has been demonstrated in exercise 1.3, rather lengthy calculations are required in that general case, to get a transfer function type characterization of the system.

- A stable system has *linear phase*, if it is FIR and if its zeros are on the unit circle or reciprocal to the unit circle. That means a zero at z_ℓ has a mirror image zero at $1/z_\ell^*$. The numerator of its transfer function is either a mirror image polynomial if a possible zero at $z = 1$ is of even order, or an anti-mirror-image polynomial, if a zero at $z = 1$ is of odd order.

- A causal system is called *minimum phase*, if its phase is the smallest possible one of all systems with the same magnitude of the frequency response. There are two types of minimum-phase systems:

  (a) Those which are invertible, i.e., systems with a transfer function $H(z)$ such that $1/H(z)$ is stable and minimum phase as well. In this case, all zeros of the $H(z)$ are inside the unit circle.

  (b) Those, which are not invertible. Here all zeros of $H(z)$ are inside or on the unit circle.

  We mention that for all invertible minimum phase systems the group delay satisfies the condition:

$$\int_0^{\pi} \tau_g(\omega)d\omega = 0\,. \tag{3.2}$$

  Obviously, (3.2) implies that there are always one or more intervals, where the group delay must be negative.

The real and imaginary parts of $\ln[H(e^{j\omega})]$ are related by the Hilbert-Transform. If $H(z)$ describes an invertible system then

$$\ln[H(e^{j\omega})] = \sum_{k=0}^{\infty} c_k e^{-jk\omega}\,, \tag{3.3}$$

where

$$c_k = \frac{1}{2\pi}\int_{-\pi}^{\pi} \ln[H(e^{j\omega})] e^{jk\omega} d\omega \tag{3.4}$$

is a causal and real sequence, being the cepstrum of the impulse response $h[n]$ of the system. Here the causality of the sequence c_k is characteristic for a minimum phase system, while $c_k \in \mathbb{R}$ is only a consequence of the assumption that the system is real. Using the c_k we get

$$\begin{aligned} \text{Re}\{\ln[H(e^{j\omega})]\} &= \ln|H(e^{j\omega})| = \sum_{k=0}^{\infty} c_k \cos k\omega && (3.5)\\ &= c_0 + \mathcal{H}\{\varphi(\omega)\}\,, \\ \text{and}\quad \text{Im}\{\ln[H(e^{j\omega})]\} &= \varphi(\omega) = -\sum_{k=1}^{\infty} c_k \sin k\omega && (3.6)\\ &= -\mathcal{H}\{\ln|H(e^{j\omega})|\}\,. \end{aligned}$$

In this case the group delay is

$$\tau_g(\omega) = \sum_{k=1}^{\infty} k c_k \cos k\omega\,. \tag{3.7}$$

- A system with the property $|H(e^{j\omega})| = \text{const.}$ is called an *allpass*. The zeros of its transfer function are mirror images of its poles with respect to the unit circle. Thus the transfer function of an allpass is

$$H_A(z) = \frac{b_0 \cdot \prod_{k=1}^{N}(1 - p_k^{-1} z^{-1})}{\prod_{k=1}^{N}(1 - p_k z^{-1})}\,; \quad |H_A(e^{j\omega})| = \left| b_0 \prod_{k=1}^{N} p_k^{-1} \right| \tag{3.8}$$

$$H_A(z) = \frac{z^{-N} + \sum_{k=1}^{N} a_k z^{k-N}}{1 + \sum_{k=1}^{N} a_k z^{-k}} = \frac{z^{-N} \cdot D(z^{-1})}{D(z)}\,. \tag{3.9}$$

A nonmininum phase system, i.e. a system with a transfer function $H(z)$ having zeros outside the unit circle, can always be described by a cascade of a minimum phase system and an allpass with the transfer functions $H_M(z)$ and $H_A(z)$, respectively:

Starting with an arbitrary $H(z)$ from (1.10) we can write

$$H(z) = b_0 \cdot \frac{\prod_{\ell=1}^{N_1}(1 - z_\ell z^{-1}) \prod_{\lambda=1}^{N_2}(1 - z_\lambda^{-1} z^{-1})}{\prod_{k=1}^{N}(1 - p_k z^{-1})} \cdot \frac{\prod_{\lambda=1}^{N_2}(1 - z_\lambda z^{-1})}{\prod_{\lambda=1}^{N_2}(1 - z_\lambda^{-1} z^{-1})}$$

$$= \quad H_M(z) \cdot H_A(z)\,. \tag{3.10}$$

where $|z_\ell| \le 1,\ \ell = 1 : N_1,\ |z_\lambda| > 1,\ \lambda = 1 : N_2,\ N_1 + N_2 = N.$

Remark: Since the orders of $H_M(z)$ and $H_A(z)$ are N and N_2, respectively, the total order of the combination is $N + N_2$, while $H(z)$ is still of order N. A corresponding cascade implementation with the minimum phase system first would have N_2 uncontrollable natural modes characterized by $z_\lambda^{-n},\ \lambda = 1 : N_2$, while in a cascade with the allpass first these natural modes are unobservable.

Furthermore an energy relation can be used to distinguish a general system with $H(z)$ from the corresponding minimum phase one, described by $H_M(z)$. Let both be excited by the same input sequence $v[n]$, yielding the output sequences $y[n]$ and $y_M[n]$ respectively. We compare the two running energies defined as

$$w_y[m] = \sum_{n=0}^{m} y^2[n] \tag{3.11}$$

and $w_{y_M}[m]$ correspondingly. It can be shown that

$$w_{y_M}[m] \ge w_y[m],\ \forall m\,. \tag{3.12}$$

That means that the output energy of a nonminimum phase system never increases faster than that of the corresponding minimum phase system.

- *Passive* and *lossless* systems

  Finally we mention a class of digital systems with properties corresponding to those of an R,L,C-network. The definition is based on a comparison of the input and output energy of a system.

  Let $w_v[m]$ be the running energy of the input sequence, defined according to (3.11) and

$$w_v[\infty] = \sum_{n=0}^{\infty} v^2[n] \tag{3.13}$$

  its total energy, which is assumed to be finite. The corresponding figures of the output sequence are $w_y[m]$ and $w_y[\infty]$ respectively, to be compared with $w_v[m]$ and $w_v[\infty]$. Now the system is called *passive*, if

$$w_y[m] \le w_v[m]\,,\ \forall m\,; \tag{3.14}$$

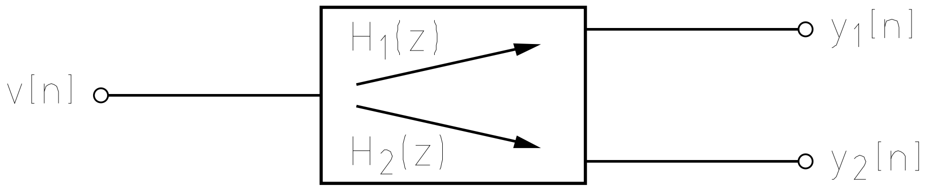
  and it is *lossless*, if

$$w_y[\infty] = w_v[\infty]\,. \tag{3.15}$$

  Using $Y(z) = H(z)V(z)$ and Parseval's equation the consequences for the transfer function can be derived. It turns out that an allpass with

$$|H(e^{j\omega})|^2 = 1\,,\ \forall \omega\,, \tag{3.16}$$

  is the only lossless system with one input and one output.



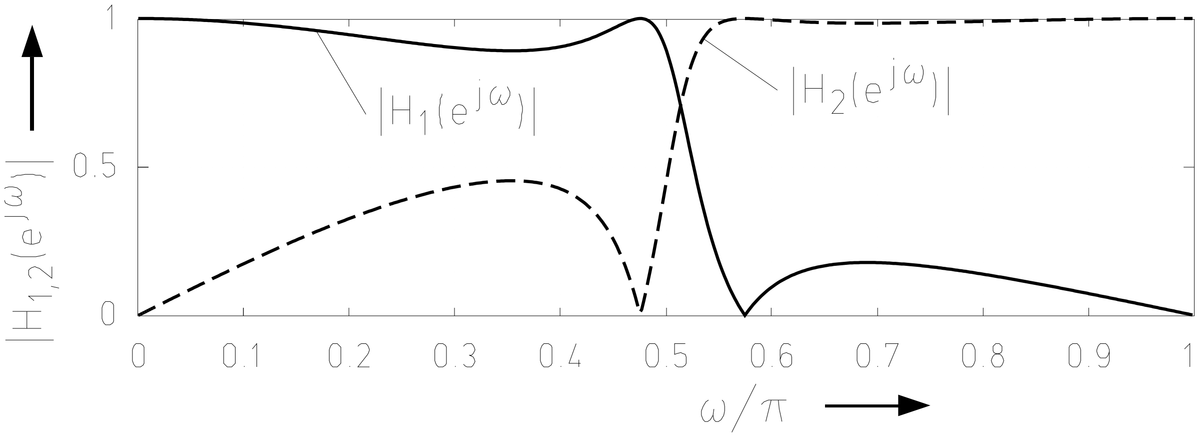


Figure 2: A lossless system with two complementary transfer functions

Now we consider another system which is more general in the sense that it has one input again, but two outputs, yielding the sequences $y_1[n]$ and $y_2[n]$. The total output energy with $\mathbf{y}[n] = [y_1[n],\ y_2[n]]^T$ is now defined as

$$w_y[\infty] = \sum_{n=0}^{\infty} \mathbf{y}^T[n]\mathbf{y}[n]\,. \tag{3.17}$$

If $H_1(z)$ and $H_2(z)$ are the two corresponding transfer functions, the relation

$$|H_1(e^{j\omega})|^2 + |H_2(e^{j\omega})|^2 = 1\,,\ \forall\omega \tag{3.18}$$

is found to be the condition for a lossless system.

The two transfer functions are complementary (see fig. 2), and they both satisfy the condition

$$|H_{1,2}(e^{j\omega})|^2 \leq 1\,. \tag{3.19}$$

Consequences in terms of sensitivity will be considered in project 3 of chapter (7).

Exercise 3.1: Stability

Suppose that the zeros of the denominator of a real system are partly given by

```
p = rand(1,5). * exp(j*pi*rand(1,5))
```

where `rand` generates a uniform distribution. Get further information with `help rand`.

(a) Find the coefficients a_k of the complete denominator polynomial $A(z)$ such that all the resulting coefficients are real.

Now the stability of the polynomial has to be checked.

(b) Using your knowledge about the generation of the polynomial $A(z)$: Do you expect stability?

(c) Calculate the roots of $A(z)$; check whether they are inside the unit circle.

(d) An alternative possibility is the Schur-Cohn stability test, which is briefly described as follows:

Starting with the given polynomial of degree N $A(z) = A_N(z)$ a sequence of polynomials $A_i(z),\ i = N : -1 : 0$ are calculated recursively according to

$$\begin{aligned} A_{i-1}(z) = &\ z^{-1}\left[A_i(z) - k_i z^i A_i(z^{-1})\right] \\ = &\ \sum_{\ell=0}^{i-1} a_\ell^{(i-1)} z^\ell\,, \quad i = N : -1 : 1\,, \end{aligned}$$

where $k_i = \dfrac{a_i^{(i)}}{a_0^{(i)}}$. Note that $z^i A_i(z^{-1})$ is a flipped version of $A_i(z)$. According to Schur-Cohn, the zeros of denominator are inside the unit circle, if

$$|k_i| < 1\,, \quad i = N : -1 : 1\,. \tag{3.20}$$

The values k_i are called the reflection coefficients (see project 4, section on the Lattice structure).

The Schur-Cohn test can be executed with the following program:

```
function k = atok(a)
%ATOK      implements the Inverse Levinson Recursion
%  usage:
%    K = atok(A) converts AR polynomial representation
%                  to reflection coefficients
%        where  A = vector of polynomial coefficients
%          and  K  "     "    reflection coefficients
%
%   NOTE: works only for vectors (possibly complex-valued)

a = a(:);   %--- make sure we are dealing with a column vector
N = length(a);
k = zeros(N-1,1);    %--- # refl. coeffs = # of roots-1 = degree-1
for i=(N-1):-1:1
   k(i) = a(i+1);
   b = flipud(conj(a));
   a = ( a - k(i)*b ) / ( 1 - k(i).*conj(k(i)) );
```

```
      a(i+1)=[];
   end
```

(e) Make up a polynomial with zeros on or outside the unit circle and show that the Schur-Cohn test detects the instability.

Exercise 3.2: Minimum and Nonminimum phase systems

The transfer function of a system is partly given by the poles

$$p_1 = 0.9 \cdot e^{j\pi/4}\,;\; p_2 = 0.8$$

and the zeros

$$z_1 = 1.5 \cdot e^{j\pi/8}\,;\; z_2 = 0.5$$

(a) Find additional poles and zeros as well as a constant gain factor, such that the transfer function $H_1(z)$ will describe a real, stable system of degree 3 with $H_1(1) = 1$. Is the system minimum phase?

(b) Change the appropriate parameters of the system described by $H_1(z)$ such that the resulting transfer functions $H_\lambda(z)$ of degree 3 have the property $|H_\lambda(e^{j\omega})| = |H_1(e^{j\omega})|$. Find all functions $H_\lambda(z),\ \lambda = 2, \ldots$ of stable real systems with this property. In order to check your result, calculate and plot $|H_\lambda(e^{j\omega})|$ for all these systems. Use `zplane` to show the pole-zero locations of the different systems. Which one of these systems is minimum phase?

(c) Calculate for the resulting transfer functions $H_\lambda(z),\ \lambda = 1, \ldots$ the phases $\varphi_\lambda(\omega)$, the group delays $\tau_{g\lambda}(\omega)$, the impulse responses $h_\lambda[n]$ and the energy-functions of the impulse responses $w_{h\lambda}[m] = \sum\limits_{n=0}^{m} h_\lambda^2[n]$. Plot the comparable functions in one diagram. Use `subplot` to get a complete description of these systems with 4 subpictures. Verify that the minimum phase system has the smallest unwrapped phase over the interval $[0, \pi]$.

(d) Verify the following statement: Let $\tau_{gM}(\omega)$ be the group delay of a minimum phase system and $\tau_{g\lambda}(\omega)$ the group delays of the nonminimum phase systems with the same $|H(e^{j\omega})|$, then

$$\tau_{gM}(\omega) < \tau_{g\lambda}(\omega),\ \forall\omega,\ \lambda\,. \tag{3.21}$$

Hint: Use `grpdelay(.)` for the calculation of $\tau_{g\lambda}(\omega)$.

(e) Calculate $\int\limits_0^\pi \tau_{g\lambda}(\omega)d\omega$ numerically for all cases. Verify that the values are $\ell\pi/2$, where ℓ depends on the number of zeros outside the unit circle.

Hint: Calculate $\tau_{g\lambda}(\omega)$ at $\omega_k = k{\cdot}\pi/N,\ k = 0 : N{-}1$. Use `sum(.)` for an approximative integration according to the rectangular rule. A value $N = 512$ is recommended.

(f) Excite the minimum phase system as well as one of the others with any input sequence, e.g. with `v=rand(1,100)`. Calculate the running energies $w_y[m]$ and $w_{ym}[m]$ of the corresponding output sequences $y[n]$ and $y_m[n]$ and verify the relation (3.12).

(g) Let $H_M(z)$ be the transfer function of the minimum phase system found in part b. Find the corresponding inverse system described by $H_{Mi}(z) = 1/H(z)$. Calculate its group delay $\tau_{gMi}(\omega)$ and compare it with $\tau_{gM}(\omega)$.

Exercise 3.3: Allpass

The transfer function of a real, stable allpass is partly known by the poles and zeros

$$p_1 = 0.9 \cdot e^{j\pi/4}\,, \quad p_2 = 0.8$$
$$z_1 = 1.5 \cdot e^{j\pi/2}\,, \quad z_2 = 1.25$$

(a) Find additional poles and zeros as well as a constant factor to complete the transfer function such that $|H_A(e^{j\omega})| = 1$ and $H_A(z)$ has the minimum possible order. Calculate and plot the group delay $\tau_g(\omega)$ and the impulse response $h[n]$. Determine the minimum value of the group delay. Prove that the group delay of a stable allpass is never negative.

(b) Find a closed form expression for $\int\limits_0^\pi \tau_g(\omega)d\omega$ in terms of the number of poles and zeros of the allpass. Calculate the integral numerically for the example given above to verify your result.

Exercise 3.4: Autocorrelation sequence of impulse responses

(a) Calculate the autocorrelation sequences $\rho_\lambda[m]$ of the impulse responses $h_\lambda[m]$ of at least two of the systems described by $H_\lambda(z)$, found in exercise 3.2, part b and for the allpass of exercise 3.3. Use the program `acimp`, developed in exercise 1.6, part a. Compare your results with those to be expected.

(b) Given the autocorrelation sequence $\rho[m]$ of the impulse response $h[n]$ of any system. Develop a program for the calculation of the transfer function $H_M(z)$ of the corresponding minimum phase system using the following basic relations [3, ch. 8.1]:

The autocorrelation sequence can be expressed as

$$\rho[m] = g[m] + g[-m]\,, \tag{3.22}$$

where

$$g[m] = \begin{cases} 0.5 \cdot \rho[0], & m = 0 \\ \rho[m], & m > 0 \\ 0 & m < 0\,. \end{cases} \tag{3.23}$$

The z-transform of (3.22) yields

$$R(z) = H(z)H(z^{-1}) = G(z) + G(z^{-1})\,. \tag{3.24}$$

The procedure works as follows:

- step 1: Design a system with the transfer function $G(z) = Z^{-1}\{g[m]\}$ using Prony's method according to exercise 1.3.

- step 2: Calculate the poles and zeros of the rational function $R(z) = G(z) + G(z^{-1})$.
- step 3: The transfer function $H_M(z)$ of the minimum phase system is determined uniquely by the poles p_k of $R(z)$ with $|p_k| < 1$ (being the poles of $G(z)$) and the zeros z_ℓ with $|z_\ell| \leq 1$, where in case of a double zero on the unit circle only one has to be used for $H_M(z)$.
- step 4: Calculate the coefficients a_k and b_ℓ of $H_M(z)$ using the selected poles p_k and zeros z_ℓ (use `zp2tf`, as developed in project 1). The required constant factor b_0 can be determined using the condition

$$|H(1)| = \sqrt{R(1)} = \sqrt{2G(1)}\,. \tag{3.25}$$

Use your program for one of the nonminimum phase systems of exercise 3.2, part b. Compare your result with the transfer function of the minimum phase system of that exercise.

Remark: Selecting roots z_ℓ partly or completely outside the unit circle yields a variety of nonminimum phase system, all with the same $\rho[m]$ and thus the same $|H(e^{j\omega})|^2$.

Exercise 3.5: Linear phase systems

We are given two of the zeros of an FIR system

$$z_1 = 0.9 \cdot e^{j\pi/5}\,,\ z_2 = 1 \cdot e^{-j\pi/2}\,.$$

(a) Find additional zeros such that the resulting transfer function of minimum degree describes a system with the frequency response

1. $H_a(e^{j\omega}) = e^{-j\omega\tau_a} \cdot H_{0a}(e^{j\omega})$,
   where τ_a is constant and $H_{0a}(e^{j\omega})$ is real
2. $H_b(e^{j\omega}) = e^{-j\omega\tau_b} \cdot H_{0b}(e^{j\omega})$,
   where τ_b is constant and $H_{0b}(e^{j\omega})$ is imaginary.

(b) Determine τ_a and τ_b. Calculate and plot the two corresponding impulse responses and the functions $H_{0a}(e^{j\omega})$ and $H_{0b}(e^{j\omega})$.

Exercise 3.6: Separation of a Nonminimum phase System.

The transfer function $H(z)$ of a linear phase system is partly described by the zeros

$$z_1 = 0.9 \cdot e^{j\pi/6}\,;\ z_2 = 0.8 \cdot e^{j\pi/4}$$

(a) Complete the description such that the first value of the impulse response is $h[0] = 1$

(b) Separate $H(z)$ according to

$$H(z) = H_A(z) \cdot H_M(z)\,,$$

where $H_A(z)$ is the transfer function of an allpass and $H_M(z)$ that of a minimum phase FIR system (see (3.10)).

(c) Let $h_A[n]$ and $h_M[n]$ be the impulse responses of the two subsystems and $h[n]$ the overall impulse response, to be expressed as

$$h[n] = h_A[n] * h_M[n] = h_M[n] * h_A[n]$$

Question: What are the lengths of these three impulse responses?

(d) Calculate and plot $h_A[n]$, $h_M[n]$ and $h[n]$ using `comb` and `subplot`. Comment on the result in terms of observability and controllability, taking the different ordering of the subsystems in consideration.

Exercise 3.7: Relation between the real and imaginary part of $H(e^{j\omega})$

(a) Given $P(e^{j\omega}) = \mathrm{Re}\{H(e^{j\omega})\} = c_0 + \sum\limits_{k=1}^{N} c_k \cos k\omega$ of a real causal system. Choose random values for the c_k using `rand('normal'), c = rand(1,N+1)`.

Find $Q(e^{j\omega}) = \mathrm{Im}\{H(e^{j\omega})\}$ and the impulse response $h[n]$. Use $N = 50$. Is the solution unique?

(b) Given $Q(e^{j\omega}) = \sum\limits_{k=1}^{N} c_k \sin k\omega$ of a real causal system, where again the coefficients are chosen with `rand('normal')` and $N = 50$, `c = rand(1,N)`.

1. Find $P(e^{j\omega})$ and the impulse response $h[n]$. Is the solution unique?
2. In case you decided that the solution is not unique modify your result such that $|H(1)| = 1$

(c) Which modifications are necessary, if the system obtained in part b is still causal but not real? Check your answer for $P(e^{j\omega})$ this time with

```
c = rand(1,N+1) + j * rand(1,N+1)
```

Exercise 3.8: Checking the minimum phase property

Given the impulse response $h[n]$ of a length-N FIR-system. We want to develop methods to check, whether or not the system has minimum phase. Furthermore, if the system is found to be nonminimum phase, we want to construct a minimum phase system with the same magnitude $|H(e^{j\omega})|$.

(a) Develop an m-file `minph` based on the roots of

$$H(z) = \sum_{n=0}^{N-1} h[n] z^{-n}$$

executing the following steps

- Calculate the zeros of $H(z)$ with

```
z = roots (h)
```

- Check `abs(z)` > 1

- If `abs(z(i))` > 1 introduce `z(i) = 1/conj(z(i))`
- Scale the transfer function such that the resulting minimum phase system satisfies

$$|H_M(e^{j\omega})| = |H(e^{j\omega})|$$

Apply your program with

```
rand('normal');
h=rand(1,10);
```

(b) Develop an m-file `minphcep` based on the cepstrum c_k as defined in (3.4), which executes the following steps:

- Calculate the c_k approximately as

  ```
  c = ifft(log(fft(h,M))),
  ```

  where $M \gg N$.
- In order to check the causality of the sequence rearrange it according to

  ```
  cr = [c(M/2:M) c(1:M/2)]
  ```

  and plot it for $k = -M/2 : M/2$
- If you find the sequence to be not causal and thus the system to be nonminimum phase find the corresponding minimum phase system described by $H_m(z)$ based on

  ```
  cm = ifft(log(abs(fft(h,M))))
  ```

  with

  ```
  Hm = exp(fft(cm(1:M/2,M)
  ```

  as

  ```
  hm = ifft(Hm).
  ```

Apply your program again with

```
rand('normal');
h=rand(1,10);
```

Exercise 3.9: Testing the lossless property

(a) You are given poles and zeros of a minimum phase system, described by $H_1(z)$

$$p_{1,2} = 0.6e^{\pm j\pi/8}\,, \quad p_3 = 0.5\,,$$
$$z_{1,2} = 0.9e^{\pm j\pi/8}\,, \quad z_3 = 0.75\,.$$

Furthermore we use the allpass, described by $H_A(z) = 1$, having the same poles.

- Determine the corresponding two transfer functions $H_1(z)$ and $H_A(z)$, such that $H_1(1) = H_A(1) = 1$.

- In order to learn about the properties of these systems calculate and plot $|H_1(e^{j\omega})|$ as well as the two group delays τ_{g1} and τ_{ga}.
- Calculate the two output sequences $y_1[n]$ and $y_a[n]$ for an input sequence

  ```
  rand('uniform'); v = [rand(1,30),zeros(1,60)].
  ```

  Plot these sequences using `subplot(2.1x)` and `comb`.
- Calculate the running energies

  $$w_x[m] = \sum_{n=1}^{m} x^2[n] \tag{3.26}$$

  for the sequences $x[n] = v[n]$, $y_1[n]$ and $y_a[n]$ and plot them in one diagram.
- Comment on your results in terms of passivity. Are the systems lossless?

(b) In this part, we construct a system with two outputs according to fig. 2 in order to study the relations given in equations (3.17), (3.14) and (3.18).

You are given the poles of two allpasses of first and second order.[17]

$$\begin{aligned} AP1: &\quad p_1 = 0.1587 \\ AP2: &\quad p_{2,3} = -0.0176 \pm j0.8820\,. \end{aligned}$$

- Determine the two allpass transfer functions $H_{A1}(z)$ and $H_{A2}(z)$.
- Excite the two allpasses with

  ```
  v=[rand(1,30), zeros(1,60)]
  ```

  using `rand ('normal')` and `filter`, yielding $u_1[n]$ and $u_2[n]$. Calculate two output sequences according to

  $$\begin{aligned} y_1[n] &= 0.5(u_1[n] + u_2[n]) \\ y_2[n] &= 0.5(u_1[n] - u_2[n])\,. \end{aligned}$$

- Calculate the running energies for $v[n]$, $y_1[n]$ and $y_2[n]$ according to (3.26) and plot them in one diagram.
- Calculate and plot as well the differences

  $$d[m] = w_v[m] - (w_{y_1}[m] + w_{y_2}[m])\,.$$

  Is the total system with the input signal $v[n]$ and two output signals $y_1[n]$ and $y_2[n]$ passive? Is it lossless?

[17] p_1, p_2, p_3 are the poles of $H(z)$, the transfer function of an elliptic filter found with ellip (3, 1, 15, 0.5) and described by

$$\begin{aligned} b &= [0.3098 \quad 0.4530 \quad 0.4530 \quad 0.3098] \\ a &= [1.0000 \quad -0.1235 \quad 0.7726 \quad -0.1235] \end{aligned}$$

- Determine the two transfer functions

$$\begin{aligned} H_1(z) &= 0.5[H_{A1}(z) + H_{A2}(z)] \\ H_2(z) &= 0.5[H_{A1}(z) - H_{A2}(z)]\,. \end{aligned}$$

  Calculate and plot in one diagram $|H_1(e^{j\omega})|^2$ and $|H_2(e^{j\omega})|^2$.
- Compare the numerator of $H_1(z)$ with the numerator b of the elliptic filter, only the denominator of which was used in this exercise. Can you explain or generalize this remarkable result?
- Calculate the unwrapped phases $b_1(\omega)$ and $b_2(\omega)$ of the two allpasses for $0 \le \omega \le \pi$. Plot

$$\Delta(\omega) = b_2(\omega) - b_1(\omega)\,.$$

  Explain the properties of the structure in terms of this phase difference.
- Calculate and plot

$$\begin{aligned} \left|G_1(e^{j\omega})\right| &= \frac{1}{2}\left|1 + e^{-j\Delta(\omega)}\right| &= \left|\cos\frac{\Delta(\omega)}{2}\right|\;; \\ \left|G_2(e^{j\omega})\right| &= \frac{1}{2}\left|1 - e^{-j\Delta(\omega)}\right| &= \left|\sin\frac{\Delta(\omega)}{2}\right|\;. \end{aligned}$$

  How are $G_{1,2}(e^{j\omega})$ related to $H_{1,2}(e^{j\omega})$?

Project 4: Structures

In this project, different possible structures for the implementation of a desired system will be considered. They will be compared in terms of their complexity or the required number of arithmetic operations, if implemented with MATLAB.

Project Description

As has been mentioned in the description of project 1, there are different algebraic expressions for the transfer function $H(z)$ which yield different elementary structures. The representation of $H(z)$ as quotient of two polynomials (see (1.4)) yields the so-called direct structures, in which the coefficients a_k and b_k are parameters. Essentially two different direct forms are available (see chapt. 6 in [1]). Direct form II is implemented in the MATLAB function `filter`. Obviously it can be used for a FIR-filter as well, i.e. if $a_k = 0, k = 1(1)N$.

Based on the representation (1.10) of the transfer function with the poles p_k and zeros z_ℓ the numerator and denominator polynomials of $H(z)$ can be written as products of polynomials of first or second degree with real coefficients. That yields the representation

$$H(z) = \prod_{\lambda=1}^{L} H_\lambda(z)\,, \tag{4.1}$$

the description of the cascade structure. Its subsystems are characterized by

$$H_\lambda^{(1)}(z) = \frac{b_{0\lambda} + b_{1\lambda}z^{-1}}{1 + a_{1\lambda}z^{-1}} \text{ and } H_\lambda^{(2)}(z) = \frac{b_{0\lambda} + b_{1\lambda}z^{-1} + b_{2\lambda}z^{-2}}{1 + a_{1\lambda}z^{-1} + a_{2\lambda}z^{-2}}\,, \tag{4.2}$$

where $a_{k\lambda}, b_{k\lambda} \in \mathbb{R}$. Since a different pairing of the zeros z_ℓ and poles p_k of $H(z)$ yields different transfer functions $H_\lambda(z)$ and since the ordering of these subsystems can be changed there are many different cascade structures beside the fact that quite a few different implementations can be used for the realization of the subsystems themselves.

The partial fraction expansion, as given in (1.11) for the case of distinct poles leads to

$$H(z) = B_0 + \sum_{\lambda=1}^{L} H_\lambda(z)\,, \tag{4.3}$$

describing the parallel structure, a parallel connection of subsystems, the transfer functions of which are now

$$H_\lambda^{(1)}(z) = \frac{b_{0\lambda}}{1 + a_{1\lambda} z^{-1}}\,;\; H_\lambda^{(2)}(z) = \frac{b_{0\lambda} + b_{1\lambda} z^{-1}}{1 + a_{1\lambda} z^{-1} + a_{2\lambda} z^{-2}}\,. \tag{4.4}$$

Here we do not have the choice between different implementations other than the possibility of choosing different structures for the subsystems.

An appropriate description of all structures is possible using state equations. An equivalent system with the same transfer function but different inside structure can be obtained by introducing a new state vector $\mathbf{q}[n]$ by $\mathbf{x}[n] = \mathbf{T}\mathbf{q}[n]$, where $\mathbf{T}$ is a nonsingular transformation matrix. The procedure will be shown using a system of second order as an example. Furthermore we choose $\mathbf{T}$ such that a so called normal system is obtained, described by a normal matrix $\mathbf{A}_N$, characterized by the property $\mathbf{A}_N^T \mathbf{A}_N = \mathbf{A}_N \mathbf{A}_N^T$.

We start with an implementation of the system in the first direct form (1.3) for the case $N = 2$

$$\mathbf{A} = \begin{bmatrix} -a_1 & 1 \\ -a_2 & 0 \end{bmatrix}, \; \mathbf{b} = \begin{bmatrix} b_1 - b_0 a_1 \\ b_2 - b_0 a_2 \end{bmatrix}, \; \mathbf{c}^T = [1 \quad 0]\,, \quad d = b_0 \tag{4.5}$$

If p and p^* are the eigenvalues of $\mathbf{A}$, the transformation matrix

$$\mathbf{T} = \frac{1}{\mathrm{Im}\{p\}} \begin{bmatrix} 0 & -1 \\ \mathrm{Im}\{p\} & \mathrm{Re}\{p\} \end{bmatrix} \tag{4.6}$$

leads to the normal matrix

$$\mathbf{A}_N = \mathbf{T}^{-1}\mathbf{A}\mathbf{T} = \begin{bmatrix} \mathrm{Re}\{p\} & \mathrm{Im}\{p\} \\ -\mathrm{Im}\{p\} & \mathrm{Re}\{p\} \end{bmatrix} \tag{4.7}$$

and

$$\mathbf{b}_N = \mathbf{T}^{-1}\mathbf{b}\,,\; \mathbf{c}_N^T = \mathbf{c}^T \cdot \mathbf{T}\,,\; d_N = d\,. \tag{4.8}$$

An interesting property of the structure will be considered in exercise 4.2. Furthermore it will be used in chapter 7.

Description of Lattice-Structures and the coupled allpasses including the synthesis procedures follows next week.

Exercise 4.1: Complexity of different implementations

You are given the coefficients of a 7th order lowpass[18]

$$\mathbf{a} = [1 \;\; -.5919 \;\; 2.1696 \;\; -1.1015 \;\; 1.5081 \;\; -0.5823 \;\; 0.3275 \;\; -0.0682]$$
$$\mathbf{b} = [0.0790 \;\; 0.2191 \;\; 0.4421 \;\; 0.5905 \;\; 0.5905 \;\; 0.4421 \;\; 0.2192 \;\; 0.0790]$$

The properties of three different implementations in terms of the required number of arithmetic operations are to be determined.

(a) Direct structure

The function filter yields a solution of

$$y[n] = \sum_{\ell=1}^{M+1} b(\ell)v[n+1-\ell] - \sum_{\ell=1}^{N} a(\ell+1)y[n-\ell]\,.$$

Determine the required number of arithmetic operations (multiplications and additions) for one output value $y[n]$ as a function of M and N.

Verify your result for the example given above with $M = N = 7$.

(b) Cascade Structure

1. Write a program for the transformation of the direct form, given by **a** and **b** into the cascade form described by 4.2
2. Write a program for the execution of the cascade form, where the subsystems are real and implemented by `filter`. (Make sure that the coefficients of your subsystems are actually real.)
3. Apply your programs for the example given above. Determine the required number of arithmetic operations per output sample

(c) Parallel Structure

1. Calculate the coefficients of the parallelform using `residuez` (see exercise 1b in project 1)
2. Write a program for the execution of the parallel form, where the subsystems are real and implemented by `filter`
3. Apply and check the performance of your program corresponding to the above described use of the program for the cascade form.

Exercise 4.2:

You are given the description of a system by

$$\mathbf{A} = \begin{bmatrix} 1.9 & 1 \\ -0.95 & 0 \end{bmatrix} \; ; \; \mathbf{b} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \; ; \; \mathbf{c}^T = [1 \quad 1] \; , \; d = 0\,.$$

[18]Elliptic filter found by `ellip(7,0.1,40,0.5)`

(a) Let the input sequence $v[n] = 0$ and the initial state vector $\mathbf{x}_i[0] = [1 \quad 1]^T$.

1. Calculate the state vector $\mathbf{x}_i[n] = [x_{i1}[n] \; x_{i2}[n]]^T$ and the output sequence $y_i[n]$ for $n = 0 : 100$.

   plot $\mathbf{x}_i[n]$ using `axis('square')` with

   ```
   plot(x1,x2,'o'),
   ```

   plot as well $y_i[n]$ using `comb`.

2. Calculate and plot the stored energy of the system, defined as

   $$w[n] = \mathbf{x}^T[n] \cdot \mathbf{x}[n] \, .$$

   We mention that a non exited system is called passive, if

   $$\Delta w[n+1] = w[n+1] - w[n] \leq 0 \, , \; \forall n \geq 0 \, .$$

   Is the system under test passive in that sense?

(b) Now we assume the initial state to be zero. Calculate the state vector $\mathbf{x}_s[n]$ and the output sequence $y_s[n]$ for an excitation by the stepsequence $u[n]$. Compare the resulting final values $\mathbf{x}_s[100]$ and $y_s[100]$ with those for $n \to \infty$, to be calculated easily out of the state equations. Is there a relation to the transfer function $H(z)$ at a specific point of z?

(c) Transform the system into a normal one according to eq. (4.6) ... (4.8) and repeat part a. Do not forget to transform the initial state.

Note that the state vector $\mathbf{q}[n]$ differs completely from $\mathbf{x}[n]$, while the output sequences in both cases are the same.

Compare the two systems in terms of passivity as defined in part a, point 2.

(d) Repeat part b for the transformed system. What are your final values $\mathbf{q}_s[100]$ and $y_s[100]$ in this case. Compare them with those to be expected.

Chapter 6

Stochastic Signals

January 17, 2007

Overview

The study of random signals and methods of power spectrum estimation is a rather challenging discipline because the level of mathematics needed for a rigorous presentation of stochastic processes is high. On the other hand, many of the algorithms used to perform spectrum estimation are relatively easy to implement in a language such as MATLAB with its vector and matrix operations and its random number generator. Furthermore, an understanding of these algorithms and their performance is often gained through simulation. The signals involved are random, so their mathematical description is based on probability distributions. However, in actual processing only one member signal out of the stochastic process is manipulated. Therefore, a simulation should capture the essential behavior of the algorithm through the processing of a few signals. A thorough simulation must be a Monte-Carlo simulation which demands many runs of the algorithm on different random signals from the same underlying process.

The objective of this chapter is to present the essential information about parameter estimation for a stochastic process and relate the theoretical formulae for quantities such as bias and variance. Initially, we consider the estimation of simple quantities such as the mean, variance, and pdf of a random signal. We also examine, by means of simulation, properties of the random signal, such as ergodicity and stationarity. The rest of this chapter is devoted to power spectrum estimation. First, methods based on the FFT are covered in detail. This leads to an understanding of the Welch-Bartlett method which is the most widely used technique based on the FFT. Then, the maximum entropy method (MEM) and other techniques of "modern" spectrum estimation are treated.

In the study of spectrum estimation, we concentrate on the implementation of several representative methods, and then on ways to characterize their performance. In order to show their merits as well as their deficiencies, we use as an example a random process with known power density spectrum, generated by passing white noise through a linear system with known transfer function. Thus, in our experiments, we can compare to the correct result and can express the observed deviations in terms of this known spectrum. In the packet on *FFT Spectrum Estimation,* the data window introduces a fundamental resolution limit due to the "uncertainty principle" of Fourier analysis. One project considers the problem of resolving two closely spaced sinusoids with different windows and at different SNR's.

Obviously, methods that circumvent the resolution limit of Fourier analysis are of interest. A number of such methods have been developed and popularized over the past twenty years, among the most well known is MEM. In the packet on *Modern Spectrum Estimation,* the projects will treat MEM and some related methods based on linear prediction. The important role of the all-pole spectrum, and of pole-zero models in general, will be stressed. Finally, the Pisarenko harmonic decomposition is introduced, along with practical estimation methods based on eigenvalues and eigenvectors, e.g., MUSIC. A key concept in this area is the idea of a signal-noise subspace representation through eigenvectors. A complete study of these methods and their application to signal analysis and spectrum estimation would easily take another entire volume, so we are content to introduce some of the well-known methods. Then we can use MATLAB to implement each one and to do a simulation to gauge their relative performance.

Background Reading

There are quite a few books on random processes and random variables, beside the classic one by Papoulis [**?**], e.g., those by Leon-Garcia [**?**] and Scharf [**?**]. A brief introduction to random signals is also given as an appendix in both [8] and [**?**], as well as Section 2.10 in [8].

Some books are devoted to spectral estimation, especially the books by Marple [**?**] and Kay [**?**]. Others have rather comprehensive chapters on the subject: Chapter 13 in [**?**], Chapters 11 and 12 of [**?**], Chapter 2 in [20] and sections 11.5–11.7 in [8]. Chapters 4, 5 and 8 in [**?**], as well as Chapters 1, 2 and 6 in [20], also address a variety of topics related to spectrum estimation and modeling of random signals. A useful algorithm for estimating the autocorrelation sequence is described in [**?**]. Finally, two reprint collections have been published by IEEE Press of the topic of Modern Spectrum Estimation [**?**, **?**].

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{Stochastic Signals}}$

Stochastic Signals

Overview

The signals we have to deal with in practice are, in most cases, not deterministic. A speech signal, for example, cannot be described by an equation. Nonetheless, it has certain characteristics which distinguish it from, say, a television signal. In fact, almost all signals that we have to handle in communications, and in many other fields of engineering and science, are of a stochastic nature (also called random).

A stochastic signal has two facets: at a fixed time instant its value is a random variable, and as a function of time the random variables might be interrelated. The definition of the random signal is done via its statistical properties: probability density function, joint density function, mean, autocorrelation, etc. In a theoretical problem, these quantitative descriptions apply to the ensemble of all realizations of the particular random process. They are deterministic functions, well behaved in the mathematical sense. However, in a practical problem they must be estimated, using measurements on a finite set of data taken from observations of the random process. Since these estimates are actually formed from random variables, they are themselves random variables. Thus we can only make probabilistic statements about the closeness of an estimated value to the true values, e.g., 95% confidence intervals.

This set of projects deals with the description and processing of stochastic signals, mainly under the assumptions of stationarity and ergodicity. Furthermore, we investigate how the ensemble averages are influenced, when a stochastic signal is processed through a linear filter or a nonlinear mapping. In most of the projects, estimates are computed in MATLAB via averaging—either time averages or ensemble averages. These estimates must then be compared to the true values known from theory.

An understanding of these results should open the way to designing systems that generate stochastic signals with desired properties. For example, a parametric description of a given stochastic signal can be obtained as the output of a fixed linear filter excited by so-called white noise, an uncorrelated stochastic signal. Related to this representation is the problem of prediction and decorrelation, i.e., the design of a system whose output sequence is approximately an advanced version of its input, or such that the correlated input sequence is transformed into an output that is a white noise sequence.

As in the case of deterministic signals, a description in both the time domain and in the frequency domain is of interest. The direct Fourier transform of the random signal is not a useful quantity, but the transform of the autocorrelation function is—it is called the *power spectrum*. An introduction into methods for spectral estimation will be given the following two packets.

Background Reading

Many books have been published on the subject of random processes and random variables. A classic text used for many years was written by Papoulis [**?**]. More recent texts are those by Leon-Garcia [**?**] and Scharf [**?**]. A brief introduction to random signals is also given as an appendix in [**?**], as well as Section 2.10 and Appendix A in [8].

Project 1: Random Variables

In this project, the elementary properties of random variables will be introduced. The exercises concentrate on estimating the mean, variance and probability density function of the random variables.

A random variable (RV) is described by a *probability density function* (pdf):

$$p_{\mathbf{v}}(v) = \frac{d}{dv} P_{\mathbf{v}}(v) \qquad \text{PROBABILITY DENSITY (pdf)} \qquad (1.1)$$

$$\text{where} \quad P_{\mathbf{v}}(v) = \text{Probability}[\mathbf{v} \le v] \qquad \text{PROB. DISTRIBUTION} \qquad (1.2)$$

Here $\mathbf{v}$ denotes the random variable while v is a particular value of $\mathbf{v}$. The pdf $p_{\mathbf{v}}(v)$ can be interpreted as giving the probability that the RV $\mathbf{v}$ will be equal to the value v.

In many cases, only certain ensemble averages of the RV need be computed—usually the mean and variance:

$$m_{\mathbf{v}} = \mathcal{E}\{\mathbf{v}\} = \int_{-\infty}^{+\infty} v \, p_{\mathbf{v}}(v) \, dv \ , \qquad (1.3)$$

$$\sigma_{\mathbf{v}}^2 = \mathcal{E}\{|\mathbf{v} - m_{\mathbf{v}}|^2\} = \int_{-\infty}^{+\infty} (v - m_{\mathbf{v}})^2 \, p_{\mathbf{v}}(v) \, dv \ . \qquad (1.4)$$

These ensemble averages are *constants*, but they cannot be determined exactly from samples of the RV. In these exercises, samples of the RV will be created by a pseudo-random number generator, the properties of which are known with sufficient accuracy. The pdf, mean and variance will be *estimated* from a finite number of these samples and then compared with the theoretical values.

Hints

The MATLAB function `rand(M,N)` will generate an $M \times N$ matrix of pseudo-random numbers. This function can produce random numbers with two different pdf's: uniform or Gaussian (also called normal).[19] The default at startup is uniform, but the pdf type can be set by invoking `rand('uniform')` or `rand('normal')` prior to calling `rand(M,N)`.

WARNING: Since these exercises require the estimation of ensemble properties via averaging, very large sample sets must be used. This can stress the memory capabilities of MATLAB on some machines and in the student version. If your machine can accommodate large vectors, use very large lengths (e.g., 8000–10000) so that your results will be nearly perfect. If you are forced to have short vectors, then recognize that the small sample size may cause the answers to deviate noticeably from the corresponding theoretical values.

The MATLAB function `hist(x,nbins)` will compute and plot a histogram (as a bar chart) for a vector `x` of pseudo-random numbers. The default for the number of bins is 10; otherwise, `nbins` can be specified.

[19] In MATLAB version 4.0, a new function `randn()` is used to generate Gaussian random numbers.

Exercise 1.1: Uniform pdf

Generate a long vector of samples of a uniform random variable; use at least several thousand samples.

(a) Use `hist`, `mean`, and `std` to make estimates of the pdf, $m_{\mathbf{v}}$, and $\sigma_{\mathbf{v}}$, respectively. Note that the histogram must be normalized to have a total area equal to one, if it is to be a legitimate estimate of the pdf.

(b) The MATLAB `rand` function produces a uniform density in the range 0 to 1. Thus it is possible to derive the theoretical values for the mean and variance ($\sigma^2_{\mathbf{v}}$). Determine these theoretical values and compare to the estimated ones.

(c) Repeat the numerical experiment of part (a) several times to see that the estimated values are not always the same. However, you should observe that the estimated values hover about the true values.

Exercise 1.2: Gaussian pdf

Generate a long vector of samples of a Gaussian random variable; use at least several thousand samples.

(a) As in Exercise 1.1, compute estimates of the pdf, $m_{\mathbf{v}}$, and $\sigma^2_{\mathbf{v}}$. Compare the mean and variance to the true values; and repeat the experiment several times to observe the variability of these two quantities.

(b) The histogram plot should approximate the true pdf—in this case, the bell shape of a Gaussian, $\mathcal{N}(m_{\mathbf{v}}, \sigma^2_{\mathbf{v}})$. The formula for the Gaussian pdf is known:

$$p_{\mathbf{v}}(v) = \frac{1}{\sigma_{\mathbf{v}} \cdot \sqrt{2\pi}} \cdot e^{-(v-m_{\mathbf{v}})^2/2\sigma^2_{\mathbf{v}}}. \tag{1.5}$$

On a plot of the scaled histogram, superimpose the function for $p_{\mathbf{v}}(v)$. See `help plot` or `help hold` for ways to put several curves on one plot. Be careful to normalize the histogram and Gaussian pdf to the same vertical scale. Experiment with the number of bins and the length of the vector of random samples to get a reasonably good match.

Exercise 1.3: Average 12 Uniform RV's to Create a Gaussian

On a computer it is trivial to generate uniformly distributed random numbers. Other pdf's are usually created by applying non-linear transformations to the uniform density (see Project 4). For the Gaussian case, there is a another simple approach based on averaging. This method relies on the central limit theorem [?] which states (loosely) that averaging independent RV's, irrespective of their pdf, will give a new RV whose pdf tends to a Gaussian (see Project 5, Exercise 5.2). This theorem involves a limiting process, but a practical example of the theorem can be demonstrated by averaging a relatively small number of uniform RV's; twelve, in fact, will be sufficient.

(a) Use `rand(12,N)` to create a $12 \times N$ matrix of uniform random variables. Take the average of each column of the matrix; this can be done cleverly by using the `mean` function which does each column separately. The result is a new random vector of length N.

(b) Estimate the pdf of this new random variable from its histogram. Also, estimate its mean and variance.

(c) Since the default random number generator in MATLAB is a non-zero mean uniform RV, derive the theoretical values for the mean and variance of the approximate Gaussian.

(d) Compare the estimated values from part(c) to the theoretical ones. Make a plot of the histogram with the theoretical Gaussian pdf superimposed.

Exercise 1.4: Independent RV's

The pseudo-random number generator (`rand`) within MATLAB can be invoked twice to produce samples of two distinct random variables. The interaction of these two RV's is described by their joint pdf, which is a function of two variables. Suppose that we call the two RV's $\mathbf{v}_1$ and $\mathbf{v}_2$. Then the joint pdf at (x, y) is the probability that $\mathbf{v}_1$ equals x <u>and</u> $\mathbf{v}_2$ equals y. For example, the 2-D Gaussian pdf can be written as:

$$p_{\mathbf{v}_1\mathbf{v}_2}(x,y) = \frac{1}{2\pi\sqrt{|\mathbf{C}|}} \cdot e^{-\frac{1}{2}(\mathbf{v}-\mathbf{m}_\mathbf{v})^T\mathbf{C}^{-1}(\mathbf{v}-\mathbf{m}_\mathbf{v})}, \tag{1.6}$$

where the vector $\mathbf{v}$ is $[\,x \quad y\,]^T$, while $\mathbf{m}_\mathbf{v} = [m_{\mathbf{v}_1}, m_{\mathbf{v}_2}]^T$, and $\mathbf{C}$ is the covariance matrix of the two zero-mean RV's, $\tilde{\mathbf{v}}_i = \mathbf{v}_i - m_{\mathbf{v}_i}$

$$\mathbf{C} = \mathcal{E}\left\{\begin{bmatrix}\tilde{\mathbf{v}}_1\\ \tilde{\mathbf{v}}_2\end{bmatrix}[\,\tilde{\mathbf{v}}_1 \quad \tilde{\mathbf{v}}_2\,]\right\} = \begin{bmatrix}\mathcal{E}\{\tilde{\mathbf{v}}_1^2\} & \mathcal{E}\{\tilde{\mathbf{v}}_1\tilde{\mathbf{v}}_2\}\\ \mathcal{E}\{\tilde{\mathbf{v}}_2\tilde{\mathbf{v}}_1\} & \mathcal{E}\{\tilde{\mathbf{v}}_2^2\}\end{bmatrix}. \tag{1.7}$$

The covariance matrix is always symmetric and positive semi-definite.

There are two ways to estimate the joint pdf: compute a 2-D histogram, or assume the pdf is Gaussian and estimate the covariance matrix for use in (1.6). To test these methods generate two data vectors containing Gaussian RV's. Generate a few thousand points of each. Make them both zero-mean, but set the variance of the first equal to 1 and the variance of the other to 3.

(a) Derive a mathematical formula for the joint pdf, which is a 2-D Gaussian. See `help meshdom` to generate the (x, y) domain for computing and plotting this formula. Plot the level curves (via `contour`) of the joint Gaussian pdf, and observe that they are elliptical in shape.

(b) Compute an estimate of the covariance matrix (1.7) by taking averages of v_1^2, v_2^2, and v_1v_2. Compare this estimate to the true covariance matrix. Plot the 2-D Gaussian function based on this covariance estimate and compare its level curves to those found in part (a).

(c) Write a MATLAB function (called `hist2`) to compute the 2-D histogram for a pair of vectors. Include an argument to specify the number of bins. Take advantage of the existing 1-D histogram M-file `hist` and also the `find` function to compute the output histogram matrix one column (or row) at a time. If your function is too slow when processing long data vectors, analyze the code in the M-file `hist` to see if you can find ways to improve your M-file.

(d) Use `hist2` to estimate the pdf directly from the data. Plot this estimated pdf on the same graph with the true pdf. Use a `contour` plot, but plot only a few level curves for each pdf so that it will be easy to compare.

(e) Since these two RV's were generated independently, they should be uncorrelated. The definition of *uncorrelated* is that the 2-D pdf will factor into the product of two 1-D pdf's. This implies that the expected value of the product $\mathbf{v}_1\mathbf{v}_2$ is the product of the expected values, which for the zero-mean case would be zero. Verify that $\mathbf{v}_1$ and $\mathbf{v}_2$ are uncorrelated by doing the following: Estimate the 1-D pdf's of $\mathbf{v}_1$ and $\mathbf{v}_2$ separately, and multiply them together to produce another estimate of the 2-D pdf. Plot this separable estimate versus the true pdf using a contour plot.

Exercise 1.5: Joint pdf for Correlated RV's

In this exercise, the joint pdf of two *correlated* random variables will be computed. Generate two Gaussian random vectors, each containing several thousand samples. Both should be zero-mean RV's with variance equal to 1 and 2, respectively. Form two new RV's by taking the sum and difference of the original RV's; these new RV's will be used for the tests in this exercise.

(a) Determine the theoretical form of the joint Gaussian pdf. Find the exact entries in the covariance matrix.

(b) Make a mesh plot of this multivariate Gaussian from its functional form. Use `meshdom` to generate the (x, y) domain for the calculation and the plot.

(c) Estimate the entries in the 2-D covariance matrix from the data vectors, and plot this estimated pdf. Compare the estimated and true pdf's by making a contour plot with a few level curves from each one.

(d) Estimate the pdf by computing the 2-D histogram. Make a contour plot (and a mesh plot) of the histogram, and compare it to the true pdf. In this case, the two RV's are correlated, so you should verify that the pdf will not factor into the product of two 1-D pdf's.

Project 2: Non-stationary, Stationary, and Ergodic Random Processes

Now we consider a random process $\mathbf{v}_n$ characterized by a particular rule, according to which its different members are generated. A random process, which is also called a stochastic process, is an ensemble of time signals, with the additional characterization that its value v_n at the time instant n is a random variable. In the most general case, the pdf of the signal value might be different for each index n, but the more common case is a *stationary* process where the pdf remains the same for all n. In addition, for a stationary process, the joint pdf between signal values at n and m will depend only on the difference $n - m$.

Since the underlying idea of processing stochastic signals is to learn something about the pdf(s) that define the process, an important problem for stochastic signal processing is how to do that estimation from one member of the stochastic process. If we only have one member from the ensemble of signals, then we cannot average across the ensemble as was done in the random variable case. Instead, we must operate under the assumption that time averages along the one recorded signal will be sufficient to learn about the pdf's. The assumption that allows us to take this approach is called *ergodicity*, which states loosely that "time averages will converge to ensemble averages." An ergodic process must be stationary because it would never be possible to estimate a time-varying pdf from just one signal.

A stochastic process $\mathbf{v}_n$, is characterized by its (time-dependent) pdf $p_{\mathbf{v}_n}(v_n, n)$, as well as the joint pdf between RV's at the time instants n and $n+m$, and all higher-order pdf's. If the pdf's do not depend on the definition of the time origin, the process is *stationary*, otherwise, it is called a *nonstationary process*.

Usually we are most interested in measuring the important *ensemble averages* which include the mean and variance:

$$m_{\mathbf{v}_n}[n] = \mathcal{E}\{\mathbf{v}_n\} = \int_{-\infty}^{+\infty} v\, p_{\mathbf{v}_n}(v,n)dv \qquad \text{(MEAN)} \tag{2.1}$$

$$\sigma^2_{\mathbf{v}_n}[n] = \mathcal{E}\{|(\mathbf{v}_n - m_{\mathbf{v}_n})|^2\} \qquad \text{(VARIANCE)}. \tag{2.2}$$

In case of a *stationary process* the time dependence will disappear, so that the mean $m_\mathbf{v}$ and variance $\sigma^2_\mathbf{v}$ become constants as in (1.3) and (1.4).

So far we have considered the whole process $\mathbf{v}_n$, consisting of an infinite number of individual signals $v_\lambda[n]$, all generated according to the same rule. For any one of these sequences, averages in time-direction can be defined, e.g., the time-averaged mean:

$$\langle v_\lambda \rangle = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} v_\lambda[n] \tag{2.3}$$

and the corresponding time-averaged variance

$$\left\langle |v_\lambda - <v_\lambda>|^2 \right\rangle = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{+N} |v_\lambda[n] - \langle v_\lambda \rangle|^2. \tag{2.4}$$

In general, these time-averaged quantities depend on the individual sequence $v_\lambda[n]$. But if $v_\lambda[n]$ is a member of an ergodic process, then the time averages do not depend on λ and are equal to the corresponding ensemble averages:

$$\langle v[n] \rangle = \mathcal{E}\{\mathbf{v}_n\} = m_\mathbf{v}, \tag{2.5}$$

$$\langle |v[n] - m_\nu|^2 \rangle = \mathcal{E}\{|\mathbf{v}_n - m_\mathbf{v}|^2\} = \sigma^2_\mathbf{v}. \tag{2.6}$$

Thus, for an *ergodic* process, one individual sequence $v_\lambda[n]$ is assumed to be representative of the whole process.

In this project, we first want to show the difference between a nonstationary and a stationary process. A second aim is to give an example of a process that is stationary but non ergodic and contrast it with an ergodic process.

Hints

The exercises below will examine properties of the signals from the following three random processes. Each one should be made into a MATLAB program `rp*.m` that will create a matrix of size $M \times N$ containing random numbers. Make sure that the files are placed on the path where MATLAB can find them.

```
function v = rp1(M,N);     %<<------- RANDOM PROCESS #1
rand('uniform');
a = 0.02;
b = 5;
Mc = ones(M,1)*b*sin((1:N)*pi/N);
Ac = a*ones(M,1)*[1:N];
v = (rand(M,N)-0.5).*Mc + Ac;

function v = rp2(M,N);     %<<------- RANDOM PROCESS #2
rand('uniform');
Ar = rand(M,1)*ones(1,N);
Mr = rand(M,1)*ones(1,N);
v = (rand(M,N)-0.5).*Mr + Ar;

function v = rp3(M,N);     %<<------- RANDOM PROCESS #3
rand('uniform');
a = 0.5;
m = 3;
v = (rand(M,N)-0.5)*m + a;
```

Exercise 2.1: Stationary or Ergodic?

First we can visualize different members of the three processes in the time domain, in order to get a rough idea about stationarity and ergodicity. For stationarity, the issue is whether or not certain properties change with time; for ergodicity, the issue is whether one member of a stationary process is representative of the entire process.

Generate four members of each process of length 100 ($M = 4$, $N = 100$) and display them with `subplot`. Decide by inspection of the 4 representative signals, whether each process is ergodic and/or stationary.

Exercise 2.2: Expected Values via Ensemble Averages

Compute the ensemble mean and standard deviation for each of the three processes, and plot versus time n. This can be done by generating many signals for each process, so let $M = 80$ and $N = 100$. Use the MATLAB functions `mean(·)` and `std(·)` to approximate the mean and standard deviation. From the plot of these quantities versus time decide (again) about the stationarity of the processes.

Remark: The ensemble average, in the case of `mean`, is a sum across λ for each n:

$$\frac{1}{M}\sum_{\lambda=1}^{M} u_\lambda[n] = \hat{m}_{\mathbf{u}_n}[n] \;\approx\; m_{\mathbf{u}_n}[n] = \mathcal{E}\{\mathbf{u}_n\}.$$

Likewise, for the ensemble average of the variance we must compute a sum of squares, after removing the mean.

$$\frac{1}{M}\sum_{\lambda=1}^{M}(u_\lambda[n]-\hat{m}_{\mathbf{u}_n}[n])^2=\hat{\sigma}^2_{\mathbf{u}_n}[n] \;\approx\; \sigma^2_{\mathbf{u}_n}[n]=\mathcal{E}\{|\mathbf{u}_n-m_{\mathbf{u}_n}[n]|^2\}.$$

Exercise 2.3: Expected Values via Time Averages

Measure approximately the time averages by calculating the time-averaged means $\langle u_\lambda\rangle$, $\lambda=1,2,3,4$, for four different members of each process.

$$\begin{aligned}\frac{1}{N}\sum_{n=0}^{N-1}u_\lambda[n] &\approx \langle u_\lambda\rangle,\\ \frac{1}{N-1}\sum_{n=0}^{N-1}(u_\lambda[n]-\hat{m}_{u\lambda})^2 &\approx \left\langle|u_\lambda-\langle u_\lambda\rangle|^2\right\rangle.\end{aligned}$$

Use $M=4$ and $N=1000$ when generating the signals. Decide (again) which of the processes are ergodic; use the results from the previous exercise on ensemble averages, if necessary.

NOTE: strictly speaking, the time average requires a limit, $N\to\infty$, but the large signal length $N=1000$ should be sufficient to approach the limiting value. Also, a nonstationary process cannot be ergodic, so the time averages will be useless in that case.

Exercise 2.4: Theoretical Calculations

Analyze the MATLAB code in the functions `rp*.m` given above, and write the mathematical description of each stochastic process. If possible, determine a formula for the underlying pdf for each process. Decide on that basis whether the processes are ergodic and/or stationary. Calculate the theoretical mean, $m_{\mathbf{v}_n}[n]$, and variance, $\sigma^2_{\mathbf{v}_n}[n]$, for each. Then compare your results with those obtained by the measurements.

Exercise 2.5: An Ergodic Process

For an ergodic process, the pdf can be estimated via time averaging.

(a) For this purpose write a function M-file `pdf` using the available MATLAB function `hist`. Test your function for the simple case of a stationary process with a Gaussian pdf: `rand('normal'); v = rand(1,N);`. Pick the length N to be 100 and 1000.

(b) For the processes `rp*.m` that are ergodic, determine approximately their pdf using your `pdf` M-file. Do the measurement for $N=100, 1000$ and 8000. Plot the three estimated pdf's and the theoretical pdf using `subplot`. Compare the results and comment on the convergence as N gets large.

(c) Is it possible to measure the pdf of a nonstationary process with your function `pdf`? Comment on the difficulties that would be encountered in this case.

Project 3: Influence of a Linear System on a Stochastic Process

In this project, we consider the influence of a linear system on the properties of an ergodic random process. Especially, we investigate how the autocorrelation sequence, the mean, and the pdf of a process are changed by filtering. The pdf, $p_{\mathbf{v}_n}(v)$, cannot depend on n, since we are assuming the process to be ergodic and thus stationary.

First, we recall some important definitions. The ergodic random process, $\mathbf{v}_n$, is described by its mean $m_{\mathbf{v}_n} = \mathcal{E}\{\mathbf{v}_n\}$ and its autocorrelation sequence:

$$\phi_{\mathbf{vv}}[m] = \mathcal{E}\{\mathbf{v}_{n+m}\mathbf{v}_n^*\}. \tag{3.1}$$

The variance $\sigma^2_{\mathbf{v}_n}$ is contained in the value at $m = 0$, because

$$\phi_{\mathbf{vv}}(0) = \sigma^2_{\mathbf{v}_n} + m^2_{\mathbf{v}_n}. \tag{3.2}$$

The Fourier transform of $\phi_{\mathbf{vv}}[m]$ yields a function called the *power density spectrum*

$$\Phi_{\mathbf{vv}}(e^{j\omega}) = \sum_{m=-\infty}^{+\infty} \phi_{\mathbf{vv}}[m]e^{-j\omega m}. \tag{3.3}$$

Every linear time-invariant system can be described by its impulse response $h[n]$, and its transfer function

$$H(z) = \sum_{n=-\infty}^{\infty} h[n]z^{-n} \ . \tag{3.4}$$

When excited by the input $v[n]$, its output sequence $y[n]$ will be a member of a random process $\mathbf{y}_n$ whose mean is multiplied by the DC gain of the filter

$$m_{\mathbf{y}_n} = m_{\mathbf{v}_n} \cdot H(e^{j0}) \tag{3.5}$$

and whose autocorrelation sequence is given by

$$\phi_{\mathbf{yy}}[m] = \phi_{\mathbf{vv}}[m] * \rho[m] = \phi_{\mathbf{vv}}[m] * \left(\sum_{n=-\infty}^{\infty} h[n] \cdot h[n+m] \right), \tag{3.6}$$

where $\rho[m] = h[m] * h[-m]$ is the (deterministic) autocorrelation sequence of the impulse response. The power density spectrum of the output process is

$$\Phi_{\mathbf{yy}}(e^{j\omega}) = \Phi_{\mathbf{vv}}(e^{j\omega}) \cdot |H(e^{j\omega})|^2. \tag{3.7}$$

Furthermore, the *cross-correlation* of the input and output sequences is often used to identify the impulse response. We obtain

$$\phi_{\mathbf{vy}}[m] = \phi_{\mathbf{vv}}[m] * h[m]. \tag{3.8}$$

Finally, if we consider the pdf of the output process $\mathbf{y}_n$, a simple answer is possible in two cases:

1. If the input process $\mathbf{v}_n$ is normally distributed, then the output process will be normally distributed as well. This fact is independent of the autocorrelation sequence $\phi_{\mathbf{vv}}[m]$ and the impulse response $h[n]$ of the system.

2. If the consecutive values of $\mathbf{v}_n$ are statistically independent, i.e. if its autocorrelation sequence is $\phi_{\mathbf{vv}}[m] = \sigma^2_{\mathbf{v}_n} \cdot \delta[m]$ and if the impulse response is sufficiently long, the output sequence will be approximately normally distributed as well. This result does not depend on the pdf or the impulse response. It is essentially a restatement of the central limit theorem (see Exercises 1.3 and 5.2).

All the relations above are based on the assumption that the output process $\mathbf{y}_n$ is ergodic. But if the system is causal and we start the excitation at $n = 0$ with $v[n]$, being a segment of the ergodic process $\mathbf{v}_n$, the corresponding output sequence $y[n]$ will be

$$y[n] = h[n] * v[n] = \sum_{k=0}^{\infty} h[k]v[n-k]. \tag{3.9}$$

This sequence $y[n]$ is obviously causal and thus cannot be a member of a stationary process. However, if we assume that the transient time of the system is approximately of finite length, i.e., $h[n] \approx 0$ for $n > L$, the output process will appear stationary for $n > L$.

In order to be more specific, we consider the mean and the variance of the output process *during the transient time.*

$$\begin{aligned} m_{\mathbf{y}_n}[n] = &\ \mathcal{E}\{\mathbf{y}_n\} = \mathcal{E}\left\{\sum_{k=0}^{n} h[k]v[n-k]\right\} \\ = &\ m_{\mathbf{v}_n} \sum_{k=0}^{n} h[k] = m_{\mathbf{v}_n} \cdot r[n], \end{aligned} \tag{3.10}$$

where $r[n]$ is the step response of the system.

A similar result for the variance holds when the input sequence consists of statistically independent values (white noise):

$$\sigma^2_{\mathbf{y}_n}[n] = \sigma^2_{\mathbf{v}_n} \cdot \sum_{k=0}^{n} |h[k]|^2. \tag{3.11}$$

Hints

In the following exercises, we will study three different filters and their impact on processing random input signals generated via `rand`. The linear systems are described by the coefficients of their rational transfer functions.

```
filter 1:     b1 = [ 0.3   0 ];
              a1 = [ 1 -0.8 ]

filter 2:     b2 = 0.06  *  [1 2 1];
              a2 = [ 1 -1.3  0.845 ];

filter 3:     b3 = [ 0.845 -1.3  1 ];
 (all-pass)   a3 = fliplr(b3);
```

Thus the MATLAB function `filter` can be used for their implementation; and the function `freqz` can be used to compute their frequency responses.

The required estimation of the auto- and cross-correlation sequences can be done with the MATLAB functions `acf` and `ccf`, respectively.[20] These programs apply the method proposed by C.M. Rader [?]. According to (3.3) the power density spectrum can be calculated approximately by applying `fft` on the measured autocorrelation sequence. More will be said about power spectrum estimation in the packet on *FFT Spectrum Estimation.*

Exercise 3.1: Sample Autocorrelation of White Noise

The `rand` function produces statistically independent samples. Generate segments of the two random input sequences via:

```
N = 5000;
rand ('uniform');
v1 = sqrt(12) * (rand(1,N) - 0.5);   %<--- zero mean

rand ('normal');    %<---- Gaussian: mean = 0, var = 1
v2 = rand(1,N);
```

(a) Call these signals $v_1[n]$ and $v_2[n]$. Determine the variance of $v_1[n]$. Compute and plot 64 values of their autocorrelation sequences using `[ phi, lags] = acf(v, 64)`. The second output `lags` gives the domain over which the autocorrelation sequence was computed.

(b) How closely do they match the true autocorrelation function, expected from theoretical considerations?

Exercise 3.2: Filtered White Noise

The signals $v_1[n]$ and $v_2[n]$ are to be used as inputs to a first-order filter.

(a) Suppose that the impulse response of the filter is $h[n] = b\,a^n,\ n \geq 0,\ |a| < 1$. Derive the theoretical autocorrelation function, and then compute the output autocorrelation sequences for both inputs through filter #1, where $a = 0.8$ and $b = 0.3$. Compare both calculated results to the theory and explain any significant differences.

(b) Estimate the pdf of the output when the input is uniform, $v_1[n]$. Repeat for the Gaussian input, $v_2[n]$. Explain why both pdf's are nearly the same.

(c) Repeat part (a) for a first-order all-pass filter.

$$H(z) = \frac{z^{-1} - a^*}{1 - az^{-1}}$$

Derive the theoretical autocorrelation function so that your result is applicable to an all-pass with any number of poles. Is there a difference to be expected for the two different input signals?

[20]These special M-files are described in Appendix A.

Exercise 3.3: Measured Autocorrelation

This exercise demonstrates the effect of a filter on the autocorrelation function.

(a) Excite filter #2 with both white noise signals: $v_1[n]$ and $v_2[n]$. Call the resulting outputs $y_{21}[n]$ and $y_{22}[n]$.

(b) Compute and display the histograms of both output sequences, $y_{2i}[n]$. Be careful to use data only from the stationary part of the signals. Explain how the impulse response $h_2[n]$ can be used to estimate the length of the transient.

(c) Measure the autocorrelation sequences of the stationary part of the two output sequences $y_{2i}[n]$. Display both input and output correlation functions using a four panel `subplot`.

(d) Measure the variances of all four signals (two inputs and two outputs) and compare them with the results of theoretical considerations.

(e) Excite filter #3 with $v_1[n]$ and $v_2[n]$. Measure the autocorrelation sequences and the histograms of the two output sequences $y_{3i}[n]$. Display your results and explain the similarities as well as the differences.

(f) Furthermore, measure the cross-correlation sequence between the input and output sequences for filters #2 and #3. Be careful to use only the stationary part of the output signal. Use the Gaussian input first, but then try the uniform input, $v_1[n]$.

(g) From the cross-correlation estimates, compute an estimate of the impulse responses of systems #2 and #3 and then plot with `comb`. Compute $h_2[n]$ and $h_3[n]$, the true impulse responses. Plot the true $h_i[n]$ as a dashed line envelope to compare with the estimate. Use `subplot(21x)` to create a two-panel display with both $h_2[n]$ and $h_3[n]$. Does the answer depend strongly on whether the input is uniform, $v_1[n]$, or Gaussian, $v_2[n]$?

Exercise 3.4: Transients

In this exercise, we investigate the transient behavior of a system, when excited by a stochastic input signal. We use filter #2 as an example and excite it by signals of length 50 out of the normally distributed process generated by

```
rand('normal');
v = alfa* rand(160,50) + beta;
```

(a) Choose α and β such that the variance $\sigma_{\mathbf{v}}^2 = 2$ and the mean $m_{\mathbf{v}} = 1$.

(b) Calculate and display the output sequences $y_\lambda[n]$ for four different input sequences $v_\lambda[n]$.

(c) Generate a 160×50 matrix of output sequences using `filter` inside an appropriate `for` loop. Compute estimates of the mean and variance as functions of n by averaging over the ensemble of $M = 160$ signals.

(d) Now calculate the true impulse response $h[n]$, the step response $r[n]$ and the sequence $w[n] = \sigma^2_{\mathbf{v}_n} \cdot \sum_{k=0}^{n} h^2[k]$. Display these results for $0 < n \leq 50$, superimposed on the estimates obtained above. Compare the results with those to be expected according to (3.10) and (3.11) and comment on the relationship between $r[n]$, $w[n]$ and the transient nature of the stochastic output signals.

Exercise 3.5: Moving Averages

The moving average operator can be used to determine approximately the time-averaged mean and variance of a random sequence out of a stationary process. The N-point moving average operator is:

$$y_1[n] = \frac{1}{N} \sum_{m=n-N+1}^{n} v[m].$$

The signal $y_1[n]$ can be used to estimate $\langle v[m] \rangle$; correspondingly, we can form $y_2[n]$ from a moving average of $v^2[m]$ and use it to determine an estimate of $\langle v^2[m] \rangle$.

In this exercise, we use an FIR filter whose impulse response is $h[n] = 1/N$, $n = 0, 1, 2, \ldots, N-1$. A reasonable value for N is $N = 100$.

(a) Generate an uniformly distributed test signal that is at least 5 times longer than the FIR filter length, via `v = rand(1, 5*N)`

(b) Calculate and display the two output sequences: $y_1[n]$ for the mean, and $y_2[n]$ for the variance. Based on the plot, explain why the random signals $y_i[n]$ are not stationary in the interval $0 \leq n < N$.

(c) For $n > N$ the sequences $y_i[n]$ are random variables as well, but now out of stationary processes. It is easy to see that the variances of $y_i[n]$ are smaller than the variance of the input sequence. Measure the means and variances of the signals $y_i[n]$ by time averaging over $N \leq n \leq 5N$. Compare the means and variances with the values expected from theory.

(d) Compute and plot the histograms of the two output sequences, using the values for $N \leq n \leq 5N$ again. Mark the mean and variance on this plot.

Exercise 3.6: Confidence Interval

The foregoing experiments illustrate that the results of a moving average are random variables as well, with a certain mean and variance. Usually it is reasonable to assume that these values are normally distributed with a variance proportional to $1/N$ and to the (unknown) variance $\sigma^2_{\mathbf{v}}$ of the input signal. In order to be more specific, we describe the situation for the measurement of the mean $m_{\mathbf{v}}$ of the random signal $v[n]$.

If we compute a single estimate of the mean by averaging N samples of the signal, then the estimated value $\hat{m}_{\mathbf{v}}$ comes from a normally distributed RV with variance $\sigma^2_{\mathbf{v}}/N$. If the estimate is computed again over another N samples, the value will change somewhat, but we would like to bound the expected change. One way to do this is to compute a "confidence interval" centered on $\hat{m}_{\mathbf{v}}$ within which we predict that all estimates of the mean will lie

95% of the time. This approach leads to the following definition: the unknown mean $m_{\mathbf{v}}$ is located inside a *confidence interval* with the probability

$$S = \text{Prob}\left\{\hat{m}_{\mathbf{v}} - \frac{\sigma_{\mathbf{v}}}{\sqrt{N}} \cdot c \leq m_{\mathbf{v}} \leq \hat{m}_{\mathbf{v}} + \frac{\sigma_{\mathbf{v}}}{\sqrt{N}} c\right\}. \tag{3.12}$$

where c is the confidence parameter. Based on the above mentioned assumption that the variables $\hat{m}_{\mathbf{v}}$ are normally distributed, we get $c = \sqrt{2} \cdot \text{erf}^{-1}(S)$, which means, for example, that with a probability of $S = 95\%$ the mean $m_{\mathbf{v}}$ is inside the limits $\hat{m}_{\mathbf{v}} \pm \dfrac{\sigma_{\mathbf{v}}}{\sqrt{N}} \cdot 1.96$.

(a) In order to test the confidence interval calculation, we use a test signal that is white Gaussian noise, as in the previous exercise. Compute one estimate of the mean with a 100-point FIR filter. Using this estimate and the *true* variance of the signal, compute the 95% confidence interval for the 'unknown" mean. This theoretical calculation can now be tested by computing a large number of mean estimates and checking what percentage lie within the confidence interval. To get a reasonable test, several hundred estimates will have to be checked.

(b) In practice, there is another difficulty because we would have to estimate the variance before computing the confidence interval. Explain how your test of the confidence interval, from the previous part, will change if a single estimate of the variance over 100 points is used in place of the true variance.

Project 4: Influence of a Nonlinear Mapping on a Random Process

In this project, we deal with a nonlinear mapping of a random signal. In particular, we consider how the pdf changes. For special cases, we compute the autocorrelation sequence and the power spectrum of the mapped process. These nonlinear mappings have an important application: a new process with a desired pdf can be created out of a given one, usually starting from a uniform distribution.

For convenience we recall some equations (see [**?**, Chapter 5]). Given a random process $\mathbf{v}_n$ with probability density function $p_{\mathbf{v}}(v)$. Its variables are mapped according to

$$x = g(v).$$

As a simplification we assume that $g(v)$ is a monotonically increasing or decreasing function, which has a unique inverse mapping

$$v = g^{-1}(x).$$

It turns out that the mapped process $\mathbf{x}$ has the new pdf:

$$p_{\mathbf{x}}(x) = p_{\mathbf{v}}[g^{-1}(x)] \left| \frac{d}{dx}[g^{-1}(x)] \right|. \tag{4.1}$$

This analytic result can be used to determine the function $g(v)$, which maps a uniformly distributed process with $p_{\mathbf{v}}(v) = 1$ for $v \in [0, 1]$ into another one with the desired probability density $p_{\mathbf{x}}(x)$. The result is

$$x = g(v) = P_{\mathbf{x}}^{-1}(v), \tag{4.2}$$

where $P_{\mathbf{x}}^{-1}(v)$ is the inverse of the distribution function of the desired process.

If the variables of a sequence out of a random process are statistically independent, a memoryless nonlinear mapping will yield independent values again. In the more general case of a process with an arbitrary autocorrelation sequence we confine the considerations to a special case: If $\mathbf{v}_n$ is a normally distributed process with zero mean and the autocorrelation sequence $\phi_{vv}[m]$ and if $x = v^2$ we get for the output process $\mathbf{x}_n$ the mean

$$\mathcal{E}\{\mathbf{x}_n\} = \phi_{\mathbf{vv}}[0] \ , \tag{4.3}$$

and the autocorrelation sequence

$$\mathcal{E}\{\mathbf{x}_{n+m}\mathbf{x}_n\} = \phi_{\mathbf{xx}}[m] = \phi_{\mathbf{vv}}^2[0] + 2\phi_{\mathbf{vv}}^2[m] \tag{4.4}$$

(see [**?**, Section 10.2]). Corresponding to (4.4) the power density spectrum will involve a convolution:

$$\Phi_{\mathbf{xx}}(e^{j\omega}) = \phi_{\mathbf{vv}}^2(0)\delta(\omega) + \frac{1}{\pi}\,\Phi_{\mathbf{vv}}(e^{j\omega}) * \Phi_{\mathbf{vv}}(e^{j\omega}). \tag{4.5}$$

Hints

We use the random signals generated in MATLAB via `rand`. When a process with statistically dependent values is needed, we can use filter #2 out of the Project 3 *Influence of a Linear System on a Stochastic Process.*

Exercise 4.1: Linear Mapping

You are given a random process $\mathbf{v}_n$ with the probability density function $p_{\mathbf{v}}(v)$ and a linear mapping according to $x = \alpha \cdot v + \beta$, α and β being real numbers.

Derive a general expression for $p_{\mathbf{x}}(x)$ in terms of an arbitrary $p_{\mathbf{v}}(v)$. Specialize to the case where the process $\mathbf{v}_n$ is normal, $\mathcal{N}(0,1)$. Generate a random signal with Gaussian pdf, then transform it via the linear mapping, and finally plot its histogram to verify the change in the pdf.

Exercise 4.2: Nonlinear Mapping

Given a process $\mathbf{v}_n$ uniformly distributed in [0,1].

(a) Map this process linearly onto the interval $[-\pi/2, \pi/2]$.

(b) Map the resulting process ξ with

$$x = \sin\xi$$

onto the interval $[-1, 1]$. Determine a formula for $p_{\mathbf{x}}(x)$.

(c) Perform the mapping with MATLAB for $N = 8000$ values. Measure approximately the probability density function using `hist` with 20 bins. Compare the results with the theoretical formula from part (b).

(d) Measure the autocorrelation sequence of $\mathbf{x}_n$ using `acf` and explain the result.

Exercise 4.3: Laplacian Noise

Starting with a uniformly distributed random process, generate a process $\mathbf{x}_n$ with a Laplacian distribution and with variance $\sigma_{\mathbf{x}}^2 = 1$:

$$p_{\mathbf{x}}(x) = e^{-\sqrt{2}|x|}/\sqrt{2} \tag{4.6}$$

The nonlinear mapping procedure will be applied to a process $\mathbf{v}_n$, uniformly distributed in [0,1].

(a) Verify mathematically that the mapping has to be done with

$$x = g(v) = \begin{cases} \ln(2v)/\sqrt{2} & 0 \le v < 0.5 \\ -\ln(2-2v)/\sqrt{2} & 0.5 \le v \le 1. \end{cases}$$

Plot this function over the range $[0,1]$.

(b) Plot the inverse function, $v = g^{-1}(x)$. In MATLAB, this is as easy as swapping the arguments in the `plot` function.

(c) Prepare an M-file `lap(M,N)` for the generation of an $M \times N$ matrix, the rows of which are sample sequences out of a random process with Laplacian distribution, zero mean and unit variance.

(d) Generate a single sequence of length 8000 with your program `lap(M,N)`. Measure approximately the probability density function using a scaled histogram with 100 bins. Display the result on a semilogarithmic scale by using `semilogy` and overlay with a plot of (4.6) to compare with the results expected from theory.

Exercise 4.4: Histogram Equalization

In image processing, an effective transformation of a photograph to enhance low-level detail is a non-linear mapping that stretches out the gray scale of the image. This can be accomplished by transforming the image so that the histogram of its gray levels is nearly uniform. Since the input image does not necessarily have a known pdf (or histogram shape), the process of histogram equalization relies on pdf estimates to synthesize the correct mapping. The key is equation (4.2) which gives the relationship between the mapping $g(v)$ and the probability *distribution* function $P_{\mathbf{x}}(v)$. Since the distribution function is just the running integral of the pdf, it can also be estimated from a histogram of the input data.

(a) Generate an input signal that is Gaussian noise. Compute the histogram, and then make a plot of the probability *distribution* function $P_{\mathbf{v}}(v)$ defined in (1.2).

(b) Create a nonlinear mapping function that will transform the input signal $v[n]$ into a new random process that is uniformly distributed. This can only be done approximately by a piecewise linear approximation of the distribution function. The exact mapping depends strongly on the number of bins used in computing the underlying histogram.

(c) Test your mapping on the Gaussian input signal. Plot the histogram of the output. Generate several results for different number of bins.

(d) Write an M-file that will perform histogram equalization on any input signal. Test it on Laplacian noise.

Exercise 4.5: Squaring a Random Signal

(a) Generate a normally distributed sequence $y[n]$ with zero mean by exciting filter #2 out of Project 3 with a white noise input sequence $v[n]$. Measure its output autocorrelation sequence $\phi_{\mathbf{yy}}[m]$.

(b) Perform the mapping $x[n] = y^2[n]$ and measure approximately the pdf $p_{\mathbf{x}}(x)$ with 40 bins and the autocorrelation sequence $\phi_{\mathbf{xx}}[m]$. Compare with the expected theoretical results.

Project 5: Combining Two Random Processes

In this project we consider the joint properties of two or more random processes, generated by simple arithmetic operations. The following properties hold (see [**?**, Chapters 6 and 7]):

1. If $\mathbf{u}_n$ and $\mathbf{v}_n$ are two random processes, statistically independent or not, and $g_1(\cdot)$ and $g_1(\cdot)$ and $g_2(\cdot)$ are two mapping functions, then the mean of the two mapped random processes is separable

$$\mathcal{E}\{g_1(\mathbf{u}_n) + g_2(\mathbf{v}_n)\} = \mathcal{E}\{g_1(\mathbf{u}_n)\} + \mathcal{E}\{g_2(\mathbf{v}_n)\}. \tag{5.1}$$

2. If $\mathbf{u}_n$ and $\mathbf{v}_n$ are statistically independent, i.e. if

$$p_{\mathbf{uv}}(u, v, m) = p_{\mathbf{u}}(u) \cdot p_{\mathbf{v}}(v), \qquad \forall m \tag{5.2}$$

   then the mean of the product is separable

$$\mathcal{E}\{g_1(\mathbf{u}_n) \cdot g_2(\mathbf{v}_n)\} = \mathcal{E}\{g_1(\mathbf{u}_n)\} \cdot \mathcal{E}\{g_2(\mathbf{v}_n)\}. \tag{5.3}$$

3. In the special case $g_{1,2}(\cdot) = e^{j\chi(\cdot)}$ that yields

$$\mathcal{E}\{g_1(\mathbf{u}_n) \cdot g_2(\mathbf{v}_n)\} = \mathcal{E}\{e^{j\chi(\mathbf{u}_n+\mathbf{v}_n)}\} = C_{\mathbf{x}}(\chi), \tag{5.4}$$

   where $C_{\mathbf{x}}(\chi)$ is the *characteristic function* of the process $\mathbf{x}_n = \mathbf{u}_n + \mathbf{v}_n$. Obviously, we get

$$C_{\mathbf{x}}(\chi) = C_{\mathbf{u}}(\chi) \cdot C_{\mathbf{v}}(\chi) \tag{5.5}$$

   and thus, the probability density of the sum of two independent processes is the convolution of their pdf's

$$p_{\mathbf{x}}(x) = p_{\mathbf{u}}(x) * p_{\mathbf{v}}(x). \tag{5.6}$$

4. The autocorrelation sequence for the sum of two independent processes is

$$\phi_{\mathbf{xx}}[m] = \phi_{\mathbf{uu}}[m] + \phi_{\mathbf{vv}}[m] + 2m_{\mathbf{u}} \cdot m_{\mathbf{v}} \ . \tag{5.7}$$

Finally, the autocorrelation sequence of the product process

$$\mathbf{y}_n = \mathbf{u}_n \cdot \mathbf{v}_n \ ,$$

where $\mathbf{u}_n$ and $\mathbf{v}_n$ are two independent processes, turns out to be

$$\phi_{\mathbf{yy}}[m] = \phi_{\mathbf{uu}}[m] \cdot \phi_{\mathbf{vv}}[m] \ . \tag{5.8}$$

All these equations can be extended to the summation or multiplication of more than two random processes. Equations (5.2–5.8) hold, if all these processes are mutually independent.

Exercise 5.1: Sum of Two Random Signals

Generate two independent random processes $\mathbf{u}_n$ and $\mathbf{v}_n$, both with uniform distribution, the first one in the interval [1,2], the other one in [0,4].

(a) Measure the mean and the variance of the sum $\mathbf{x}_n = \mathbf{u}_n + \mathbf{v}_n$; and compare with the results to be expected theoretically.

(b) Measure the mean and the variance of the product $\mathbf{y}_n = \mathbf{u}_n \cdot \mathbf{v}_n$; and compare with the results to be expected theoretically.

(c) Measure approximately the pdf of the sum $\mathbf{x}_n = \mathbf{u}_n + \mathbf{v}_n$ and compare with the results to be expected according to (5.6).

Generate now two independent, but not white, random processes $\mathbf{y}_{1n}$ and $\mathbf{y}_{2n}$ by exciting filters #1 and #2 out of Project 3 with $\mathbf{u}_n$ and $\mathbf{v}_n$, respectively.

(d) Measure the mean and the autocorrelation function of the two individual processes.

(e) Measure the mean and the autocorrelation function of their sum $\mathbf{y}_n = \mathbf{y}_{1n} + \mathbf{y}_{2n}$ and compare with the results to be expected according to (5.1) and (5.7).

Finally, generate two statistically dependent random processes $\mathbf{w}_{1n}$ and $\mathbf{w}_{2n}$ by exciting filters #1 and #2 out of Project 3 with $\mathbf{u}_n$.

(f) Measure the mean, the variance and the pdf of both processes individually.

(g) Measure the mean and the autocorrelation function of their sum $\mathbf{w}_n = \mathbf{w}_{n1} + \mathbf{w}_{n2}$ and compare as far as possible with the results to be expected theoretically.

Exercise 5.2: Pdf of a Sum

Generate M sequences of length N out of independent random processes $\mathbf{v}_{n\mu}$, all uniformly distributed in the interval $[-1, 1]$, with `rand(M,N)`. Use a large length, say $N = 2000$.

(a) Calculate the summation processes $\mathbf{x}_{nM} = \sum_{\mu=1}^{M} \mathbf{v}_{n\mu}$, using `sum` for $M = 2, 3, 4, 5$. Compute and plot their pdf's using `hist` with 20 bins. Display these results all together using a four-panel `subplot`.

(b) What type of pdf do you expect for increasing M?

Exercise 5.3: Sum and Product of Two Processes

This exercise deals with two independent random processes $\mathbf{u}_n$ and $\mathbf{v}_n$ with zero mean, where $\mathbf{u}_n$ is uniformly distributed in $[-1, 1]$, while $\mathbf{v}_n$ is normally distributed. The autocorrelation sequences are $\phi_{\mathbf{uu}}[m]$ and $\phi_{\mathbf{vv}}[m]$. The sum and the product of these processes are to be investigated.

The process $\mathbf{u}_n$ must be generated in a special way, so that its power density spectrum will not be constant. Two steps are involved: filter a white Gaussian noise input process, and then apply a nonlinear mapping to the output process to change the distribution to uniform. To modify the spectrum we filter a zero-mean normal process; use filter #2 from Project 3, and take the input signal length to be $N = 8000$. The filter's output process will be normally distributed as well, but with a non-constant power density spectrum. This process can be transformed into a uniformly distributed one by using the mapping technique described in Project 4. It turns out that the "error function" `u = erf`$(v/\sigma_{\mathbf{v}} \cdot \sqrt{2})$ maps a normally distributed process $\mathbf{v}_n$ with zero mean and variance $\sigma_{\mathbf{v}}^2$ into a process $\mathbf{u}_n$, that is uniformly distributed on $[-1, 1]$.

Remark: This nonlinear mapping will change the power density spectrum and thus the autocorrelation sequence, but it will still be non-constant.

(a) Do all the steps necessary to produce the uniformly distributed process $\mathbf{u}_n$, so that it has a non-constant power spectrum. Measure the pdf in order to check your result.

(b) Measure the autocorrelation sequences of both processes using $N = 8000$ samples of the processes.

(c) Measure the mean, variance and the autocorrelation sequences of the sum and the product of both processes. Compare your results with those expected from theory.

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{FFT Spectrum Estimation}}$

FFT Spectrum Estimation

Overview

This set of projects deals with FFT-based methods for spectrum estimation, the approximate determination of the power density spectrum $\Phi_{\mathbf{yy}}(e^{j\omega})$ of a given real ergodic random process $\mathbf{y}_n$.

$$\Phi_{\mathbf{yy}}(e^{j\omega}) = \sum_{m=-\infty}^{\infty} \phi_{\mathbf{yy}}[m]\, e^{-j\omega m} \tag{0.1}$$

In practice, its measurement cannot be done according to the definition (0.1) because the autocorrelation sequence

$$\begin{aligned}\phi_{\mathbf{yy}}[m] &= \mathcal{E}\{\mathbf{y}_{n+m}\mathbf{y}_n\} \\ &= \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{+N} y_\lambda[n+m] y_\lambda[n]\end{aligned}$$

can only be approximated when a finite segment of length N of one member $y_\lambda[n]$ out of the process is available.

In this packet, we shall study the performance of several different methods. In order to show their merits as well as their deficiencies, we use as an example a random process with known power density spectrum, generated with a linear system with known transfer function, excited by a white Gaussian process $\mathbf{v}_n$ with variance $\sigma_{\mathbf{v}}^2 = 1$. Thus in our experiment we can compare to the correct result and can express the observed deviations in terms of this known spectrum. Since this additional information about the true spectrum is not available when processing an unknown signal, we also consider how to derive confidence interval expressions for our estimates.

Background Reading

There are books devoted to spectral estimation, especially the books by Marple [**?**] and Kay [**?**]. Others have rather comprehensive chapters on the subject: Chapter 13 in the book by Papoulis [**?**], Chapter 2 in Lim and Oppenheim [20] and Sections 2.10 and 11.5–11.7 in Oppenheim and Schafer [8]. A useful algorithm for estimating the autocorrelation sequence is described in [**?**].

Project 1: Periodogram

In this project, we investigate the periodogram, an estimate that is based on one segment of $y[n]$, extracted by windowing with $w[n]$, a window sequence of length N:

$$x[n] = w[n] \cdot y[n]\,. \tag{1.1}$$

The periodogram is defined as

$$I_N(e^{j\omega}) = \frac{1}{N}\left|\sum_{n=0}^{N-1} x[n]\cdot e^{-j\omega n}\right|^2 . \tag{1.2}$$

Obviously, samples of $I_N(e^{j\omega})$ in (1.2) at the points $\omega = \omega_k = (2\pi/N)k$ can be calculated very efficiently with the FFT. In addition, it can be rewritten as

$$I_N(e^{j\omega}) = \frac{1}{N} \sum_{m=-(N-1)}^{N-1} \rho_{\mathbf{xx}}[m] e^{-j\omega m}\,, \tag{1.3}$$

where the autocorrelation sequence of the finite length sequence $x[n]$ is computed via:

$$\rho_{\mathbf{xx}}[m] = \sum_{n=0}^{N-1-|m|} x[n]\, x[n+|m|] \tag{1.4}$$

The similarity of the definition of the power density spectrum, given in the overview (0.1) and eq. (1.3) is the reason for calling the periodogram the "natural estimate" of the power density spectrum. But, as we shall show, it yields rather inaccurate results, if used without modification.

A theoretical investigation of the properties of the periodogram yields the following results (see Section 13.2 in [**?**], Chapter 2 in [20] or Section 11.6 in [8]).

Bias. First, the expected value of $\rho_{\mathbf{xx}}[m]$ in (1.4) reduces to

$$\mathcal{E}\{\rho_{\mathbf{xx}}[m]\} = \rho_{ww}[m] \cdot \phi_{\mathbf{yy}}[m]\,, \tag{1.5}$$

where $\phi_{\mathbf{yy}}[m]$ is the true autocorrelation, and $\rho_{ww}[m]$ is the autocorrelation sequence of the window:

$$\rho_{ww}[m] = \sum_{n=0}^{N-1-|m|} w[n] w[n+|m|] \tag{1.6}$$

Thus the expected value of the periodogram becomes

$$\begin{aligned} \mathcal{E}\{I_N(e^{j\omega})\} &= \frac{1}{N} \sum_{m=-(N-1)}^{N-1} \mathcal{E}\{\rho_{\mathbf{xx}}[m]\} e^{-j\omega m} && (1.7) \\ &= \frac{1}{2\pi N}\, \Phi_{\mathbf{yy}}(e^{j\omega}) * \left|W(e^{j\omega})\right|^2 . && (1.8) \end{aligned}$$

Here $W(e^{j\omega}) = \sum_{n=0}^{N-1} w[n] e^{-j\omega n}$ is the DTFT of the window. Finally, the bias, which is the difference between the expected value of the estimate and the true mean, can be written as:

$$\begin{aligned} \text{BIAS} &= \mathcal{E}\{I_N(e^{j\omega_k})\} - \Phi_{\mathbf{yy}}(e^{j\omega_k}) && (1.9) \\ &= \frac{1}{2\pi N} \Phi_{\mathbf{yy}}(e^{j\omega}) * \left|W(e^{j\omega})\right|^2 - \Phi_{\mathbf{yy}}(e^{j\omega}) && (1.10) \end{aligned}$$

Example: We consider the case of the rectangular window whose autocorrelation sequence is a triangle. The corresponding transform is

$$\left|W(e^{j\omega})\right|^2 = \left[\frac{\sin N\omega/2}{\sin \omega/2}\right]^2 . \tag{1.11}$$

According to (1.8) the convolution of this asinc-squared function with $\Phi_{\mathbf{yy}}(e^{j\omega})$ yields the mean of the periodogram, which thus turns out to be a smoothed version of the true power spectrum, $\Phi_{\mathbf{yy}}(e^{j\omega})$. Since the convolution kernel $\left|W(e^{j\omega})\right|^2$ has a mainlobe of width $2\pi/N$, we conclude that

$$\lim_{n\to\infty} \mathcal{E}\left\{I_N(e^{j\omega})\right\} = \Phi_{\mathbf{yy}}(e^{j\omega}) \ . \tag{1.12}$$

at every point of continuity of $\Phi_{\mathbf{yy}}(e^{j\omega})$. It follows that $I_N(e^{j\omega})$ is an unbiased estimator of $\Phi_{\mathbf{yy}}(e^{j\omega})$ in the limit as $N \to \infty$, called *asymptotically unbiased.*

This result (1.12) can be generalized to all windows, if two conditions hold:

1. the window is normalized so that $\sum_{n=0}^{N-1} w^2[n] = N$, and
2. the width of the mainlobe of the spectrum $\left|W(e^{j\omega})\right|^2$ decreases as $1/N$.

On the other hand, for finite N, (1.12) provides a means to calculate the bias of the result, if the true spectrum about $\Phi_{\mathbf{yy}}(e^{j\omega})$ is known (see Exercise 1.1).

Variance. Now we consider the variance of $I_N(e^{j\omega})$:

$$\mathrm{var}\{I_N(e^{j\omega})\} = \mathcal{E}\{I_N^2(e^{j\omega})\} - [\mathcal{E}\{I_N(e^{j\omega})\}]^2 \ .$$

If the process $\mathbf{y}_n$ is Gaussian, a rather long calculation yields

$$\mathrm{var}\{I_N(e^{j\omega})\} = [\mathcal{E}\{I_N(e^{j\omega})\}]^2 + \left|\frac{1}{2\pi N}\int_{-\pi}^{\pi} \Phi_{\mathbf{yy}}(\eta)\cdot W(e^{j(\eta-\omega)})\cdot W^*(e^{j(\eta+\omega)})d\eta\right|^2 \ . \tag{1.13}$$

Since the first term in (1.13) does not go to zero as $N \to \infty$, the periodogram is not a consistent estimator.

Example: In the special case of a rectangular window we get

$$\mathrm{var}\{I_N(e^{j\omega})\} = \Big[\mathcal{E}\{I_N(e^{j\omega})\}\Big]^2 + \left[\sum_{m=-(N-1)}^{N-1} \frac{\sin\omega(N-|m|)}{N\sin\omega}\,\phi_{\mathbf{yy}}[m]\right]^2 \tag{1.14}$$

If $\mathbf{y}_n$ has no significant correlation beyond lag m_0, i.e., $\phi_{\mathbf{yy}}[m] \approx 0$ for $|m| > m_0$, then for $N \gg m_0$ this expression yields

$$\mathrm{var}\{I_N(e^{j\omega})\} \quad \approx \quad [\mathcal{E}\{\Phi(e^{j\omega})\}]^2 \left[1 + \left(\frac{\sin\omega N}{N\sin\omega}\right)^2\right] \ . \tag{1.15}$$

Using the fact that the mean is asymptotically unbiased, we get for $N \to \infty$

$$\mathrm{var}\{I_N(e^{j\omega})\} = \begin{cases} 2\,\Phi_{\mathbf{yy}}^2(e^{j\omega}) \ , & \omega = 0, \pi \\ \Phi_{\mathbf{yy}}^2(e^{j\omega}) \ , & \text{elsewhere.} \end{cases} \tag{1.16}$$

The essential point of this example is that even for very long windows the standard deviation of the periodogram estimate is as large as the mean, the quantity to be estimated.

Hints

For the exercises below we need a test signal consisting of filtered noise. We start with a signal that is a normally distributed white random sequence $v[n]$ with zero mean and variance $\sigma_{\mathbf{v}}^2 = 1$, generated with the MATLAB function `rand('normal')`. The test signal will be created by passing $v[n]$ through a known filter $H(e^{j\omega})$. The MATLAB function `filter` is used to produce the random signal. However, its output sequence will be stationary only after the transient is over; this transient length can be estimated by checking the length of the impulse response.

The function `freqz` can be used to calculate $|H(e^{j\omega})|^2$ which is also the correct power density spectrum $\Phi_{\mathbf{yy}}(e^{j\omega})$. Thus our estimates, which are just approximations, can be compared to this standard. In addition, the true autocorrelation sequence, $\phi_{\mathbf{yy}}[m]$, which is the inverse DTFT of $\{\Phi_{\mathbf{yy}}(e^{j\omega})\}$, can be approximated by calculating the inverse FFT of the sampled power spectrum.

In several exercises, the periodograms will be calculated with different values of N, yielding a different number of samples of the estimate of the power density spectrum. For a better comparison with the theoretical function $\Phi_{\mathbf{yy}}(e^{j\omega}) = |H(e^{j\omega})|^2$, they should all be interpolated to the same grid. This can be accomplished simply by padding the windowed sequence $x[n]$, by $M - N$ zeros, prior to taking the FFT needed for the periodogram (a convenient choice would be $M = 512$ or 256).

Exercise 1.1: Formulae for bias and variance

In this exercise, we will verify the mathematical expressions for the true values of the bias and variance. For the filter applied to the WGN, we use the following coefficients

```
b = 0.06 * [ 1  2  1 ]
a = [ 1   -1.3   0.845 ] .
```

The transient portion of the output must be skipped. The poles, which lie at a radius of approximately 0.92, can be used to estimate the duration of the transient.

(a) Calculate the true power spectrum, $\Phi_{\mathbf{yy}}(e^{j\omega}) = |H(e^{j\omega})|^2$, and plot versus ω.

(b) To compute bias of the periodogram, it is easier to start from the autocorrelation domain as in (1.5) and (1.7). First, we need a segment of the true autocorrelation function, $\phi_{\mathbf{yy}}[m]$. Compute an approximation to $\phi_{\mathbf{yy}}[m]$ using the `ifft` function. Determine the length needed in the inverse FFT, so that the approximation error is negligible. Write an M-file that will return a specified section of $\phi_{\mathbf{yy}}[m]$. Then plot $\phi_{\mathbf{yy}}[m]$ for $-100 \leq m \leq 100$.

If possible, derive a formula for $\phi_{\mathbf{yy}}[m]$ and use this formula to calculate the values.

(c) Now we can find the expected value of the periodogram as in (1.5). First, calculate the expected value of the autocorrelation function of the windowed signal $\mathcal{E}\{\rho_{\mathbf{xx}}[m]\}$ for the case of a rectangular window of length N. Then use the DTFT to compute $\mathcal{E}\{I_N(e^{j\omega_k})\}$, as suggested in (1.7). Plot the expected value of the periodogram, and overlay with the true value of the power spectrum as a dashed line. Repeat for four different cases, $N = 32$, 64, 128, and 256. Display the results together in a four-panel `subplot`.

(d) Now determine the bias of the periodogram (1.10), and plot it versus ω. Make the plots for all four values of N and justify that the bias will go to zero as $N \to \infty$. Explain why the bias is greatest at the peak of the spectrum.

(e) Calculate and plot the variance of the periodogram according to (1.14); again for $N = 32$, 64, 128, and 256. Verify that it is *not* decreasing with N.

(f) In order to check the accuracy of the approximation in (1.15), calculate and plot the difference of the two functions given in (1.14) and (1.15) for $N = 128$ and 256.

Exercise 1.2: Measuring the periodogram with a rectangular window

Now we compute the periodogram of $y[n]$ directly from the data, using a rectangular window.

(a) Calculate four periodograms with $N = 32$, 64, 128, and 256. Plot all four together using a four-panel `subplot`. Overlay with the true power spectrum, as a dashed line for comparison. Describe features that improve with increasing N, and relate these observations to the fact that the bias is decreasing.

(b) Calculate and plot together 10 periodograms, all for the case where the window length is $N = 64$. In order to get independent results, non-overlapping sections must be used and the transient must be skipped. Repeat for $N = 256$. Describe the nature of these plots in view of the fact that the variance is *not* decreasing as $N \to \infty$. For instance, measure all the peak heights to see whether they are less variable for larger N.

Exercise 1.3: Measuring the periodogram with a tapered window

We now repeat the previous exercise, but with the Hamming window $w_1[n]$, or the von Hann (`hanning`) window $w_2[n]$:

$$w_1[n] = \gamma_1\,[0.46 - 0.54\cos(2\pi n/N)] \qquad \text{for } 0, 1, \ldots, N-1 \tag{1.17}$$

$$w_2[n] = \gamma_2\,[\tfrac{1}{2} - \tfrac{1}{2}\cos(2\pi n/N)] \qquad \text{for } 0, 1, \ldots, N-1 \tag{1.18}$$

Pick one of these windows to use throughout this exercise.

(a) Determine the gain factor γ_i needed to normalize the energy in the window, so that the windowed periodogram estimate is asymptotically unbiased.

(b) In order to investigate the properties of this window calculate and plot its autocorrelation sequence and the corresponding transform; overlay with a plot of asinc-squared to compare with the rectangular window case. Use $N = 64$ and the MATLAB function `xcorr` for the calculation of $\rho_{ww}[m]$. In the plot of the transform, it would be wise to use a semilogarithmic representation to see details in the sidelobes, and a linear plot to view the mainlobe. Calculate approximately the mainlobe width of the tapered window as a function of N. Will the windowed periodogram estimate be asymptotically unbiased?

(c) Compute and plot the bias for the tapered window, as in Exercise 1.1, parts (b) – (d).

(d) Repeat Exercise 1.2, part (b) with the tapered window. Compare your results with those obtained for the rectangular window and with the theoretical function $\Phi_{\mathbf{yy}}(e^{j\omega})$. In which frequency range do you get an improvement and why?

Project 2: Periodogram Averaging

In Project 1, we found the periodogram to be an easily calculable, but biased and inconsistent estimate of the power density spectrum. Here we consider methods for improving the performance of the periodogram. Quite a few modifications have been introduced to improve the results without losing the advantage of an efficient calculation. The essential points of the changes are:

- *averaging* over a set of periodograms of nearly independent segments
- *windowing* applied to the segments
- *overlapping* the windowed segments for more averaging.

If we are given M data points from a signal $y[n]$ out of an ergodic process $\mathbf{y}_n$, this data block can be divided into $K = M/N$ segments of length N, where we assume K to be an integer for convenience. According to Bartlett's procedure (see Section 11.6.3 in [8]) we calculate an estimate of the power density spectrum as the average of K periodograms:

$$\Phi_B(e^{j\omega}) = \frac{1}{K}\sum_{r=0}^{K-1} I_N^{(r)}(e^{j\omega}) \; , \tag{2.1}$$

where the periodogram of the r^{th} segment is

$$I_N^{(r)}(e^{j\omega}) \; = \frac{1}{N}\left|\sum_{n=0}^{N-1} x_r[n]e^{-j\omega n}\right|^2 \tag{2.2}$$

This segment was created by windowing $y[n]$:

$$x_r[n] = w[n]\cdot y[n + r(N - N_o)] \; . \tag{2.3}$$

Strictly speaking, (2.3) describes Bartlett's procedure only if $w[n]$ is the rectangular window, and the overlap N_o is zero. The generalization to other windows and, especially, to the case of overlapping windows, where $N_o > 0$, is called *Welch's procedure.*

This averaging process improves the variance. First, we calculate the expected value

$$\mathcal{E}\{\Phi_B(e^{j\omega})\} = \frac{1}{K}\sum_{r=0}^{K-1}\mathcal{E}\{I_N^{(r)}(e^{j\omega})\} = \mathcal{E}\{I_N(e^{j\omega})\} \; . \tag{2.4}$$

Since the periodogram is a biased estimate of the desired power spectrum for a finite length N (see Project 1), the averaging does not yield an improvement. The bias is only a function of the window length and window type. In fact, if only a finite length signal is available, segmentation will increase the bias, since it forces the window to be shorter.

For the variance, however, we get

$$\text{var}\{\Phi_B(e^{j\omega})\} = \frac{1}{K_{\text{eff}}}\text{var}\{I_N^{(r)}(e^{j\omega})\} \; . \tag{2.5}$$

If the segments were truly independent, then $K_{\text{eff}} = K$. However, in general, there is correlation between the segments, or the segments are overlapped, so that $K_{\text{eff}} < K$. The number K_{eff} indicates the effective number of averages being accomplished.

Hints

We use again the random sequence $y[n]$, generated as described in Project 1. For an easy comparison with the theoretical function $\Phi_{\mathbf{yy}}(e^{j\omega})$, the power spectra should all be interpolated to the same grid—probably 512 frequency samples would be best.

The determination of empirical results for the expected values in Exercise 2.2 could be done by calculating a matrix of estimated power density spectra and using the MATLAB functions `mean` and `std` for calculating these values per column.

Exercise 2.1: A program for the Welch-Bartlett procedure.

Write an M-file called `wbpsd` for the Welch-Bartlett spectral estimation technique. It should be possible to specify some or all of the following input parameters:

M	the length of the input sequence
N_o	the size of the overlapping part of the sections
K	the number of segments
N	the length of the periodogram (a power of 2)
$w[n]$	the window (so different types can be tried)
N_{fft}	the number of frequency samples after interpolation (default $N_{\text{fft}} = 512$)

In order to make sure that a full size segment is always used for calculating the averaged periodogram according to (2.2), the number of segments $K = (M - N_o)/(N - N_o)$ must be an integer. If the selected parameters M, N and N_o do not satisfy this condition, then M should be reduced accordingly by ignoring an appropriate number of data at the end of the given sequence.

Exercise 2.2: Averaged periodograms

Use the `wbpsd` M-file from the previous exercise to process the test signal. Use a data length of $M = 512$ with $N = 64$ and the following window types, overlaps and number of sections:

Rectangular window	$N_o = 0$	$K = 8$
Hann window	$N_o = 0$	$K = 8$
Hann window	$N_o = 32$	$K = 15$

(a) Calculate and plot together 10 Welch-Bartlett estimated power spectral densities for these three cases, using independent length-M sequences each time.

(b) Calculate 100 Welch-Bartlett estimated power spectral densities for the above given parameters with 100 independent sequences of length M. Use these results for the calculation of the empirical bias and variance, as a function of ω. Plot the empirical bias together with the true bias. Is there good agreement between theory and the empirical result, or should the experiment be redone for more than 100 trials?

(c) Use the spectra from the previous part to calculate an empirical variance for the Welch-Bartlett estimate. Compare your result to the theoretical formula studied in Project 1. Determine an estimate for K_{eff}, the effective number of averages using (2.5). Explain why the value of K_{eff} is always less than K; in addition, determine which case has the most variance reduction.

(d) Show empirical evidence that the variance will converge to 0, as $M, K \to \infty$, keeping N fixed. This may require that you generate one more test case with a much longer data sequence ($M \gg 512$).

Exercise 2.3: The MATLAB M-file `spectrum`

The M-file `spectrum` is MATLAB's realization of the Welch-Bartlett spectral estimator. It is contained in the signal processing toolbox along with a companion function `specplot` that will plot its outputs. The data window `hanning` is hard-coded into `spectrum`, although it could be changed by editing. The length N of the averaged periodogram and the size N_o of the overlapping part of the sections can be specified as input arguments.

It turns out that `spectrum` introduces a scaling of the power spectral density estimate, $\Phi^{(s)}_{\mathbf{yy}}(e^{j\omega}) = (1/\alpha) \cdot \Phi_{\mathbf{yy}}(e^{j\omega})$. This scaling can be a problem when experimental data must be calibrated to known units.

(a) Use `spectrum` to estimate a scaled power spectral density from a length $M = 512$ data sequence. Choose the overlap to be zero ($N_o = 0$); and try three different section lengths $N = 64$, 128 and 256. Repeat the same estimates with the M-file `wbpsd` written in Exercise 2.1. Determine the scaling factor $\alpha(N)$ by comparing the two estimates. Comment on the possible cause of the different scalings.

(b) Determine whether or not the scaling α is dependent on the frequency ω.

Exercise 2.4: Empirical Study of Confidence Intervals

The variance of a spectral estimate, if measured empirically, requires that the estimation process be repeated many times. However, in practice one wants to know whether a single estimate is any good; and whether or not a repetition of the experiment is likely to yield an estimate that is close to the first. From one estimate, the procedure is to compute a "confidence interval" that marks the upper and lower limits on further estimates. The M-file `spectrum` offers an estimate of the 95% confidence interval, based on a gaussian assumption for the PSD estimates. In theory, 19 out of every 20 estimates should lie within the confidence interval bounds.

(a) Use `spectrum` to calculate both a PSD-estimate and its confidence interval from a length $M = 512$ data sequence for $N = 64$ and $N_o = 32$. Plot the results using the M-file `specplot` to display the confidence interval as dashed lines above and below the estimated spectrum.

(b) In order to test the 95% factor, redo the spectrum estimate 20 times and plot all the estimates on the same graph with the confidence interval. Visually compare the estimates to the upper an lower bounds. Does the confidence estimate seem to be a realistic one? Which frequencies seem correct, which ones are definitely wrong?

(c) It is possible to count the number of times the estimates lie outside the confidence bounds. This can be done for all 20 estimates and for all the frequencies in the estimate. From this count determine what percentage of the estimates lie inside the confidence limits and compare your empirical result to the projected 95% figure.

(d) The confidence limit assumes that the bias is zero, so that one estimate is sufficient to characterize the mean. However, this is not true for the finite sample case. Redo the plot of the confidence interval using the true mean and true variance of the PSD estimate. Compare to the confidence intervals created by `spectrum` and explain any significant differences. Repeat the empirical counting process in the previous part to see whether or not these confidence limits are more realistic.

(e) The confidence limit in `spectrum` is based on a Gaussian assumption, where the mean and variance have to be approximated. For the PSD estimate, the mean is taken to be the computed value of $I(e^{j\omega})$ and the variance is set equal to $(1/K_{\text{eff}})$ times the mean squared. The 95% point on a Gaussian pdf can be computed using the function `inverf`. Use these ideas to write an M-file that will add confidence intervals to the function `wbpsd` written in Exercise 2.1.

The Gaussian assumption is only an approximation, and it is a poor one when the ratio of the variance to the mean is large. In reality, the distribution is chi-squared.

Project 3: Narrowband Signals

Quite often spectral estimates are desired for systems whose frequency response contains narrow peaks, e.g., passive sonar. In this case, the resolution limit imposed by the data window is the main concern—in fact, it causes a bias in the spectral estimate. On the other hand, the approximation to the variance (1.15) no longer holds. In this project, we investigate a system with two narrowband peaks, and the estimation of its power spectrum via the Welch-Bartlett procedure.

Hints

Long data sequences may be needed to carry out the estimation procedures needed in this project. Even on computers with limited memory, the filtered output can be produced for very long sequences by using the initial and final state saving capability of `filter`.

Exercise 3.1: Resolution of Peaks

Construct a transfer function $H(z)$ with four poles at:

$$\text{POLES} = \{0.99\angle \pm 88^\circ, 0.99\angle \pm 92^\circ\} \tag{3.1}$$

The numerator should be a constant, so the system function $H(z)$ is all-pole.

(a) Plot the magnitude squared of the transfer function $|H(e^{j\omega})|^2$, which is also the true power spectrum.

(b) From the plot of the true power spectrum, find the frequency separation of the two peaks. Then determine the minimum window length needed to resolve the two peaks when using a rectangular window, also determine the window length needed for a Hamming window.

(c) Generate the filter's output signal, $y[n]$. Plot a section of the *random* signal, and comment on its structure. In this case the transient will be rather long, so make sure that you ignore enough points at the beginning of the signal.

(d) Apply the Welch-Bartlett method with the two different windows. and with averaging over 8 segments. You should verify whether or not the window length is sufficient to resolve the peaks. If not, increase the window length until you get resolution.

(e) Consider the spectrum estimates generated for different numbers of segments: $K = 4$, 8, 16 and 32. From these results, determine whether or not variance is a significant problem with estimating the narrowband spectrum. In particular, observe whether more averaging improves the estimate.

(f) Consider the bias of the spectrum estimate in the following way: Does the estimated power spectrum have the correct shape near the peaks? It will be necessary to "blow up" the plot near the peak locations and make a plot with the estimate superimposed on the true spectrum to answer this question. If they are different, this is a manifestation of bias. Assuming they are different, how much more would the length of the window have to increased to remove this bias? Hint: compare the bandwidth of the true peak to the mainlobe width of the window.

(g) *Optional:* Change the system slightly by placing zeros in the transfer function at $z = \pm j$. Redo the experiment for the Hamming window, to see whether or not the minimum window size drops.

Exercise 3.2: Different Windows

The mainlobe of the window function is the primary factor in resolution. However, we cannot ignore the sidelobe structure. In the previous exercise, the spectrum contained two equal amplitude peaks. Now we consider another system in which the peak heights are quite different. It is easy to produce an example where some windows will always fail due to their sidelobe structure.

(a) Use the following system to generate a random output signal with known power spectrum.

$$H(z) = \frac{2.2 - 2.713z^{-1} + 1.9793z^{-2} - 0.1784z^{-3}}{1 - 1.3233z^{-1} + 1.8926z^{-2} - 1.2631z^{-3} + 0.8129z^{-4}}$$

Plot the true power spectrum for this system.

(b) Process the signal with the Welch-Bartlett method using the rectangular window. Use 8 sections of data, and a window length of 64 which should easily resolve the two peaks.

(c) Redo the Welch-Bartlett processing with a length-64 Hamming window. Are the two peaks now visible in the plot of the spectrum estimate?

(d) Change the length of the rectangular window and reprocess the data in an attempt to get resolution of both peaks. Compare to the processing with a longer Hamming window. Why is the Hamming window capable of resolving the peaks while the rectangular window cannot? If the length of the rectangular window is made extremely large will it eventually be possible to "see" the two peaks when producing the spectral estimate with rectangular windowing?

Exercise 3.3: Sine Wave Plus Noise

A related case is that of sine waves in noise. This problem arises in radar, for example, when processing Doppler returns. Once again the resolution issue is the main one, but now the signal is deterministic in a background of noise.

The test signal for this exercise should contain three sinusoids in a background of WGN.

#1	$\omega_1 = \pi/3$	$A_1 = 50$	$\phi_1 = 45^\circ$
#2	$\omega_2 = 0.3\pi$	$A_2 = 10$	$\phi_2 = 60^\circ$
#3	$\omega_3 = \pi/4$	$A_3 = 2$	$\phi_3 = 0^\circ$

The noise variance should be set at $\sigma_{\mathbf{v}}^2 = 10$, although other values could be used.

(a) Create a signal that has only sinusoid #3 and the noise. If you make a time-domain plot the sinusoid will not be visible. However, after processing with a long FFT the peak at the sinusoidal frequency will appear. Try FFT lengths from 32 through 256 to verify this behavior. What is the minimum FFT length needed to pull the sinusoid out of the background of noise? How does this FFT length depend on the SNR?

(b) Create the signal containing three sinusoids in a background of WGN, as described in the table above. Use one very long section of the signal (1024 points) to compute an FFT and display the magnitude to verify that the three spectral lines are in the correct place.

(c) Based on the known characteristics of the three sinusoids, determine the minimum lengths needed for a Hamming and rectangular window to resolve the closely spaced sinusoids. Process the data with the Welch-Bartlett method, using 8 sections. Make sure that the three peaks are resolved; if not, reprocess with a longer window. Compare the results for the estimates made with the two different window types.

(d) Process the data with the Welch-Bartlett method, using $K = 1$ and $K = 8$. Describe the difference in the spectrum estimates for different amount of averaging.

Project 4: Cross-Spectrum

Related to the auto-power spectrum of one signal is the cross power spectrum between two signals. It is defined as the Fourier transform of the cross-correlation sequence:

$$\Phi_{\mathbf{xy}}(e^{j\omega}) = \sum_{m=-\infty}^{\infty} \phi_{\mathbf{xy}}[m] e^{-j\omega m} \tag{4.1}$$

Since the Fourier transform of the cross correlation between two signals is the multiplication of one transform with the conjugate of the other, we can define a cross spectral estimate directly in the frequency domain:

$$P_{\mathbf{xy}}(e^{j\omega_k}) = X^*[k] \cdot Y[k] \tag{4.2}$$

This is analogous to the periodogram definition, so it must be averaged to yield a worthwhile estimate.

Exercise 4.1: Function for Cross Spectrum

Since the cross-spectrum is nearly identical to the auto-spectrum, modify your previous program (`wbpsd`) to accept an additional input, the signal $y[n]$, and then compute the cross spectrum between $x[n]$ and $y[n]$. Allow for the possibility of sectioning and averaging during the computation.

Exercise 4.2: Identification of the Transfer Function

One interesting property of the cross spectrum is its use in system identification. If the two signals $x[n]$ and $y[n]$ are related by a transfer function $H(z)$, such that $x[n]$ is the input and $y[n]$ is the output, the cross spectrum is:

$$\Phi_{\mathbf{xy}}(e^{j\omega}) = H(e^{j\omega})\,\Phi_{\mathbf{xx}}(e^{j\omega}) \tag{4.3}$$

(a) The equation above suggests a method of system identification—namely, record the input and output of a system and take the ratio of the cross spectrum to the auto-spectrum of the input to find the complete frequency response. One of the benefits of this method is that it finds both the magnitude and the phase of $H(e^{j\omega})$. If, on the other hand, you took the ratio of the auto-spectra the result would be $|H(e^{j\omega})|^2$.

(b) Use the following system to test this system identification idea.

```
b = 0.2 * [ 1   0   1 ]
a = [ 1   -1.3   0.845 ] .
```

Generate the frequency response of the system and plot both the magnitude and the phase.

(c) Generate a long stream of output data, but also keep the input data. For now make the input WGN. Compute the cross-spectral estimate by averaging together 8 sections when using a Hann window. Plot the result—magnitude and phase.

(d) Now compute the auto-spectrum of the input and divide to get the frequency response. Plot the estimated frequency response and the true $H(e^{j\omega})$ on the same graph. Compare both the magnitude and the phase. Comment on the differences, especially if there are regions where the estimate is very bad.

(e) Redo the experiment for an input signal that is not white. For example, use a first-order filter to "color" the noise and show that the division will remove the effect of the input signal. Experiment with lowpass and highpass filtered noise for the input.

Obviously, a flat input spectrum would be the best test signal, but sometimes lowpass or highpass is all that is available. State a condition on the input spectrum that is necessary for this sort of system identification to work.

Computer-Based Exercises

for

Signal Processing

Modern Spectrum Estimation

Modern Spectrum Estimation

Overview

The methods of spectrum estimation based on the FFT are all limited by the resolution of Fourier analysis which is governed by the "uncertainty principle." In particular, the resolution of two sinusoids separated by $\Delta\omega$ in frequency requires a data length that is greater than $2\pi/\Delta\omega$. However, in some situations the amount of available data is limited, so it might not be possible to increase the window length. For example, the data may have been recorded in an experiment where the event of interest lasts for only a short time, or the memory in the data acquisition hardware may be limited. In the related problem of beamforming, the amount of data available is equal to the number of sensors in the spatial array which, in turn, is limited by the spatial aperture of the array.

Therefore, methods that circumvent the resolution limit of Fourier analysis are of interest. A number of such methods have been developed and popularized over the past twenty years. One of the most well known is the Maximum Entropy Method (MEM). In this packet, the projects will be directed at the use of MEM for spectrum estimation and its relation to modeling and prediction.

Most "modern" spectrum estimation methods are based on either pole-zero models, signal modeling, or eigenvector analysis of the signal covariance matrix. We will only consider a few of these; namely, the autocorrelation method of linear prediction (which is also the MEM spectrum estimate for a Gaussian process), the covariance method of linear prediction, and some signal/noise eigenspace methods suggested by the Pisarenko harmonic decomposition [?]. A complete study of these methods and their application to signal analysis and spectrum estimation would easily take another entire volume.

Background Reading

The methods of modeling for high-resolution spectrum estimation are treated in the books by Marple [?] or Kay [?], as well as Chapters 1, 2 and 6 in [20] and Chapters 4, 5 and 8 in [?]. In addition, two IEEE Press reprint books devoted to Modern Spectrum Estimation are available [?] and [?]. The linear prediction formulation can be found in the tutorial article by Makhoul [?] as well as many other recent texts. The use of eigenvector methods for decomposing the covariance matrix was first introduced by Pisarenko [?]. A practical spectrum estimation method based on this idea is the MUSIC algorithm [?], as well as [?].

Project 1: Maximum Entropy Method

The maximum entropy method (for a Gaussian process) produces a spectrum estimate in the form of an all-pole spectrum [?]

$$P_{\text{mem}}(e^{j\omega}) = \frac{G^2}{|A(e^{j\omega})|^2} = \frac{G^2}{|1 + a_1 e^{-j\omega} + a_2 e^{-j2\omega} + \ldots + a_N e^{-jN\omega}|^2} \tag{1.1}$$

where N is the number of poles. The calculation of the parameters $\{G, a_1, a_2, \ldots, a_N\}$ can be done by solving a set of $(N{+}1) \times (N{+}1)$ linear equations, which are the normal equations of a least-squares minimization problem [**?**, 6]:

$$\begin{bmatrix} r[0] & r^*[1] & r^*[2] & \dots & r^*[N] \\ r[1] & r[0] & r^*[1] & r^*[2] & \dots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & r^*[1] \\ r[N] & \dots & r[2] & r[1] & r[0] \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ a_2 \\ \vdots \\ a_N \end{bmatrix} = \begin{bmatrix} G^2 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{1.2}$$

These equations are based on the autocorrelation sequence $r[\ell]$ of the signal being analyzed.

$$r[\ell] = \frac{1}{L} \sum_{n=0}^{L-1} x^*[n] x[n+\ell] \tag{1.3}$$

The signal $x[n]$ must be extended by padding with zeros whenever the argument $(n+\ell)$ in (1.3) is greater than $L-1$. This method is also referred to as the "autocorrelation method" for obvious reasons [**?**]. An alternate method, called the "covariance method" will be treated later in Exercise 2.1.

Hints

For computing the autocorrelation when the sequence is very long, the MATLAB function `xcorr` is not suitable, because it computes $r[\ell]$ for all possible lags. It would be better to write a function based on the FFT that will produce only the lags $\ell = 0, 1, 2, \dots, N$ needed in (1.2). See the M-file `acf` in Appendix A which was also used in Project 3 of the Packet on *Stochastic Signals.*

In order to present the results that show variability of the spectral estimates, run the experiment about 10 times and make a scatter plot of the complex roots of the denominator polynomial $A(z)$. See `help zplane` for plotting in the format of Fig. 1. Another way to make the comparison is to plot $P_{\text{mem}}(e^{j\omega})$ for a number of runs as shown in Fig. 2.

Exercise 1.1: MEM function

Write a MATLAB function that will compute the MEM spectral estimate via (1.1–1.3). This consists of two steps:

1. Autocorrelation estimation according to (1.3).
2. Solution of the normal equations (1.2). Be careful, because there are unknowns on both sides of (1.2). The lower N equations must be solved first for the coefficients $\{a_k\}$, then G can be found.

The function should return two outputs: the vector of coefficients $\{a_k\}$ of the denominator polynomial, and the gain term G which can then be used in (1.1).

Exercise 1.2: MEM Estimate

Test your MEM function on the following system, which was also used in the Packet on *FFT Spectrum Estimation.*

```
b = 5
a = [ 1   -1.3   0.845 ]
```

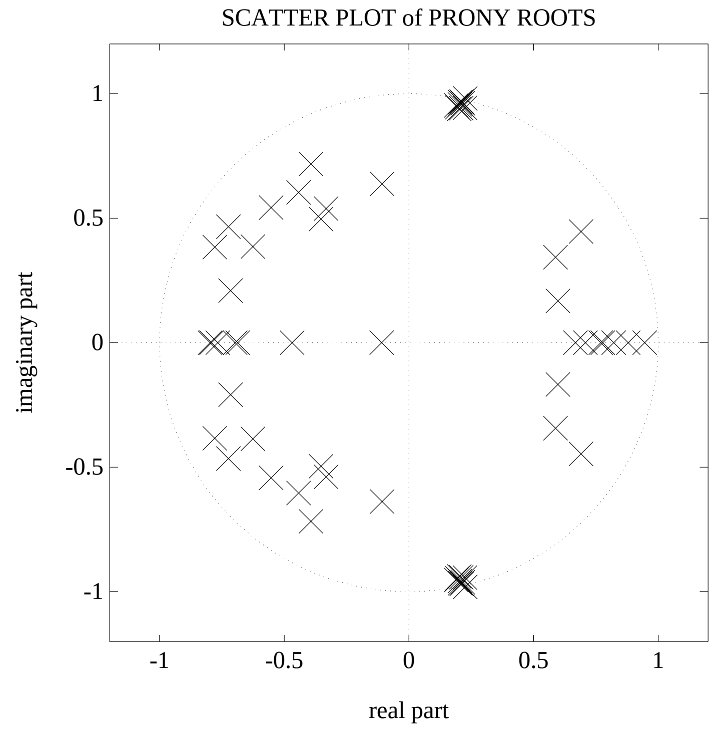


Figure 1: Scatter plot of the roots from ten different 6-pole models of a sinusoid in noise. Note the tight clustering of roots on the unit at the frequency $\omega = 2\pi(0.217)$.

The vectors `a` and `b` define the coefficients of an all-pole filter that will be driven by white Gaussian noise to produce the test signal.

(a) Find the frequency response of the filter, and then plot the true power spectrum of the output process. Notice that this is an "all-pole" spectrum

(b) Take a 256-point section of the output signal (after the transient has died out) and apply MEM to compute the all-pole spectrum estimate with a model order of $N = 2$. Since the system has two poles, this choice for N is large enough that MEM might give an exact match to `a`. Compare the estimated coefficients $\{a_k\}$ to the true values.

In order to make a fair comparison, this experiment must be run many times to observe the variability in the estimated $\{a_k\}$. In order to present the results, run the experiment 10 or 20 times and make a scatter plot of a_2 versus a_1 each time. Alternatively, you could generate a scatter plot of one of the complex roots of the denominator polynomial $A(z)$ as in Fig. 1. Comment on the variability that you observe in these scatter plots.

(c) Another way to make the comparison is to plot $P_{\text{mem}}(e^{j\omega})$ and the true power spectrum on the same graph. As in the previous part, this comparison should be repeated many times and then all the estimated spectra plotted together as in Fig. 2. Alternatively, you could plot the difference between the true and estimated spectra and observe the maximum deviation between the two. Explain why the largest deviation seems to lie near the peak in the true spectrum.

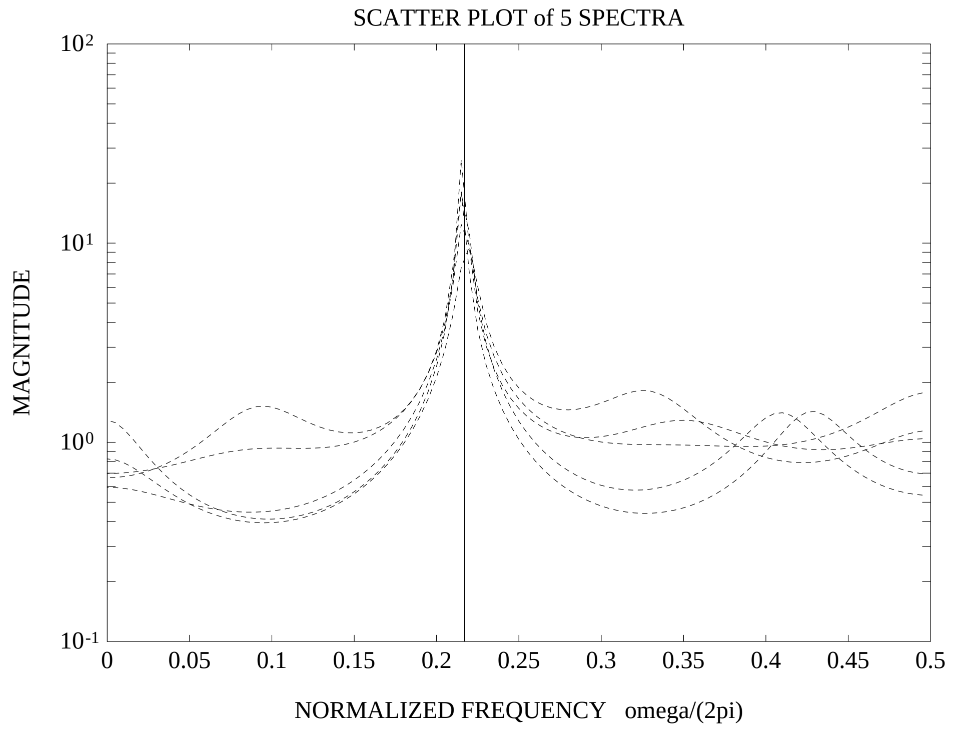


Figure 2: Scatter plot of the spectra for five different models of a sinusoid in noise. Note the tight clustering of peaks near the correct frequency $\omega = 2\pi(0.217)$.

(d) Now try to determine experimentally whether or not the MEM estimate will ever be exactly the same as the true power spectrum. Try section lengths that are hundreds of points long, and also thousands of points long (if feasible on your computer). Compare on the basis of the power spectrum scatter plot, and also by making a scatter plot of the coefficients $\{a_k\}$. From your tests is it possible to claim that the variability of the estimates (as shown via scatter plots) decreases as L increases?

(e) In an attempt to minimize the length of the input data used, while still getting a good estimate, the data can be windowed with something other than a rectangular window. Experiment with using a Hamming window on the data prior to computing the autocorrelation estimate (1.3). Rerun the scatter plot of the poles, as in part (b), for $L = 128$ and $L = 256$, in order to compare the Hamming results to those of the boxcar window.

Exercise 1.3: MEM with high model order

In MEM, the number of poles must be selected *a priori*. It is an advantage to know the true value of N, but the MEM estimation procedure is robust enough that overestimating the model order N is not disastrous. In fact, with noisy data MEM seems to give better answers when the model order is a bit higher than the minimum. In this exercise, we study the performance of MEM when the model order is increased.

(a) Generate a signal using the same all-pole filter as in Exercise 1.2. Try higher model orders, e.g., $N = 3, 4, 6, 8, 12, \ldots$, and, for each N, make scatter plots of five spectral estimates to illustrate the variability in the match between the true spectrum and the MEM estimate. Keep the length of the data sequence fixed, but very long, say $300 < L < 1000$. Does the quality of the estimate improve as N is increased? Does the variability in the estimate change as the model order is increased? The answer may depend on whether you look only near the peak of the true spectrum.

(b) Make a scatter plot of the pole locations for these higher order models and compare to the true pole locations. Explain which poles are the most significant in matching the true spectrum by looking not only at the pole positions, but also at the residues of the poles in a partial fraction expansion of the all-pole transfer function (see `residuez` in MATLAB). Note that the poles (for the autocorrelation method) must always be inside the unit circle.

Exercise 1.4: Pole-Zero case

Now take a test spectrum that is not all-pole.

```
b = [ 1   +1.3   0.845 ]
a = [ 1   -1.3   0.845 ]
```

Redo the previous exercise, with a fixed value for L and with a model order of $N = 6$. Comment on the differences caused by the numerator $B(e^{j\omega})$ in the true spectrum.

Project 2: Spectrum Estimates based on Linear Prediction

Some variations on MEM can provide better performance in situations such as determining frequencies for narrowband signals, and for sinusoids in noise. In this project, we consider three variations of MEM which are all based on the same underlying linear prediction problem. Each of these will produce an all-pole spectral estimate of the form (1.1).

The MEM and its relative, the covariance method, both rely on the solution of a linear prediction problem to find the denominator polynomial $A(z)$ in an all-pole model for the signal. This linear prediction operation can be written as a set of linear equations, which is usually over-determined, i.e., more equations than unknowns. In matrix form, the prediction error equations are:

$$\begin{bmatrix} x[0] & 0 & 0 & \ldots & 0 \\ x[1] & x[0] & 0 & \ldots & 0 \\ x[2] & x[1] & x[0] & \ldots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots \\ x[N-1] & x[N-2] & x[N-3] & \ldots & x[0] \\ \vdots & \ddots & \ddots & \ddots & \ddots \\ x[L-2] & x[L-3] & x[L-4] & \ldots & x[L-N+1] \\ \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ldots & x[L-1] & x[L-2] & x[L-3] \\ 0 & \ldots & 0 & x[L-1] & x[L-2] \\ 0 & \ldots & 0 & 0 & x[L-1] \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_N \end{bmatrix} = - \begin{bmatrix} x[1] \\ x[2] \\ x[3] \\ \vdots \\ x[N] \\ \vdots \\ x[L-1] \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad (2.1)$$

In a more compact form, we have

$$\mathbf{X}\mathbf{a} = -\mathbf{x} \tag{2.2}$$

which can be solved for $\mathbf{a}$ by minimizing the squared error between the two sides of (2.2), $\mathbf{e} = \mathbf{X}\mathbf{a} - (-\mathbf{x})$.

$$\min_{\mathbf{a}} \mathbf{e}^H \mathbf{e} \; = \; \min_{\mathbf{a}} (\mathbf{X}\mathbf{a} + \mathbf{x})^H (\mathbf{X}\mathbf{a} + \mathbf{x}) \tag{2.3}$$

The minimum error solution in (2.3) is the solution to the normal equations:

$$(\mathbf{X}^H \mathbf{X})\mathbf{a} = -\mathbf{X}^H \mathbf{x} \tag{2.4}$$

assuming that the covariance matrix $\mathbf{\Phi} = \mathbf{X}^H\mathbf{X}$ is invertible.

The covariance method differs from the autocorrelation method only in that $\mathbf{X}$ is formed from a subset of the rows in (2.1). The autocorrelation method uses all the rows; the covariance method, only those rows of $\mathbf{X}$ and $\mathbf{x}$ that contain no zeros, i.e., starting from the row whose first entry is $x[N-1]$ and ending with the row whose first entry is $x[L-2]$. In MATLAB the matrix $\mathbf{X}$ can be generated with the M-file `convmtx`, or with the M-file `convolm` in Appendix A.

Exercise 2.1: Covariance Method

The covariance method computes the denominator $A(e^{j\omega})$ for (1.1) via a slightly different set of normal equations. If we reduce (2.4) to a form similar to (1.2), the (i, j)-th entry in the matrix $\mathbf{\Phi}$ of the normal equations is:

$$\phi(i,j) = \sum_{n=N}^{L-1} x^*[n-i]x[n-j] \qquad\qquad i,j = 1,2,\ldots,N \tag{2.5}$$

In effect, the range of summation is changed so that the same number of non-zero terms $(L - N)$ are always involved. The right-hand side of (2.4) is a vector whose elements are $-\phi(i, 0)$.

The MATLAB function for solving the covariance method is called `prony`, because Prony's method of exponential modeling gives the same answer. See `help prony` from the MATLAB signal processing toolbox for details. Other aspects of Prony's method are treated in Chapter 11 in the project on *Linear Prediction.*

(a) Use the covariance method on the same example as in Exercise 1.2. Make a scatter plot of the pole locations, for the case $N = 2$ and $L = 256$. Compare to the results from the autocorrelation method (MEM), in order to determine if this estimate has lower variability.

(b) Plot several spectra as was done in Exercise 1.2(c) and then compare to the MEM spectrum estimate.

(c) Test the covariance estimate for a higher model order, e.g., $N = 8$. Plot the pole locations for ten different runs and observe whether or not the poles ever move outside the unit circle. In this case, two of the poles should closely model the true poles of the second-order system, while the other six could lie anywhere.

Exercise 2.2: Forward-Backward Covariance Method

The covariance method has a small drawback in that it limits the number of prediction error equations used when forming $\mathbf{X}$ in (2.2). Furthermore, to have more equations than unknowns, we need $L-N \geq N$. Getting more equations L would require more data. If this is not feasible, it is possible to form more equations by using the "forward-backward" method of linear prediction. This strategy is based on the idea that a random signal has the same properties if it is flipped in time. Therefore, backward prediction of the signal can be done with the same prediction operator $A(z)$.

For the covariance method, this works as follows: augment the matrix $\mathbf{X}$ by appending $L-N$ rows formed by running the predictor over a signal that is the flipped and conjugated version of $x[n]$. In this case, the covariance matrix $\mathbf{\Phi}$ generated in (2.4) will have entries:

$$\phi_{fb}(i,j) = \sum_{n=N}^{L-1} x^*[n-i]x[n-j] + \sum_{n=N}^{L-1} x_r^*[n-i]x_r[n-j] \qquad i,j = 1,2,\ldots,N \tag{2.6}$$

where $x_r[n] = x^*[L–1-n]$ is the contribution from the backward prediction. The right-hand side of (2.4) has elements $-\phi_{fb}(i,0)$.

(a) Write an M-file to implement the forward-backward covariance method. The input arguments will be the signal $x[n]$ and the model order N.

(b) Test this function on the same case as in Exercise 2.1. Redo parts (a)–(c) of that exercise. Since this method works well for sinusoids in noise, reserve judgment on its performance until testing it in the next project.

Exercise 2.3: An SVD-based Solution

Tufts and Kumaresan [?] have proposed a solution to the linear prediction problem that gives more robust behavior in lower SNR cases. It is based on the singular value decomposition (SVD). The solution of (2.4)

$$\mathbf{a} = -[(\mathbf{X}^H\mathbf{X})^{-1}\mathbf{X}^H]\,\mathbf{x} = -\mathbf{X}^\dagger\,\mathbf{x}$$

involves a special case of the "pseudo-inverse" of $\mathbf{X}$, denoted by $\mathbf{X}^\dagger$. Computation of the pseudo-inverse $\mathbf{X}^\dagger$ generally relies on the singular value decomposition (SVD) of the matrix $\mathbf{X}$. The book by Golub and Van Loan [?] is an excellent reference for a theoretical background on SVD.

In MATLAB, the M-file `svd` will compute the singular value decomposition, and the M-file `pinv` implements a pseudo-inverse by first computing an SVD and then discarding relatively small singular values prior to forming the inverse. Consult the MATLAB `help` on these functions, as well as [?, 6] for more details on SVD and the pseudo-inverse. The M-file below is a modification of the MATLAB function `pinv` to compute a rank-r pseudo-inverse. For example, if we specify $r = 3$, then the three largest elements along the diagonal of `S` will be used to form the rank-3 inverse. The function `pseudinv` can be used later to solve the prediction error equations.

```
function Ainv = pseudinv(A,r)
%PSEUDINV   Pseudo-inverse of rank r.
```

```
% usage:    Ainv = PINV(A,r)
%   produces the rank-r inverse of A, from the SVD of A.
%   Only r singular values are retained for the inverse
%   with the exception that singular values less than
%   MAX(SIZE(A)) * S(1) * EPS are discarded.
%      Ainv = PINV(A) uses all possible singular values.
% See also  SVD and PINV

[U,S,V] = svd(A);
S = diag(S);
keep = sum( S > (max(size(A)) * S(1) * eps) );
if (nargin == 1)
   r = keep;
else
   r = min( r, keep );
end
S = diag( ones(r,1) ./ S(1:r) );
Ainv = V(:,1:r) * S * U(:,1:r)';
```

One view of the pseudo-inverse $\mathbf{X}^{\dagger}$ is that it is a low-rank approximation to the inverse of the original matrix. The rank of the inverse can be controlled by the number of singular values that are retained when the pseudo-inverse is constructed. With this view, both the covariance method and the autocorrelation method would be called "full-rank" inverses, because the solution of (2.4) is the same as an SVD inverse in which all singular values are kept.

When applied to linear prediction, the philosophy is the following: The signal matrix $\mathbf{X}$ contains values that are due to both the signal and the noise. If we know *a priori* the number of signal components is r, then in the noise-free case the rank of $\mathbf{X}$ would be r. However, the noise usually increases the rank to its maximum. Since we need to "invert" $\mathbf{X}$ to solve for $\mathbf{a}$, it is reasonable that we should design an inverse based solely on the signal. The assumption of the rank-r inversion is that the larger singular values and their singular vectors correspond to the signal components, because they span the same subspace as the signal vectors. Therefore, the rank parameter r is chosen to match the expected number of signal components. For example, if the signal were generated by a second-order filter, we would take $r = 2$.

(a) Write an M-file to implement the SVD low-rank covariance method. The input arguments will be the signal $x[n]$ and the rank r which is the suspected number of signals present. The data matrix $\mathbf{X}$ can be defined by either the forward-backward prediction error, or the forward only. The model order N can be larger, but only r poles should be significant.

(b) Test this function on the same case as in Exercise 2.1. Redo parts (a)–(c) of that exercise. Again, this method should work well for sinusoids in noise, so its performance will be tested in the next project.

Project 3: Narrowband Signals

This project is nearly the same as Project 3 in the Packet on *FFT Spectrum Estimation.* However, we now have four different all-pole spectral estimators (from Project 2) to evaluate. For this project, we synthesize the test signal using a filter with two narrowband peaks, and compare the estimation of its power spectrum among the four all-pole estimators and to the Welch-Bartlett result done in Project 3 of the Packet entitled *FFT Spectrum Estimation.* For the most part, we will investigate the resolution of closely spaced narrow peaks and try to minimize the data length needed. Finally, the case of sinusoids in noise and the influence of SNR on the estimation of frequency will be studied.

Hints

Make sure that you have a M-files for doing all four all-pole estimates considered in Project 2. As in the previous project, the spectrum estimates will vary for different realizations of the random process under scrutiny. This variation is of some interest, so a scatter plot of the spectra or of pole locations is the preferred way to present the results.

Exercise 3.1: Resolution of Peaks

Construct a transfer function $H(z)$ with four poles at:

$$\text{POLES} = \{0.99\angle \pm 88^\circ, 0.99\angle \pm 92^\circ\} \tag{3.1}$$

The numerator should be a constant, so that $H(z)$ is all-pole.

(a) Determine the transfer function $H(e^{j\omega})$, and then plot the true power spectrum. From this plot, determine the frequency separation of the two peaks, $\Delta\omega$. From the value of $\Delta\omega$, estimate the FFT length that would be need to resolve the two peaks via Fourier analysis with a Hamming window.

(b) Generate the synthetic test signal by exciting $H(z)$ with white Gaussian noise to produce the output signal, $y[n]$. In addition, determine the length of the transient by synthesizing the impulse response of the filter. In this case, the transient will be rather long, so make sure that you ignore enough points at the beginning of the signal when selecting a segment for spectrum estimation.

(c) Choose one of the four all-pole estimators and process the signal. The primary objective is to resolve the two peaks in the spectral estimate with the minimum segment length L. Therefore, some experimentation with the window length L, and perhaps with the model order N, or rank r, will be necessary. You might also try different window types. When you have finally settled on specific values of L and N, run ten cases and plot all the spectra together. This type of scatter plot will show how the resolving capability varies for different parts of the random signal. Finally, since only resolution is of interest, the region near the peaks should be blown up when making the plot.

(d) Compare the segment length L determined in part (c) to the FFT length needed in the Welch-Bartlett procedure, using the Hamming window. In addition, compare plots of the spectral estimates for both cases and comment on the differences near the peaks and in the sidelobe regions. This part requires that you refer to Project 3 in the Packet on *FFT Spectrum Estimation.*

(e) The bias in the spectrum estimate can be studied by concentrating on the shape of $P_{\text{mem}}(e^{j\omega})$ near the peaks. The shape of the the spectrum estimates must be compared to the true shape at the peaks. Thus it will be necessary to zoom in on the plot near the peak locations and make a plot with all the estimates superimposed on the true spectrum.

In order to distinguish bias effects from variance, consider the following two questions: Are the differences you observe consistent over all the spectrum estimates? If the estimated spectra were averaged, would the result be closer to the true spectrum?

If feasible, try processing the data with a longer window length to see whether or not the bias can be reduced by increasing L.

(f) *Optional:* Change the system slightly by placing zeros in the transfer function at $z = \pm j$, and redo the entire experiment.

Exercise 3.2: Small Peaks

In the FFT, the sidelobes presented a problem because small peaks could be obscured by the sidelobes of a large peak. Apparently, an all-pole spectrum estimate such as MEM has no sidelobes, so it may not miss a small peak. Test this idea on the example below (which is also from the Packet entitled *FFT Spectrum Estimation*).

(a) Use the following system to generate a random output signal with known power spectrum.

$$H(z) = \frac{2.2 - 2.713z^{-1} + 1.9793z^{-2} - 0.1784z^{-3}}{1 - 1.3233z^{-1} + 1.8926z^{-2} - 1.2631z^{-3} + 0.8129z^{-4}}$$

Plot the true spectrum; and then evaluate the peak heights which should be quite different.

(b) Process the signal with each of the four all-pole estimators. Try to minimize the window length subject to the constraint that the two peaks be resolved and the relative peak heights be preserved. Recall that using a Hamming window with the FFT method is crucial if the small peak is to be identified at all.

(c) *Optional:* Redo the experiment with a new transfer function in which the smaller peak is even closer to the larger one. Choose the poles and residues in a partial fraction expansion to control the true spectrum. See how close you can bring the smaller one to the larger one before the methods fail.

Exercise 3.3: Sine Wave Plus Noise

For the case of sine waves in noise, the model-based methods should work well because each sinusoid corresponds to a pole-pair in the z-transform. Since the sine wave component of the signal is deterministic, it should turn out that Prony's method will do an excellent job when the SNR is high. For lower SNR, the SVD-based method should be superior.

(a) Create a signal containing three sinusoids in a background of white Gaussian noise

#1	$\omega_1 = \pi/3$	$A_1 = 100$	$\phi_1 = 45°$
#2	$\omega_2 = 0.3\pi$	$A_2 = 10$	$\phi_2 = 60°$
#3	$\omega_3 = \pi/4$	$A_3 = 1$	$\phi_3 = 0°$

The noise variance should be set at $\sigma_{\mathbf{v}}^2 = 100$, although other values of SNR need to be tried later on.

(b) First, do the estimation with Prony's method, i.e., the "covariance" method. Show how to extract the frequencies of the sinusoids directly from the roots of the predictor polynomial $A(z)$. Experiment with a model order that may be higher than the minimum needed. Try to do the processing with about 50 data points; then try to minimize the length of data segment used. Obviously, there will be a tradeoff among the parameters L, N and the accuracy of the frequency estimates. Run ten trials of the processing and plot the results together to illustrate the variability of the estimate due to additive noise.

(c) Next try MEM, the "autocorrelation" method, with a Hamming window. For the same data length L and model order N, compare the accuracy of the frequency estimation to that of Prony's method. This will necessitate running 10 estimates and presenting the results as a scatter plot for evaluation.

(d) Repeat part (b) for the forward-backward covariance method of Exercise 2.2.

(e) Repeat part (b) for the SVD-based method from Exercise 2.3.

(f) Test for different SNRs, $\sigma_{\mathbf{v}}^2 = 10$ and 1000. Show that an excellent estimate of the frequencies can be obtained with a very short data segment when $\sigma_{\mathbf{v}}^2 = 10$. Determine which methods will succeed when $\sigma_{\mathbf{v}}^2 = 1000$.

(g) For these all-pole estimators, if you want to determine the power in the sinusoid, devise a calculation that will do this. Consider the fact that the peak height in the all-pole spectrum may not be equal to the power in the sinusoid. In fact, the peaks in the all-pole spectrum cannot be correct if they have finite bandwidth, since the true spectrum for the sinusoidal case consists of impulses in ω.

Project 4: Eigenvector-Based Methods

When the spectrum estimation problem is primarily concerned with extracting sinusoids in noise, some recent methods based on the eigen-decomposition of the covariance matrix are among the best for high SNR situations [**?**, Chap. 13]. These methods have also been popularized for angle-of-arrival (AoA) estimation in array processing. In this case, the direction of an arriving plane wave gives a sinusoidal variation across the array, so the measurement of this "spatial" frequency amounts to direction finding.

Hints

See `eig` for computation of the eigenvectors, and `svd` for the singular value decomposition [**?**].

Exercise 4.1: Pisarenko's Method

Many of the eigenvector-based methods are descended from Pisarenko's method [?], which is based on an amazing property of the covariance matrix.

> Starting with any positive-definite Toeplitz covariance matrix, it is possible to derive a representation for the covariance matrix as a sum of sinusoids plus white noise. If $\mathbf{\Phi}$ is an $L \times L$ covariance matrix, then the representation can be written:
>
> $$\mathbf{R} = \sum_{k=1}^{\mu} \alpha_k \mathbf{v}(\omega_k)\mathbf{v}^H(\omega_k) \; + \; \sigma^2 \mathbf{I} \tag{4.1}$$
>
> where $\mu = L-1$, and $\mathbf{v}(\omega_k)$ is a "steering" vector
>
> $$\mathbf{v}(\omega_k) = [\, 1 \quad e^{j\omega_k} \quad e^{j2\omega_k} \quad \ldots \quad e^{j(L-1)\omega_k} \,]^T \tag{4.2}$$

In the array processing problem, the frequency ω_k can be related to direction of arrival θ via

$$\omega_k = \frac{2\pi\Delta x}{\lambda} \sin\theta$$

where Δx in the inter-sensor spacing and λ is the wavelength of the propagating plane wave. In this case, it is natural to call $\mathbf{v}(\omega_k)$ a "steering vector."

The algorithm that produces this representation (4.1) must find the frequencies $\{\omega_k\}$, which it does by factoring a polynomial. Amazingly enough, the polynomial of interest is defined by one of the eigenvectors of $\mathbf{\Phi}$. The resulting procedure suggests many generalizations.

(a) For the experiment in this exercise, create a positive-definite Toeplitz matrix. This can be done with `rand` and `toeplitz`, but some care is needed to satisfy the positive-definite constraint. One shortcut to building a positive-definite matrix is to increase the value of the diagonal elements—eventually, the matrix will become positive definite. After you create the Toeplitz covariance matrix, verify that it is positive definite.

(b) Since all the eigenvalues of the covariance matrix will be positive, we can identify the minimum eigenvalue and its corresponding eigenvector—called the "minimum eigenvector." It turns out that the noise power (σ^2) needed in (4.1) is equal to the minimum eigenvalue of $\mathbf{\Phi}$.

For your matrix $\mathbf{\Phi}$, determine σ^2 and extract the minimum eigenvector for use in the next part. There is a small possibility that your matrix from part (a) might have a minimum eigenvalue ($\lambda_{\min}$) with multiplicity greater than one. In this case, the minimum eigenvector is not unique and the following part will not have the correct property. If so, generate another matrix $\mathbf{\Phi}$ so that ($\lambda_{\min}$) has multiplicity one.

(c) The amazing property of $\mathbf{\Phi}$ discovered by Pisarenko [?] is that the frequencies (ω_k) are obtained from the roots of a polynomial defined by the minimum eigenvector, $\mathbf{v}_{\min}$. Each eigenvector of $\mathbf{\Phi}$ has L elements, so these can be used as the coefficients of a degree $L-1$ polynomial:

$$V_{\min}(z) = \sum_{\ell=0}^{L-1} v_\ell z^{-\ell} \tag{4.3}$$

Verify the following property for your example. All roots of the minimum eigenvector polynomial (4.3) lie on the unit circle, i.e., have magnitude one. Plot the root locations with `zplane` (Appendix A) to see them all on the unit circle.

(d) In order to complete the verification of the sinusoid plus noise representation of the covariance matrix, it is necessary to find the amplitudes α_k in (4.1). This is relatively easy because the α_k enter the problem linearly. Once the values of ω_k and σ^2 are known, the first column of $\mathbf{\Phi}$ can be written in terms of the α_k to get $L-1$ equations in $L-1$ unknowns. Use MATLAB to set up and solve these equations.

(e) Now that all the parameters of the representation are known, synthesize $\mathbf{\Phi}$ according to the right-hand side of (4.1) and check that the result is equal to $\mathbf{\Phi}$ from part (a).

Exercise 4.2: Orthogonality of signal and noise subspaces

The eigenvectors of $\mathbf{\Phi}$ can be used to decompose $\mathbf{\Phi}$ into two orthogonal subspaces, called the noise subspace and the signal subspace. The noise subspace is spanned by the minimum eigenvector; the signal subspace by the steering vectors. The dimensionality of the noise subspace could be greater than one if the matrix $\mathbf{\Phi}$ has a repeated minimum eigenvalue.

(a) The roots of $V_{\min}(z)$ are all on the unit circle and, therefore, all of the form $z = e^{j\omega_k}$. Furthermore, the polynomial evaluation $V_{\min}(z)$ at $z = e^{j\omega_k}$ is equivalent to the inner product of $\mathbf{v}_{\min}$ with $\mathbf{v}(\omega_k)$.

$$V_{\min}(e^{j\omega}) = \mathbf{v}^H(\omega)\mathbf{v}_{\min} \tag{4.4}$$

Each root $z = e^{j\omega_k}$. can be used directly to define a steering vector. Use MATLAB to create the $L-1$ steering vectors, and verify that each one is orthogonal to the minimum eigenvector. This equation (4.4) is essential in the generalizations to follow.

(b) The algorithm described in Exercise 4.1 becomes a bit more complicated when the minimum eigenvalue is repeated. In this case, less than $L-1$ sinusoids are needed in the representation (4.1), and every one of the minimum eigenvector polynomials will have only a subset of roots on the unit circle. Interestingly enough, all of these polynomials will have just this subset of roots in common.

To illustrate this behavior, process the 5×5 positive-definite Toeplitz matrix below, which has a repeated minimum eigenvalue. Apply the procedure as before to derive the representation (4.1). In addition, plot the roots of all the minimum eigenvector polynomials to show which roots are in common.

$$\mathbf{\Phi} = \begin{bmatrix} 3 & 1-j2 & -1-j\sqrt{2} & -1 & 1-\sqrt{2} \\ 1+j2 & 3 & 1-j2 & -1-j\sqrt{2} & -1 \\ -1+j\sqrt{2} & 1+j2 & 3 & 1-j2 & -1-j\sqrt{2} \\ -1 & -1+j\sqrt{2} & 1+j2 & 3 & 1-j2 \\ 1-\sqrt{2} & -1 & -1+j\sqrt{2} & 1+j2 & 3 \end{bmatrix}$$

(c) Write an M-file that will do the three steps of processing required in Pisarenko's method: find $\lambda_{\min}$, then factor $V_{\min}(z)$ to get ω_k, and finally compute α_k. Make your M-file work correctly for the extended case of repeated eigenvalues, by using the idea suggested in part (b) of this exercise. Save this file for testing later on in Project 5.

Exercise 4.3: MUSIC

In the application of Pisarenko's method to actual data, the covariance matrix $\mathbf{\Phi}$ must be estimated. This is a significant issue because the representation (4.1) is exact. In practice, a method based on approximation might be preferable. An important algorithm along these lines is MUSIC (MUltiple SIgnal Classifier) introduced by Schmidt [**?**]. The MUSIC algorithm avoids the factoring step and produces a spectrum estimate as in MEM.

In Pisarenko's method, we assume that the noise subspace can be characterized exactly by the minimum eigenvector. Sometimes the minimum eigenvalue has a multiplicity greater than one, so the dimensionality of the noise subspace is also greater than one. In fact, if we knew that the data had only 2 sinusoids, then we would expect the noise subspace to have dimension $L-2$. But with real data it is unlikely that $\mathbf{\Phi}$ would have a minimum eigenvalue that repeats $L-2$ times. In practice, we would probably observe 2 large eigenvalues and $L-2$ small ones. In order to identify the correct number of significant eigenvalues we need a threshold below which all small eigenvalues are associated with the noise subspace.

The MUSIC estimate is

$$P_{\text{music}}(e^{j\omega}) = \frac{1}{\mathbf{v}^H(\omega)\mathbf{\Phi}_n\mathbf{v}(\omega)} \tag{4.5}$$

where $\mathbf{\Phi}_n$ is the noise subspace portion of $\mathbf{\Phi}$.

$$\mathbf{\Phi} = \mathbf{\Phi}_s + \mathbf{\Phi}_n = \sum_{i=1}^{\mu} \mathbf{v}_i\mathbf{v}_i^H + \sum_{j=\mu+1}^{L} \mathbf{v}_j\mathbf{v}_j^H$$

(a) Write an M-file to implement the MUSIC technique. The inputs should be the covariance matrix $\mathbf{\Phi}$ and a threshold percentage that determines how many λ_i will be assigned to the noise subspace. If this percentage is called η, then λ_i belongs to the noise subspace when

$$|\lambda_i - \lambda_{\min}| < \eta|\lambda_{\max} - \lambda_{\min}|$$

Finally, the MUSIC function should return two outputs: the power spectrum $P_{\text{music}}(e^{j\omega})$ and the number of signal components found μ.

(b) Test your function by generating a 7×7 covariance matrix $\mathbf{\Phi}$ for data that contains two sine waves plus white Gaussian noise with a variance equal to $\sigma_{\mathbf{v}}^2 = 0.01$. Make the amplitudes os the sines equal, and pick the frequencies at random. Take $\mathbf{\Phi}$ to be the true covariance in the form (4.1), but perturb the matrix entries slightly. Make sure that the perturbed $\mathbf{\Phi}$ is still Toeplitz. Set the threshold $\eta = 0.2$, or higher, so that the MUSIC process finds exactly two complex signal components. Plot the spectrum to see if the estimated frequency is correct.

Extensive testing of the MUSIC algorithm on synthetic data will be undertaken in the next project.

Exercise 4.4: Kumaresan-Tufts Method

An improved estimate of the polynomial (4.4) for annihilating the signal subspace can be found using the singular value decomposition [**?**]. When the signal/noise subspace decomposition is performed, the spectral estimate is constructed by finding steering vectors that are orthogonal to the noise subspace. This is equivalent to finding a polynomial $D(z)$ whose coefficients satisfy:

$$\mathbf{E}_s^H \mathbf{d} = \mathbf{0} \tag{4.6}$$

where the $L \times \mu$ matrix $\mathbf{E}_s$ is formed from the signal subspace eigenvectors—one in each column. The vector $\mathbf{d} = [\,1 \quad d_1 \quad d_2 \quad \ldots \quad d_{L-1}\,]$ will be orthogonal to the signal subspace, so it can be used to find steering vectors that satisfy $\mathbf{v}^H(\omega)\mathbf{d} = 0$. Some of the roots of $D(z)$ should lie on the unit circle and give the frequencies of the sinusoidal components in the data. In (4.6) the polynomial coefficients $\mathbf{d}$ are seen to annihilate the members of the signal subspace.

Once $\mathbf{d}$ is calculated, the spectral estimate is formed in a way similar to the all-pole forms:

$$P_{\text{KT}}(e^{j\omega}) = \frac{1}{|\mathbf{v}^H(\omega)\mathbf{d}|^2} \tag{4.7}$$

Equation (4.6) is under-determined, but can be solved with a pseudo-inverse computed from the SVD (singular value decomposition). See Exercise 2.3 for a discussion of the pseudo-inverse, including an M-file.

(a) Write an M-file to compute the Kumaresan-Tufts estimate (4.7). The inputs should be the covariance matrix and a threshold η for separating the signal and noise subspaces.

(b) Repeat the test in part (b) of Exercise b.

(c) In addition, to showing the spectrum estimate, plot the root locations of $D(z)$ to see where the extra roots go.

Project 5: Testing with Synthetic Signals

In order to test the various signal-noise subspace methods, we need a method for generating noisy data. The typical test involves estimating the frequencies of a few sinusoids in a background of white noise. Therefore, we need to generate data according to the model given in (4.1). The crucial difference, however, is that the covariance matrix $\mathbf{\Phi}$ must be estimated from a finite segment of the signal. Thus $\mathbf{\Phi}$ might not be exactly correct for the data being observed. In fact, $\mathbf{\Phi}$ might not even have the correct form, i.e., Toeplitz.

In this project, we develop an M-file for generating noisy data and we use it to test several of the algorithms discussed in the previous project. The test scenario corresponds to the direction estimation problem. Additional test cases for extracting sinusoids in noise can be found in [**?**, **?**].

Exercise 5.1: Generating Sinusoids in Noise

The model in (4.1) states that the covariance matrix $\mathbf{\Phi}$ is composed of two parts: sinusoids plus white noise. Therefore, we can generate a signal according to that rule and see how closely the form of $\mathbf{\Phi}$ is matched when $\mathbf{\Phi}$ is estimated directly from the data.

The following equation describes the signal generation process:

$$x[n] = \sum_{i=1}^{\mu} A_i e^{j(\omega_i n + \phi_i)} + \frac{\sigma_n}{\sqrt{2}} v[n] \qquad \text{for } n = 0, 1, 2, \ldots, L-1 \tag{5.1}$$

where the phase ϕ_i must be random, and uniformly distributed in the interval $[-\pi, \pi)$; otherwise, the sinusoids will be correlated with one another. The noise $v[n]$ is a complex Gaussian process where both the real and imaginary parts have variance equal to one. The function `rand` can be used to generate uniformly distributed random values, as well as Gaussian white noise with unit variance. The ratio A_i/σ_n determines the SNR; in dB, it is $20\log_{10}(A_i/\sigma_n)$.

(a) The simplest case would be one complex exponential plus noise in (5.1). For $L = 3$, generate $x[n]$ when $\omega_1 = 2\pi(0.234)$, $A_1 = 1.0$ and $\sigma_n = 0.1$. Use this signal to generate an estimate of the covariance matrix $\mathbf{\Phi}$. Explain why the estimate $\mathbf{\Phi}_x$ can be formed by computing the outer product:

$$\mathbf{\Phi}_x = \mathbf{x}\mathbf{x}^H = \begin{bmatrix} x[0] \\ x[1] \\ x[2] \end{bmatrix} [\, x^*[0] \quad x^*[1] \quad x^*[2]\,] \tag{5.2}$$

In particular, verify that this $\mathbf{\Phi}_x$ is Hermitian symmetric. Is $\mathbf{\Phi}_x$ a Toeplitz matrix? Compare the estimated $\mathbf{\Phi}_x$ in (5.2) to the true value from (4.1).

(b) In an actual estimation problem, many "snapshots" of the signal vector $\mathbf{x}_m$ would have to be collected and averaged in order to get an adequate estimate of $\mathbf{\Phi}$ from the data. This can be accomplished by averaging a number of outer products (5.2).

$$\mathbf{\Phi}_x = \frac{1}{M}\sum_{m=1}^{M} \mathbf{x}_m \mathbf{x}_m^H$$

For the two cases considered in parts (a) and (b), compute $\mathbf{\Phi}_x$ for 10 snapshots, and then 100 snapshots or more. Compare these estimates to the true value of $\mathbf{\Phi}$. Does the estimate converge to a Toeplitz form?

(c) Repeat the previous part for $A_1 = 0$ and $\sigma_n = 1$. This is the noise-only case, so the proper form for $\mathbf{\Phi}$ is a scaled identity matrix. Is that what you observe?

Exercise 5.2: M-files for Simulation

(a) Write an M-file to synthesize $x[n]$. The inputs to this function are: the number of sinusoids μ, their amplitudes A_i, the noise power σ_n^2, and the number of signal samples L. Its output is a vector containing the signal values.

(b) Write an M-file to estimate the covariance matrix $\mathbf{\Phi}$ from the signal $x[n]$. This function needs an input to specify the number of snapshots and could then call the M-file for synthesizing $x[n]$.

(c) Write an M-file to generate the true covariance matrix $\mathbf{\Phi}$ given the correct values for ω_i, A_i and σ_n.

(d) Generate a signal containing two sinusoids in noise, and estimate its 4×4 covariance matrix from a large number of snapshots. Let $\omega_1 = 2\pi(0.2)$, $\omega_2 = 2\pi(0.3)$, $A_1 = A_2 = 1.0$ and $\sigma_n^2 = 0.01$. Determine how many shapshots are needed to approximate the true values of $\mathbf{\Phi}$ within 5%.

Exercise 5.3: Testing MUSIC

In this exercise, the MUSIC algorithm will be tested under different signal-to-noise ratios. Refer to Exercise 4.3 for the M-file that implements MUSIC. The test signal should be formed as follows: $x[n]$ consists of two complex exponentials ($\mu = 2$) in noise. The number of data samples (or sensors) is $L = 7$. The amplitudes are equal $A_1 = A_2 = 1.0$, and the noise power is $\sigma_n^2 = 0.01$. The frequencies are $\omega_1 = 2\pi(0.227)$ and $\omega_2 = 2\pi(0.2723)$ If these frequencies are converted to angle θ for an AoA problem, $\omega = 2\pi\Delta x \sin\theta/\lambda$, the angles would be $\theta = 27°$ and $33°$ if the inter-sensor spacing is $\Delta x = \frac{1}{2}\lambda$. This same test signal will be used to evaluate the other algorithms.

(a) Run MUSIC once to see what the eigenvalue spread will be. Then pick a threshold η so that MUSIC will use a noise subspace with $L-2$ eigenvectors. Make sure that the two frequencies can be resolved in a plot of $P_{\text{music}}(e^{j\omega})$; otherwise, lower the noise power σ_n^2.

(b) Run MUSIC ten times with the number of snapshots $M = 16$. Plot all the spectra together to see how well the method performs when resolving the two closely spaced spectral peaks. Zoom in on the frequency range to show only $2\pi(0.2) \leq \omega \leq 2\pi(0.3)$. Furthermore, check whether the correct relative peak heights are obtained.

(c) Since this is a frequency estimation problem, one way of summarizing the behavior of the MUSIC estimate would be to pick the frequency estimates as the location of the two largest peaks in ω and then collect these results from all the runs. From these estimates, the mean and variance of the two frequency estimates can be computed.

(d) Rerun the experiment with $M = 50$ snapshots and with $M = 5$ snapshots and/or with a different SNR ($\sigma_n^2 = 0.25$ and $\sigma_n^2 = 0.0001$). Comment on the performance for these cases.

Note: when the SNR is very high the method should give nearly perfect results, i.e., the variance of the frequency estimates should converge to zero. Likewise, when the number of snapshots becomes very large the variability of the estimate should decrease

Exercise 5.4: Roots of Pisarenko's Method

In the application of Pisarenko's method to actual data, the covariance matrix $\mathbf{\Phi}$ must be estimated. This is a significant issue because the representation (4.1) is exact. Nonetheless, we can use the test case in Exercise 5.3 to examine the root distribution of the minimum eigenvector polynomial.

(a) Generate an estimated covariance matrix $\mathbf{\Phi}_x$ and extract its eigenvectors. Plot the root locations for $V_{\text{min}}(z)$.

(b) Relate the roots near the unit circle to the true frequencies in the data.

(c) Select the $L-2$ smallest eigenvalues and their eigenvectors. Extract the roots of all of these and plot together with `zplane`. Comment on the expected clustering of roots near the true frequencies $z = e^{j\omega_1},\ e^{j\omega_2}$.

Exercise 5.5: Testing the K-T Method

Use the M-file for computing the Kumaresan-Tufts estimate (4.7). Refer to Exercise 5.3 for the test signal needed to study the K-T method.

(a) Run the K-T method once with the dimension of the signal subspace set equal to 2. Make sure that the two frequencies can be resolved; otherwise, lower the noise power σ_n^2.

(b) Run K-T ten times with the number of snapshots $M = 16$. Plot all the spectra together to see how well the method performs when resolving the two closely spaced spectral peaks. Furthermore, check whether the correct relative peak heights are obtained.

(c) Since this is a frequency estimation problem, pick the frequency estimate as the location of the two largest peaks in ω, and calculate the mean and variance of the two frequency estimates. Compare to the results found in Exercise 5.3 (c).

(d) Rerun the experiment with more snapshots and with fewer snapshots and/or with a lower SNR. Comment on the performance for these cases.

(e) Plot the root locations of the $D(z)$ polynomial to see where the extra roots go.

(f) *Optional:* Run the SVD-based covariance method from Exercise 2.3 on this same test data set. Compare to MUSIC and the K-T method.

Chapter 7

Wordlength Effects

January 17, 2007

Overview

This chapter presents simulations of finite wordlength effects that arise in digital filters and in A/D conversion systems.

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{Wordlength Effects}}$

Wordlength Effects

Overview

Under ideal conditions, i.e. if implemented with unlimited wordlength of coefficients and variables, a digital filter behaves as expected, if the design has been done properly. In this case the choice of one of the numerous structures influences only the complexity and thus the achievable speed, if a particular hardware is used. Besides that, the performance will always be the same. But, in reality, the situation is different and by far more complicated:

Coefficients as well as data can be implemented with finite wordlengths only. Quantized coefficients will lead at least to a more or less erroneous behavior of the system, e.g. a different frequency response. The deviation from the expected performance will depend on the chosen structure, i.e. on its sensitivity. It might even happen that the quantization of coefficients turns a stable system into an unstable one.

Another effect will even change the character of the system: Arithmetic operations done with numbers of limited wordlengths usually yield results of larger wordlengths. The necessarily following reduction down to the wordlength the system can handle is a nonlinear process. The resulting nonlinear system might behave completely differently from the desired linear one. That is especially so, if limit cycles occur as a result of an instability.

Of course it can be expected that the various errors become smaller, if the wordlengths of coefficients and data are increased. Therefore it is an important part in the design, to find out the required minimum wordlengths for a satisfying operation of the system to be built or to check, if a satisfying performance with tolerable error can be achieved in a particular case, e.g. if a given integrated signal processor with fixed wordlength is supposed to be used.

For these reasons a rather thorough investigation of the effects explained above is of interest. But since a general analysis of a system working with limited wordlength is rather impossible, a stepwise investigation of the different influences is usually done. That requires, in addition, a model of the system and its signals and thus an approximate description of the real effects. So the influence of quantizing coefficients is studied while ignoring the nonlinearity of the real system. The effect of quantizing the input signal or the result of an arithmetic operation like multiplication by rounding is described by an additional noise source with certain properties, concerning e.g. probability density, power density spectrum and its correlation to the original unquantized signals. Finally, the existence of limit cycles and methods to avoid them have been studied primarily for the rather simple case of a block of second order.

In all cases these separate investigations should not only yield information about the error in particular cases, but also they should lead to rules for the design of a system, such that the desired properties can be achieved with minimum expense.

The objective of this chapter is to give some insight into the different quantization effects. Especially it deals with

(a) the analysis and modelling of signals of limited wordlength. We use the well known description of the error sequence and check by measurements the validity of these common models.

(b) the analysis and modelling of arithmetic operations, when the necessary reduction of the wordlength is done by different methods.

(c) a demonstration of large scale and granular limit cycles. Methods will be shown to avoid them.

(d) the comparison of some structures in terms of coefficient sensitivity by considering their complex frequency response. In particular, we show the favorable sensitivity properties of so-called lossless structures.

(e) the performance of a digital filter, expressed in terms of its complex frequency response and the power density spectrum of the output noise, produced by internal arithmetic errors. These quantities will be measured using a special method, explained in the appendix and compared with the result, found with the usual analysis.

Background Reading

Ch 6 in Discrete-Time Signal Processing by A.V. Oppenheim and R.W. Schafer

Section 8.6 in Advanced Topics in Signal Processing, edit. by J.S. Lim and A.V. Oppenheim.

Project 1: Quantized Signals

In this project the properties of an A/D converter will be investigated. Included in this study are measurements of its I/0 characteristic and the S/N-ratio as a function of the noise power as well as an investigation of the noise. Especially we examine the validity of the common model for describing the noise by its probability density function, its autocorrelation and the crosscorrelation with the input signal.

Project Description

We consider an A/D-converter, the input signal of which is the continuous function $v_0(t)$. Usually its output sequence is described by

$$v[n]_Q = k \cdot Q = v[n] + r[n], \tag{1.1}$$

where

$Q = 2^{-(w-1)}$ is the quantization stepsize in a fixed-point number of wordlength w,

k is an integer with $-2^{w-1} \leq k \leq 2^{w-1} - 1$,

$v[n] = v_0(t = n \cdot T)$, with T being the sampling interval.

Assuming that $v[n]_Q$ is generated by rounding the samples $v[n]$, the error sequence $r[n]$ is modelled as a random sequence with the following properties:

it is uniformly distributed in $\left[-\frac{Q}{2}, \frac{Q}{2}\right)$,

thus it has a variance $\sigma_r^2 = Q^2/12$,

it has a constant power density spectrum,

it is not correlated with $v_0(t)$, nor with $v[n]$.

A first and obvious condition for these properties of $r[n]$ is that no overflow occurs, i.e. we assume

$$-1 \le v_0(t) < 1, \quad \forall t.$$

Furthermore, a sufficient condition is that $v_0(t)$ is uniformly distributed in all quantization intervals $[kQ, (k+1)Q);\ -2^{w-1} \le k < 2^{w-1} - 1$.

The objective of this project is the verification of the model, described by (1.1), if appropriate signals $v_0(t)$ are used, but also to show its limitation in other cases.

Furthermore possibilities for reducing the quantization error by oversampling and error-feedback will be investigated.

Hints

For the investigation of an A/D-converter we use samples of signals $v_0(t)$ generated with floating-point arithmetic. They can be regarded as being unquantized with sufficient accuracy. We shall use two types of random input signals, one uniformly distributed in [-1, 1), the other normally distributed with different variances. See `help rand` for further information. In addition we shall use sinusoidal sequences of different amplitude.

The quantization, which simulates the A/D converter, is done using the m-file `fxquant`, written for the simulation of fixed-point arithmetic with choosable wordlength w. The program allows the choice of different types of quantization (rounding, 2's complement truncation, magnitude truncation) and different modes to handle the overflow. For more information see `help fxquant`.

```
function X = fxquant( s, bit, rmode, lmode )
%X = fxquant( S, BIT, RMODE, LMODE )  simulated fixed-point arithmetic
%   fxquant  returns the input signal S reduced to a word-length
%   of BIT bits and limited to the range [-1,1). The type of
%   word-length reduction and limitation may be chosen with
%   RMODE:  'round'     rounding to nearest level
%       'trunc'     2's complement truncation
%       'magn'      magnitude truncation
%   LMODE:  'sat'       saturation limiter
%       'overfl'    2's complement overflow
%       'triangle'  triangle limiter
%       'none'      no limiter
```

For the investigation of the error sequence $r[n]$ the programs `hist`, `spectrum`, `acf` and `ccf` can be used. Information about these programs can be found with `help` as well.

The probability densities of $r[n]$ should be determined for 20 bins. If `hi` is the output of `hist` the required normalization can be achieved with `p = hi/sum(hi)`.

The correlation sequences $\phi_{rr}[m]$ and $\phi_{vr}[m]$ should be calculated for $m = -16 : 15$.

Exercise 1.1: The quantizer

We start with an investigation of the m-file `fxquant` determining experimentally the non-linear relationship between input and output signal and the characteristics of the error sequence. Choose as the input signal a sequence $v[n]$ linearly increasing between -1.1 and $+1.1$ e.g. with stepsize 2^{-6}. Generate the corresponding quantized sequence having a wordlength of $w = 3$ bits using rounding and a saturation overflow characteristic with

```
vq = fxquant (v, 3, 'round', 'sat').
```

Calculate `dq = vq - v`; plot separately `vq` versus `v` and `dq` versus `v`. Which deviations from the above mentioned model do you observe so far?

Exercise 1.2: The S/N-ratio of an A/D-converter

Now we use a normally distributed input sequence with zero mean. After separating the error sequence

$$r[n] = v[n]_Q - v[n]$$

we measure the ratio of the signal power $S = \sigma_{\mathbf{v}}^2$ to the noise power $N = \sigma_{\mathbf{r}}^2$ under different conditions.

(a) Choose $\sigma_{\mathbf{v}}^2 = 0.01$; measure and plot S/N in dB as a function of the wordlength w for $w = 3 : 1 : 16$. Use 8000 samples for each point. What is the improvement of the S/N-ratio per bit?

(b) The following experiment will be done with $w = 8$ bits. Measure and plot S/N in dB as a function of S in dB for S $= -60$ dB : 5 : 20 dB.

In a certain range the S/N-ratio increases linearly with S. Determine and explain the points where this property is lost. Especially verify experimentally that for $\sigma_{\mathbf{v}} \approx 0.3$ the power of the error due to overflow has the same order of magnitude as the rounding error.

Exercise 1.3: The quantization error

Now we investigate the properties of the error sequences $r[n]$ for different input signals $v[n]$.

Generate `len` values of the error sequence $r[n] = v[n]_Q - v[n]$

with e.g. `len` = 8000 and determine the following properties approximately for wordlength $w = 8$, rounding and saturation overflow characteristic:

- the mean and the variance using `mean` and `std`,
- the probability density of $r[n]$ with `hist`,
- its power density spectrum with `spectrum`,
- its autocorrelation sequence either with `acf` or as `abs(ifft(spectrum(.)))`,
- the cross spectral density of $r[n]$ and $v[n]$ with `spectrum(r,v,.)`,
- the crosscorrelation sequence with of $r[n]$ and $v[n]$ either with `ccf` or as `ifft (r,v,n)`.

These investigations should be done with the following signals:

(a) $v_1[n]$ being a white random signal, uniformly distributed in [-1, 1),

(b) $v_2[n]$ being a white random signal, normally distributed with variance 1,

(c) $v_3[n] = a * \sin[n * \omega]$ with $\omega = 2\pi/\texttt{len}$, $a = 1.0,\ 1/32$.

In case c) the measurements should be done with the wordlength $w = 4$ as well.

Comment on the deviations from the theoretical model of the quantization error. Especially explain the reasons for these differences.

Exercise 1.4: Oversampling

In this exercise, we investigate the reduction of the quantization noise by oversampling plus an appropriate filtering. Suppose we are given a continuous signal $v_0(t)$, the spectrum of which is approximately zero for $f \geq f_c$, thus requiring a sampling rate of $f_{s1} \geq 2f_c$. Instead we use an A/D-converter with a much higher sampling rate $f_{s2} = r \cdot f_{s1}$, where r is an integer > 1. We want to verify that a following filtering with a lowpass of cutoff frequency f_c combined with a subsampling by a factor r yields a reduction of the noise power by a factor of r or $10 \cdot \log_{10} r$ dB in comparison with a direct A/D-conversion of $v_0(t)$ with the sampling frequency f_{s1}.

Remark: This method is applied for the C/D player.

Since we deal with sampled signals in our computer experiments we have to generate a digital version of a bandlimited signal first (see fig.1). For this purpose we use a lowpass filter with the transfer function $H_0(z)$, having a stopband cutoff frequency $\omega_c = \pi/r$. It yields the desired sequence $v[n]$, corresponding to $v_0(t)$, but with sampling rate f_{s2}. In the upper branch of the figure, it is shown that after subsampling by a factor r the usual A/D-conversion with f_{s1} (corresponding to a quantization in our experiment) yields the output sequence $y_1[n]$, while the extraction of the error sequence $r_1[n]$ can be done as in exercise 1.3. In the lower branch the conversion by oversampling is done. Another lowpass filter described by $H_1(z)$, having a passband of a width corresponding to the bandwidth of $v_0(t)$ eliminates the noise spectrum for $f > f_c$. A subsampling by the factor r finally generates the desired signal $y_2[n]$ in this case. The separation of the noise sequence $r_2[n]$ requires an identical filter for the generation of a signal out of $v[n]$ for comparison (why?).

The following steps have to be done:

- Use a lowpass, e.g. an elliptic filter $H_0(z)$ designed with `ellip(7,.1,60,.195)`, having a stopband cutoff frequency $\omega_c \approx \pi/4$ and an attenuation of 60 dB in the stopband to be excited by white noise with zero mean.
- Use e.g. $w = 10$ bits for the quantizers, representing the A/D-converters. Later on try other wordlengths.
- Use FIR-filters ($H_1(z)$) with linear phase, appropriate cutoff-frequency and attenuation of 60 dB in the stopband.
- Generate the sequences $r_1[n]$ and $r_2[n]$ working with input signals of length 8000. Measure and compare their variances. Comment on the results.

Exercise 1.5: Sigma-Delta A/D converter

A further reduction of the noise is possible with a "Sigma-Delta A/D converter", working with error-feedback. A discrete model of a simple example is shown in fig. 2.

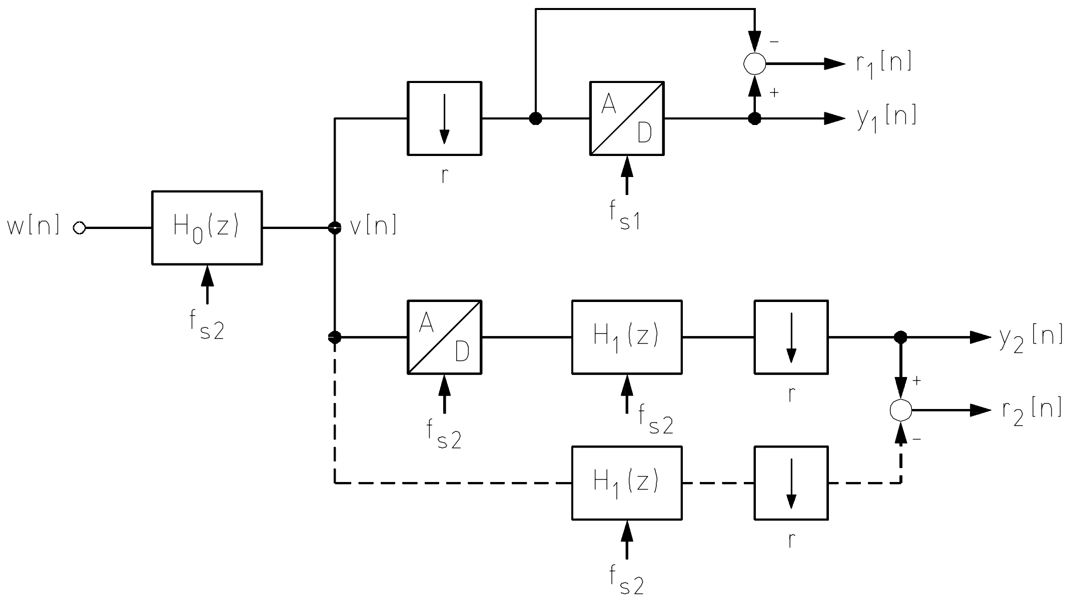


Figure 1: Structure for comparative investigation of an oversampling A/D-converter

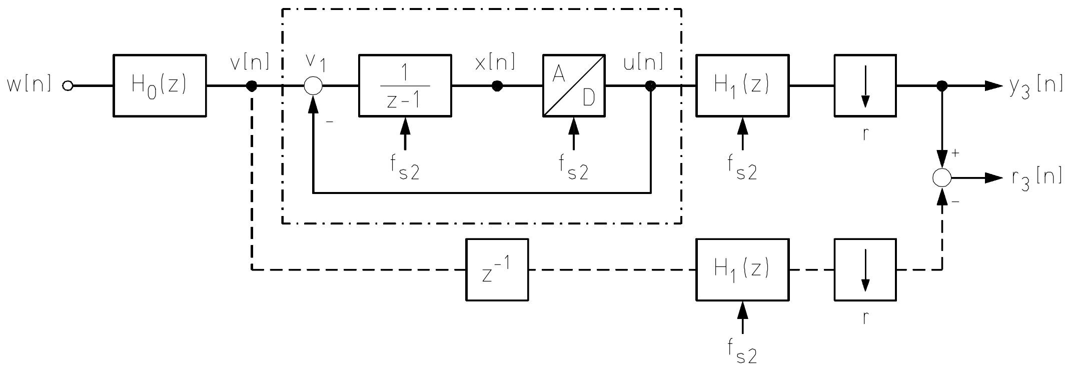


Figure 2: Structure for investigating an A/D-converter working with oversampling and error-feedback

The analysis of the basic block yields for the Z-transform of its output signal

$$U(z) = Z\{u[n]\} = \frac{1}{z}V(z) + \frac{z-1}{z}R(z)\,, \tag{1.2}$$

where $R(z)$ is the Z-transform of the noise sequence at the output of the quantizer. Thus the noise transfer function turns out to be

$$|H_R(e^{j\omega})|^2 = 4\sin^2\omega/2 = (2 - 2\cos\omega)\,. \tag{1.3}$$

The lowpass filter $H_1(z)$ eliminates the noise for $\omega > \pi/r$, so the power of the noise sequence $r_3[n]$ at the output becomes in case of an ideal LPF

$$\sigma_r^2 = \frac{Q^2}{12}\frac{1}{\pi}\int\limits_0^{\pi/r} |H_R(e^{j\omega})|^2 d\omega = \frac{Q^2}{12}\frac{2}{r}\left(1 - \frac{\sin\pi/r}{\pi/r}\right) \tag{1.4}$$

In comparison with the structure in fig. 1 the "noise shaping" as described by (1.3) yields a further reduction of the noise by a factor $2\left(1 - \sin\frac{\pi}{r}/\frac{\pi}{r}\right)$, which $\to 0$ as $r \to \infty$.

In this exercise, we want to verify these theoretical results by measuring approximately the power of $r_3[n]$ and its power density spectrum. Besides the filters and subsamplers used previously in exercise 1.4 we have to simulate the basic block exactly. This can be done using the following for-loop:

```
x = 0,

for n = 2 : length(v),

   u(n-1) = fxquant(x,w,'round','sat');

   v1 = v(i) - u(n-1);

   x = v1+x;

end.
```

- Generate the sequence $r_3[n]$ working with input signals $w[n]$ of length 8000 and wordlengths $w = 8$ and $w = 10$ for the quantizer.
- In order to verify the result (1.3) for $\omega < \pi/r$ measure approximately its power (using `std`) and its power density spectrum (using `spectrum`).

Exercise 1.6: A different measuring method

Furthermore we use another method for the investigation of the basic block, shown in fig. 2. The procedure is explained in some detail in the appendix of this chapter, where a blockdiagram and (partly) a MATLAB-program is given as well. It is recommended to read this section first before proceeding. As described with (A.1) the required ensemble of periodic input signals is generated as

$$\tilde{v}_\lambda[n] = \text{IFFT}\{V_\lambda[k]\} = \frac{1}{N}\sum_{k=0}^{N-1} V_\lambda[k] w_N^{-kn} \in \mathbb{R}\,, \tag{1.5}$$

where N is the period. Here we use the following modification: With

$$V_\lambda[k] = \begin{cases} |V| \cdot e^{j\varphi_\lambda[k]} & , \quad k = 0(1)(N/2r-1) \\ \\ 0 & , \quad k = N/2r(1)N/r-1 \end{cases} \tag{1.6}$$

and $V_\lambda[N-k] = V_\lambda^*[k]$, we get a periodic real signal having precisely the desired bandlimitation. Thus the filtering by $H_0(z)$, as shown in fig. 2 is not necessary anymore. As is explained in the cited appendix the phases $\varphi_\lambda[k]$ are independent random variables, thus yielding an ensemble of periodic but independent sequences $v_\lambda[n]$.

In this experiment we are interested in the estimate $\hat{\phi}_{rr}(e^{j\omega})$ of the power density spectrum of the noise. Eq. (A.11) gives the desired result for the points $\omega = \omega_k = k \cdot 2\pi/N$, $k = 0(1)N/2-1$. Here we can use the output sequence $u[n]$ of the quantizer directly, the spectrum $U_\lambda[k]$ of which replaces $Y_\lambda[k]$ in (A.11). The variance of the noise in the frequency band up to $\omega_c = \pi/r$ is obtained by a simple summation:

$$\sigma_r^2 = \frac{\pi}{N} \sum_{k=0}^{N/2r-1} \hat{\phi}_{rr}(e^{j\omega_k})\,. \tag{1.7}$$

Neither a filtering with $H_1(z)$ nor a separation of the noise as explained in fig. 2 is required.

Use this method for measuring $\hat{\phi}_{rr}(e^{j\omega_k})$ and σ_r^2 with $w = 8(2)12$ and $r = 4$. The procedure should be applied with $N = 1024$ and $L = 40$ (the number of experiments to be averaged).

Note that the modification made here requires two changes of the MATLAB program, given in the appendix: The definition of the phase as explained above and in the calculation of `SumH`. (Why?)

Project 2: Modelling a Multiplier

In this project we investigate a real multiplier used for the multiplication of a sequence of random numbers $v[n]$ by a constant factor c, which might be a filter coefficient in a practical application. Dealing with fixed point arithmetic we assume the magnitudes of both factors are not larger than one. The wordlengths are w and w_c respectively. Thus, in general, the product $p[n] = c \cdot v[n]$ will have a wordlength $w_p = w + w_c - 1$. Its further processing e.g. in a digital filter requires a quantization of the result down to w bits, the processor wordlength. This project deals with an investigation of the error sequences produced by different methods for the quantizing the product. The goal is a model of a real multiplier, applicable to the noise analysis of a digital system.

Project Description

The real multiplier, we are interested in, is modelled by an ideal multiplier combined with a quantizer. It yields signals

$$p_Q[n] = [c \cdot v[n]]_Q = c \cdot v[n] + e[n]\,, \tag{2.1}$$

having the required wordlength w. We investigate the properties of the error sequence

$$e[n] = p_Q[n] - c \cdot v[n]\,.$$

In the following

$Q = 2^{1-w}$ is the quantization step of the input signal $v[n]$ and of $p_Q[n]$,

$Q_c = 2^{1-w_c}$ that of the coefficient c and

$Q_p = 2^{1-(w+w_c)}$ that of $p[n]$.

The quantization will be done by one of the following methods

- 2's complement rounding, described in MATLAB notations by

$$\texttt{pqr = Q * floor(p/Q} + \texttt{.5)}, \tag{2.2}$$

- 2's complement truncation, described by

$$\texttt{pqt = Q * floor(p/Q)}, \tag{2.3}$$

- magnitude truncation, described by

$$\texttt{pqm = Q * fix(p/Q) = sign(p).*(Q * floor(abs(p/Q)))}. \tag{2.4}$$

The simple and most common models for the description of the error sequences $e[n]$ are in the three cases

- 2's complement rounding:

  $e[n]$ is a member of a random process $\mathbf{e}_r$, being uniformly distributed in the interval $(-Q/2, Q/2]$. Thus it has the properties

$$m_r = \mathcal{E}\{\mathbf{e}_r\} \approx 0,\ \sigma_r^2 = Q^2/12\,. \tag{2.5}$$

  Its power density spectrum is constant, it is not correlated with the input process $\mathbf{v}$.

  Note the similarity with the description of an A/D-converter.

- 2's complement truncation:

  Now the random process $\mathbf{e}_t$ is uniformly distributed in the interval $(-Q, 0]$. Thus its properties are

$$m_T = \mathcal{E}\{\mathbf{e}_t\} = -Q/2,\ \mathcal{E}\{\mathbf{e}_t^2\} = Q^2/3,\ \sigma_t^2 = Q^2/12\,. \tag{2.6}$$

  Again the power density spectrum is assumed to be constant, and $\mathbf{e}$ is assumed to be uncorrelated with the input process $\mathbf{v}$.

- magnitude truncation

  The random process is uniformly distributed either in the interval $(-Q, 0]$, if $p \geq 0$ or in $[0, Q)$, if p is negative. It can be expressed as

$$e_m[n] = -\frac{Q}{2}\,\mathrm{sign}\,[p] + e'_m[n]\,, \tag{2.7}$$

  where $e'_m[n]$ has the same properties as $e_r[n]$, the error sequence in case of rounding. If p is e.g. normally distributed with zero mean we get

$$m_m = \mathcal{E}\{\mathbf{e}_m\} = 0\ ;\ \sigma_m^2 = Q^2/3\,. \tag{2.8}$$

  Usually the obvious correlation between $\mathbf{e}_m$ and $\mathbf{v}$ will be ignored.

The above given characteristics of the error sequences have been found to be useful, if Q_c is small enough and if the product $p[n] = c \cdot v[n]$, to be quantized is large enough. That implies a dependence on the variance $\sigma_{\mathbf{v}}^2$ of $\mathbf{v}$ and on the coefficient $c = \lambda_c \cdot Q_c$, where λ_c is an integer. Note that there might be a big difference between the nominal quantization stepsize Q_c and the effective one Q_{ce} in an actual case. E.g. $c = 0.$ x 1 000 000 with x $\in (0,1)$ is a number with nominal $Q_c = 2^{-8}$ but as well one out of two possible positive numbers with the effective stepsize $Q_{ce} = 2^{-2}$. Obviously the number of possible different values $e[n]$ is equal to $[Q_{ce}]^{-1}$. A careful investigation of the means and variances of the error sequences yields some modifications of former results. With $c = \lambda_c Q_c$ and $lcd(\lambda_c, Q_c^{-1})$, being the largest common divisor of λ_c and Q_c^{-1}, the *effective quantization stepsize* becomes

$$Q_{ce} = Q_c \cdot lcd(\lambda_c, Q_c^{-1})\,. \tag{2.9}$$

Now we get in the different cases

- 2's complement rounding: Instead of (2.5)

$$m_r = \frac{1}{2} Q Q_{ce};\; \mathcal{E}\{\mathbf{e}_r^2\} = \frac{Q^2}{12}(1 + 2Q_{ce}^2)\,;\; \sigma_r^2 = \frac{Q^2}{12}(1 - Q_{ce}^2)\,. \tag{2.10}$$

- 2's complement truncation: Instead of (2.6):

$$m_t = -\frac{Q}{2}(1 - Q_{ce});\; \mathcal{E}\{\mathbf{e}_t^2\} = \frac{Q^2}{3}(1 - \frac{3Q_{ce}}{2} + \frac{Q_{ce}^2}{2})\,;\; \sigma_t^2 = \frac{Q^2}{12}(1 - Q_{ce}^2)\,. \tag{2.11}$$

- Magnitude truncation: With $m_m = 0$ as before now instead of (2.8)

$$\sigma_m^2 = \frac{Q^2}{12}(1 - \frac{3Q_{ce}}{2} + \frac{Q_{ce}^2}{2})\,. \tag{2.12}$$

  Furthermore, in cases of rather large values Q_{ce} but small variances σ_v^2 of the signal $v[n]$ a correlation between $e[n]$ and $v[n]$ will be observed, even in case of rounding. Here the error sequence $e[n]$ includes a term proportional to $v[n]$, but dependent on c and σ_v. Thus we get instead of (2.1)

$$p_Q[n] = c \cdot [1 + \Delta c(c, \sigma_v)] \cdot v[n] + r[n]\,, \tag{2.13}$$

  where now the correlation of $r[n]$ and $v[n]$ is rather small. But the result is an incorrect coefficient

$$c' = c \cdot [1 + \Delta c(c, \sigma_v)]\,. \tag{2.14}$$

Hints

In these experiments we use either the expressions (2.2)...(2.4) for doing the different quantization or `fxquant` (see project on quantized signals). Use `help` to get further information about `round`, `floor`, `fix` and `fxquant`. For measuring the power spectral density of the noise as well as a possible deviation Δc of the coefficient we use the measuring procedure, explained in the appendix. Furthermore the programs `hist`, `spectrum`, `acf` and `ccf` can be used.

As input signal $v[n]$ of the multiplier we use a quantized version of a normally distributed sequence with zero mean, generated with `rand('normal')` and different variances, to be specified. The quantization down to the wordlength w of the multiplier is working with, should be done with rounding, according to (2.2).

Exercise 2.1: The quantizers

We want to determine the non-linear I/0 characteristics of the three quantizers, described by eq. (2.2)...(2.4). Measure `pq` and `e` = `pq` − `p` as a function of p for $w = 3$ and

a) $p = -1 : 2^{-6} : 1$ (corresponding to $Q_{ce} = 2^{-4}$)

b) $p = -1 : 2^{-4} : 1$ (corresponding to $Q_{ce} = 2^{-2}$)

Plot with `subplot(22.)`; `pq` (case a), `pq` (case b), `e` (case a), `e` (case b).

Exercise 2.2: The error sequences in the usual case

For the following measurements we use an input signal $v[n]$ with deviation $\sigma_v = 0.25$ and `length(v)` = 8000. Furthermore we apply $w = 10$.

(a) What is the effective wordlength Q_{ce} if we choose $c = 63/128$?

(b) Measure approximately the means and the variances of the error sequences $e[n]$.

(c) Measure approximately and plot the power spectrum density of $e[n]$ and the cross spectral density of $e[n]$ and $v[n]$ using `spectrum`. Alternatively measure the autocorrelation sequence with `acf` and the crosscorrelation sequence of $e[n]$ and $v[n]$ with `ccf`. Comment your results, especially the differences of the quantizers with regard to the crosscorrelation.

Exercise 2.3: The error sequences in special cases

Now we investigate the properties of the different quantizers under other conditions. Especially we use coefficients c with larger effective quantization step Q_{ce} and in addition smaller variances of the input signal. The wordlength is again $w = 12$.

(a) Measure the means and the variances of the error sequences $e[n]$ for the three quantizers using the following parameters:
$\sigma_v =$ 0.01 and 0.05 ; length(v) = 8000.
$c =$ $\lambda_c Q_c$ with $Q_c = 2^{-7}$ and two values of λ_c such that the effective quantization step becomes $Q_{ce} = 2^{-4}$, (e.g. $c = 8/128$ and $72/128$)

Compare your results with those to be expected according to eq. (2.10...2.12).

(b) Now we want to measure the effective coefficient c', as defined by (2.14) and the power density spectrum $\hat{\phi}_{rr}(e^{j\omega})$ of the remaining noise. The measurement should be done with the method outlined in the appendix using $N = 512$ and $L = 50$ trials. The magnitude of the input signal has to be selected such that the desired values as in part a are obtained.

The result $H(e^{j\omega})$ of the measurement corresponds to c' (why?), furthermore we get the desired power density spectrum. Comment on your results.

Project 3: Sensitivity of Filter Structures

In this project we investigate the properties of different structures for the implementation of a digital filter, described by the same transfer function $H(z)$. Of course these systems will have an identical input–output behavior if they work with unlimited wordlength, provided that the design was done properly. But the necessary quantization of coefficients as well as data causes deviations of different size for different structures.

In order to simplify the problem we investigate first the influence of quantizing the coefficients, disregarding the arithmetic errors. It is essential that such a system is still linear and thus it can be described by its frequency response. Its deviation from the ideal frequency response, to be calculated with coefficients of unlimited wordlength gives the required measure of the sensitivity of different structures.

Project Description

In project 4 of the chapter **Systems and Structures** some different structures for the implementation of a certain transfer function $H(z)$ have been introduced. The influence of limiting the wordlength of its coefficients will be investigated here by calculating or measuring the frequency response after rounding the parameters to a specified wordlength. The following structures will be investigated:

The direct form 2 (implemented in `filter`)

The cascade form, the parallel form, both with two different structures for the block of second order.

The Lattice structure

The lossless system, implemented with two coupled allpasses, introduced in project 3 and 4 of the chapter **Systems and Structures**. We add a few remarks about this structure by showing that it has a "built in" insensitivity. That results out of its structural property

$$|H(e^{j\omega})| \leq 1\,. \tag{3.1}$$

Suppose the coefficients a_{ki} of the two allpasses are somewhat incorrect, due to the required limitation of their wordlength. Thus we have $a_{ki} + \Delta a_{ki}$ instead of a_{ki}. Furthermore, we assume that the structure of the allpass systems is such that their main property $|H_{Ai}(e^{j\omega})| = 1$ for all ω as well as the stability of the system is not affected by the erroneous coefficients, a condition which can be satisfied easily. Under these assumptions eq. (3.1) still holds. Now let ω_λ be a point where $|H_i(e^{j\omega_\lambda})| = 1$, with ideal coefficients. Due to eq. (3.1) any change of any coefficients a_{ki} will result in a decrease of $|H_i(e^{j\omega_\lambda})|$ regardless of the sign of Δa_{ki}. Thus the slope of $|H_i(e^{j\omega})|$ as a function of any a_{ki} is precisely zero at $\omega = \omega_\lambda$. So the first order sensitivity in respect to a_{ki}, defined as

$$S_{ki} = \frac{a_{ki}}{|H_i(e^{j\omega})|} \cdot \frac{\partial |H_i(e^{j\omega})|}{\partial a_{ki}} \tag{3.2}$$

is zero at all points ω_λ, where $|H_i(e^{j\omega})|$ is unity. Thus, under rather mild conditions we can expect a small sensitivity of the system in the passband.

Hints

For the investigation of the sensitivity of the structures described above use transfer functions, either of an FIR or an IIR-system, which satisfy the lowpass tolerance scheme described by

cutoff frequency of the passband $\omega_p = 0.4$;

cutoff frequency of the stopband $\omega_s = 0.44$;

tolerated deviation in the passband $\Delta|H| \leq 0.01 \hat{=} -0,0873\,\text{dB}$;

maximum value in the stopband $|H| \leq 0.01 \hat{=} -40\,\text{dB}$.

Systems which satisfy this tolerance scheme have to be designed first with `remez` and `ellip` respectively. Using their numerator and denominator polynomial $B(z)$ and $A(z)$ the zeros c_k and poles d_k of the IIR-system as well as the partial fraction expansion can be calculated. In order to find the parameters of the cascade structure use the "nearest neighbour strategy" for pairing poles and zeros to real subsystems of second order: start with the pole with maximum magnitude and combine it with the next zero, use in addition the linear factors of their complex conjugate values for getting one subsystem. Continue with the remaining values.

The design of the two allpasses should be done as follows: Arrange the poles $d_k = |d_k > e^{j\varphi_k}$ with $\varphi_k \geq 0, k = 1(1)(N+1)/2$ such that $\varphi_{k+1} < \varphi_k$. Now appoint these poles alternately to $H_{A1}(z)$ and $H_{A2}(z)$. Taking the complex conjugate values into account the required real systems will be achieved. It is recommended to work with a cascade of blocks of first or second order, each to be implemented with `filter`. The allpass property for all coefficients in the stable domain can be guaranteed (why?).

Comment: May be designing a filter during this project is not a good idea. Instead we can either combine the project with another one in the filter design packet in the sense that we use a filter, already designed there or we have to provide coefficients on a disq (?).

Hints

While the direct structure can be implemented immediately using `filter`, the other structures require an appropriate combination of the results obtained by using `filter` for the subsystems. Starting with the coefficients of the different structures obtained during the design step, perform a quantization by rounding using $[a]_Q =$ `Q*floor(a/Q+.5)` (see eq. (2.2) in project 2). The frequency responses of the resulting filters with quantized coefficients can be calculated using the m-file `frequenz`. The result can be used immediately inthe case of the direct structure, while in the other cases the frequency responses of the subsystems obtained with `frequenz` have to be combined appropriately.

Exercise 3.1: Direct form

(a) FIR-system

Calculate the frequency response of the system with coefficients $[b_k]_Q$, obtained by rounding b_k to $w = 8$ bits. It is

$$H(e^{j\omega})_Q = \sum_{k=0}^{N} [b_k]_Q \cdot e^{-j\omega k}\,.$$

Calculate furthermore the error-frequency response with

$$\Delta b_k = [b_k]_Q - b_k \qquad \text{as}$$

$$\Delta H(e^{j\omega}) = \sum_{k=0}^{N} \Delta b_k e^{-j\omega k} .$$

$H(e^{j\omega})_Q$ and $\Delta H(e^{j\omega})$ should have linear phase (why?). Test whether that is correct by calculating

$$H(e^{j\omega})_Q \cdot e^{j\omega N/2} ;\; \Delta H(e^{j\omega}) \cdot e^{j\omega N/2}$$

Which result do you expect?

Use subplot (21.) for visualizing $\text{Re}\{H(e^{j\omega})_Q \cdot e^{j\omega N/2}\}$ and $\text{Re}\{\Delta H(e^{j\omega}) e^{j\omega N/2}\}$.

Why do we recommend $\text{Re}\{\cdot\}$ to be plotted?

Furthermore plot $|H(e^{j\omega})_Q|$ in dB.

Comment on the maximum deviations:

$\Delta H(e^{j\omega})$ should satisfy a condition $|\Delta H(e^{j\omega})| \leq m \cdot Q$; what is m?

Does the system still satisfy the tolerance scheme we started with?

(b) IIR-system

The system to be implemented has at least to be stable. Determine the required minimum wordlength $\min w$ of the quantized coefficients $[a]_Q$ for guaranteed stability by calculating the magnitudes of the roots of the different polynomials defined by these coefficients.

Calculate and plot $|H(e^{j\omega})|$ with coefficients having the wordlength $\min w$. Does the system satisfy the tolerance scheme?

Increase the wordlength further up to a value yielding a system of satisfying properties.

Exercise 3.2: Cascade-form

(a) Recall the stability conditions for a polynomial in z of second order. Calculate and plot the zeros of all polynomials with the coefficients $a_{1\lambda} = \lambda_1 \cdot Q$ and $a_{2\lambda} = \lambda_2 \cdot Q$ for $Q = 2^{-4}$ while choosing λ_1 and λ_2 such that these zeros are inside the unit circle.

(b) Use the subsystem having the largest coefficient $a_{2\lambda}$ to test, whether rounding can yield an unstable system. In case determine the required wordlength for guaranteed stability.

(c) The zeros of the transfer function are located on the unit circle. Can it happen that this property gets lost by rounding the coefficients $b_{k\lambda}$ of the numerator polynomials of the different subsystems?

(d) Round the coefficients of the subsystems using $w = 8$ bits. Calculate and plot $|H(e^{j\omega})|_Q$ for these quantized coefficients in dB in comparison with $|H(e^{j\omega})|$ with the original parameters. Where do you observe the largest deviation, in the passband or stopband?

(e) Does the filter satisfy the tolerance scheme? If not find the required wordlength for a satisfying performance.

Exercise 3.3: Parallel-form

(a) While the situation of the denominator coefficients is the same as in the cascade case, the situation is different for the numerator of the transfer function. Why?

(b) Quantize all coefficients by rounding using $w = 8$ bits. Calculate and plot $|H(e^{j\omega})|_Q$ in dB again in comparison with the ideal frequency response $|H(e^{j\omega})|$. Where do you observe the largest deviation now?

(c) Does the filter satisfy the tolerance scheme? If not find the required coefficient wordlength for a satisfying performance.

Exercise 3.4: Coupled allpasses

(a) Test if the allpass property of the two cascades of allpasses of first and second order is still preserved after rounding the coefficients to any wordlength w, if these systems are implemented with filter. Under what condition will this statement hold if the two allpasses are implemented with the direct form?

(b) Round all coefficients using $w = 8$ bits. Calculate and plot again the corrresponding frequency response $|H(e^{j\omega})|_Q$ in dB in comparison with the ideal one. Plot in addition the frequency response in the passband to an enlarged scale. How about the stopband? Are the zeros of the transfer function still on the unit circle as they should? Comment on your results.

Project 4: Limit Cycles

As has been mentioned in the overview, an implemented digital system is not linear anymore but nonlinear due to two reasons:

1. The range of all numbers, to be represented in a digital system is limited. If the summation of two or more numbers yields a result beyond this limit, an overflow and thus an error occurs, the value of which depends on the nonlinear characteristic, choosen in the particular implementation, and on the actual signal. This might result in a nonstability of the total system, yielding an oscillation of rather large amplitude in spite of the fact that no excitation is applied. It is called a "large scale-" or "overflow limit cycle".

2. The second nonlinear effect is caused by the required limitation of the wordlength after a multiplication. It has been considered in Project 2. Here we are not interested in a model of a real multiplier, introduced there by using a random error source, but in a possible nonstability caused by this nonlinearity, yielding an oscillating behavior of the system. Since the resulting limit cycles have usually (but not necessarily) a small amplitude and since their values are expressed as multiples of the quantization step Q, they are called "small scale-" or "granular limit cycles".

The possibility and the size of limit cycles depend on

- the type of the nonlinearity
  i.e. the overflow characteristic on one hand and the quantization method (rounding or truncation) on the other,

- the location of the poles of the transfer function of the linear system, to be implemented. Besides its influence on the size they determine the periode of the limit cycle,
- the structure of the system,
- the actual excitation. A limit cycle can occur, if the input signal is zero or constant or periodic.

A complete investigation of this rather complicated problem is not possible here. The following exercises are restricted to two points:

1. The existence of both types of limit cycles and its dependance on the conditions mentioned above will be demonstrated with examples for the special case of systems of first or second order with zero input.
2. Methods for avoiding limit cycles will be discussed and demonstrated.

Project Description

For the case of fixed point arithmetic there are essentially three possibilities to limit the actual result $s[n]$ of a summation such that

$$-1 \leq f(s[n]) < 1\,, \tag{4.1}$$

assuming a 2's complement representation of numbers:

- the 2's complement characteristic can be used directly (see fig. •).
- the saturation characteristic can be applied, described by

  $$f(s[n]) = \begin{cases} -1 \quad , & s[n] \leq -1 \\ 1-Q \ , & s[n] \geq 1 \end{cases}$$

  (see fig. •).
- the triangle characteristic can be used (see fig. •)

Hints

The m-file `fxquant` can be used for the implementation of the different overflow modes. For more information use `help fxquant`.

Project 5: QUANTIZATION NOISE IN DIGITAL FILTERS

The influence of arithmetic errors inside a digital filter is investigated by measuring the power density spectrum of the error sequence. The results are compared with those obtained by theory, based on the usual model of a real multiplier.

State: The exercises have been tried out using m-files for the quantization. Result: In order to finish an exercise under Matlab in tolerable time the experiments have to be restricted to a system of second order. Nothing has been written up so far in English.

Appendix

A method for measuring the performance of an implemented digital system

The system to be investigated is only approximately linear due to the effects of limited wordlength inside the filter. It can be modelled by a parallel connection of a linear system, the output of which is $y_L[n]$, and another one, yielding the noiselike errorsequence $r[n]$. The separation should be done such that $y_L[n]$ and $r[n]$ are orthogonal to each other. While the linear subsystem will be described by its frequency response $H(e^{j\omega})$, the other one is characterized by the power density spectrum $\phi_{rr}(e^{j\omega})$ of its output sequence. The method to be explained yields samples of estimates of these two functions at $\omega_k = k \cdot 2\pi/N[\cdot]$.

We perform a sequence of measurements, using as excitation members $\tilde{v}_\lambda[n]$ of an ensemble of signals, being periodic for $n \geq 0$. In the most simple case, to be used here, they are generated as

$$\tilde{v}_\lambda[n] = \text{IFFT}\,\{V_\lambda[k]\} = \frac{1}{N}\sum_{k=0}^{N-1} V_\lambda[k] w_N^{-kn} \in \mathcal{R}, \tag{A.1}$$

where the spectral values

$$V_\lambda[k] = |V| e^{j\varphi_\lambda[k]} \tag{A.2}$$

have always the same magnitude for all k and all λ, while $\varphi_\lambda[k]$ is a random variable uniformly distributed in $[-\pi, \pi)$ and statistically independent in respect to k and λ. The condition $\varphi_\lambda[k] = -\varphi_\lambda[N-k]$ has to be observed in order to get a real signal $\tilde{v}_\lambda[n]$. It turns out that the sequences $\tilde{v}_\lambda[n]$ are approximately normally distributed.

The system under test will be excited by these $\tilde{v}_\lambda[n]$, $\lambda = 1(1)L$, where L is the number of trials to be chosen such that the desired accuracy of the result is achieved.

After the transient time the output sequence of the system will be periodic, to be described by

$$\tilde{y}_\lambda[n] = \tilde{y}_L[n] + \tilde{r}_\lambda[n]. \tag{A.3}$$

Note that in the digital system, to be tested here with a periodic excitation, the error sequence $\tilde{r}_\lambda[n]$ will be periodic as well. The term $\tilde{y}_L[n]$ on the right hand side can be expressed as

$$\tilde{y}_L[n] = \text{IFFT}\,\{H(e^{j\omega_k}) V_\lambda[k]\}, \tag{A.4}$$

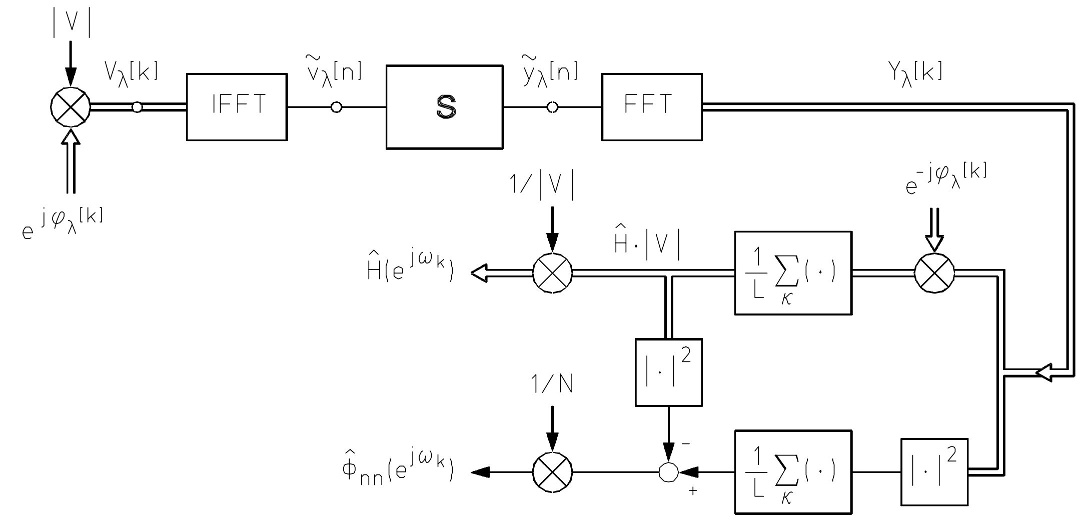
where the $H(e^{j\omega_k})$ are samples of the frequency response of the linear subsystem to be determined. Minimizing

$$\mathcal{E} = E\{|\tilde{r}_\lambda[n]|^2\} = E\left\{|\tilde{y}_\lambda[n] - \frac{1}{N}\sum_{k=0}^{N-1} H(e^{j\omega_k}) V_\lambda[k] w_N^{-nk}|^2\right\} \tag{A.5}$$

with respect to the $H(e^{j\omega_k})$ yields after some calculations for the special case described by (A.2)

$$H(e^{j\omega_k}) = E\left\{\frac{Y_\lambda[k]}{V_\lambda[k]}\right\}, \tag{A.6}$$

where $Y_\lambda[k] = \text{FFT}\,\{y_\lambda[n]\}$. Due to the orthogonality principle the sequence $y_{L\lambda}[n]$ is indeed orthogonal with regard to $r_\lambda[n]$, if the linear subsystem is described by these $H(e^{j\omega_k})$.



The expected value can be calculated approximately by averaging over L trials:

$$H(e^{j\omega_k}) \approx \hat{H}(e^{j\omega_k}) = \frac{1}{L}\sum_{\lambda=1}^{L}\frac{Y_\lambda[k]}{V_\lambda[k]}. \tag{A.7}$$

Now the power density spectrum of the noise

$$\Phi_{rr}(e^{j\omega_k}) = \frac{1}{N}E\{|\text{FFT}\left\{r_\lambda[n]\}|^2\right\} \tag{A.8}$$

can be computed as

$$\phi_{rr}(e^{j\omega_k}) = \frac{1}{N}E\left\{|Y_\lambda[k] - H(e^{j\omega_k})\cdot V_\lambda[k]|^2\right\} \tag{A.9}$$

$$\approx \frac{1}{N}\left[E\{|Y_\lambda[k]|^2\} - |\hat{H}(e^{j\omega_k})|^2|V|^2\right] \tag{A.10}$$

An estimation will be obtained as

$$\hat{\phi}_{rr}(e^{j\omega_k}) \approx \frac{1}{N}\left[\frac{1}{L}\sum_{\lambda=1}^{L}|Y_\lambda[k]|^2 - |\hat{H}(e^{j\omega_k})|^2\cdot|V|^2\right] \tag{A.11}$$

It can be shown that the results found with (A.6) and (A.11) are at least asymptoticly unbiased and consistent. That means the accuracy can be increased by increasing the number L of measurements.

The following flow-diagram describes the required steps. Furthermore a MATLAB program is given with one missing line to be filled in.

```
\% parameters
N   =  256;      % period length of signal         ⎫
TR  =  100;      % number of transient samples     ⎬ to be specified
L   =   20;      % number of trials                ⎭

    % array initializations
    SumH = zeros(1,N);       % H(k) accumulation
    SumY2 = SumH;            % |Y(k)|^ 2 accumulation
```

```
for i =  1 : L;
         %   create one period of input signal
         Phase = . . .

         VP = exp(j*Phase);                      % magnitude 1
         vp = real(ifft(VP));                    % input signal

         %   add initial samples (transient removal)
         v = [vp(N-TR+1:N) vp ];

         %   device under test, using the function filter as an example
         y = filter(.,.,v);

         %   output signal: 1 period in steady state
         y = y( TR+1 : N+TR );

         %   accumulation
         YP = fft(y);
         SumH = SumH  +YP ./ VP;
         SumY2 = SumY2+real(YP .* conj(YP));
end;
% postprocessing
H = SumH / L;
LDS = ( (SumY2 / L) - real(H .* conj(H)) ) / N;
```

The program results in $N/2$ values $\hat{H}(e^{j\omega_k})$ and $\hat{\phi}_{rr}(e^{j\omega_k}),\ \omega_k = k\cdot 2\pi/N$, $k = 0(1)N/2-1$.

Chapter 8

Discrete-Time Filter Design

January 17, 2007

Overview

One of the most powerful operations of discrete-time signal processing is that of filtering. As the name implies, a filter tries to separate parts of a signal according to some criterion, e.g. separating a desired signal from noise, or separating two radio stations. One might want to separate the weekly stock market price variations from the yearly variations.

There are two types of discrete-time filters: FIR (finite impulse response) filters and IIR (infinite impulse response) filters. For both filter types, there are two distinct parts in the design problem: the approximation problem where one tries to approximate a desired filter characteristic by an allowed one, and the realization problem where one implements a transfer function in hardware or software. This chapter deals with the approximation problem; whereas Chapters 5 and 7 cover realization. We assume the reader is familiar with the basic notions of difference equations, frequency response, rational z-transforms, etc. If not, please consult one of the appropriate earlier chapters.

For many practical problems, the specifications for a filter are given in terms of its frequency response. The approximation part of the filter design process can be broken down into four related stages.

- Choose a desired ideal response, usually in the frequency domain,
- Choose an allowed class of filters (e.g., a length-L FIR filter),
- Choose a measure or criterion of how good the approximation is, and
- Develop a method to find the "best" member of the allowed class of filters according to the criterion of approximation.

These four steps are often repeated several times to get an acceptable filter. After the "best" filter is designed and evaluated, the desired response or the allowed class or the measure of quality might be changed and the filter redesigned.

There are three commonly used approximation error measures: least squared error, Chebyshev, and maximally flat. The average squared error is important because it uses the power or energy as a measure of the size of the error. The Chebyshev error has important physical meaning because it is the maximum of the difference between what you want and what you have. While the squared error and the Chebyshev error measures are global measures, the Taylor's series approximation is a local method that maximizes the smoothness of the approximation by matching as many derivatives at a point as possible.

Most of the projects and exercises of the chapter will use the basic lowpass filter as an example, but the ideas extend to other desired frequency responses. Two forms of ideal frequency responses will be examined. The first having a passband with unity transmission and a stopband with zero transmission. The second will have a transition band between the pass and stopband which will allow a much better approximation for a given length or order. It is very important in any approximation or optimization problem to choose the error criterion and the desired ideal response carefully *and explicitly.*

This chapter will consider the properties of both FIR and IIR filters to develop insight and intuition into their characteristics. It is this insight that one should use to choose the allowed class of filters or to choose the appropriate criterion of approximation. We will design and analyze a set of filters with the goals of understanding the basic properties of discrete-time filters, gaining insight into the design process, and learning the characteristics of several standard design methods.

6 Background Reading

The basic methods of filter design are covered in most general DSP text books. Notation and a general introduction to filters are contained in Chapter 1 of this book. Two older books with good chapters on the topic are by Rabiner and Gold [1] and Gold and Rader [2]. Two books dealing specifically with the filter design problem are by Parks and Burrus [3] and Taylor [4]. An excellent coverage of analog or continuous-time filter designs often used as prototypes for digital IIR filters is presented by Van Valkenburg [5]. A good book on discrete least squared error approximation is by Lawson and Hanson [6].

Computer-Based Exercises for Signal Processing

Discrete Design of FIR Filters

Discrete Design of FIR Filters

Overview

The DFT of the impulse response of an FIR filter gives samples of its frequency response. This suggests a method for designing a filter. Choose an ideal frequency response, sample this ideal with L equally spaced samples, and take the inverse DFT of these samples to give the impulse response of the filter. This method is called *frequency sampling design*. The main shortcoming of this approach is the complete lack of control of the frequency response between the samples. A second design method is formulated that uses a larger number of frequency samples than the length of the filter. Under these conditions the actual response will not pass through all the specified samples, but one can easily design a filter whose response has the least average squared error over these sample frequencies. Since both of these design methods operate on discrete samples of the frequency response, they are both developed and investigated in this section.

Background Reading

Details of frequency sampling design can be found in [1, 3] and discussions of discrete least squared error approximation can be found in [6].

Project 1: FIR filter design by frequency sampling

This method designs a filter whose frequency response passes exactly through specified samples of the desired frequency response and, therefore, is an interpolation technique. Since the DFT of the impulse response of an FIR filter is a set of equally spaced samples of its frequency response, the inverse DFT of the samples should be the impulse response of the filter. That is indeed the case and it is the basis of this widely used filter design method.

Project Description

The frequency response of a length-L FIR filter is given by the discrete-time Fourier transform (DTFT) of the impulse response

$$H(e^{j\omega}) = \sum_{n=0}^{L-1} h[n]\, e^{-j\omega n}. \tag{1.1}$$

The length-L DFT of $h[n]$ is the set of L evenly spaced samples of $H(e^{j\omega})$ over ω from zero to 2π given by

$$H_k = H(e^{j2\pi k/L}) = \sum_{n=0}^{L-1} h[n]\, e^{-j2\pi nk/L} \tag{1.2}$$

with $k = 0, 1, 2, \cdots, L-1$. If the length of the filter is equal to the number of frequency samples, the IDFT of the samples of the desired frequency response $H_d(e^{j\omega})$ is the impulse response

$$h[n] = \mathcal{IDFT}\{H_d(e^{j\omega_k})\} = \frac{1}{L}\sum_{k=0}^{L-1} H_k\, e^{j2\pi kn/L}. \tag{1.3}$$

The frequency response of this filter will exactly interpolate the samples of the desired frequency response.

For the general case, both $h[n]$ and $H(e^{j\omega})$ are complex-valued which means there are $2L$ degrees of freedom and $2L$ equations needed to determine the $2L$ unknowns. The samples of the frequency response are evenly spaced over the frequency range of ω from $-\pi$ to π or from 0 to 2π.

Most practical filter design problems have constraints. The impulse response $h[n]$ is usually real which means the real part of $H(e^{j\omega})$ must be an even function and the imaginary part must be odd. Thus there are only L degrees of freedom. If the frequency response has linear phase, the impulse response is symmetric and, therefore, has about half as many degrees of freedom as its length. There is also an inherent difference in the frequency response of even and odd-length filters that can be important. The frequency response of an even-length linear-phase filter must be zero at $\omega = \pi$. The time delay (or phase slope) is an integer for an odd length and an odd multiple of one half for an even length.

The frequency response of a filter can be expressed in several ways. $H(e^{j\omega})$ can be decomposed into its real part and its imaginary part, both real-valued functions of ω, but this is usually not what is desired. In most cases, certainly for linear phase filters, the magnitude and its associated phase is the preferred form. This is

$$H(e^{j\omega}) = |H(e^{j\omega})|\, e^{j\phi(\omega)} \tag{1.4}$$

where

$$|H(e^{j\omega})| = \sqrt{\Re e\{H(e^{j\omega})\}^2 + \Im m\{H(e^{j\omega})\}^2} \tag{1.5}$$

and

$$\phi(\omega) = \tan^{-1}\left(\frac{\Im m\{H(e^{j\omega})\}}{\Re e\{H(e^{j\omega})\}}\right). \tag{1.6}$$

This magnitude–phase description of the complex valued $H(e^{j\omega})$ has problems when $H(z)$ has zeros exactly on the unit circle. $|H(e^{j\omega})|$ will not be analytic, it will have cusps at its zeros, and the phase will have discontinuities equal to an odd multiple of π. These problems can be eliminated by using the amplitude $A(\omega)$ rather than the magnitude $|H(e^{j\omega})|$ where

$$H(e^{j\omega}) = A(\omega)\, e^{j\theta(\omega)} \tag{1.7}$$

and

$$A(\omega) = \pm|H(e^{j\omega})| \tag{1.8}$$

is real but may take positive or negative values determined by what is necessary to make $A(\omega)$ smooth and to remove the discontinuities of π in the phase. $\theta(\omega)$ is the phase response consistent with $A(\omega)$ and is equal to $\phi(\omega)$ with the discontinuities removed. If $h[n]$ is real, $A(\omega)$ and $|H(e^{j\omega})|$ are even real functions of ω and $\phi(\omega)$ and $\theta(\omega)$ are odd functions. An important distinction between the two descriptions for linear phase filters is that $H(e^{j\omega})$ and $|H(e^{j\omega})|$ are periodic in ω with period 2π. But $A(\omega)$ is periodic with period 2π if L is odd and with period 4π if L is even.

The usual definition of a linear phase filter is one whose amplitude–phase description has the phase given by

$$\theta(\omega) = K\,\omega. \tag{1.9}$$

A strictly linear phase filter with $\phi(\omega) = K\omega$ is usually too restrictive.

Rather than use the general IDFT, special design formulas can be derived that include the restrictions of $h[n]$ being real and the phase being linear. These are in terms of L samples of the amplitude $A(\omega)$ over ω from zero to 2π and is given by

$$A_k = A\left(\frac{2\pi k}{L}\right). \tag{1.10}$$

The impulse response for L odd can be derived from (1.3) and (1.7) to be

$$h[n] = \frac{1}{L}\left[A_0 + \sum_{k=1}^{M} 2A_k \cos(2\pi(n-M)k/L)\right] \tag{1.11}$$

where the phase constant of linearity in (1.9) is given by

$$K = -M = -\tfrac{1}{2}(L-1). \tag{1.12}$$

If the length is even, the impulse response is

$$h[n] = \frac{1}{L}\left[A_0 + \sum_{k=1}^{L/2-1} 2A_k \cos(2\pi(n-M)k/L)\right] \tag{1.13}$$

with the same phase constant of linearity as (1.12).

The schemes discussed above assume the frequency samples are $\omega_k = 2\pi k/L$. The other possible evenly spaced sampling scheme is $\omega_k = (2k+1)\pi/L$ which gives the design formula for an odd length as

$$h[n] = \frac{1}{L}\left[\sum_{k=0}^{M-1} 2A_k \cos(2\pi(n-M)(k+\frac{1}{2})/L) + (-1)^n A_M\right] \tag{1.14}$$

again with $M = (L-1)/2$. The impulse response for an even length filter is

$$h[n] = \frac{1}{L}\left[\sum_{k=0}^{L/2-1} 2A_k \cos(2\pi(n-M)(k+\frac{1}{2})/L)\right]. \tag{1.15}$$

If the amplitude has unequally spaced samples, the IDFT cannot be used; instead, the set of simultaneous complex equations formed from

$$H(e^{j\omega_k}) = \sum_{n=0}^{L-1} h[n]\, e^{-j\omega_k n} \tag{1.16}$$

must be solved. If the filter has linear phase and the length is odd, the equations are real and given by

$$A(\omega_k) = \sum_{n=0}^{M-1} 2h[n]\cos(\omega_k(M-n)) + h(M) \tag{1.17}$$

with $M = (L-1)/2$. Since there are $M+1$ unknown $h[n]$ values, there must be $M+1$ equations which require $M+1$ samples of $A(\omega)$ given.

If the filter has linear phase and the length is even, the equations are real and given by

$$A(\omega_k) = \sum_{n=0}^{L/2-1} 2h[n]\cos(\omega_k(M-n)) \tag{1.18}$$

This case requires $L/2$ samples and equations.

We will consider three methods of frequency sampling design of linear phase FIR filters: the use of the IDFT, the use of explicit formulas, and the solution of simultaneous equations. Each has its advantages and disadvantages.

Most text books call the odd and even length linear phase FIR filters type I and type II filters. If the phase response is linear plus a constant $\pi/2$, the odd and even length FIR filters are called type III and IV. These are used for differentiators and Hilbert transformers where the constant plus linear phase response is desired. They have design formulas similar to the type I and II but with sine expansions [3].

Hints

In using the IDFT for the frequency sampling filter design method with MATLAB, the `fft` command is used. It is fairly fast if the length L is composite (very fast if $L = 2^M$), but slow if the length is prime. The inverse FFT is implemented by an M-file that calls `fft`. If unequally spaced samples are used, the equations given by (1.17) and (1.18) are written as $A = Fh$ where A is the column vector of samples of the desired frequency response amplitude, F is the square matrix of cosines from (1.17) or (1.18), and h is the unknown vector of half the impulse response. These are solved in MATLAB via: `h = F \ A`.

When plotting any function of an integer, such as the impulse response of a filter $h[n]$, use `comb`. The frequency response of the filters designed in this chapter can be calculated via `freqz`. In most cases, a fairly large number of frequency response values should be calculated to give a smooth graph when plotted. In most cases both the magnitude and phase should be calculated and plotted.

The location of the zeros of the transfer function of the filter can be calculated and plotted easily in MATLAB. The command `z = roots(h)` will factor the polynomial with $h[n]$ as coefficients and `plot(z,'o')` will plot the imaginary part vs the real part and place small circles at the location of the zeros of the polynomial.[21]

Most of the exercises will consider an ideal linear phase lowpass filter with an amplitude response of

$$A(\omega) = \begin{cases} 1 & \text{if } 0 \le \omega \le \omega_0 \\ 0 & \text{if } \omega_0 < \omega \le \pi \end{cases} \tag{1.19}$$

and some may include a transition band between the pass and stopbands such as

$$A(\omega) = \begin{cases} 1 & \text{if } 0 \le \omega \le \omega_p \\ \dfrac{\omega_s - \omega}{\omega_s - \omega_p} & \text{if } \omega_p < \omega < \omega_s \\ 0 & \text{if } \omega_s < \omega \le \pi \end{cases} \tag{1.20}$$

where ω_p is the edge of the passband, ω_s is the edge of the stopband, and ω_0 is the average band edge $\omega_0 = (\omega_p + \omega_s)/2$. These and perhaps others must be sampled in order to design the filter.

[21]See CASPER special `zplane.m`.

The first exercise illustrates the explicit form of the necessary MATLAB commands. Some of these should be put in the form of a function to be used in the other exercises or in other projects. The later exercises expect you to create the appropriate command or command sequence.

Note: MATLAB starts its addressing of elements in a vector at one, yet many mathematical formulas start their variables at zero. Be careful to take this difference into account.

Exercise 1.1: Design a Lowpass Filter

Design a length-23 linear phase FIR lowpass filter to approximate an ideal response which has a passband edge of $\omega_0 = 0.3\pi$. Assume a unity sampling rate which gives a Nyquist frequency of $\omega = \pi$.

(a) Form a vector of the samples of the ideal amplitude which will be ones for frequencies from zero to the band edge and zero from there up to π. Show the following MATLAB commands do this.

```
pass = fix(w0*L/(2*pi));
if rem(L,2)==0, s = -1; else s = 1; end;
Ad = [ones(1,pass), zeros(1,L-2*pass+1), s*ones(1,pass-1)];
```

Plot `Ad` to see the ideal amplitude frequency response. What happens if one of the samples falls on ω_0? Explain why the line with the `if` statement is necessary.

(b) Create a vector that when multiplied point-by-point by the amplitude, gives the complex linear phase frequency response. Show this is done by

```
M1 = (L-1)/2;
k = [0:L-1];
p = exp(2*pi*j*(-M1)*k/L); %--- j = sqrt(-1)
```

The sampled frequency response vector is simply a term by term product of this and the amplitude done by `H = Ad.*p;`.

(c) Design the filter by using the IDFT with the MATLAB command

```
h = ifft(H).
```

This should be the real, symmetric impulse response $h[n]$. Remove the zero imaginary part by `h = real(h);` . Plot $h[n]$ using the `comb` command.

(d) Test the filter by calculating its magnitude frequency response. This can be easily done by

```
Mag = abs(fft(h,512)); %--- or use freqz( )
w = [0:255]*pi/256;
plot(w, Mag(1:256));
```

Does it look like a good lowpass filter with the correct cutoff frequency? Why do we plot only half of `Mag`?

(e) Calculate its amplitude response by removing the linear phase from $H(e^{j\omega})$. Show this can be done by

```
M1 = (L-1)/2;
k = [0:L-1];
p = exp(2*pi*i*M1*k/L);
Amp = real(p.*fft(h,512));
plot(Amp(1:256));
```

The magnitude and amplitude should be the same except where the amplitude is negative. Is that true for your plots?

(f) Test to see if the frequency response passes through the specified points. This is done by appending a multiple of 23 zeros to $h[n]$ and taking the DFT. Some of the calculated values will be at the same frequencies of the original desired samples. Compare the frequency response of your designed filter at these points to see if the magnitude is the appropriate one or zero. Plot the magnitude or amplitude and the ideal magnitude or amplitude on the same graph to show the interpolation.

(g) Plot the phase to see if it is linear with the appropriate constant of linearity. Notice the size and location of the jumps in the phase plot. Notice the unpredictable phase where the magnitude is zero.

(h) Plot the location of the zeros on the complex z-plane with `zplane.m` or via

```
plot(roots(h),'o');
```

Relate the zero locations to the shape of the frequency response plots.

Exercise 1.2: Use the wrong phase

Experiment with designing filters with the same desired amplitude, but with a phase that is not consistent with an integer length filter. Try other phase responses and discuss the effects on the final design and its actual amplitude response.

Exercise 1.3: An Alternate Approach

To see an alternate approach that uses only real functions, repeat exercise 1.1. but with zero phase. In other words, use $H(e^{j\omega}) = A(\omega)$. Note the results are simply a time shift which gives a non-causal impulse response symmetric about the origin. This works only for odd length filters. Why?

Exercise 1.4: Design of Even Length Filters

In order to see the difference in odd and even length FIR filters, repeat exercise 1.1 for $L = 22$. In this case the constant of linearity for the phase $M = (L-1)/2$ is not an integer. It is an odd multiple of 0.5. Calculate and plot the magnitude and phase frequency response. Check values at the sample points. Plot the location of the zeros. Why does a symmetric even length linear phase FIR filter always have a zero at $\omega = \pi$?

Exercise 1.5: Derive Design Formulas

Derive (1.11) from the linear phase condition and (1.3). Derive (1.14).

Exercise 1.6: Alternate Sampling

Since formulas can be used for frequency sampling design of FIR filters, use (1.11) to design an FIR filter with the same specifications as given in exercise 1.1. Use the alternative sampling scheme implemented in (1.14) with the same specifications. What is the difference in the frequency response of the two filters? Which seems better and why? Can you devise a method to use the IDFT and achieve the alternative sampling scheme design?

Exercise 1.7: Gibbs Phenomenon

A Gibbs type phenomenon occurs in the frequency sampling design of filters, much the same as it does with a direct Fourier expansion of a desired frequency response with a discontinuity. In the case of the Fourier transform the peak of the overshoot is approximately 9% of the size of the discontinuity. What is the size of the corresponding overshoot in the frequency response of a filter designed by the frequency sampling method?

Exercise 1.8: A Transition Band

The above exercises in this project all use ideal amplitudes which are one or zero at each frequency sample. This means the transition from pass to stop band takes place in one frequency sampling interval which is a fast transition but the filters all also have rather large amounts of oscillation in the frequency response near the band edge. Modify the transition to be two sampling intervals by changing the sample nearest the band edge to one-half rather than one or zero and design an FIR filter with the same specification as used in exercise 1.1. Compare the frequency response of this filter with that in exercise 1.1 in terms of the rate of drop-off between the pass and stop bands and in terms of the overshoot. Notice the trade-off of these two characteristics. To further investigate the relationship of transition band width and overshoot or Gibbs effect, make the transition three sample intervals wide with samples of 0.667 and 0.333. This should give a wider or slower transition and less overshoot.

Exercise 1.9: Optimize in the Transition Band

Create a transition band between the passband and stopband. Rather than use the linear transition function shown in (1.20), experiment with values of the frequency samples in the transition band to reduce the overshoot or maximum oscillations in the pass and stopbands. What changes from the use of a straight line do you find? How much reduction in overshoot can you obtain? This process can be automated by the use of linear programming [1].

Exercise 1.10: Don't Care Transition Band

Create a transition band between the passband and stopband in the ideal frequency response where there are no samples. This will cause the frequency samples to be unevenly spaced and, therefore, prevent the use of the IDFT or the formulas in (1.11) through (1.15). Use the formula (1.17) to obtain equations which can be solved by MATLAB to design the filter. How does the frequency response and zero locations compare with the filters designed by the IDFT or formulas?

Project 2: FIR filter design by discrete least squared error approximation

The square of a signal or the square of an error is a measure of the power in the signal or error. This is clear if the signal is a voltage, current, velocity, or displacement. The time integral of the power of a signal is its energy and is often an important measure of the signal. In many practical problems, the integration cannot be mathematically carried out and is, therefore, approximated by a finite summation. It is this finite sum of the square of the difference of the desired frequency response and the actual frequency response that we will use as our approximation measure in this project. For equally spaced frequency samples, Parseval's theorem states that an optimal frequency domain approximation implies an optimal time domain approximation.

Project Description

The discrete squared error measure is defined by

$$\epsilon = \frac{1}{N}\sum_{k=0}^{N-1}\left|H_d(e^{j\omega_k}) - H(e^{j\omega_k})\right|^2 \tag{2.1}$$

where $H_d(e^{j\omega})$ is the desired ideal frequency response, $H(e^{j\omega})$ is the actual response of the length-L filter given by (1.2) and (1.17), and N is the number of frequency points over which the error is calculated. If the number of independent filter coefficients $h[n]$ is equal to the number of requirements or equations set in (1.2), it is possible to choose the $h[n]$ such that there is no error. This is what was done in the frequency sampling design method in project 1. By choosing $N \gg L$, the summed squared error in (2.1), appropriately normalized, approaches the integral squared error which is often what is actually wanted in approximating $H_d(e^{j\omega})$.

Using Parseval's theorem, one can show that symmetrically truncating a length-N filter designed by frequency sampling will give a length-L filter whose frequency response is an optimal approximation to $H_d(e^{j\omega})$ in the sense that ϵ in (2.1) is minimized. This is true only for equally spaced samples because that is the requirement of Parseval's theorem. This result is similar to the fact that a truncated Fourier series is an optimal least squared error approximation to the function expanded. The long filter that is to be truncated may be designed by using the IDFT or the formulas in (1.11) and (1.13).

If the frequencies are not equally spaced, truncation will not result in an optimal approximation. If $N > L$, the equations given by (1.17) and (1.18) are over determined and may be written in matrix form as

$$A = Fh \tag{2.2}$$

where A is the length-N vector of samples of $A(\omega)$, F is the $N \times L$ matrix of cosines, and h is the length-L vector of the filter coefficients. Because of $A(\omega)$ being an even function, only $L/2$ terms are needed in A. Although these equations are over determined, they may be approximately solved in MATLAB with `h = F \ A`. MATLAB implements this operation with an algorithm that minimizes the error in (2.1).

The goal of this project is to learn how to design filters that minimize the discrete squared error, to understand the properties of this design method, and to examine the properties of the filters so designed.

Hints

The DFT implemented in the MATLAB function `fft` or an M-file implementation of the formulas (1.11) through (1.15) will be used to design the long filter to be truncated for most of the exercises in this project. In some cases, we will solve sets of overdetermined equations to obtain our approximations. Read the manual and use `help` on \ and / to learn about the approximate solution of over determined equations. It might be helpful to read about least squared error methods in references such as [6].

Many of the design and analysis methods used in this project are extension of those in project 1. Review the description and discussion in that project.

To analyze, evaluate, and compare the filters designed by the various methods in this project, magnitude or amplitude frequency response plots, plots of zero locations, and plots of the filter itself should be made. It would be efficient to create special M-file functions that efficiently make these plots.

Exercise 2.1: Design an Odd Length Lowpass Filter

Design a length-23 linear phase FIR lowpass filter to approximate an ideal response which has a passband edge of $\omega_0 = 0.3\pi$. Assume a unity sampling rate which gives a Nyquist frequency of $\omega = \pi$. Use the frequency sampling method described in project 1 to design three filters of lengths 45, 101, and 501. Truncate them to symmetric filters of length-23 and compare them with each other. Compare the frequency responses and zero locations.

Exercise 2.2: How Many Samples Should be Used?

If one really wants to minimize the integral squared error but must use the discrete squared error method, what is the ratio of N, the length of the filter to be truncated, to L, the length of the filter, to obtain close results?

Exercise 2.3: Residual Error

Verify Parseval's relation of the time-domain and frequency-domain sums of squares by calculating the approximation error of the length-23 filter designed in exercise 2.1 using a frequency sampling length of 101. Do this in the frequency domain using (2.1) and the time domain from the sum of the squares of the truncated terms of $h[n]$.

Exercise 2.4: Use of Overdetermined Equations

Design a length-23 filter with the specification from exercise 2.1 but use the solution of overdetermined equations of (1.17) rather than truncation of a longer filter. Do this for the three values of N of 45, 101, and 501. You should get the same results as in exercise 2.1. How does the design time compare? Are there numerical problems?

Exercise 2.5: Use a Transition Band

Design a length-23 filter with specifications similar to those in exercise 2.1, but with a transition band. Let the passband be the range of frequencies $\{0 \leq \omega \leq .25\pi\}$, the stopband be the range $\{.35\pi \leq \omega \leq \pi\}$, and the transition be a straight line connecting the two as described in (1.20). Use the same least discrete squared error criterion used in exercise 2.1

for the same three number of frequency samples. Compare the results with each other and with those in exercise 2.1. Plot the frequency response of the three filter on the same graph to compare. Note the reduction of pass and stopband ripple vs. the increase of transition band width.

Exercise 2.6: Don't Care Transition Band

Design a length-23 filter with the same specifications as exercise 2.5, but with no frequency samples in the transition band. This means the frequency samples are not equally spaced and simultaneous equations from (1.17) will have to be solved approximately as was done in exercise 2.4. Do this for the same three numbers of frequency samples. Compare with the results of exercise 2.5 by plotting the frequency responses on the same graph.

Exercise 2.7: Weighting Functions

It is possible to use a weighting function in the definition of error.

$$\epsilon = \sum_{k=0}^{L-1} W_k \left| H_d(e^{j\omega_k}) - H(e^{j\omega_k}) \right|^2 \tag{2.3}$$

Derive a matrix formulation for the set of over determined simultaneous equations describing this problem. Design a length-23 filter with the same specifications as in exercise 2.6 but with 10 times the weight on the stopband squared error as on the passband squared error. Discuss the result.

Exercise 2.8: Numerical Problems

If long filters with wide transition bands are designed, the simultaneous equations to be solved will be nearly singular. This ill-conditioning seems to be a function of the product of the filter length and the transition band width. For what length-bandwidth product does this start to occur? Plot the frequency response of some filters designed by solving ill-conditioned equations. What are their characteristics?

Computer-Based Exercises for Signal Processing

$\boxed{\text{Least Squares Design of FIR Filters}}$

Least Squares Design of FIR Filters

Overview

The use of an approximation measure which is the integral of the square of the error is often used in filter design since it is in some ways a measure of the energy of the error. It is also attractive because Parseval's theorem states that an optimal approximation in the frequency domain using the integral squared error criterion will also be an optimal approximation in the time domain. In general, one cannot analytically solve this optimization problem, but in several important cases, analytical solutions can be obtained. This section will investigate the design of lowpass, highpass, bandpass, band reject filters, with and without transition bands. It will then look at the use of window functions to reduce the Gibbs effect that sometimes occurs.

Background Reading

The basic least integral squared error of the no-transition band lowpass filter is discussed in most DSP text books as the Fourier series expansion method. It is discussed in [3] as a least squared error method. The use of spline transition bands is developed in [3, 7].

Project 1: FIR filter design by least integral squared error approximation

In many filter design problems, it is the least integral squared error approximation that is desired. Although in most of these cases, the problem cannot be solved, there are a few important cases that do have analytical solutions. This project will examine the ideal linear phase lowpass filter with no transition band which gives a $\sin(x)/x$ form impulse response. It will also consider ideal frequency response with transition bands and will compare results with those obtained numerically using discrete least squared error methods.

Project Description

The integral square error measure is defined by

$$\epsilon = \frac{1}{2\pi}\int_{-\pi}^{\pi}\left|H_d(e^{j\omega}) - H(e^{j\omega})\right|^2 \, d\omega \tag{1.1}$$

where $H_d(\omega)$ is the desired ideal frequency response and $H(\omega)$ is the actual frequency response of the length-L filter. Because of the orthogonality of the basis functions of the discrete-time Fourier transform, Parseval's theorem states this same error can be given in the time domain by

$$\epsilon = \sum_{n=-\infty}^{\infty} |h_d[n] - h[n]|^2 \tag{1.2}$$

where $h_d[n]$ is the inverse DTFT of $H_d(\omega)$ and $h[n]$ is the length-L impulse response of the filter being designed.

If $h_d[n]$ is infinitely long, but symmetric, and $h[n]$ is of length-L, for an odd length (1.2) can be written in two parts as

$$\epsilon = \sum_{n=-M}^{M} |h_d[n] - h[n]|^2 + \sum_{n=M+1}^{\infty} 2h_d^2[n]. \tag{1.3}$$

It is clear that choosing the unknown $h[n]$ to be equal to the given $h_d[n]$ minimizes ϵ in (1.3) and, therefore, in (1.1). In other words, symmetric truncation of $h_d[n]$ gives the optimal approximation of the frequency response. The only problem in using this result is the IDTFT of a desired frequency response can often not be calculated analytically.

The ideal, no transition band lowpass filter has a frequency response described by

$$A_d(\omega) = \begin{cases} 1 & \text{if } 0 \le \omega \le \omega_0 \\ 0 & \text{if } \omega_0 < \omega \le \pi \end{cases} \tag{1.4}$$

where ω_0 is the band edge between the pass and stopbands. The IDTFT of $A_d(\omega)$ is

$$h_d[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} A_d(\omega)\, e^{j\omega n}\, d\omega = \frac{\sin(\omega_0 n)}{\pi n} \tag{1.5}$$

Since we inverted $A_d(\omega)$ rather than $H_d(e^{j\omega})$, this impulse response must be shifted as well as truncated to give the optimal result of

$$h[n] = \begin{cases} \dfrac{\sin(\omega_0(n-M))}{\pi(n-M)} & \text{for } 0 \le n \le L-1 \\ 0 & \text{otherwise} \end{cases} \tag{1.6}$$

If a transition band is included in the ideal frequency response, a transition function must be specified. If that transition function is a straight line (first order spline), the ideal amplitude is

$$A(\omega) = \begin{cases} 1 & \text{if } 0 \le \omega \le \omega_p \\ \dfrac{\omega_s - \omega}{\omega_s - \omega_p} & \text{if } \omega_p < \omega < \omega_s \\ 0 & \text{if } \omega_s \le \omega \le \pi \end{cases} \tag{1.7}$$

where ω_p is the edge of the passband, ω_s is the edge of the stopband, and ω_0 is the average band edge $\omega_0 = (\omega_p + \omega_s)/2$. The IDTFT of $A(\omega)$ is

$$h_d[n] = \frac{\sin(\omega_0 n)}{\pi n} \frac{\sin(\Delta n/2)}{\Delta n/2} \tag{1.8}$$

where $\Delta = (\omega_s - \omega_p)$ is the transition band width. This has the form of the ideal no transition band impulse response of (1.5) multiplied by a wider envelope controlled by the transition band width. The truncated and shifted version of this has a frequency response which is an optimal approximation to (1.7).

Although these formulas are described here for odd length FIR filters, they hold for even lengths as well. Other transition functions also can be used but must be chosen such that the IDTFT can be taken. For cases where the integral in the IDTFT cannot be carried out, one must use the numerical methods used in projects on discrete methods.

Hints

This project will require designing filters whose coefficients are given by formulas. These should be programmed in M-files so they may easily be used on different specifications. The resulting filters will be examined by plotting their magnitude or amplitude response, transfer function zero locations, and impulse response. This should be reviewed from the projects on discrete methods.

An important analysis of filter length can be made by plotting the approximation error in (1.1) vs. the filter length L. This can be calculated by approximating the integral by a summation over a dense grid of frequency samples or by calculating the major part of the second term in (1.3). An approximation error evaluating function should be written to calculate this error efficiently. It can be put in a loop to calculate the needed error vs. length data.

Exercise 1.1: Design an Odd Length Lowpass Filter

Design a length-23 linear phase FIR lowpass filter to approximate an ideal response which has a passband edge of $\omega_0 = 0.3\pi$. Assume a unity sampling rate which gives a Nyquist frequency of $\omega = \pi$. Plot the impulse response with the `comb` command. Plot the magnitude or amplitude frequency response of the filter. Plot the transfer function zero locations. How do these compare with the designs of filters with the same specifications in the projects using discrete methods? The results should be fairly close to that of the discrete squared error designs with large numbers of frequency samples but should be noticeably different for the designs with fewer samples or with the frequency sampling method. The pure Gibbs phenomenon predicts a maximum overshoot of approximately 9% of the discontinuity. Is that observed in the frequency response of this design?

Exercise 1.2: Design a Long FIR Filter

Design a length-51 FIR filter to the same specifications used in exercise 1 above. Plot the impulse response, frequency response, and zero location of the transfer function. You may want to use a log plot of the magnitude response plot of longer filters to better see details in the stopband. How do they compare with similar plots from exercise 1? How does the maximum overshoot compare? How does the integral squared error of the two filters compare?

Exercise 1.3: Approximation Error

Plot the approximation error of an optimal length-L filter designed to the specifications of exercise 1 vs. the length of the filter. Derive an empirical formula relating the error to the filter length.

Exercise 1.4: Use a Transition Band

Design a length-23 filter with specifications similar to those in exercise 1, but with a transition band. Let the passband be the range of frequencies $\{0 \leq \omega \leq 0.25\pi\}$, the stopband be the range $\{0.35\pi \leq \omega \leq \pi\}$, and the transition be a straight line connecting the two. Use the formula (1.8). This filter should be compared to the no transition band design in exercise 1 by plotting the frequency response. In particular, compare the overshoot, the rate of change from pass to stopband, and the approximation error.

Exercise 1.5: Approximation Error

Plot the approximation error of an optimal length-L filter designed to the specifications of exercise 4 vs. the length of the filter. This curve will have regions that are relatively flat?

Explain this curve by looking at the formula (1.8) and considering the effects of truncation. Derive a formula for the locations of these flat regions from (1.8).

Exercise 1.6: Spline Transition Function

The simple straight line transition function used in (1.7) can be generalized to a p^{th} order spline [3] which gives an ideal impulse response of

$$h_d[n] = \frac{\sin(\omega_0 n)}{\pi n}\left(\frac{\sin(\Delta n/2p)}{\Delta n/2p}\right)^p \tag{1.9}$$

Design three length-23 filter to the specifications of exercise 4 using values of $p = 1$, 2, and 10 and one filter with no transition band. Plot their amplitude response on the same graph. How does the value of p affect the frequency response? ($p = \infty$ is equivalent to no transition band)

Exercise 1.7: Optimal Order Spline Transition Band

Use the spline transition function method of (1.9) to design length-L FIR filters with specifications of exercise 4. Make a graph which contains plots of approximation error vs length for values of $p = 1$, 2, 3, 4, and 5.

Exercise 1.8: Error Formula

Plot the approximation error of an optimal FIR filter with a transition band vs. the transition band width Δ for a length of 23 and an average band edge of $\omega_0 = .3\pi$. Derive an empirical formula relating the error to the transition band width. Does this result depend significantly on ω_0?

Exercise 1.9: Optimal Filter

Analyze the spline transition function method for various lengths, transition band widths, and values of p. From equation (1.9) and from the empirical evaluation of error vs. length and other parameter curves, derive an empirical formula for an optimal value of p as a function of L and Δ. Write an M-file program that will design an optimal filter from L, ω_p, ω_s by choosing it own value of p and evaluating (1.9). Evaluate the designs.

Exercise 1.10: A Comparison

Because the inverse discrete-time Fourier transform of many ideal responses cannot be analytically evaluated, the numerical methods of the project on discrete methods must be used. Compare truncated long discrete least squared error designs to the designs by formula of this project. How many error samples should one take to give results close to the integral? How long can the filters be designed by the two methods?

Project 2: Design of highpass, band pass and band reject least squared error FIR Filters

Earlier projects on discrete methods developed FIR filter design methods but illustrated them only on low pass specifications. This project will consider the approximation problems of highpass, bandpass, and band reject filters using techniques that convert lowpass designs. These techniques are interesting mostly for the analytic methods developed in project 1 since the numerical methods in the projects on discrete methods can be applied directly to the new specifications. These methods will work only with unweighted least squared error designs.

Project Description

The bandpass filter with no transition bands has an ideal amplitude response of

$$A_d(\omega) = \begin{cases} 0 & \text{for } 0 \le \omega \le \omega_1 \\ 1 & \text{for } \omega_1 < \omega < \omega_2 \\ 0 & \text{for } \omega_2 \le \omega \le \pi \end{cases} \tag{2.1}$$

where the lower passband edge is ω_1 and the upper is ω_2.

This can be obtained by subtracting the responses of two lowpass filters. It is the response of a lowpass filter with band edge at ω_1 minus the response of a lowpass filter with band edge at ω_2. Since the ideal responses are

$$A_{bp}(\omega) = A_{lp1}(\omega) - A_{lp2}(\omega) \tag{2.2}$$

and the IDTFT is linear,

$$h_{bp}[n] = h_{lp1}[n] - h_{lp2}[n]. \tag{2.3}$$

This also holds if the ideal bandpass response has transition bands by simply using transition bands on the lowpass filters. Indeed, it allows different width transition bands.

The bandpass filter can also be generated by modulating a lowpass filter. Multiplying the impulse response of a lowpass filter by sampled sinusoid will shift its frequency response. This property of the DTFT allows designing a prototype lowpass filter and then multiplying its impulse response by a sinusoid of the appropriate frequency to obtain the impulse of the desired bandpass filter. This method, if used with transition bands, will not allow independent control of the two transition bands.

The ideal highpass filter with no transition band has an amplitude response of

$$A_d(\omega) = \begin{cases} 0 & \text{for } 0 \le \omega < \omega_0 \\ 1 & \text{for } \omega_0 \le \omega \le \pi \end{cases} \tag{2.4}$$

It can be generated by subtracting the response of a lowpass filter from unity or by multiplying the impulse response of a prototype lowpass filter by $(-1)^n$ which shifts the stopband of the lowpass filter to center around $\omega = 0$.

The ideal band reject filter with no transition bands has an amplitude response of

$$A_d(\omega) = \begin{cases} 1 & \text{for } 0 \le \omega \le \omega_1 \\ 0 & \text{for } \omega_1 < \omega < \omega_2 \\ 1 & \text{for } \omega_2 \le \omega \le \pi \end{cases} \tag{2.5}$$

where the lower reject band edge is ω_1 and the upper is ω_2.

This can be obtained by adding the response of a highpass filter from that of a lowpass filter or of subtracting the response of a bandpass filter from unity. It also can be obtained through modulation by shifting the stopband region of a lowpass prototype filter to the reject band of the new filter.

Hints

The tools for working this project are the same as the earlier projects of this chapter. You will need to be able to easily and reliably design lowpass filters with and without transition bands. The "Hints" section of Project 1 and its exercise 1 in the packet on discrete methods are very helpful for all of the projects in this chapter.

Exercise 2.1: Design a Bandpass Filter

Design a least squared error linear phase bandpass FIR filter with the lower passband edge at $\omega_1 = 0.2\pi$ and the upper passband edge at $\omega_2 = 0.3\pi$ using no transition bands and a length of 31. Use the design method that subtracts the designs of two lowpass filters. Plot the impulse response, the amplitude response, and the zero locations. Does it seem like a good approximation to the ideal? Show this by plotting the ideal and actual amplitude response on the same graph. Do the location of the zeros make sense and agree with the frequency response?

Exercise 2.2: Design a Bandpass Filter using Modulation

Design a bandpass filter using the same specifications as in exercise 1 but use the modulation design method. That will require some care in choosing the band edge of the prototype lowpass filter and the frequency of the sinusoid to achieve the desired ω_1 and ω_2. Check by comparing with the design of exercise 1.

Exercise 2.3: Design a Band Reject Filter

Design a band reject filter using the same band edges as the bandpass filter in exercise 1. Analyze its impulse response, frequency response, and zero locations. Compare with the ideal.

Exercise 2.4: Design a Highpass Filter

Design a length-23 highpass FIR filter with a band edge at $\omega_0 = 0.3\pi$. Design it by both the subtraction and the shifting methods. Analyze the filter by plotting the impulse response, amplitude response, and zero locations. Show that you cannot design a highpass even length FIR filter. Why is this?

Exercise 2.5: Use a Transition Band

Design a length-31 bandpass filter with transition bands. Set the lower stopband as $\{0 \le \omega \le 0.08\pi\}$, the passband as $\{0.1\pi \le \omega \le 0.3\pi\}$, and the upper stopband as $\{0.4\pi \le \omega \le \pi\}$. Analyze its frequency response by plotting the amplitude response on the same graph as the ideal. Look at its other characteristics.

Exercise 2.6: Design a Multi-Passband Filter

Design a multi-passband filter of length 51 with no transition bands. Set one passband as $\{0 \le \omega \le 0.2\pi\}$ and a second as $\{0.3\pi \le \omega \le 0.4\pi\}$. The first passband should have a gain of one and the second should have a gain of one half. Plot the amplitude response and the ideal on the same graph.

Project 3: FIR filter design using window functions

Although the least squared error approximation design methods have many attractive characteristics, the Gibbs phenomenon of a relatively large overshoot near a discontinuity in the ideal response is sometimes objectionable. It is the truncation of the infinitely long ideal impulse response that causes the overshoot, so the use of window functions to more gently truncate the sequence has been developed. The result is a hybrid method that starts out with a least squared error approximation but modifies it to reduce the Chebyshev error. The window function method is an alternative to defining $H_d(e^{j\omega})$ in the transition bands as in project 1. This project develops and analysis several standard window based FIR filter design methods.

Project Description

The window method of FIR filter design starts with the design of a least squared error approximation. If the desired filter has a basic lowpass response, the impulse response of the optimal filter given in (1.5) is

$$\hat{h}_d[n] = \frac{\sin(\omega_0 n)}{\pi n} \tag{3.1}$$

The shifted and truncated version is

$$h[n] = \begin{cases} \dfrac{\sin(\omega_0(n-M))}{\pi(n-M)} & \text{for } 0 \le n \le L-1 \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

for $M = (L-1)/2$. The truncation was obtained by multiplying (3.1) by a rectangle function. Multiplication in the time domain by a rectangle is convolution in the frequency domain by a sinc function. Since that is what causes the Gibbs effect, we will multiply by a window function that has a Fourier transform with lower sidelobes.

One method of smoothing the ripples caused by the sinc function is to square it. This results in the window being a triangle function, also called the Bartlett window; see `triang` and `bartlett` in MATLAB.[22]

The four generalized cosine windows are given by[23]:

$$W[n] = \begin{cases} a - b\cos\left(\dfrac{2\pi n}{L-1}\right) + c\cos\left(\dfrac{4\pi n}{L-1}\right) & \text{for } 0 \le n \le L-1 \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

The names of the windows and their parameters are:

Window	MATLAB name	a	b	c
Rectangular	`boxcar`	1	0	0
Hann	`hanning`	0.5	-0.5	0
Hamming	`hamming`	0.54	-0.46	0
Blackman	`blackman`	0.42	-0.5	0.08

[22]In MATLAB the `triang` and `bartlett` functions give *different* answers.

[23]as defined in MATLAB. In some definitions, the denominator will be taken as L.

The Kaiser window is given by

$$W[n] = \begin{cases} \dfrac{I_0(\beta\sqrt{1-[2(n-M)/(L-1)]^2})}{I_0(\beta)} & \text{for } 0 \le n \le L-1 \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

where $M = (L-1)/2$, $I_0(x)$ is the zero-th order modified Bessel function of the first kind and β is a parameter to adjust the width and shape of the window.

The generalized cosine windows have no ability to adjust the trade-off between transition band width and overshoot and, therefore, are not very flexible filter design tools. The Kaiser window, however, has a parameter β which does allow a trade-off and is known to be an approximation to an optimal window. An empirical formula for β that reduces the Gibbs overshoot is

$$\beta = \begin{cases} 0.1102(A-8.7) & \text{for } 50 < A \\ 0.5842(A-21)^{.4} + 0.07886(A-21) & \text{for } 21 < A < 50 \\ 0 & \text{for } A < 21 \end{cases} \tag{3.5}$$

where

$$A = -20\log_{10}\delta \tag{3.6}$$

$$\Delta = \omega_s - \omega_p \tag{3.7}$$

$$L - 1 = \frac{A-8}{2.285\Delta} \tag{3.8}$$

with δ being the maximum ripple in the pass and stopbands.

Because the Bartlett, Hanning and Blackman windows are zero at their end points, multiplication by them reduces the length of the filter by two. To prevent this shortening, these windows are often made $L+2$ in length. This is not necessary for the Hamming or Kaiser windows.

These windows not only can be used on the classical ideal lowpass filter given in (3.1) or (3.2) but can be used on any ideal to smooth out a discontinuity.

Hints

MATLAB has window functions programmed but you will learn more and understand the process better by writing your own window M-files. However, you will find it instructive to examine the MATLAB M-files using the `type` command. The standard filter analysis tools described in Project 1 are useful.

Exercise 3.1: Design a Lowpass Filter using Windows

Design a length-23 linear phase FIR lowpass filter with a band edge of $\omega_0 = 0.3\pi$ using the following five different windows.

(a) Rectangular

(b) Triangular or Bartlett

(c) Hanning

(d) Hamming

(e) Blackman

Plot the impulse response, amplitude response, and zero locations of the four filters. Compare the characteristics of the amplitude response of the five filters. Compare them to an optimal Chebyshev filter designed with a transition band.

Exercise 3.2: Design a Bandpass Filter using Windows

Take the bandpass filter designed in exercise 2.5 and apply the five windows. Analyze the amplitude response.

Exercise 3.3: Use the Kaiser Window

(a) Plot the relationship in (3.5) to see the usual range for β. Why is $\beta = 0$ for $A < 21$?

(b) Design a length-23 filter using the same specifications as in exercise 3.1, but using a Kaiser window with $\beta = 4$, 6, and 9. Plot the impulse response, amplitude response, and zero locations of the three filters. Compare them with each other and with the results of exercise 3.1. How does the trade-off of transition band width and overshoot vary with β?

Exercise 3.4: Design of Bandpass Filters

Apply the Kaiser window with the three values of β given in exercise 3.3 to the bandpass filter as was done in exercise 3.2. Analyze the amplitude response and compare with the results of exercise 3.2.

Exercise 3.5: Chebyshev Error of the Kaiser Window

(a) Set specifications of a length-23 lowpass FIR filter with passband in the range $\{0 \leq \omega \leq 0.3\pi\}$ and stopband in the range $\{0.35\pi \leq \omega \leq \pi\}$. Design a set of filters using the Kaiser window with a variety of values for β. Calculate the Chebyshev error over the passband and stopband using the `max` command in Matlab. Plot this Chebyshev error vs β and find the minimum. Compare with the value given by the empirical formula in (3.5).

(b) *Another Chebyshev Error Computation.* Repeat part (a) but use the squared error calculated only over the pass and stopband rather than the total $0 \leq \omega \leq \pi$. Run experiment over various lengths and transition band widths to determine an empirical formula for β that minimizes the squared error.

Computer-Based Exercises

for

Signal Processing

Chebyshev Design of FIR Filters

Chebyshev Design of FIR Filters

Overview

One of the most important error measures in the design of optimal FIR filters is the maximum difference between the desired frequency response and the actual frequency response over the range of frequencies of interest. This is called the Chebyshev error and is what is most obvious in a visual evaluation of a frequency response. When this error is minimized, the error takes on shape that oscillates with equal size ripples. The minimization of this error in a filter design problem is usually done using the Parks-McClellan algorithm or linear programming. The characteristics of the solution are described by the alternation theorem. This section will investigate the Remez exchange algorithm used by Parks and McClellan and the use linear programming.

Background Reading

Details of the Parks-McClellan algorithm and the Remez exchange algorithm can be found in [1, 8, 3]. The basic ideas of linear programming can be found in a number of books such as [9] and the application to filter design can be found in [10]. Programs implementing the Parks-McClellan are in the MATLAB command `remez` and linear programming algorithms implemented through the simplex, quadratic programming, or Karmarkar's methods are available in the MATLAB optimization toolbox or other sources.

Project 1: FIR filter design by the Parks-McClellan method

This project will design filters using the Parks-McClellan method to observe the speed of the method, the characteristics of Chebyshev approximations, the extra ripple phenomenon, and to observe the relationship between L, δ_p, δ_s, and $\Delta = \omega_s - \omega_p$.

Project Description

Consider the characteristics of an optimal length-21 linear phase lowpass FIR filter as the passband edge is changed. Consider the alternations, ripple, extremal frequencies, and root locations of this filter designed by the Parks-McClellan algorithm in MATLAB. The important band edge frequencies are given below. We have $L = 21$, therefore, $M = (L-1)/2 = 10$. The alternation theorem states that the optimal Chebyshev approximation must have at least $M+2 = 12$ extremal frequencies and in the case of the extra ripple filter will have $M+3 = 13$ extremal frequencies.

Hints

Use the MATLAB command `remez` to design the filter. You might also try the program developed in project 1 and/or the linear programming approach of project 2 to check answers and compare design speeds. Use the `help remez` statement to learn more about the remez command.

All filter design functions in MATLAB use a frequency scaling that is somewhat non-standard. When entering the cutoff frequencies ω_p and ω_s (units of radians), the values must be divided by π. Thus a cutoff frequency specified as $\omega_p = 0.22\pi$ would be entered as

`0.22` in MATLAB. This scaling is not the usual normalized frequency where the sampling frequency is one. Therefore, if the specifications call for a cutoff frequency of 1000 Hz when the sampling frequency is 5000 Hz, then you must enter `0.4` in MATLAB, because the cutoff frequency in radians is $\omega_c = 2\pi(1000/5000) = 2\pi(0.2) = 0.4\pi$. In this example, the normalized frequency would be $1000/5000 = 0.2$, so the MATLAB frequency is twice the normalized frequency.

Exercise 1.1: Design a length-21 FIR filter

Use the MATLAB command `h = remez(20, [0,0.4,0.5,1], [1,1,0,0])` to design a length-21 filter with a passband from 0 to $\omega_p = 0.4\pi$ and a stopband from $\omega_s = 0.5\pi$ to π with a desired response of one in the passband and zero in the stopband. Plot the impulse response, the zero locations, and the amplitude response. How many "ripples" are there? How many extremal frequencies are there? How many "small ripples" are there that are not extremal frequencies and, if there are any, are they in the passband or stopband? Are there zeros that do not directly contribute to a ripple?

Exercise 1.2: Characteristics vs. passband edge location

Design and answer the questions posed in exercise 1.1 for a set of different passband edges ($f_p = \omega_p/\pi$). Use `h = remez(20, [0,fp,0.5,1], [1,1,0,0])`. Do this for passband edges of

```
fp =  0.2000, 0.207, 0.2100, 0.222, 0.2230, 0.224, 0.225, 0.230
      0.2310, 0.240, 0.2900, 0.300, 0.3491, 0.385, 0.386, 0.401
      0.4015, 0.402, 0.4049, 0.412, 0.4130, 0.500.
```

These frequencies include points where the characteristics change. Discuss these in light of the alternation theorem.

Exercise 1.3: Characteristics of narrow band filters

Design a family of narrow band filters and discuss how their characteristics compare with the filters above and how they change with a changing passband edge.

Project 2: The alternation theorem and the Remez exchange algorithm

The *alternation theorem* [11] states that an optimal Chebyshev approximation will have an error function which oscillates with a given number of equal magnitude ripples that alternate in sign. The *Remez exchange algorithm* is a clever method of constructing that equal ripple Chebyshev approximation solution. Rather than directly minimizing the Chebyshev error, this algorithm successively exchanges better approximations for the locations of error ripples. It is guaranteed to converge to the optimal equiripple solution under rather general conditions and it is the basis of the Parks-McClellan algorithm for designing linear phase FIR filters. This project examines the mechanics and characteristics of this important and powerful algorithm.

Project Description

The frequency response of a length-L FIR filter with impulse response $h[n]$ is given by the DTFT as

$$H(e^{j\omega}) = \sum_{n=0}^{L-1} h[n]\, e^{-j\omega n}. \tag{2.1}$$

For an odd length, linear phase FIR filter, the impulse response has even symmetry and (2.1) becomes

$$H(e^{j\omega}) = e^{-jM\omega} \sum_{n=0}^{M} a[n]\, \cos(\omega n) \tag{2.2}$$

where $M = (L-1)/2$ is the group delay of the filter and the constant of linearity for the phase. This can be written

$$H(e^{j\omega}) = e^{-jM\omega} A(\omega) \tag{2.3}$$

where

$$A(\omega) = \sum_{n=0}^{M} a[n]\, \cos(\omega n) \tag{2.4}$$

is an even real-valued function called the amplitude. The $a[n]$ coefficients of the cosine terms are related to the impulse response by

$$a[n] = \begin{cases} h(M) & \text{for } n = 0 \\ 2h(n-M) & \text{for } 0 < n \le M \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

This can be written in matrix form as:

$$\mathbf{A} = \mathbf{C}\,\mathbf{a} \tag{2.6}$$

with $\mathbf{A}$ being a vector of samples of $A(\omega)$, $\mathbf{C}$ being a matrix of cosines terms defined in (2.4), and $\mathbf{a}$ being the vector of filter coefficients defined in (2.5).

The Chebyshev approximation filter design problem is to find the $a[n]$ (and from (2.5) the $h[n]$) which minimize the error measure

$$\epsilon = \max_{\omega \in \Omega} \; |A_d(\omega) - A(\omega)| \tag{2.7}$$

where Ω is a compact subset of the closed frequency band $\omega \in [0, \pi]$. It is the union of the bands of frequencies the approximation is over. These bands are the pass and stopbands of our filter and may be isolated points.

The *alternation theorem* from Chebyshev theory states that if $A(\omega)$ is a linear combination of r ($r = M+1$ for M odd) cosine functions (e.g. (2.4)), a necessary and sufficient condition that $A(\omega)$ be the unique optimal Chebyshev approximation to $A_d(\omega)$ over the frequencies $\omega \in \Omega$ is that the error function $E(\omega) = A(\omega) - A_d(\omega)$ have *at least* $r+1$ extremal frequencies in Ω. These extremal frequencies are points such that

$$E(\omega_k) = -E(\omega_{k+1}) \qquad \text{for } k = 1, 2, \cdots, r \tag{2.8}$$

where

$$\omega_1 < \omega_2 < \cdots < \omega_r < \omega_{r+1} \tag{2.9}$$

and

$$|E(\omega_k)| = \delta = \max_{\omega \in \Omega} |E(\omega)| \qquad \text{for } 1 \le k \le r+1. \tag{2.10}$$

The alternation theorem states the optimal Chebyshev approximation necessarily has an equiripple error, has enough ripples, and is unique.

The *Remez exchange algorithm* is a method which constructs an equiripple error approximation which satisfies the alternation theorem conditions for optimality. It does this by exchanging old approximations to the extremal frequencies with better ones. This method has two distinct steps. The first calculates the optimal Chebyshev approximation over $r+1$ distinct frequency points by solving

$$A_d(\omega_k) = \sum_{n=0}^{r-1} a[n]\, \cos(\omega_k n) + (-1)^k \delta \qquad \text{for } k = 1, 2, \cdots, r+1. \tag{2.11}$$

for the r values of $a[n]$ and for δ. The second step finds the extremal frequencies of $A(\omega)$ over a dense grid of frequencies covering Ω. This is done by locating the local maxima and minima of the error over Ω. The algorithm states that if one starts with an initial guess of $r+1$ extremal frequencies, calculates the $a[n]$ over those frequencies using (2.11), finds a new set of extremal frequencies from the $A(\omega)$ over Ω using (2.4), and iterates these calculations, the solutions converge to the optimal approximation.

Hints

This project has three groups of exercises. Exercises 2.1 and 2.2 will manually step through the Remez exchange algorithm to observe the details of each step. Exercise 2.1 and 2.3 through 2.5 will develop a sequence of functions which implement the complete Remez exchange algorithm, pausing between each iteration to plot the frequency response and observe how the algorithm converges. Exercises 2.6 through 2.11 are generalizations.

The Remez algorithm has two basic steps which are iterated. Exercise 2.1 develops the first step as a function `cheby` where (2.11) is solved to give an optimal Chebyshev approximation over the $M+2$ frequencies in the vector `f`. Exercise 2.3 develops a function `update` which finds a new set of extremal frequencies by searching for the local maxima of the error over a dense grid of frequencies in the pass and stopbands.

In a practical implementation, the number of grid points would be chosen approximately 10 times the length of the filter. In this project, to simplify calculations, frequency will be normalized to π, rather than given in radians per second; the number of grid points will be set at 1000 so that a frequency value of $f_p = \omega_p/\pi = 0.25$ will correspond to an address in the frequency vector of 250. Remember that MATLAB uses 1 as the address of the first element of a vector and we often want zero.

The results of this project will be checked by using the built-in MATLAB `remez` function.

Exercise 2.1: The Basic Chebyshev Alternation Equations

Create a MATLAB M-file function that computes the best Chebyshev approximation over $r+1$ frequencies by completing the following code. Put the proper statements in place of the `?????????`.

```
function [a,d] = cheby0(f,Ad)
% [a,d] = cheby0(f,Ad)  calculates the a(n) for a
%  Chebyshev approx to Ad over frequencies in f.
%  For an odd length L linear phase filter with
%  impulse response h(n) and delay M = (L-1)/2, the
%  M+1  values of cosine coefficients a(n) are
%  calculated from the  M+2  samples of the desired
%  amplitude in the vector  Ad  with the samples
%  being at frequencies in the vector  f.  These
%  extremal frequencies are in bands between 0 and
%  0.5 (NORMALIZED FREQUENCY).  The max error, delta, is d.
%
M = length(f) - 2;          %Filter delay
C = cos(2*pi*f'*[0:M]);     %Cosine matrix
s = ???????????;            %Alternating signs
C = [C,s'];
a = C\Ad';                  %Solve for a(n) and delta
d = ??????;                 %Delta
a = ?????????;              %a(n)
end;
```

Test this function by applying it to

```
f = [0 .1 .2 .25 .3 .35 .4 .5];
Ad = [1 1 1 0 0 0 0 0];}
```

Plot `Ad` vs. `f` with: `plot(f,Ad); hold on; plot(f,Ad,'o'); hold off`.
Plot the amplitude over a dense grid with

```
A = real(fft(a,1000)); A = A(1:501);
plot(A); hold on; plot(f,A(f),'o'); hold off.
```

Explain how this a[n] is optimal over `f` but not over the total pass and stopband.

Exercise 2.2: Design an Odd Length FIR Filter

Design a length-11 linear phase FIR filter using the Remez exchange algorithm. Use an ideal lowpass $A_d(\omega)$ with a passband edge of $f_p = \omega_p/\pi = 0.2$ and a stopband edge of $f_s = \omega_s/\pi = 0.25$.

(a) Form a vector `f` of $M + 2 = 7$ initial guesses for the extremal frequencies. Form a vector `Ad` of 7 ones and zeros at samples of the ideal amplitude response at the frequencies in `f`. Plot `Ad` vs `f`.

(b) Solve the Chebyshev approximation over the 7 frequencies with the function created in exercise 2.1. Plot the total amplitude response of the resulting filter using `plot(A)`. Plot the samples of `A` at the initial guesses of extremal frequencies.

(c) Visually locate the 7 new extremal frequencies where $|A(\omega) - A_d(\omega)|$ has local maxima over the pass and stopbands and exchange these new estimates of the extremal frequencies for the old ones in f. Always include the edges of the pass and stopbands: `fs = 0.2` and `fp = 0.25`. Include $\omega = 0$ and/or $\omega = \pi$, whichever has the larger error. Solve for a new set of $a[n]$ over these new estimates of extremal frequencies using the `Cheby` function from exercise 2.1. Plot the amplitude response.

(d) Repeat this update of `f` and recalculation of $a[n]$ until there is little change.

How many iteration steps were needed for the algorithm to converge? How close are the results to those obtained from the built-in Remez command?

Exercise 2.3: The Remez Exchange

In this and following exercises of this project, a full Remez exchange algorithm will be developed and investigated. Create a MATLAB function that will automatically find the new extremal frequencies from the old extremal frequencies by completing the following M-file code.

```
function [f2,Ad] = update(f1,fp,fs,a)
% [f2,Ad] = update(f1,fp,fs,a)  Finds the
%  (L-1)/2 + 2 new extremal frequencies f2 and
%  samples of Ad(f) consistent with f2 from the old
%  extremal freqs f1 by searching for the extremal
%  points over the pass and stopbands of A(f)
%  calculated from the (L-1)/2+1 = D values of a(n).
%  For odd length-L and even symmetry h(n).
%
D = length(a);                      %Degrees of freedom
A = real(fft(a,1000));
A = A(1:501);                       %Amplitude response
nx = [ ]; Ad = [ ];
np = fp*1000;
E = abs(A(1:np+1) - ones(1:np+1)); %passband error
for n = 2:np                        %search passband for max
   if (E(n-1)<E(n))&(E(n)>E(n+1))  %find local maximum in PB
      nx = [nx, n]; Ad = [Ad, 1];  %save location of max
   end
end
ns = 1000*fs;
nx = [nx,np,ns]; Ad = [Ad,1,0];    %add transition band edges
for n = ns:500                      %search stopband
   if (abs(A(n-1))<abs(A(n)))&(abs(A(n))>abs(A(n+1)))
      nx = [nx, n]; Ad = [Ad, 0];  %save location of max in SB
   end
end
?????????????                       % add extremal frequencies
?????????????                       % at f = 0 and/or .5
f2 = nx/1000;
end;
```

Test this function by applying it to data similar to that used in exercise 2.2 to see if it does what you want in a variety of examples.

Exercise 2.4: Initial Extremal Frequencies

Write a MATLAB function called `init.m` that will take the specifications of filter length L, passband edge $f_p = \omega_p/\pi$, and stopband edge $f_s = \omega_s/\pi$, and generate the initial guesses of extremal frequencies as a vector `f` and the associated vector of samples `Ad`, i.e., $A_d(\omega)$. Test it on several example specifications. Plot the results.

Exercise 2.5: The Remez Filter Design Program

Write a MATLAB program that executes the `init.m` function to calculate the initial extremal frequencies and `Ad` followed by a loop which executes the Chebyshev approximation in `cheby0` and the updating of the extremal frequencies in `update`. Put a plot of the amplitude response in the loop to see that each step of the algorithm does. If you are running this on a very fast computer, you may want to put a pause after the plot. Design a length-23 linear phase FIR lowpass filter with passband edge $f_p = \omega_p/\pi = 0.2$ and stopband edge $f_s = \omega_s/\pi = 0.25$ using this design program. Comment on how the extremal frequencies and δ change with each iteration. How do you decide when to stop the iterations?

Exercise 2.6: How Robust is the Remez Exchange Algorithm?

Rewrite the `init.m` function to start with all of the extremal point guesses in the stopband (excluding $\omega = 0$ and ω_p). This will illustrate how robust the algorithm is and how it moves the excess extremal points from the stop to the passband. Run this for several different pass and stopband edges and discuss the results.

Exercise 2.7: Weighting Functions

Generalize the `cheby.m` function to include a weighting function. Design the filter in exercise 2.5 but with a weight of 10 on the stopband error.

Exercise 2.8: Design Even Order Filters

Write a new set of functions to design even length filters. Recall that an even length linear phase FIR filter must have a zero at $\omega = \pi$, therefore, cannot have an extremal point there. Design a length-22 lowpass filter with the band edges given in exercise 2.5. Comment on the convergence performance of the algorithm.

Exercise 2.9: Design Type III and IV Filters

Write a set of functions that will design Type III and IV filters which have an odd-symmetric $h[n]$ and an expansion in terms of sines rather than the cosines in (2.4) and (2.11). Design a length-11 and a length-10 differentiator with these programs. Discuss the results. Note the problems with the odd-length differentiator.

Exercise 2.10: Design Bandpass Filters

Write new `init.m` and `update.m` functions that will design a bandpass filter. For the simple two-band lowpass filter, there are usually $M + 2$ extremal frequencies but possibly $M + 3$ for the "extra ripple" case. For more than two bands as in the bandpass filter, there may be still more extremal frequencies.

Exercise 2.11: How Good are the Programs?

What are the maximum lengths these programs can design? How does the maximum length depend on ω_p and ω_s? How does the execution time compare with the built-in Remez function?

Project 3: FIR filter design using linear programming

Linear programming has proven to be a powerful and effective optimization tool in a wide variety of applications. The problem was first posed in terms of economic models having linear equations with linear inequality constraints. In this project we investigate how this tool can be used to design optimal Chebyshev linear phase FIR filters.

Project Description

The Chebyshev error is defined as

$$\varepsilon = \max_{\omega\in\Omega} |A(\omega) - A_d(\omega)| \tag{3.1}$$

where Ω is the union of the bands of frequencies that the approximation is over. The approximation problem in filter design is to choose the filter coefficients to minimize ε.

It is possible to pose this in a form that linear programming can be used to solve it [12, 10]. The error definition in (3.1) can be written as an inequality by

$$A_d(\omega) - \delta \leq A(\omega) \leq A_d(\omega <) + \delta \tag{3.2}$$

where the scalar δ is minimized.

The inequalities in (3.2) can be written as

$$A \leq A_d + \delta \tag{3.3}$$

$$-A \leq -A_d + \delta \tag{3.4}$$

or

$$A - \delta \leq A_d \tag{3.5}$$

$$-A - \delta \leq -A_d \tag{3.6}$$

which can be combined into one matrix inequality using (2.6) by

$$\begin{bmatrix} C & -1 \\ -C & -1 \end{bmatrix} \begin{bmatrix} a \\ \delta \end{bmatrix} \leq \begin{bmatrix} A_d \\ -A_d \end{bmatrix}. \tag{3.7}$$

If δ is minimized, the optimal Chebyshev approximation is achieved. This is done by minimizing

$$\varepsilon = \begin{bmatrix} 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} a \\ \delta \end{bmatrix} \tag{3.8}$$

which, together with the inequality of (3.7), is in the form of the dual problem in linear programming [9, 13].

Hints

This can be solved using the `lp()` command from the MATLAB Optimization Toolbox [14] which is implemented in an M-file using a form of quadratic programming algorithm. Unfortunately, it is not well suited to our filter design problem for lengths longer than approximately 11.

Exercise 3.1: Formulate the FIR filter design problem as a linear program

A MATLAB program that applies its linear programming function `lp.m` to (3.7) and (3.8) for linear phase FIR filter design is given below. Complete the program by writing the proper code for the lines having `??????????` in them. Test the program by designing a filter having the same specifications as given in exercise 2.1 in the project on the Remez exchange algorithm. Compare the this design with the one done using Remez or by using the Parks-McClellan algorithm in the MATLAB command `remez`.

```
% lpdesign.m  Design an FIR filter from L, f1, f2, and LF using LP.
%  L is filter length, f1 and f2 are pass and stopband edges, LF is
%  the number of freq samples.  L is odd.  Uses lp.m
%         csb 5/22/91
L1 = fix(LF*f1/(.5-f2+f1));  L2 = LF - L1;     %No. freq samples in PB, SB
Ad = [ones(L1,1); zeros(L2,1)];                %Samples of ideal response
f  = [[0:L1-1]*f1/(L1-1), ([0:L2-1]*(.5-f2)/(L2-1) + f2)]';  %Freq samples
M  = (L-1)/2;
C  = cos(2*pi*(f*[0:M]));                      %Freq response matrix
CC = ??????????????????????                    %LP matrix
AD = [Ad; -Ad];
c  = [zeros(M+1,1);1];                         %Cost function
x  = lp(c,CC,AD);                              %Call the LP
d  = x(M+2);                                   %delta or deviation
a  = ????????????                              %Half impulse resp.
h  = ????????????                              %Impulse response
```

Exercise 3.2: Design a bandpass filter

Design a bandpass FIR filter with the specifications given in exercise 2.3 in the Remez project above but using linear programming. Compare the solutions.

Exercise 3.3: Analysis of the speed of linear programming

For the specification of a lowpass filter used in exercise 3.1, design filters of odd length from 5 up to the point the algorithm has convergence problems or takes too much time. Time these designs with the `clock` and `etime` commands. From a plot of time vs. length, determine a formula that predicts the required time. This will differ depending on whether the Simplex, the Karmarkar, or some other algorithm is used. Try this on as many different algorithms as you have access to and discuss the results.

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{Design of IIR Filters}}$

Design of IIR Filters

Overview

The infinite-duration impulse response (IIR) discrete-time filter is the most general linear signal processing structure possible. It calculates each output point via a recursive difference equation:

$$y[n] = -\sum_{k=1}^{N} a_k \, y[n-k] + \sum_{m=0}^{M} b_m \, x[n-m] \tag{0.1}$$

which assumes a_k and b_m are not functions of the discrete-time variable n. Notice the FIR filter is a special case when $N = 0$ and only past inputs are used. The design problem is to take a desired performance in the form of specifications, usually in the frequency domain, and to find the set of filter coefficients a_k and b_m that best approximates or satisfies them. The IIR discrete-time filter is analogous to the RLC circuit continuous-time filter or the active RC filter. Indeed, one of the standard methods of IIR filter design starts with the design of a continuous-time prototype which is converted to an equivalent discrete-time filter.

The advantage of the IIR filter over the FIR filter is greater efficiency. One can usually satisfy a set of specifications with a significantly lower order IIR than FIR filter. The disadvantages of the IIR filter are problems with stability and with quantization effects, the impossibility of exactly linear phase frequency response, and the more complicated design algorithms. Most of the good and bad characteristics come from the feedback inherent in the IIR filter. It is the feedback of past output values that can cause the filter to be unstable or amplify the effects of numerical quantization, and it is feedback that gives the infinite duration response.

Several design methods for IIR filters use a continuous-time (analog) filter as a prototype which is converted to a discrete-time (digital) IIR filter. Some of the exercises concern characteristics of analog filters, but goal is digital filter design.

The purpose of this section is to become familiar with the characteristics of IIR filters and some of the standard design methods. You should not only evaluate the performance of these filters, you should compare them with the alternative, the FIR filter.

Background Reading

All DSP text books have some coverage of IIR filters. Particularly good discussions can be found in [1, 3, 4] and the details of analog filter design useful as prototypes for discrete-time filters can be found in [5].

Project 1: Characteristics of IIR filters

This project will use MATLAB to quickly design several types of IIR discrete-time filters and analyze their characteristics. There are three descriptions of a filter that must be understood and related. First there is the impulse response which is the most basic time domain input-output description of a linear system. Second, there is the magnitude and phase frequency response which is the most basic frequency domain input-output description of a linear, time invariant system. Third, there is the pole-zero map in the complex plane which is the most basic transfer function description. This section will not consider state space descriptions since they are more related to implementation structures than approximation.

Project Description

There are four classical IIR filters and their analog counterpoints: 1) Butterworth, 2) Chebyshev, 3) Chebyshev II, and 4) elliptic function. They represent four different combinations of two error approximation measures. One error measure uses the Taylor's series. This method equates as many derivatives of the desired to those of the actual response as possible. The other approximation method minimizes the maximum difference between the desired and actual response over a band of frequencies.

1. The analog *Butterworth* filter is based on a Taylor's series approximation in the frequency domain with expansions at $\omega = 0$ and $\omega = \infty$. This filter is also call a maximally flat approximation since it is optimal in the sense that as many derivatives as possible equal zero at $\omega = 0$ and $\omega = \infty$. This approximation is local in that all the conditions are applied at only two points and it is the smoothness of the response that influences its behavior at all other frequencies. The formula for the magnitude squared of the normalized frequency response of an N^{th} order analog Butterworth lowpass filter is given by:
$$|H(\Omega)|^2 = \frac{1}{1+\Omega^{2N}} \tag{1.1}$$
This response is normalized so that the magnitude squared is always $\frac{1}{2}$ at $\Omega = 1$ for any N. Replacing Ω by Ω/Ω_0 would allow an arbitrary band edge at Ω_0.

2. The analog *Chebyshev* filter has a minimum maximum error over the passband and a Taylor's approximation at $\Omega = \infty$. The maximum error over a band is called the Chebyshev error. This terminology can be confusing since the Chebyshev filter only minimizes the Chebyshev error over one band. One of the interesting and easily observed characteristics of a Chebyshev approximation is the error oscillates with equal size ripples. A Chebyshev approximation is often called an equal ripple approximation, but that can be misleading since the error must not only be equal ripple, there must be enough ripples.

3. The analog *Chebyshev II* filter (sometimes called the inverse Chebyshev filter) is a Taylor's series approximation at $\Omega = 0$ and has minimum Chebyshev error in the stopband. This is often a more practical combination of characteristics than the usual Chebyshev filter.

4. The analog *elliptic* function filter (sometimes called the *Cauer* filter) uses a Chebyshev approximation in both the passband and stopband. The Butterworth, Chebyshev, and Chebyshev II filters have formulas that can calculate the location of their poles and zeros using only trigonometric functions, however, the elliptic function filter requires evaluation of the Jacobian elliptic functions and the complete elliptic integrals and, therefore, the theory is considerably more complicated. Fortunately, MATLAB does all the work for you and all are equally easy to design.

These four optimal analog filters can be transformed into optimal digital filters with the bilinear transform which is investigated in the next project. The IIR filter design programs in MATLAB take care of the analog filter design and the bilinear transformation into the digital form automatically.

Hints

One can calculate samples of the frequency response of an IIR filter via `freqz` which computes:

$$H(e^{j\omega_k}) = H(e^{j2\pi k/L}) = \frac{DFT\{[\,b_0\; b_1\; \ldots b_M\,]\}}{DFT\{[\,a_0\; a_1\; \ldots a_N\,]\}} \tag{1.2}$$

The location of the poles can be plotted on the complex z plane by using `plot(roots(a),'x')` where `a` is a vector of the denominator coefficients; the zeros with `plot(roots(b),'o')`.[24] The impulse response can be generated via `filter` and plotted with `comb`.

The `buttap`, `cheby1ap`, `cheby2ap`, and `ellipap` commands design analog prototypes which must be transformed using the `bilinear` command into a digital IIR form. The `butter`, `cheby1`, `cheby2` and `elliptic` commands design digital (discrete-time) IIR filters by automatically prewarping the specifications, calling the appropriate prototype design program and then applying the bilinear transformation. You should read the MATLAB manual about these commands as well as using the `help` command. You should also examine the M-file programs that implement the designs.

All filter design functions in MATLAB use a frequency scaling that is somewhat nonstandard. When entering the cutoff frequencies ω_p and ω_s (units of radians), the values must be divided by π. Thus a cutoff frequency specified as $\omega_p = 0.22\pi$ would be entered as `0.22` in MATLAB. This scaling is not the usual normalized frequency where the sampling frequency is one. Therefore, if the specifications call for a cutoff frequency of 1000 Hz when the sampling frequency is 5000 Hz, then you must enter `0.4` in MATLAB, because the cutoff frequency in radians is $\omega_c = 2\pi(1000/5000) = 2\pi(0.2) = 0.4\pi$. In this example, the normalized frequency would be $1000/5000 = 0.2$, so the MATLAB frequency is twice the normalized frequency.

Exercise 1.1: How is the Analog Butterworth filter optimal?

From (1.1), show how many derivatives of the magnitude squared are zero at $\Omega = 0$ for a 5^{th} order analog Butterworth filter. How many are zero at $\Omega = \infty$?

Exercise 1.2: Analyze a 5^{th} order digital Butterworth lowpass IIR filter

Use the MATLAB command `butter` to design a 5^{th} order lowpass IIR filter with a sampling frequency of 2 Hz and a band edge of 0.6 Hz. Plot the magnitude and phase frequency responses. Plot the pole and zero location diagram. Plot the significant part of the impulse response. Discuss how each pair of these three plots might be inferred from the third.[25]

Exercise 1.3: Analyze 5^{th} order Chebyshev and elliptic filters

Use the MATLAB command `cheby1` to design a 5^{th} order lowpass IIR filter with a sampling frequency of 2 Hz and a band edge of 0.6 Hz and a passband ripple of 0.5 db. Plot the

[24]See CASPER special `zplane.m` which does all this and draws the unit circle.

[25]See also the chapter on *Systems and Structures*.

magnitude and phase frequency responses. Plot the pole and zero location diagram. Plot the significant part of the impulse response. Discuss how each pair of these three plots might be inferred from the third. Repeat for an 5^{th} elliptic filter with a passband ripple of 0.5 db, band edge of 0.6 Hz, and stopband ripple that is 30 db less than the passband response.

Exercise 1.4: Compare the order of the four designs

The filtering specifications for a particular job had a sampling rate of 2 Hz, passband ripple of 0.1, passband edge of 0.28 Hz, stopband edge of 0.32 Hz, and stopband ripple below 30 db. What order Butterworth, Chebyshev, Chebyshev II, and elliptic filters will meet these specifications? Use the `buttord cheb1ord cheb2ord ellipord` commands. Discuss these results and experiment with other specifications. Why does the elliptic filter always have the lowest order?

Exercise 1.5: Compare IIR and FIR filters

After working exercise 4, design a Chebyshev FIR filter to meet the same specifications. What length filter designed by the Parks-McClellan method will meet them? What length filter designed using the Kaiser window will meet them? Discuss these comparisons.

Project 2: Use the bilinear transformation

The four classical IIR filter designs are usually developed and implemented by first designing an analog prototype filter then converting it to a discrete-time filter using the bilinear transformation. This is done for two reasons. First, analog filter methods use the Laplace transform where frequency is simply the imaginary part of the complex Laplace transform variable. The approximations are much more easily developed and described in the rectangular coordinates of the Laplace transform than the polar coordinates of the z-transform. Second, one method can be used to design both discrete and continuous time filters. This project investigates this method.

Project Description

The bilinear transform is a one-to-one map of the entire analog frequency domain $-\infty \leq \Omega \leq \infty$ onto the discrete time frequency interval $-\pi \leq \omega \leq \pi$. The mapping is given by

$$s = \frac{2}{T}\frac{z-1}{z+1} \tag{2.1}$$

Substituting this equation into a Laplace transfer function gives a discrete-time z-transfer function with the desired map of the frequency response. The analog and discrete-time frequencies are related by

$$\Omega = \frac{2}{T}\tan(\frac{\omega}{2}) \tag{2.2}$$

where T is the sampling period in seconds. From this formula, it is easily seen that each frequency in the half infinite analog domain is mapped to a frequency in the zero to π discrete-time domain. This map is non-linear in the sense that near $\omega = 0$, the analog and discrete-time frequencies are very close to each other. However, as the discrete-time frequency nears π, the analog frequency goes to infinity. In other words, there is a nonlinear warping of the analog frequency range to make it fit into the finite discrete-time range. There is an ever increasing compression of the analog frequency as the digital frequency nears π. The effects of this frequency warping of the bilinear transform must be taken into account when designing IIR filters by this method. Thus the bilinear method is suited for mapping frequency responses that are piecewise constant over frequency bands, such a lowpass and bandpass filters. It will not work well in designing the group delay or matching a magnitude characteristic such as a differentiator.

The design of IIR filters using the bilinear transformation has the following steps:

1. Modify the discrete-time filter design specifications so that after the bilinear transform is used they will be proper. This is call "prewarping".

2. Design the analog prototype filter using the prewarped specifications.

3. Convert the analog transfer function into a discrete-time z-transform transfer function using the bilinear transform.

This process is automatically done in the MATLAB design programs, however, to understand the procedure, this project will explicitly go through the individual steps.

Hints

Remember that frequency specifications are often given in f with units of Hz or cycles per second but the theory is often developed in ω with units of radians per second. The relation is $\omega = 2\pi f$. Also, recall the discrete-time filter consists of the coefficients a_k and b_m. The sampling frequency and other frequency normalizations and transformations affect only the band edge.

The MATLAB functions for analog prototypes will produce filters whose cutoff frequency is normalized to $\Omega_c = 1$, see `buttap`, `cheb1ap`, `cheb2ap`, and `ellipap`.

Exercise 2.1: Bilinear Transformation

Plot the relationship between the analog frequency Ω and the digital frequency ω specified by the bilinear transformation. Use several values for $(2/T)$ and plot the curves to together. If an analog prototype has a cutoff frequency at $\Omega_c = 1$, how will the digital cutoff frequency change as T increases?

Exercise 2.2: Prewarp the specifications

A 4^{th} lowpass discrete-time Butterworth filter with a sampling frequency of 40 kHz is to be designed for a band edge of 8 kHz. What is the prewarped band edge?

Exercise 2.3: Design the analog prototype Butterworth filter

Find the Laplace transform continuous-time transfer function for the 4^{th} order Butterworth filter using the prewarped band edge. Do this by hand or use the MATLAB `buttap` command.

Exercise 2.4: Apply the bilinear transformation

Find the z-transform discrete-time transfer function from the continuous-time transfer function using the bilinear transform. Do this by hand or use the MATLAB `bilinear` command. Compare this with the design done directly by MATLAB.

Project 3: Design of highpass, band pass and band reject IIR Filters

It is possible to design a highpass, bandpass, or band reject IIR filter by a frequency transformation (change of variables) on a lowpass filter. This is possible for both analog and discrete-time IIR filters and is done by a different process than used for an FIR filter.

Project Description

If one has a lowpass analog filter transfer function of the complex variable s, simply replacing s by $1/s$ will convert the lowpass filter into a highpass filter. The replacement essentially replaces the "point" at infinite in the complex s plane by the origin and the origin by the point at infinite. It turns the s plane "inside out". This has the effect of converting a lowpass filter into a highpass filter.

For a discrete time z-transform transfer function, the point of plus one in the complex z plane and the point minus one are interchanged to achieve the same conversion.

A lowpass analog filter can be converted into a bandpass filter by replacing s by

$$s \rightarrow \frac{s^2 + \omega_0}{s} \tag{3.1}$$

where $\omega_0 = \sqrt{\omega_1 \omega_2}$ is the geometric mean of the two band edges. This mapping doubles the order of the filter and is a one-to-two mapping of the frequency domain. The reciprocal of this transformation will produce a band reject filter from a lowpass filter.

NOTE: For the discrete-time case, the bandpass frequency transformations can be accomplished via all-pass functions which warp the ω axis.

MATLAB does these frequency conversions automatically. This project is to analyze the characteristics of these filters.

Exercise 3.1: A highpass Butterworth filter

Use the MATLAB command `butter` to design a 5th order highpass Butterworth IIR filter with a sampling frequency of 2 Hz and a band edge of 0.7 Hz. Plot the magnitude and phase frequency responses. Plot the pole and zero location diagram. Plot the significant part of the impulse response. Compare with the lowpass design done above. Discuss how each pair of these three plots might be inferred from the third. Compare the impulse response to that of a Butterworth LPF.

Exercise 3.2: Design highpass filters

Use the MATLAB command `cheby1` to design a 5th order highpass Chebyshev IIR filter with a sampling frequency of 2 Hz and a band edge of 0.7 Hz and a passband ripple of 1 db. Plot the magnitude and phase frequency responses. Plot the pole and zero location diagram. Plot the significant part of the impulse response. Discuss how each pair of these three plots might be inferred from the third. Repeat for an elliptic filter with a stopband ripple of -40 db, passband edge of 0.25 Hz, and stopband edge of 0.3 Hz.

Exercise 3.3: Bandpass Transformation

Plot the frequency mapping specified by (3.1). Explain how it transforms a lowpass prototype into a bandpass filter. Where are the bandedges of the BPF located, if the LPF has a cutoff at $\Omega_c = 1$. Also, explain what the result would be if the transformation were applied to a high-pass filter.

Exercise 3.4: Design bandpass filters

Use the MATLAB command `cheby1` to design a 5^{th} order bandpass Chebyshev IIR filter with a sampling frequency of 2 Hz, a lower band edge of 0.5 Hz, an upper band edge of 0.8 Hz, and a passband ripple of 1 db. Plot the magnitude and phase frequency responses. Plot the pole and zero location diagram. Plot the significant part of the impulse response. Discuss how each pair of these three plots might be inferred from the third.

Exercise 3.5: Details

Design the bandpass filter in the preceding exercise by explicitly going through each step of the transformation using the `lp2bp bilinear cheb1` commands.

Project 4: IIR filter design in the time domain by Prony's method

Most IIR filters are designed from specifications given in the frequency domain, but there are situations where a time domain design is desired. A particularly powerful method uses Prony's method (or Padé's method) to design an IIR filter by approximating or interpolating the first portion of a desired impulse response. Prony's method is a very useful tool for many applications and should be studied and understood.

Project Description

The z-transform transfer function of an IIR filter defined by (0.1) is given by

$$H(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_M^{-M}}{a_0 + a_1^{-1} + \cdots + a_N^{-N}} = h[0] + h[1]\,z^{-1} + h[2]\,z^{-2} + \ldots \tag{4.1}$$

where the impulse response, $h[n]$, has an infinite number of terms.

In the usual formulation of the approximation problem for filter design, one defines a solution error by

$$\epsilon_s(\omega) = \left[H(e^{j\omega}) - \frac{B(e^{j\omega})}{A(e^{j\omega})} \right] \tag{4.2}$$

and chooses the filter coefficients a_k and b_m to minimize some norm of ϵ_s. This is a linear problem for a polynomial as we saw in the FIR filter design problem, but it is non-linear here where the coefficients enter into both the numerator and denominator of the rational function (4.1). This problem can be reformulated to become linear by defining an *equation error*:

$$\epsilon_e(\omega) = (A(e^{j\omega})\,H(e^{j\omega}) - B(e^{j\omega})) \tag{4.3}$$

which can minimized by solving linear equations. The time domain version of this is called Prony's method [3, 15] and the frequency domain version is implemented by `invfreqz` in MATLAB and discussed in [3]. Clearly in the time domain, since there are an infinite number of $h[n]$ terms, a finite number of them must be chosen. If the number of impulse samples used is the same as the unknown coefficients, $L = M + N + 1$, the problem can be solved exactly so that the impulse response achieved is exactly the same as what was desired for $0 \leq n \leq L$ and with nothing said about what happens for $n > L$. If $L > M + N + 1$, coefficients are found so that the sum of the squared equation error of (4.3) is minimized.

If coefficients can be found that makes either ϵ_s or ϵ_e equal zero, the other is zero also. If the minimum error is not zero but is small, usually the coefficients that minimize $||\epsilon_e||$ are close to those that minimize $||\epsilon_s||$. If the minimum is not small, the solutions can be quite different. However, remember that the solution to both problems is optimal but according to different error measures. We usually want to minimize (4.2) but it is much easier to minimize (4.3) and that can be done by Prony's method. Notice that there is no control of stability.

Hints

Study the MATLAB M-file¡ `prony` using `help` and `type`. This is discussed in [3, 15].

Exercise 4.1: Exact interpolation of desired response

It is desired to design an IIR filter that will exactly match the following 10 impulse response values:

```
h(n) = [ 1 4 6 2 5 0 1 0 0.1 0 0 ]
```

Find the IIR filter coefficients a_k and b_m using Prony's method for:

(a) $M = 5$ and $N = 5$.

(b) $M = 3$ and $N = 7$.

(c) $M = 7$ and $N = 3$.

Using `filter`, calculate the first 20 values of the impulse response of these three filters designed by Prony's method. Do the first 10 match? Do the second 10 match? Try other combinations of M and N and discuss the results. How do the frequency responses compare? Analyze the pole and zero locations and discuss. Are the filters designed stable?

Repeat this exercise with the impulse response:

```
h(n) = [ 1 4 6 2 5 0 1 1 0 0.1 0 ]
```

compare the results using the two desired impulse responses.

Exercise 4.2: Minimum equation error approximation

It is desired to design an IIR filter that will approximately match the following 20 impulse response values:

```
h(n) = [ 1 4 6 2 5 0 1 1 0 0.1 0 0 0 0 0 0 0 0 0 0 ]
```

Find the IIR filter coefficients a_k and b_m using Prony's method for:

(a) $M = 5$ and $N = 5$.

(b) $M = 3$ and $N = 7$.

(c) $M = 7$ and $N = 3$.

Using `filter`, calculate the first 20 values of the impulse response of these three filters designed by Prony's method and compare them with the desired values of $h[n]$. Repeat with the last ten values equal to one. Analyze and compare with the results of the preceding exercise.

Exercise 4.3: Parameter identification by Prony's method

Use the MATLAB command `butter` to design a 5^{th} order lowpass IIR filter with a sampling frequency of 2 Hz and a band edge of 0.6 Hz as was done in an earlier exercise. Calculate the first 11 terms of the impulse response using `filter`. Apply Prony's method for $M = 5$ and $N = 5$ to this impulse response and compare the designed filter coefficients with those that generated the impulse response. Try taking 15 terms of the impulse response and using $M = 7$ and $N = 7$. What are the results? Discuss your observations of this case.

Exercise 4.4: Effects of noise or error

Prony's method is somewhat sensitive to error or noise. Repeat the preceding exercise but add noise to the impulse response of the Butterworth filter before applying Prony's method. Do this by adding `K*rand(10,1)` to $h[n]$ for several values of K ranging from values that make the noise very small compared to the impulse response up to values that make the noise comparable to the impulse response. Repeat this for the case where 15 terms are used. Repeat this for still larger numbers of terms. What conclusion can you draw?

Project 5: Special topics

This project is loosely structured to investigate two less well known methods of IIR filter design. The first is a frequency domain version of Prony's method where the squared equation error is minimized. This is a very effective method but requires a complex desired frequency response (both magnitude and phase). The second is a method borrowed from the spectral estimation world and applied to filter design. The Yule-Walker method can be applied to a desired magnitude without requiring the phase to be specified, much as was done with the Butterworth and other classical methods. It gives an approximate least squared error approximation.

Project Description

The least equation error method is similar to Prony's method but has the advantage that, in the frequency domain, one can sample all of the region $0 \leq \omega \leq \pi$ and not have to truncate terms as was necessary in the time domain. It has the disadvantage of requiring complex values of the frequency domain samples where Prony's method used real values [3]. This method is implemented by `invfreqz` for discrete-time filters and `invfreqs` for continuous-time filters. The problems with using this method for many filter design cases is the requirement of providing the phase.

In this project we will use the complex frequency response of an already designed filter upon which to first apply our method. This will give us a realistic phase upon which to work. Notice from the name of the MATLAB command, the method is in a sense an inverse to the `freqz` command.

The advantages of the Yule–Walker method is it only requires the magnitude of the desired response. The disadvantage is that it only approximately finds the optimal solution. It is implemented in MATLAB by `yulewalk`.

Exercise 5.1: The inverse frequency response

Use the MATLAB command `butter` to design a 5^{th} order lowpass IIR filter with a sampling frequency of 2 Hz and a band edge of 0.3 Hz as was done in earlier exercises. Calculate the complex frequency response using `freqz` at 100 equally spaced points. Apply the design method using `invfreqz` to this sampled frequency response and compare the designed filter coefficients with the Butterworth filter whose frequency response we used. Try a different number of frequency points and comment on the results.

Exercise 5.2: The least equation error filter design method

As was done in the previous exercise, use the MATLAB command `butter` to design a 5^{th} order lowpass IIR filter with a sampling frequency of 2 Hz and a band edge of 0.6 Hz. Calculate magnitude and phase frequency response of the filter from the output of `freqz`. Using that magnitude and a new desired phase curve that is linear in the passband and constant in the stopband (¡close to the designed phase), create a new complex frequency response and apply `invfreqz` to it. Look at the magnitude of the frequency response of the resulting filter and comment on it. Try several other piecewise linear phase curves in an effort to get a ”good” approximation to the magnitude and to the linear phase. Discuss the results.

Exercise 5.3: A second least equation error design

This exercise requires working the previous one. Take the best linear phase obtained in exercise 2 and couple it with a desired magnitude of one in the passband and zero in the stop band. If you have numerical problems here, add a transition band and/or use a small but non-zero value in the stopband. Experiment with various desired ideal frequency responses in an effort to understand how this new method works. Take your best design and analyze the time-domain, pole-zero location, and frequency response characteristics as was done for the four classical methods. Discuss your observations.

Exercise 5.4: The Yule–Walker method

Design a 5^{th} IIR filter to the same specifications as the exercises above using the Yule–Walker method in the MATLAB command `yulewalk`. It may be necessary to have a small transition band to avoid too rapid a change in magnitude from one to zero. Experiment with this and compare the results with those designed by the least equation error and by the four classical methods. Design a 13^{th} order IIR filter to the same specifications and analyze its characteristics. Notice in particular, the zero locations.

Chapter 9

DFT and FFT Algorithms

January 17, 2007

Overview

The fast Fourier transform (FFT) is the name of a family of algorithms for efficiently calculating the discrete Fourier transform (DFT) of a finite length sequence of real or complex numbers. The various forms of Fourier transforms have proven to be extraordinary important in almost all areas of mathematics, science and engineering. While the Laplace transform, Fourier transform, Fourier series, z-transform, and discrete-time Fourier transform are important in analytical work, it is the DFT that we can actually calculate. Indeed, the DFT and its efficient implementation by the FFT are among the most important tools in digital signal processing.

The basic idea behind the FFT is the elimination of redundant calculations in the direct evaluation of the DFT from its definition. It is the clever organization, grouping, and sequencing of operations that can give the minimum amount of arithmetic. In algorithm theory, this process is sometimes called "divide and conquer" but that name is misleading because simply dividing a problem into multiple small ones does not necessarily reduce the work. The process should be called "organize and share" to point out it is the sharing of common operations that eliminates redundant operations and saves arithmetic. In working the exercises in this chapter, one should look for these ideas and try to understand them in the most general way.

This chapter will consider several approaches to calculating the DFT. The first step in deciding what algorithm to use is to carefully pose a question. The answer might be very different if you want all of the DFT values or if you want only a few. A direct calculation or the Goertzel algorithm might be the best method if only a few spectral values are needed. For a long prime length DFT, the chirp z-transform might be the best approach. In most general cases and for many lengths, the Cooley-Tukey FFT or the prime factor FFT will probably be the best choice. To understand these ideas and to learn how to answer questions with other considerations are the goals of this chapter.

There are three organizations used by most FFT algorithms, one requires that the length of the sequence to be transformed have relatively prime factors. This is the basis of the prime factor algorithm (PFA) and the Winograd Fourier transform algorithm (WFTA). The second can be applied to any factoring of the length into prime or composite factors but requires additional arithmetic. This is used in the Cooley-Tukey fixed-radix FFT and mixed-radix FFT and also in the split-radix FFT. Still another approach converts the DFT into a filter. The chirp z-transform, Rader's method, and Goertzel's algorithm do that. These approaches are all covered in the following projects and exercises.

It should be remembered that while most of the exercises in this chapter address the Fourier transform, the ideas can also apply to many other transforms and signal processing operations.

6 Background Reading

Most general discrete-time signal processing text books will have one or more chapters on the DFT and FFT. They will certainly cover the basic Cooley-Tukey FFT. More specific information can be found in the books by Burrus and Parks [16], Brigham [17], Blahut [18], McClellan and Rader [19], or chapter 4 in Lim and Oppenheim [20]. An excellent overview article on modern FFT techniques has been published by Duhamel and Vetterli [21]. Programs can be found in Burrus and Parks [16] and in the IEEE DSP Program Book [22]. MATLAB itself has an interesting implementation of the FFT in its `fft` command (version 3.5 or later).

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{Direct Calculation of the DFT}}$

Direct Calculation of the DFT

Overview

Calculating the discrete Fourier transform (DFT) of a number sequence from the definition (1.1) of the DFT is the most basic direct method. Since there are N DFT values, each calculated from N input values, there will be on the order of N^2 floating point multiplications and additions required. This section will investigate two approaches to this direct calculation.

Background Reading

Details of direct DFT calculations and Goertzel's algorithm can be found in [3, 8].

Project 1: Calculation of the DFT from the Definition

If only a few values of the DFT of a number sequence are needed, a direct calculation of the DFT may be best. As you better understand the FFT, you will see that the efficiency gained by "sharing" operations is mainly lost when there are only a few final values to share the calculations. Also, if the length of the DFT is prime or has few factors, the FFT does not give any or much advantage although the chirp z-transform covered in a later packet will be considerably faster for large prime lengths.

Project Description

MATLAB uses an interpreted language with very efficient individual commands such as vector products or matrix multiplications but with fairly inefficient execution of `for` loops. Consequently, it can be used to simulate the hardware operation of a vector architecture computer which has very fast vector operations and slower scalar operations because of pipelining. In this project we will measure the execution time and the number of floating point operations (flops) of a direct calculation of the DFT using three different organizations of the algorithm.

The definition of the DFT is given by

$$X[k] = \sum_{n=0}^{N-1} x[n]\, W_N^{nk} \tag{1.1}$$

for

$$W_N = e^{-j2\pi/N} = \cos(2\pi/N) - j\sin(2\pi/N) \tag{1.2}$$

and $k = 0, 1, \ldots, N-1$.

This can be viewed as two nested loops of scalar operations consisting of a complex multiplication and a complex addition (or accumulation). It can also be viewed as a single loop which calculates each $X[k]$ by an inner product of the data vector made up of $x[n]$ and an N-point basis vector made up of W_N^{nk} for $n = 0, 1, \cdots, N-1$ and k fixed. Equation (1.1) can also be thought of as a single matrix multiplication (where the matrix is generated by MATLAB in `dftmtx` or as `fft(eye(N))`).

Hints

The evaluation of the algorithms in this project will be in terms of the execution time and in terms of the number of floating point operations (flops) required. The timing can be done by preceding the operation being timed by the MATLAB statement: `t0 = clock;` and following it by `time = etime(clock,t0);`. The evaluation of the number of flops required is done in a similar manner using `f0 = flops;` before and `fl = flops - f0;` after the operation. There may be inaccuracies with the timings on a time-shared computing system which will have to be dealt with by averaging several runs. Generate a test sequence of random complex numbers using: `x = (rand(1,N) + j*rand(1,N));`. The lengths to be used depend on the speed and memory size of the computer.

Remember that MATLAB starts the addressing of all vectors and arrays at one while the DFT formulas start at zero. Care must be taken in writing an m-file program or function from a mathematical formula to use the proper indexing. Because of the way MATLAB counts flops, there will be some differences in the number of flops for evaluation of `exp()`, `cos()`, and `sin()`.

Exercise 1.1: Two loop program

Write a program (script m-file) or function in MATLAB to evaluate the DFT in (1.1) using two nested `for` loops with the inner loop summing over n and the outer loop indexing over k. Time the program for several lengths using the `clock` and `etime` commands mentioned above. Evaluate the number of flops required for several lengths and compare with what you would expect from the formula. Check the manual and use the `help` command to see how `flops` counts operations. Compare the times and flops of your DFT program with the built-in MATLAB command `fft` for the same lengths. Take into account that the `flops` command will count all arithmetic operations; exponential computation and index arithmetic as well as data arithmetic.

Exercise 1.2: One loop program

Write a DFT program using one loop which steps through each value of k and executes an inner product. Time the program and evaluate the number of flops as was done for the two-loop program and compare the results for the same set of lengths. Explain the results obtained.

Exercise 1.3: No loop program

Write a DFT program using a single matrix multiplication. Write your own DFT matrix rather than using the built-in `dftmtx`. Use the `exp` command with the exponent formed by an outer product of a vector of `n = 0:(N-1)` and a vector of `k = 0:(N-1)`. Time and evaluate flops for this program as was done for the two previous programs. Experiment with an evaluation which includes the formation of the DFT matrix and one which includes only the matrix multiplication. This can be done by precomputing the matrix for a given N. Comment on the differences and on the comparisons of the three implementations of the DFT formula. How many flops are used in generating the complex exponentials?

Exercise 1.4: Time and Flops vs. Length

For each of the above formulations, write a program with a loop which will calculate the DFT with lengths from one up to some maximum value partially determined by the speed and memory of your computer. In each program, form a vector of times and a vector of flops. A plot of these vectors gives a picture of execution time vs. length and of flops vs. length. The theory predicts an N^2 dependence. Is that the case? Given that a single complex multiplication requires four real multiplications and two real additions, can you account for all the measured flops?

Exercise 1.5: Formula for flops vs. length

Use the MATLAB command `polyfit` on the flop vs. length data to determine the form and specific coefficients of the flop count.

Project 2: Goertzel's Algorithm

The Goertzel algorithm is a "direct" method to calculate the DFT which also requires order N^2 operations. However, in some cases it uses approximately half the number of multiplications used by the direct methods investigated in the first project and illustrates a different organization.

Project Description

The evaluation of the length-N DFT can be formulated as the evaluation of a polynomial. If a polynomial is formed from the data sequence by

$$X(z) = \sum_{n=0}^{N-1} x[n]\, z^n \tag{2.1}$$

it is easily seen from the definition of the DFT in (1.1) that the k^{th} value of the DFT of $x[n]$ is found by evaluating $X(z)$ at $z = W_N^k$. This data polynomial is similar to the z-transform of $x[n]$, but the use of positive powers of z turns out to be more convenient.

An efficient method for evaluating a polynomial is Horner's method (also called nested evaluation). Horner's method uses a grouping of the operations illustrated in the following example.

$$X(z) = [[[x[4]z + x[3]]\, z + x[2]]\, z + x[1]]\, z + x[0]. \tag{2.2}$$

This sequence of operations can be written recursively as a difference equation in the form

$$y[m] = z\, y[m-1] + x[N-m] \tag{2.3}$$

with the initial condition, $y[0] = 0$ and the evaluated polynomial being the solution of the difference equation at $m = N$.

$$X(z) = y[N] \tag{2.4}$$

The DFT value $X[k]$ is then the value of $y[n]$ when $z = W_N^k$ and $n = N$.

This means $X[k]$ can be evaluated by a first-order IIR filter with a complex pole at $z = W_N^k$ and an input of the data sequence in reverse order. The DFT value is the output of the filter *after N iterations.*

A reduction of the number of multiplications required by Goertzel's algorithm can be achieved by converting the first-order filter into a second-order filter in such a way as to eliminate the complex multiplication in (2.3). This can be done by multiplying the numerator and denominator of the first order filter's transfer function by $z - W_N^{*k}$. If the number of multiplications are to be reduced, it is important *not* to implement the numerator until the last iteration.

Hints

This project will use the MATLAB `polyval` command as well as the `filter` command and a personally-written m-file to implement the Goertzel difference equation. Details on the use of these commands can be found in the manual and by use of the `help` command. "External" MATLAB functions are actually m-files that can be examined with the `type` command. "Internal" non m-file commands or functions are generally faster than m-files. The calculations can be timed with the `clock` and `etime` commands as done in the first project and the number of floating point operations used can be evaluated by the `flops` command.

Remember that MATLAB starts the addresses of all vectors and arrays at one while the DFT formulas start at zero. Care must be taken in writing an m-file program or function from a mathematical formula to use the proper indexing.

Exercise 2.1: Horner's method

Verify that equations (2.3) and (2.4) are implementations of Horner's polynomial evaluation in (2.2). Write a MATLAB m-file program to calculate the DFT of a sequence by using the command `polyval` to evaluate (2.1) at $z = W_N^k$. After this is tested to be correct, put the statements in a loop to evaluate all N DFT values. Write a version of this program that does not use a loop, but rather calls `polyval` with a vector argument to do all the evaluations at once. Measure the flops of both versions for several values of N and compare the results with those from project 1 and discuss.

Exercise 2.2: Use the `filter` Command

Write a program to evaluate the DFT at one k value using the MATLAB `filter` command. After this is tested to be correct, put the statements in a loop to evaluate all N DFT values. Compare the times and flops for several values of N to the results of the direct evaluation from project 1 and exercise 2.1 above. Explain any differences.

Exercise 2.3: Use a Difference Equation

In order to better understand the implementation of Goertzel's algorithm, write an m-file implementation of the difference equation (2.3) rather than use the `filter` command. The DFT values should be the same as in the filter implementation of exercise 2.1 or 2.2 or a direct calculation from (1.1). After this implementation is tested to be correct and put in a loop (or written to operate on a vector of inputs) to give all DFT outputs, compare the flops with the results of exercises 2.1 and 2.2. Are they what you would expect? Compare execution times with the results from exercise 2.1, 2.2 and, perhaps, project 1 for several different lengths.

Exercise 2.4: Trig function evaluations

Compare the number of trigonometric function evaluations of Goertzel's method with the direct method.

Exercise 2.5: Second order Goertzel's method

The first order Goertzel algorithm can be modified into a second order filter that uses only real multiplications and, therefore, reduces the number of required multiplications. The details can be found in [8] or [3]. Write a second order Goertzel realization by an m-file which implements the second order difference equation and evaluate its timings and flops. It should have approximately the same number of additions and one half the multiplications. Do you find that?

Exercise 2.6: Real and complex data

It is inefficient to use a general DFT program that can take complex inputs on real input data. Evaluate the first and second order Goertzel algorithms in terms of number of operations required to calculate the DFT of real data in comparison to complex data.

Computer-Based Exercises

for

Signal Processing

The Cooley-Tukey FFT

The Cooley-Tukey FFT

Overview

The Cooley-Tukey FFT is the name of a family of algorithms that use the decomposition of the DFT described in the original paper by Cooley and Tukey. The basic idea behind the FFT is the elimination of redundant calculations in the direct evaluation of the DFT from its definition in (1.1). This is done by factoring the length N and calculating multiple short DFT's with lengths equal to the factors. The Cooley-Tukey FFT allows any factoring of the length. The factors may be all the same as is the case when the length is $N = R^M$. Here R is called the radix and the FFT is called a radix-R FFT. If the factors are different, such as $N = N_1 N_2 N_3 \cdots N_M$, the algorithm is called a mixed radix FFT and the factors may or may not be relatively prime. This is in contrast to the prime factor FFT which requires all factors to be relatively prime (see packet on *Prime Factor Algorithms*).

Since the Cooley-Tukey FFT allows any factoring of the length, it is more versatile than the prime factor FFT. The disadvantage is that the Cooley-Tukey FFT requires an extra set of multiplication by what are called *twiddle factors* that the prime factor FFT does not. The FFT usually achieves its greatest efficiency when the length can be factored in the largest number of factors. The most popular algorithms are the radix-2 and radix-4 FFT's.

This set of projects investigates three approaches to the efficient calculation of the FFT. All use the Cooley-Tukey structure, which requires twiddle factors, but each develops a different aspect or point of view.

Background Reading

The Cooley-Tukey FFT is covered in all DSP books but details can be found in [16, 17].

Project 1: A Recursive Derivation of the FFT

A powerful and versatile algorithmic strategy is the use of recursion. The idea of recursion is both descriptive and enlightening when formulating the class of algorithms that implement the "divide and conquer" strategy. It can also be a very compact way of programming an algorithm, but is sometimes inefficient in implementation. This project will use the decomposition possible with a composite length DFT to derive the fundamentals of the FFT using a recursive realization. The original FFT was derived and programmed this way.

Project Description

In this project we will use basic properties of the DFT to write a recursive program that efficiently evaluates

$$X[k] = \sum_{n=0}^{N-1} x[n]\, W_N^{nk} \tag{1.1}$$

for

$$W_N = e^{-j2\pi/N}.$$

Let $N = P \times K$ where N is the length of the original data sequence $x[n]$, K is the sampling interval, and P is the length of the sequence of samples $x[Kn]$. The *sampling* *property* gives the length-P DFT of the sequence of samples in terms of K shifted and summed DFT's of the original sequence as

$$\mathcal{DFT}\{x[Kn]\} = \frac{1}{K}\sum_{m=0}^{K-1} X[k+Pm]. \tag{1.2}$$

The *shift property* relates the DFT of a shifted sequence to the DFT of the original sequence by

$$\mathcal{DFT}\{x[n+S]\} = W_N^{-Sk} X[k]. \tag{1.3}$$

Now take the case where $N = 2 \times N/2$, i.e., $K = 2$. One can show that a length-$N = 2^M$ DFT can be calculated from two length-$N/2$ DFT's. The *sampling property* states the length-$N/2$ DFT of the even terms of $x[n]$ is

$$\mathcal{DFT}\{x[2n]\} = \tfrac{1}{2}\left(X[k] + X[k+N/2]\right). \tag{1.4}$$

Applying the shift then the sampling properties and noting that $W_N^{N/2} = -1$ gives the length-$N/2$ DFT of the odd terms of $x[n]$ as

$$\mathcal{DFT}\{x[2n+1]\} = \frac{W_N^{-k}}{2}\left(X[k] - X[k+N/2]\right). \tag{1.5}$$

Solving these equations for $X[k]$ gives

$$X[k] = \mathcal{DFT}\{x[2n]\} + W_N^k\, \mathcal{DFT}\{x[2n+1]\} \tag{1.6}$$

$$X[k+N/2] = \mathcal{DFT}\{x[2n]\} - W_N^k\, \mathcal{DFT}\{x[2n+1]\} \tag{1.7}$$

for k and $n = 0, 1, \cdots, N/2-1$. This states the length-N DFT of $x[n]$ can be calculated in two length-$N/2$ parts from the half-length DFT of the even terms of $x[n]$ plus the DFT of the odd terms multiplied by an exponential factor. This particular formulation of the evaluation is called a decimation-in-time (DIT) algorithm because the input is divided into two parts by taking the even terms for one and odd terms for the other. The exponential factor is called a *twiddle factor* because it is part of an extra non-DFT operation necessary to account for the shift of time index by one.

An alternate set of relationships which uses a decimation-in-frequency (DIF) organization is given using length-$N/2$ DFT's by

$$X[2k] = \mathcal{DFT}\{x[n] + x[n+N/2]\} \tag{1.8}$$

$$X[2k+1] = \mathcal{DFT}\{[x[n] - x[n+N/2]]W_N^n\} \tag{1.9}$$

for $k = 0, 1, \cdots, N/2-1$. Both the DIT and DIF formulas define a length-2^M DFT in terms of two length-2^{M-1} DFT's and those are evaluated in terms of four length-2^{M-2} DFT's and if this process repeated until the length is one, the original DFT can be calculated with M steps and no direct evaluation of a DFT. This formulation is perfect for recursive programming which uses a program that calls itself.

Hints

MATLAB supports recursive functions. In order for a recursive program or function to execute properly, it must have a stopping condition. In our case for the DFT, after M steps, when the length is reduced to one, the single DFT value is the signal value, otherwise, the length is reduced further. This can be realized in a program using an `if` control which will keep the program calling itself and reducing the length by a factor of two until the length is one where the DFT is set equal to the signal value. Although certainly not necessary, some reading about recursive programming might enable you to get more from this project and to apply the ideas to other algorithms. Recursion is fundamental to certain general purpose programming languages such as Lisp or Scheme.

Exercise 1.1: Example of a Recursive Program

The following MATLAB function will compute a sum of the elements in a vector recursively. It is given as an example, so that you can analyze how to write a recursive function in MATLAB. For a length-10 vector, determine how many times `recsum` will be called.

```
function  out = recsum(in)
%RECSUM
%   recursive summation
%
if( isempty(in) )
  out = 0;
else
  out = in(1) + recsum( in(2:length(in)) );
end
```

Exercise 1.2: Derive the Recursive Formulas

Derive the sampling and shift properties of (1.2) and (1.3) from the definition of the DFT in (1.1). Derive the DIT recursive formulas of (1.6) and (1.7) from the sampling and shift properties.

Exercise 1.3: Recursive Decimation-in-Time FFT

Write a MATLAB M-file function to evaluate a length-2^M DFT by recursively breaking the data vector into two half-length vectors with the DIT approach. Using (1.6) and (1.7), construct the DFT vector from the half-length DFT's of the even and odd terms of the data vector. Time and measure the flops for this program for several values of $N = 2^M$.

Exercise 1.4: Number of Floating Point Operations Used

Derive a formula for the number of floating point operations used in the recursive program and show that it is of the form $N \log_2(N)$. Compare with the values measured in exercise 1.3.

Exercise 1.5: Radix-3 and Radix-4 Recursive FFT's

Derive the recursive formulas for a length-$N = 3^M$ DFT in terms of three length $N/3$ DFT's. Write and evaluate a recursive program for a length-3^M DFT. Derive the appropriate formulas and repeat for a length-4^M. Compare and evaluate the results noting any difference in the number of required flops using the radix-2 and radix-4 algorithms on a data sequence of the same length.

Exercise 1.6: Recursive Decimation-in-Frequency FFT

Derive the DIF recursive formulas of (1.8) and (1.9). Write a recursive decimation-in-frequency DFT program. Compare its timings and flop count with the decimation-in-time program and with theoretical calculations.

Project 2: A Two-Factor FFT with Twiddle Factors

The basic ideas and properties of the Cooley-Tukey FFT can be shown and evaluated with a simple two-factor example. All of the commonly used methods for developing FFT algorithms for long lengths involve factoring the length of the transform and then using the factorization to reduce the transform into a combination of shorter ones. The approach studied in this project can be applied to any factorization, whether the factors of N are relatively prime[26] or not. The only requirement is that the length itself not be prime. If the length is $N = R^M$, the resulting algorithm is called a radix-R FFT; if the length is $N = N_1 N_2 \cdots N_M$, the resulting algorithm is called a mixed-radix FFT. The goal of this project is to understand the principles behind this approach through a change of index variables rather than the recursion used in the preceding project.

Project Description

If the proper index map or change of variables is used, the basic DFT of (1.1) can be changed into a form of two-dimensional DFT. If the length is composite (not prime), it can be factored as

$$N = N_1 N_2 \tag{2.1}$$

and an index map defined by

$$n = N_2 n_1 + n_2 \tag{2.2}$$

$$k = k_1 + N_1 k_2. \tag{2.3}$$

For the case of three or more factors, N_1 would be replaced by N/N_2 in (2.3).

Substituting these definitions into the definition of the DFT in (1.1) gives

$$X[k_1 + N_1 k_2] = \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x[N_2 n_1 + n_2] W_{N_1}^{n_1 k_1} W_N^{n_2 k_1} W_{N_2}^{n_2 k_2} \tag{2.4}$$

with n_1 and $k_1 = 0, 1, 2, \cdots, N_1 - 1$ and n_2 and $k_2 = 0, 1, 2, \cdots, N_2 - 1$. Equation (2.4) is a nested double sum that can be viewed as multiple short DFT's. The inner sum over n_1

[26]relatively prime or co-prime means the factors have no common divisors, e.g. 8 and 9 are relatively prime although neither are individually prime.

is evaluated for each value of n_2. It is N_2 length-N_1 DFT's. The resulting function of k_1 and n_2 is multiplied by the set of $W_N^{k_1 n_2}$ which are called *twiddle factors.* The outer sum over n_2 is N_1 length-N_2 DFT's. If the length-N_1 and length-N_2 DFT are done by direct methods requiring N^2 operations, the number of complex multiplications is

$$\#\ \textsc{mult} = N_2 N_1^2 + N_1 N_2^2 + N = N(N_1 + N_2 + 1) \tag{2.5}$$

where the last term of N accounts for the twiddle factor multiplications.

If the length N has M factors, the process can be continued down to the complete factoring of N into its smallest prime factors. This will result in a larger number of nested summations and DFT's and a multiplication count of

$$\#\ \textsc{mult} = N(N_1 + N_2 + \cdots + N_M + (M-1)) \tag{2.6}$$

This is clearly smaller than the N^2 operations that direct evaluation or Goertzel's algorithm would require. Indeed, the greatest improvement will occur when N has a large number of small factors. That is exactly the case when $N = 2^M$.

The goal of the following exercises is to examine this decomposition of the DFT into multiple short ones inter-leaved with necessary twiddle factors. The details of the index map itself are examined in a later project.

Hints

In a practical implementation, the short DFT's are called butterflies (because of the shape of the length-2 DFT's flow graph) and are directly programmed. In the following exercises, we will use MATLAB's built-in `fft` command for the short DFT's and will write programs to combine them. If `fft` is applied to a vector, it returns the DFT vector. If it is applied to a matMrix, it returns a matrix with columns that are DFT's of the columns of the original matrix. This is exactly what we need for our decomposition. We will use the element-by-element operator of `.*` to multiply the arrays by the twiddle-factor array.

Exercise 2.1: A Length-15 Cooley-Tukey FFT

Calculate a length-15 DFT using 5 length-3 DFT's, 3 length-5 DFT's, and one set of twiddle-factor multiplications. First form a 3×5 matrix using the index map in (2.2). Then do the first set of DFT's with one `fft` command. Next do the twiddle-factor multiplications with one statement using the element-by-element matrix multiplication and the `exp` function of a matrix of n_2 and k_1 values formed with a vector outer product. This is followed by a second application of the `fft` function, but after transposing the matrix to change the rows into columns. Finally, the matrix is converted back into a vector using (2.3). The program should be written for two general factors but applied to $N = 15$ for this exercise. Check the results against those calculated by a direct method. Note the order or sequence of the output is not the same as the order of the input. Why? Time and measure the flops for this program. Time and measure the flops of the length-3 and length-5 DFT's and from these results compare your program with what you would expect.

Exercise 2.2: A Length-16 Cooley-Tukey FFT

Calculate a length-16 DFT using two stages of 4 length-4 DFT's and the appropriate twiddle factors in a similar way to exercise 2.1. Time and measure flops and compare to the results in exercise 2.1. Repeat for several other composite lengths. Try several long examples and comment on the improvements.

Exercise 2.3: Equivalent Formulas

Show that the formulas for combining the two half length DFT's in (1.6) and (1.7) and the formulas for combining three one-third length DFT's in project 1 are the same as short length two and three DFT's and the appropriate twiddle factors in this project.

Exercise 2.4: In-Place Calculation of the C-T FFT

Because the decomposition of (2.4) uncouples the row and column calculations, it is possible to write the results of each short DFT over its data since that data will not be used again. Illustrate that property with your FFT program. Turn in your code and label where this in-place calculation is done.

Project 3: The Split-Radix FFT

In 1984 a new FFT algorithm was described which uses a clever modification of the Cooley-Tukey index map to give a minimum arithmetic implementation of an FFT for $N = 2^M$. It is known that the Split-Radix FFT (SRFFT) uses the theoretical minimum number of multiplications for allowed lengths up to 16 and, while not optimal in terms of multiplications, it seems to give minimum number of total floating point arithmetic operations for lengths above 16. This project examines the structure of the SRFFT.

Project Description

All of the algorithms and index maps for the Cooley-Tukey FFT, mixed-radix FFT, PFA, and WFTA are organized by stages where each stage is a certain number of certain length DFT's. For the fixed-radix FFT, each stage is the same DFT; for the mixed-radix FFT and PFA, each stage is a different length DFT; and for the WFTA, the stages are partitioned to an even greater number and permuted to nest and combine all of the multiplications between the stages of additions. The split-radix FFT applies a different strategy by using two different DFT's and two different index maps in each stage. A radix-2 index map is used on the even terms and a radix-4 on the odd terms. This is shown in the following reformulation of the definition of the DFT into a decimation-in-frequency form. For the even spectral values

$$X[2k] = \sum_{n=0}^{N/2-1} (x[n] + x[n+N/2])W_N^{2nk}, \tag{3.1}$$

and for the odd terms

$$X[4k+1] = \sum_{n=0}^{N/4-1} ((x[n] - x[n+N/2]) - j(x[n+N/4] - x[n+3N/4]))W_N^n W_N^{4nk} \tag{3.2}$$

and

$$X[4k+3] = \sum_{n=0}^{N/4-1} ((x[n] - x[n+N/2]) + j(x[n+N/4] - x[n+3N/4]))W_N^{3n} W_N^{4nk}. \quad (3.3)$$

This decomposition is repeated until the length is two when a single length-2 DFT is necessary.

Although it might appear that allowing other lengths such as eight or sixteen might further improve efficiency, they do not. This simple mixture of two and four is the best that can be done using this general organization and it seems to the be the best for reducing multiplications and additions of any organization if $N = 2^M$. A careful analysis of the SRFFT shows it to be only slightly more efficient than a highly optimized radix-4 or 8 FFT.

Hints

The same ideas of recursive programming used in project 1 are needed here. Two stopping conditions will be needed: a simple $X[k] = x[n]$ if the length is one and a length-2 DFT if the length is two; otherwise, the program should call itself. The multidimensional approaches used in project 2 will have to be modified to use two different maps. There will be some problems in obtaining a realistic count of the flops because MATLAB counts multiplication by j as a multiplication. Details of the SRFFT can be found in [20, 21, 23].

Exercise 3.1: Derive Basic Equations

Derive equations (3.1) - (3.3) from the definition of the DFT in (1.1) and the appropriate radix-2 and 4 index maps.

Exercise 3.2: A Recursive SR FFT

Write a recursive implementation of a decimation-in-frequency SRFFT as described in (3.1) - (3.3) using the same approach described in project 1. Test and debug until it gives correct DFT values for all k. The recursion will have to be stopped before the last stage since the last stage is different from the general case.

Exercise 3.3: Compare the Recursive SR FFT and C-T FFT

Compare the execution times and number of flops of the recursive SRFFT with those of the recursive radix-2 FFT from project 3. Remember that MATLAB counts multiplication by j as a multiplication although it really isn't one.

Exercise 3.4: The Decimation-in-Time SR FFT

Derive the appropriate equations and write a recursive decimation-in-time SRFFT.

Exercise 3.5: Non-Recursive SR FFT

Write a multidimensional formulation (i.e. non-recursive) of the SRFFT as was done in project 2.

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{Prime Factor FFT's}}$

Prime Factor FFT's

Overview

There are two organizations used by most FFT algorithms, the first requires that the length of the sequence to be transformed have relatively prime factors. This is the basis of the prime factor algorithm (PFA) and the Winograd Fourier transform algorithm (WFTA). The second can be applied to any factoring of the length into prime or composite factors but requires additional arithmetic. This is used in the Cooley-Tukey fixed-radix FFT and mixed-radix FFT and also in the split-radix FFT. Still another approach converts the DFT into a filter. The chirp z-transform, Rader's method, and Goertzel's algorithm do that.

Project 1: A Two Factor Prime Factor Algorithm FFT

Although the index map used in the CT FFT can be used for all cases of a composite length, a special case occurs when the factors are relatively prime[27]. When the two factors of N have no common factors themselves, it is possible to choose an index map that will give multiple short DFT's as before, but this time, there are no twiddle factors. This approach is used with the prime factor algorithm (PFA) and the Winograd Fourier transform algorithm (WFTA). We again consider the evaluation of the DFT as defined by

$$X[k] = \sum_{n=0}^{N-1} x[n]\; W_N^{nk} \tag{1.1}$$

where

$$W_N = e^{-j2\pi/N}.$$

Project Description

This project is a companion to the project on the Cooley-Tukey FFT, therefore, the material in its description is applicable here. We will now use an index map of the form

$$n = N_2 n_1 + N_1 n_2 \mod N \tag{1.2}$$

$$k = K_3 k_1 + K_4 k_2 \mod N \tag{1.3}$$

where K_3 is a multiple of N_2 and K_4 is a multiple of N_1 chosen in a way to remove the twiddle factors and cause the short summations to be short DFT's. The details of this map will be investigated in project 2, but will not be needed here. Applying this map to the DFT in (1.1) gives

$$X[K_3k_1 + K_4k_2] = \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x[N_2n_1 + N_1n_2]\, W_{N_1}^{n_1k_1}\, W_{N_2}^{n_2k_2} \tag{1.4}$$

with n_1 and $k_1 = 0, 1, 2, \cdots, N_1-1$ and n_2 and $k_2 = 0, 1, 2, \cdots, N_2-1$. This is the same form as found in the project on the Cooley-Tukey FFT, but now with no twiddle factors. We have reduced the amount of arithmetic and found a cleaner decomposition of the DFT, but

[27]relatively prime or co-prime means the factors have no common divisors, e.g. 8 and 9 are relatively prime although neither are individually prime.

it requires a somewhat more complicated index map. The idea can be extended to multiple factors as long as they are all relatively prime. It is a pure conversion of a one-dimensional DFT into a multi-dimensional one. The goal of this project is to understand how this more general index map removes the twiddle factors.

Hints

The same use of the MATLAB `fft` command will be used as it was in the previous project. We will have to evaluate some of the index calculations modulo N. That can be done by using the `rem` command or by using `if n > N, n = n - N; end;` This second form will work here because stepping by N_1 and N_2 in (1.2) never causes n to exceed $2N$ and, therefore, a single subtraction will always suffice.

Exercise 1.1: A Length-15 PFA FFT

Calculate a length-15 DFT using the same approach as in exercise 1 of project 4, but using the index map

$$n = 5n_1 + 3n_2 \bmod 15 \tag{1.5}$$

$$k = 10k_1 + 6k_2 \bmod 15 \tag{1.6}$$

which allows the removal of the twiddle-factor multiplications. Note the output is in scrambled order. Why?

(a) Time and measure the flops of this program and compare with the one using twiddle factors. Why can't this approach allow the twiddle-factor removal for a length-16 DFT?

(b) Show that for the PFA, the short transforms can be done in either order, 5-point ones first, then 3-point ones, or vice versa. Is the same true of the mixed-radix algorithm, assuming that the index map is held fixed?

Exercise 1.2: A Decimation-in-Time PFA FFT

Reverse the time and frequency maps in exercise 1 to be

$$n = 10n_1 + 6n_2 \bmod 15 \tag{1.7}$$

$$k = 5k_1 + 3k_2 \bmod 15 \tag{1.8}$$

and show it still works. This is similar to the DIF and DIT Cooley-Tukey FFT's. How do the number of flops compare with exercise 1.1?

Project 2: The General Linear Index Map

The index map or change of variable used to develop the Cooley-Tukey FFT is a rather straight forward application of decimation in time or frequency. The PFA and WFTA require a somewhat more complicated index map but this map removes the need for twiddle factors. This project develops the general theory of the index map used for almost all types of FFT's and shows how it can be used to remove the necessity of an unscrambler or the equivalent of the bit-reverse-counter. Although not necessary, some reading about basic number theory would give a deeper insight into some of the results [19, 24, 25]. Some practice with number theoretic concepts is included in a later project.

Project Description

The basic one-dimensional single-summation DFT of (1.1) for $N = N_1 N_2$ can be converted into a multi-dimensional nested summation form by a linear change of variables given by

$$n = \langle K_1 n_1 + K_2 n_2 \rangle_N \tag{2.1}$$

$$k = \langle K_3 k_1 + K_4 k_2 \rangle_N \tag{2.2}$$

where

$$n = 0, 1, 2, \cdots, N-1 \tag{2.3}$$

$$n_1, k_1 = 0, 1, 2, \cdots, N_1 - 1 \tag{2.4}$$

$$n_2, k_2 = 0, 1, 2, \cdots, N_2 - 1. \tag{2.5}$$

The notation $\langle n \rangle_N$ means the residue of n modulo N. In the following description, all the indices are evaluated modulo N. After a substitution of index variables, (1.1) becomes

$$X[K_3 k_1 + K_4 k_2] = \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x[K_1 n_1 + K_2 n_2] W_N^{(K_1 n_1 + K_2 n_2)(K_3 k_1 + K_4 k_2)} \tag{2.6}$$

$$= \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x[K_1 n_1 + K_2 n_2] W_N^{K_1 K_3 n_1 k_1} W_N^{K_2 K_3 n_2 k_1} W_N^{K_1 K_4 n_1 k_2} W_N^{K_2 K_4 n_2 k_2} \tag{2.7}$$

The question is now, what values of K_i give interesting and useful results. There are three requirements to be considered.

(a) The map should be unique or one-to-one. That is to say, each pair of n_1, n_2 should correspond to a unique n and vice versa. The same is true for k_1 and k_2. This is necessary if (2.7) is to calculate all of the DFT values.

(b) The map should result in a reduction of the required arithmetic compared to (1.1). This will occur if one or both of the middle two W_N terms in (2.7) becomes unity, which, in turn, happens if one or both of the exponents of W_N is zero modulo N.

(c) The map should cause the uncoupled calculations to be short DFT's themselves. This in not necessary (or, in some cases, not possible), but gives a cleaner, more flexible formulation.

The mathematical statements of these requirements are:

(a) The necessary and sufficient conditions for the map of (2.1) to be unique are stated in two cases depending whether N_1 and N_2 are relatively prime or not. We will use the notation of (N, M) for the greatest common divisor of N and M and use a and b as integer multipliers.

1. Case 1 for N_1 and N_2 relatively prime, $(N_1, N_2) = 1$.

$$K_1 = aN_2 \text{ and/or } K_2 = bN_1 \text{ and } (K_1, N_1) = (K_2, N_2) = 1. \tag{2.8}$$

2. Case 2 for N_1 and N_2 not relatively prime, $(N_1, N_2) > 1$.

$$K_1 = aN_2 \text{ and } K_2 \neq bN_1 \text{ and } (a, N_1) = (K_2, N_2) = 1. \tag{2.9}$$

or

$$K_1 \neq aN_2 \text{ and } K_2 = bN_1 \text{ and } (K_1, N_1) = (b, N_2) = 1. \tag{2.10}$$

In case 1, one or both of the equalities may hold while in case 2, one and only one may be equal.

(b) The calculation of the DFT by (2.7) rather than (1.1) does not necessarily reduce the arithmetic. A reduction occurs only if the calculations are "uncoupled" by one of the middle W_N terms being unity. This occurs if one or both of the exponents are zero modulo N. This requires

$$K_1K_4 = \alpha N \text{ and/or } K_2K_3 = \beta N \tag{2.11}$$

for α and β integers, since a multiple of N is zero modulo N. If the map is unique, one of these conditions can always be made true.

1. If N_1 and N_2 are not relatively prime (if they have a common integer factor) only one of the conditions in (2.11) can be satisfied and (2.6) becomes

$$X[K_3k_1 + K_4k_2] = \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x[K_1n_1 + K_2n_2] W_N^{K_1K_3n_1k_1} W_N^{K_2K_3n_2k_1} W_N^{K_2K_4n_2k_2} \tag{2.12}$$

which uncouples the calculations and reduces the amount of required arithmetic. This is also known as the common factor map. Recall the exponents are all evaluated modulo N.

2. If both N_1 and N_2 are relatively prime, there is a choice of satisfying one or both of the conditions in (2.11). If the K_i are chosen so that only one of the conditions is satisfied, the form of the uncoupled DFT is the same as (2.12), but if both are satisfied, it becomes

$$X[K_3k_1 + K_4k_2] = \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x[K_1n_1 + K_2n_2] W_N^{K_1K_3n_1k_1} W_N^{K_2K_4n_2k_2} \tag{2.13}$$

with no twiddle factors. This is also known as the prime factor map.

(c) In order for the uncoupled summations of (2.12) and (2.13) to be short DFT's, the following must also hold.

$$\langle K_1K_3 \rangle_N = N_2 \text{ and } \langle K_2K_4 \rangle_N = N_1 \tag{2.14}$$

Under these conditions, (2.12) becomes

$$X[K_3k_1 + K_4k_2] = \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x[K_1n_1 + K_2n_2] W_{N_1}^{n_1k_1} W_N^{K_2K_3n_2k_1} W_{N_2}^{n_2k_2} \tag{2.15}$$

which is a two-dimensional DFT with twiddle factors and (2.13) becomes

$$X[K_3k_1 + K_4k_2] = \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x[K_1n_1 + K_2n_2] W_{N_1}^{n_1k_1} W_{N_2}^{n_2k_2} \tag{2.16}$$

which is a two-dimensional DFT with no twiddle factors.

The exercises will examine the various possibilities of these requirements and their consequences. They will also interpret them in terms of the traditional Cooley-Tukey and PFA forms.

Hints

To use MATLAB, a program or function will be needed that evaluates an integer modulo a second integer. This can be simply done by use of the MATLAB function `rem(n,N)`. An alternative is to repeatedly subtract N from n until the remainder is less than N. It may also be useful to have a program that will factor an integer into its prime factors. Some reading of basic number theory might be helpful [19].

Exercise 2.1: Factor an Integer

Write a program to factor an integer n into its prime factors. Use a loop that indexes possible divisors from 2 to $n-1$ and test each with `rem(n,d) == 0`. This might look like

```
factor = [ ];
for d = 2:n-1
   for rem(n,d) == 0
      n = n/d;
      factor = [factor, d];
   end
end
if factor == [ ], factor = n; end
```

(a) Explain why it is only necessary to test `d` up to the square root of n. Explain why it is only necessary to test 2 and the odd integers greater than 2 up to the square root of n. Explain why it is only necessary to test prime values of `d` up to the square root of n (although this would require a table of primes up to the square root of n to be precomputed and available).

(b) Modify the original program to reflect these conditions and test it.

(c) Demonstrate the program above fails (sometimes) when n has repeated roots. Modify it to handle repeated factors correctly.

Exercise 2.2: Index Maps for the FFT

While the form of the index maps allows a wide (infinite) variety of coefficients, there are practical advantages to choosing the smallest positive values that satisfy the required conditions. For $N_1 = 7$ and $N_2 = 8$ (relatively prime), the time index map satisfying (2.8) with the smallest positive coefficients giving no twiddle factors uses $a = b = 1$ giving

$$n = \langle N_2 n_1 + N_1 n_2 \rangle_N \tag{2.17}$$

Find the frequency index map (2.2) with the smallest positive coefficients satisfying (2.8) with both equalities and (2.11) for $N_1 = 7$ and $N_2 = 8$.

Exercise 2.3: CT Index Map

For $N = 56$, the same common factoring in exercise 2.2, and using the smallest positive coefficients allowing twiddle factors we have $a = K_2 = 1$ and

$$n = \langle N_2 n_1 + n_2 \rangle_N. \tag{2.18}$$

Find the smallest positive coefficients for the frequency index map (1.3) satisfying (2.8) and (2.11).

Exercise 2.4: DIF

Repeat exercise 2.2, but for $k = \langle 7k_1 + 8k_2 \rangle_N$ and finding the time map (1.2).

Exercise 2.5: DIT

Repeat exercise 2.3, for $n = \langle n_1 + 8n_2 \rangle_N$.

Exercise 2.6: In-Order, In-Place PFA FFT Index Maps

If both the time and frequency index maps are forced to be the same, there will be no scrambling of order caused by in-place calculations. Set

$$n = \langle 7n_1 + 8n_2 \rangle_N \tag{2.19}$$

$$k = \langle 7k_1 + 8k_2 \rangle_N \tag{2.20}$$

Show that the uncoupling conditions are both met but the short DFT conditions cannot be met. Show that the short transformations are simply DFT's with a permuted order.

Exercise 2.7: Twiddle Factor Array

For $N = 4^M$, examine the twiddle factor array for its general structure. Create a 16 by 4 array with entries W_{64}^{nk}. The 16 rows multiply each of the 16 length-4 DFT's after the first stage of a length-64 radix-4 FFT. The first row and first column will always be unity. There will also always be one entry of $j = \sqrt{-1}$, and four with the real part equal to the imaginary part. What are the locations of these special twiddle factors? Repeat for $N = 128$ and, perhaps others. Give general locations for these special values.

Exercise 2.8: Remove Trivial Operations

For a $N = 4^M$ radix-4 FFT, how many multiplications by 1 or $\pm j$ exist? How many twiddle factors have equal real and imaginary parts? How can this be used to save multiplications and additions in a general FFT?

Project 3: A Prime Length DFT Method and Some Basic Ideas from Number Theory

For a time after the FFT was discovered, it was thought that no major improvements could be made for a prime length FFT over the Goertzel algorithm. In 1968 Charles Rader published a short paper showing how to convert a prime length DFT into a length-(P-1) cyclic convolution [19]. Later Winograd used this same idea to design DFT algorithms which use the absolute minimum number of multiplications. These optimal algorithms turned out to be practical for only a few short lengths, but those are very important when used with an index map to achieve longer DFT's. This project shows how to convert a prime length DFT into cyclic convolution using Rader's permutation of the sequence orders. It also develops ideas in number theory useful to signal processing.

Project Description

The arithmetic system used for the indices in this theory is over a finite ring or field of integers. All indexing arithmetic operations are performed modulo some finite integer modulus. If the modulus is a prime number, all non-zero elements will have a unique multiplicative inverse (i.e. division is defined) and the system is called a *field*. If the modulus is composite, some elements will not have an inverse and the system is called a *ring*. The process of calculating the remainder of a number modulo another is called residue reduction and the relationship of numbers having the same residues is called a congruence. These ideas and definitions are discussed in any introductory book on number theory [19, 24, 25].

Several definitions are needed to develop the ideas of this chapter. Euler's totient function, $\phi(N)$, is defined as the numbers of integers in $\mathcal{Z_N} = \{n | 1 \leq n \leq N-1\}$ that are relatively prime to N. For example, $\phi(3) = 2, \phi(4) = 2, \phi(5) = 4$.

Fermat's theorem states that for any prime number P and for all non-zero numbers $\alpha \in \mathcal{Z_P}$,

$$\langle \alpha^{P-1} \rangle_P = 1 \tag{3.1}$$

and a more general form called Euler's theorem states that for any N and for all non-zero $\alpha \in \mathcal{Z_N}$ that are relatively prime to N,

$$\langle \alpha^{\phi(N)} \rangle_N = 1. \tag{3.2}$$

When it exists, the inverse of an integer $n \in \mathcal{Z_N}$ is the integer $m \in \mathcal{Z_N}$, denoted n^{-1}, where $\langle mn \rangle_N = 1$. Using Euler's theorem, the inverse can be calculated from

$$m = \langle n^{-1} \rangle_N = \langle n^{\phi(N)-1} \rangle_N \tag{3.3}$$

The integer α is called an N^{th} root of unity modulo M if

$$\langle \alpha^N \rangle_M = 1 \tag{3.4}$$

and

$$\langle \alpha^L \rangle_M \neq 1 \text{ for } L < N \tag{3.5}$$

Other terminology for the N^{th} root of unity is that α is of order N or that α belongs to the exponent N. Notice from Euler's theorem that N exactly divides $\phi(M)$, i.e. $\phi(M)$

is an integer multiple of N. If $N = \phi(M)$, α is called a primitive root. Primitive roots are important in several applications. It can be shown that they exist if and only if $M = 2, 4, P^r, or 2P^r$ and there are $\phi(\phi(M))$ of them.

If an integer N is prime, a primitive root r exists such that

$$m = \langle r^n \rangle_N \tag{3.6}$$

generates all of the non-zero integers between $m = 1$ and $m = N-1$ for $n = 0, 1, 2, \cdots, N-2$. There may be several primitive roots belonging to a modulus, each generating the same set of integers, but in a different order. In the finite field of integers $\mathcal{Z_N} = \{0 \le n \le N-1\}$, n is similar to a logarithm. Because this process generates a string of non-repeating integers, a modification of it is sometimes used as a pseudo-random number generator [26].

We now use this integer logarithm to convert a DFT into a cyclic convolution. The form of the DFT is

$$X[k] = \sum_{n=0}^{N-1} x[n] W^{kn} \tag{3.7}$$

and the form of cyclic convolution is

$$y[k] = \sum_{n=0}^{N-1} x[n] h[k-m] \tag{3.8}$$

with all indices evaluated modulo N.

The integer logarithm changes the product of k and n in the DFT into the difference in the cyclic convolution. Let

$$n = \langle r^{-m} \rangle_N \text{ and } k = \langle r^s \rangle_N \tag{3.9}$$

with $\langle n \rangle_N$ denoting the residue of n modulo N. The DFT becomes

$$X[r^s] = \sum_{m=0}^{N-2} x[r^{-m}] W^{r^s r^{-m}} + x[0] \tag{3.10}$$

for $s = 0, 1, 2, \cdots, N-2$. and

$$X[0] = \sum_{n=0}^{N-1} x[n] \tag{3.11}$$

New functions are defined which are simply permutation in order of the old functions.

$$\tilde{x}[m] = x[r^{-m}] \tag{3.12}$$

$$\tilde{C}[s] = X[r^s] \tag{3.13}$$

$$\tilde{W}[n] = W^{r^n} \tag{3.14}$$

This results in the DFT being

$$\tilde{C}[s] = \sum_{m=0}^{N-2} \tilde{x}[m] \tilde{W}[s-m] + x[0] \tag{3.15}$$

which is a cyclic convolution of length $N-1$ (plus $x[0]$) of $x[n]$ and W^n in a permuted order.

Hints

All of the residue reductions can be calculated with the `rem()` function. Try to avoid using loops in implementing the various operations. Details of the ideas in this project can be found in [19, 20, 18, 16]. Some of the ideas in this project relate to those in projects 7, 10 and 12. A bit of reading of basic number theory, especially congruency theory [24, 25], would be helpful.

Exercise 3.1: Residue Reduction

The array of the residues

$$m = \langle \alpha^n \rangle_M \tag{3.16}$$

for $M = 5$ with rows for $\alpha = 1, 2, 3, \cdots, 6$ and columns for $n = 0, 1, 2, \cdots, 5$ is

$$m = \langle \alpha^n \rangle_M = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 3 & 1 & 2 \\ 1 & 3 & 4 & 2 & 1 & 3 \\ 1 & 4 & 1 & 4 & 1 & 4 \\ * & 0 & 0 & 0 & * & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{3.17}$$

where $*$ is undefined. The second row is 2 raised to successively higher powers evaluated modulo 5. The second column is α raised to the first power. The fifth column is for the power or exponent equal to $n = \phi(5) = 4$ which illustrates Euler's theorem. From this array, determine the integer inverses of all non-zero elements of $\mathcal{Z}_5$. There are $\phi(\phi(5)) = 2$ primitive roots. One is $\alpha = 2$, what is the other? The elements of the rows and columns of this array are periodic. What is the period in α? What is the period in n? Study this array carefully to better understand the definitions and properties discussed in the project description or in your other reading.

Exercise 3.2: Roots of Unity

Form five arrays of the residues

$$m = \langle \alpha^n \rangle_M \tag{3.18}$$

for each $M = 6, 7, 8, 9,$ and 11. Let the arrays have rows for $\alpha = 1, 2, 3, \cdots, M+2$ and columns for $n = 0, 1, 2, \cdots, M+1$. Note that all α relatively prime to the modulus belong to some exponent. For each array indicate this exponent. All α relatively prime to the modulus have an inverse and it is the value just to the left of unity in each row of each array. Why? If β is the N^{th} root of unity modulo M, show in the arrays where β^q is the $(N/p)^{th}$ root of unity if q and M have a largest common factor of p.

Exercise 3.3: Primitive Roots

For the six arrays formed in exercises 3.1 and 3.2, which (which moduli) have primitive roots? Find the primitive roots in the arrays that have them and verify that there are

$\phi(\phi(M))$ of them. When the modulus is prime, all non-zero elements of $\mathcal{Z}_{\mathcal{M}}$ are generated by the primitive roots. When the modulus is composite and primitive roots exist, all non-zero elements are generated that are relatively prime to the modulus and these have inverses. This system is a ring. What is the pattern of the sequences generated by non primitive roots when the modulus is prime? when the modulus is composite and some primitive roots exist? when no primitive roots exist?

Exercise 3.4: Euler's Theorem

For each of the arrays formed in exercises 3.1 and 3.2, indicate the cases where $\alpha^{\phi} = 1$. When does that not occur? Indicate the cases where $\alpha^{N} = 1$ for some $N < \phi(M)$. When does that not occur?

Exercise 3.5: Permutations

Using the arrays for $M = 5, 7, 11$, what are permutation of the data and the permutations of the exponentials that will convert the DFT into cyclic convolution? For each M, how many choices are possible?

Exercise 3.6: Rader's Conversion

Convert the length-11 DFT matrix into a convolution matrix and calculate the DFT by convolution.

Computer-Based Exercises

for

Signal Processing

General Length FFT's

General Length FFT's

Overview

There are some cases where one wants an algorithm for efficiently calculating the DFT of an arbitrary length sequence. This is the case for the `fft()` command in MATLAB. In this packet we will investigate the MATLAB FFT and the Chirp Z-Transform method for calculating the FFT.

Project 1: Evaluation of the MATLAB FFT

Matlab, version 3.5, has implemented a clever and general FFT algorithm in its `fft` function. The goal of this project is to analyze this function in such a way that using our knowledge of FFT algorithms will enable us to speculate on how it is implemented.

Project Description

In earlier projects we have seen how composite length DFT's can be implemented efficiently by breaking them down into multiple shorter ones. Here we will carefully time and measure the flops of the MATLAB `fft` function for a large number of lengths and then plot the time or flops vs. the length. From this we can determine what kind of algorithm might be used.

Hints

Timing of the FFT is done by the following code. `x = rand(1,N); time = clock; fft(x); time = etime(clock,time);`.[28] This approach has two problems. Other users on a time-shared system can cause errors in the timing of your programs and the resolution of the clock may not be good enough to accurately time the shorter DFT's. These problems can be partially corrected by executing the FFT several times and averaging and/or by trying to do the timing when no one else is using the computer. A more consistent evaluation of an algorithm is the measurement of the number of floating point operations (flops). For many algorithms, the floating point operations take most of the time and, therefore, the timings and flops measurements are simply a multiple of each other. The flops are measured by the following code. `x = rand(1,N); f0 = flops; fft(x); fl = flops - f0;`. The evaluation of the performance will involve generating a vector of time or flops for lengths varying from zero to several hundred or several thousand depending on the speed and memory of your computer. The plots of time or flops vs. length should not have the values connected by lines but should simply be "dots". If the vector of times (or flops) is the variable `t`, the plot should be made with the command `plot(t,'.')`.

To analyze the FFT it will be necessary to know the factors of the lengths. While this can be done by hand for a small number of lengths, it would be helpful to write a program to factor an integer into its smallest prime factorization. A bit of reading or review of elementary number theory might be helpful [24, 25].

[28]In the MATLAB `fft` there is a difference in the flop count depending on whether the input vector is real or complex; for example, try `x = rand(1,N) + j*rand(1,N);`
Also the flop counter may only be a 16-bit counter in some versions of MATLAB, so that it would overflow for large values of `N`.

Exercise 1.1: Execution Times and Flops for the Matlab FFT

Create a vector of execution times for the Matlab function `fft` by putting the timing of a single FFT in a `for` loop which steps through the lengths from zero to several hundred or several thousand depending on the computing system being used. Also create a vector of flops in the same manner. These measurements could be made at the same time in the same loop. Plot the execution times and the number of flops in separate graphs vs. the lengths of the FFT. Make the plots with simple points representing the times and flops rather than connected lines.

Exercise 1.2: Structure of Flops vs. Length Plot

The plots made in the first exercise will have a very distinct structure which you should evaluate. The slowest times and largest numbers of flops will correspond to prime data lengths. What are the characteristics of the lengths of the other distinct time and flops groups? How do the lengths which are powers of two compare?

Exercise 1.3: What is the Algorithm?

From the shape of the slowest times or greatest flops vs. N plot, you should be able to conjecture what kind of algorithm is being used for a single prime length FFT. From the next few distinct groupings, you should be able to determine if a decomposition is being used and if so, what kind. You should be able to tell if twiddle factors are being used by checking a length which is three to some high power which, of course, must use twiddle factors. From all of this, write a program which calculates the times or flops from a formula based on your idea as to how the command is implemented. Compare the results of this formula with the measured data and correct until you have fairly good agreement. What is the algorithm being used?

Exercise 1.4: A Formula for Flops

Based on the above analysis of `fft`, develop a formula that will simulate the measured flops. Compare this with the theory.

Project 2: The Chirp Z-Transform

The chirp z-transform is a method of evaluating the DFT using an FIR filter. The Goertzel method in project 3 also used a filter but the approach was very different. Rader's permutation used a cyclic convolution, but it could only be applied to prime length DFT's (or with less efficiency to lengths which are a prime to a power). The chirp z-transform can be used on any data length and, while not as efficient as a Cooley-Tukey FFT or the PFA, it can be implemented using of the order of $N \log(N)$ arithmetic operations.

Project Description

The Cooley-Tukey FFT, PFA, and Rader's method all used linear index maps to reorganize the basic DFT so that it could be evaluated more efficiently. Here we use a non-linear index map to change the DFT into a non-cyclic convolution. Applying the identity

$$(k-n)^2 = k^2 - 2kn + n^2 \tag{2.1}$$

$$nk = \left(n^2 - (k-n)^2 + k^2\right)/2 \tag{2.2}$$

to the definition of the DFT gives

$$X[k] = \left[\sum_{n=0}^{N-1} (x[n]\, W^{n^2/2}) W^{-(k-n)^2/2}\right] W^{k^2/2} \tag{2.3}$$

This has the form of first multiplying the data by a chirp sequence, then convolving that with the inverse of the chirp, and finally multiplying the output of the convolution by the chirp. If posed as a filter, the impulse response of the filter is

$$h[n] = W^{n^2/2} \tag{2.4}$$

and the chirp transform of (2.3) becomes

$$X[k] = \Big((x[k]h[k]) * h^{-1}[k]\Big)\, h[k] \tag{2.5}$$

Care must be taken to implement the finite length non-cyclic convolution properly to obtain the correct length-N output sequence indicated in (2.3). It can be calculated directly or by use of the DFT. Indeed, the FFT is the way it is possible to improve efficiency over the Goertzel method.

Although discussed here as a means of calculating the DFT, the chirp z-transform is very flexible and can evaluate the z-transform on contours in the z-plane other than the unit circle and can efficiently evaluate a small number of values.

Hints

The implementation of the chirp z-transform can be done with the `conv` or the `filter` commands or with the `fft` and `ifft` commands. When using fast convolution with the FFT, be careful to take the proper part of the output. The evaluation can use the `flops` and/or the `clock` commands. Details of the methods can be found in [8, 16].

Exercise 2.1: Write a Chirp DFT Program

Write a MATLAB function that will calculate the DFT using the chirp z-transform algorithm. Create a chirp vector with `n = 0:N-1; W = exp(-j*pi*n.*n/N);`. Multiply this times the data vector and convolve the result with a properly extended version of W using the `conv` command. The appropriate length-N segment of the output should be multiplied by W and that will be the DFT. Check the function against the `fft` function to make sure it is correct. Measure the flops and execution times for a variety of lengths and compare with the FFT as was done in project 7. Plot the number of required flops vs. the length of the DFT and explain the numbers and shape.

Exercise 2.2: Use the FFT in the Chirp DFT

Write a MATLAB function to calculate the DFT using the chirp z-transform as was done in exercise 1, but implement the convolution by using the MATLAB `fft` and `ifft` functions. Plot the flops and execution times vs. length and compare with the FFT. Explain both the numbers of flops required and the dependency on N.

Exercise 2.3: Compare Chirp FFT with Other Methods

Make a comparison of the Goertzel algorithm, the MATLAB `fft` command, and the chirp z-transform implemented with the FFT from exercise 2. Do this by making a plot of "flops" vs length for lengths up to 130 or more on a faster computer. Make the plot with "dots" for the `fft` , "x"'s for the chirp, and "o"'s for the Goertzel rather than connected lines. From these results, how would you implement a general purpose FFT?

Exercise 2.4: Improvements for a Single Length

If one wants to execute a single DFT repeatedly on different data, a special purpose program can be written that will pre-calculate the chirp in (2.4) and its FFT needed for the convolution. Write a MATLAB program that will count only the flops needed for execution assuming all possible pre-calculations have been done. Compare these results with the `fft` . From this analysis over different lengths, how would you implement a special purpose FFT?

Exercise 2.5: Calculation of Sub-Set of the DFT values

The usual DFT is the set of N equally spaced samples of the z-transform of a length-N data sequence. Modify the chirp z-transform MATLAB function to calculate N samples of the z-transform on the unit circle between $\omega = 0$ and $\omega = \pi/2$. Does it require any more or less arithmetic that the version in exercise 1 or 2?

Exercise 2.6: Use of the Chirp Z-Transform off the Unit Circle

Modify the chirp z-transform program of exercise 2 and 3 to evaluate the z-transform of a length-N data sequence over L samples on a contour other than the unit circle. What contours can the method be used for? Derive a formula for the number of arithmetic operations needed as a function of N and L.

Chapter 10

Applications

January 17, 2007

Overview

This chapter presents speech and radar applications. MATLAB is very effective for simulating such applications. In addition, for the speech case, actual signals can be brought in and analyzed.

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{Radar Simulation}}$

Radar Simulation

Overview

This set of projects will introduce the concepts of range and velocity measurements in a radar system. Of particular importance is the development of properties of the linear-FM (LFM) chirp signal.

Background Reading

1. Rabiner & Gold: *Theory and Applications of DSP*, P-H, 1975.

Project 1: A Sampled Chirp in MATLAB

Project Description

In this project, the characteristics of the LFM chirp signal will be investigated: its Fourier transform and its autocorrelation function (which is the output of a matched filter).

The pulse should have the following characteristics:

CHIRP SIGNAL PARAMETERS

Parameter	Value	Units
Pulse Length	50	μsec
Swept Bandwidth	2	MegaHz
Sampling Frequency	8	MegaHz
TW Product	100	dimensionless

Table 1. Desired Parameters for a Chirp Signal

Hints

The *chirp* radar signal is defined by the formula:

$$s(t) = e^{j\pi W t^2/T} \qquad -\frac{T}{2} \le t \le \frac{T}{2}$$

This is the baseband form of the chirp, so it is a complex-valued signal. It has a time duration equal to T seconds and a swept bandwidth of W Hertz (from $-\frac{1}{2}W$ to $+\frac{1}{2}W$).

Exercise 1.1: A Sampled Chirp Signal

(a) Consider a sampled chirp signal whose time-bandwidth product is TW. Perform the sampling at p times Nyquist. Give the equation for this signal in the form:

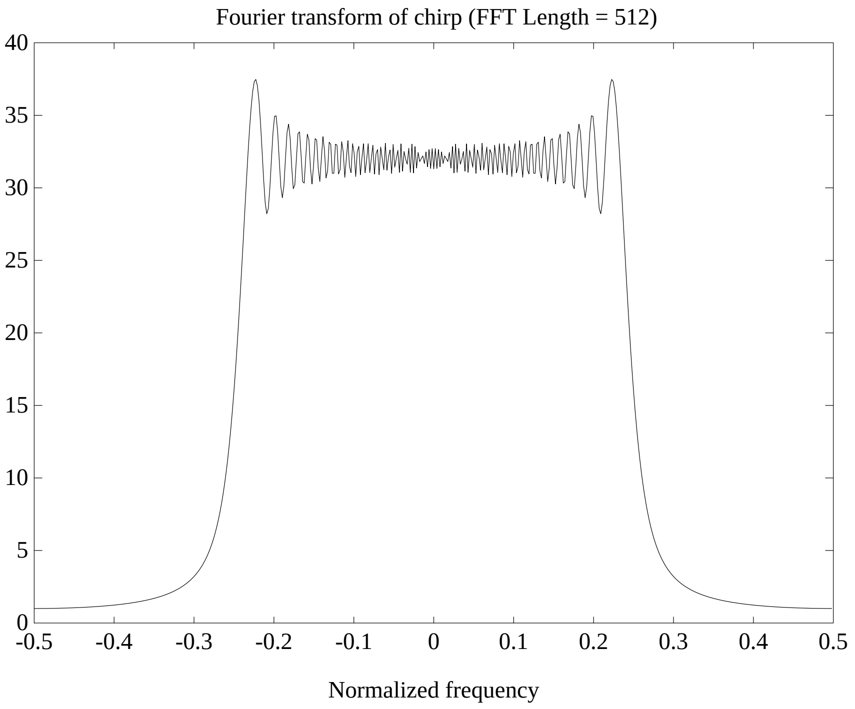
$$s[n] = \exp(j2\pi\alpha n^2) \qquad 0 \le n \le N-1$$

Determine the appropriate values for α and N, in terms of p and TW. Be careful with the definition of the sampling times: start at $t_n = -T/2$ and take N samples up to (but not including) $t = T/2$. This is because the analog signal is symmetric in t about zero, but the sampled signal is not.

(b) Generate a sampled chirp signal whose time-bandwidth product is $TW = 100$. Perform the sampling at twice Nyquist. Use the formula from part (a) to create a signal vector in MATLAB with the correct values.
Plot the real and imaginary parts of the chirp signal.

Exercise 1.2: Fourier Transform of a Chirp

The Fourier transform of a chirp is *approximately* a rectangle in frequency, *if the time-bandwidth product is large*, i.e., $TW > 100$. If we assume that $s(t)$ is the chirp, and $\tilde{S}(f)$ is its *approximate* Fourier transform, then $|\tilde{S}(f)|$ is a rectangle that extends from $f = -W/2$ to $f = +W/2$. Shown here is a true $|S(f)|$, so the rectangle approximation is not bad.



(a) Compute the Fourier transform of $s[n]$ by using the `fft()` function (with zero padding) in MATLAB.
Is the bandwidth of the chirp spectrum correct, according to the value expected from the continuous-time analysis done in class?
Try the FFT length equal to N, and also at 4 times N (padding with zeros). Note: the zero padding is used to plot a frequency response that is interpolated so that it appears to be continuous. Does the longer FFT buy anything? Why, or why not?
Note: you just want to plot a frequency response that appears to be continuous.

(b) Find the height of $\tilde{S}(f)$ by assuming $\tilde{S}(f) \approx S(f)$, and then using Parseval's theorem:

$$\int_{-\infty}^{\infty} |s(t)|^2 \; dt = \int_{-\infty}^{\infty} |S(f)|^2 \; df$$

(c) Use the results of Problem 4.1 to verify that your answer in part (a) is correct. Compute the inverse Fourier transform (at $t = 0$) of $|\tilde{S}(f)|^2$, while still using the approximation that the spectrum is a rectangle. Since $|\tilde{S}(f)|^2$ is the Fourier transform of the output of the matched filter, this value at $t = 0$ should match $y(t)$ found in the previous problem (at $t = 0$).

(d) In the previous part, explain why the height of the spectrum has the value observed (for the sampled chirp). This can be done by using Parseval's theorem to equate the energies in the time and frequency domains. Give a general formula for the height of the spectrum in terms of TW, N and p.

Exercise 1.3: Matched Filter Output

In class we used several approximations in arriving at the conclusion that the output of a pulse-compression matched filter is a "sinc" function. The purpose of this problem is to calculate the exact form for the output of the matched filter. Specifically, assume that the *chirp* radar signal is

$$s(t) = e^{j\pi W t^2/T} \qquad -\frac{T}{2} \le t \le \frac{T}{2}$$

Use the matched filter $h(t) = s^*(-t)$ and compute the output via convolution: $y(t) = s(t) * h(t)$. Be careful with the limits on the convolution integral. Note: the final output will be the product of 3 terms: a quadratic phase term, a window function, and a 'sinc" function.

(a) Generate a sampled chirp signal with $TW = 128$. Sample at 8 times Nyquist. Use this signal in a matched filter, creating the output by using DFT (FFT) convolution. Make sure you pad with zeros to avoid circular convolution. Plot the output, and show that it has the form predicted from the analysis of the continuous-time signal: i.e., a windowed "sinc" function. Check that the windowed "sinc" function has the correct mainlobe width.
NOTE: you may have to plot the magnitude after going through the `ifft()`, because the signal ends up being complex-valued due to the quadratic phase.

(b) Implement a "mis-matched" filter by multiplying in the frequency domain by a Hamming window. Note that this Hamming weighting can be implemented during the FFT convolution. Compare the "sidelobes" of the compressed pulse output in this case to the sidelobes for the compressed pulse from the *true* matched filter.

Exercise 1.4: Fourier Transform of a Burst

If a burst of pulses is considered as a single transmitted waveform, the formula defining the waveform is

$$s[n] = \sum_{\ell=0}^{L-1} p[n - \ell M]$$

where M is the inter-pulse period, $p[n]$ is a boxcar of length N, and L is the number of pulses.

(a) Determine $S(e^{j\omega})$, the DTFT of $s[n]e^{j\omega_o n}$, when $\omega_o = 2\pi/5$. This corresponds to a Doppler shifted burst. In order to do the DTFT without resorting to excessive algebraic grinding, consider that $s[n]$ is produced by convolving a finite sequence of pulses spaced by M with $p[n]$. Use the windowing and convolution properties of the DTFT.

(b) Use MATLAB to plot an example spectrum. Take $L = 8$, $N = 7$, and $M = 19$.

(c) Explain all the features in the plot from part (b): the number of spectral lines, their widths, their heights, and the shape of the spectral envelope.

Project 2: Radar System Simulation

Project Description

This project requires that you develop a radar processing system that will extract the range and velocity of several "point" targets. The parameters of the transmitted LFM (linear-FM) radar signal are given in the table below:

RADAR PARAMETERS		
Parameter	Value	Units
Radar Frequency	7	GigaHz
Pulse Length	7	μsec
Swept Bandwidth	7	MegaHz
Sampling Frequency	8	MegaHz
Inter-pulse Period	60	μsec
Number of Pulses	11	*none*
Time Delay to Start Receive Window	25	μsec
Time Delay to End Receive Window	50	μsec

Table 1. Radar Parameters of the Transmitted Waveform.

Hints

The data file that is available for processing contains 4–8 point targets. It is synthetic data, and was produced by the m-file called `radar.m`. You should consult the listing of this file for detailed questions about how the simulated radar returns were actually derived. The exact parameter values (for velocity and range) are, of course, unknown, but `radar.m` could be used to generate other synthetic data sets.

SIMULATION
The parameters of the radar simulation and the targets were chosen to match those that might be expected in an ATC (Air Traffic Control) application. One possible application might be short to medium range tracking, needed to control the final approach patterns of commercial aircraft to an airport. This scenario would require a burst waveform to measure velocity accurately; on the other hand, the maximum range is somewhat limited due to the

relatively high PRF (pulse repetition frequency) used in the burst. You should use this "scenario" to check the values of answers you get for velocities (especially) and for ranges.

SIGNAL PROCESSING

In order to create the processing program, it will be necessary to analyze the radar waveform for its delay and Doppler content. This will require the following steps (at least):

(a) Process the return with a matched filter to compress the LFM pulse that was used by the transmitter. This requires that you re-synthesize the transmitted *chirp* waveform according to the parameters given above, using the sampling frequency as given. From this transmitted pulse, the impulse response of the matched filter can be obtained. NOTE: there is a function called `chirp.m` that will produce an arbitrary LFM pulse.

(b) The transmitted signal is a burst waveform consisting of 11 identical LFM pulses. Because MATLAB may present memory limitations, each pulse must be processed separately by the matched filter.

(c) From the compressed pulses, it is then necessary to perform a spectrum analysis of the data across the 11 pulses of the burst to extract the Doppler frequency. This will require a DFT method of spectrum analysis, but only at the ranges where there are likely to be targets. Identify the major velocity components from peaks in the frequency domain plot. Make sure that you consider the possibility of positive and negative velocities.

(d) The returned radar signal contains a very large clutter component, *so you must implement some sort of pre-processing to reduce the clutter return.*

(e) The valid peaks in range and/or Doppler need to be identified by a peak picking algorithm. Automatic peak picking requires that you define a threshold that depends on the level of the additive noise. In this project, visual identification of the peaks is OK, but you should state where you set the threshold (visually).

(f) Be careful to convert all range plots and Doppler frequency plots to the correct units (i.e., Hertz, meters/sec, or kilometers, etc.) This involves the use of the sampling frequency (in range or Doppler), and the length of the FFT.

RECEIVE WINDOW

The radar returns can be spread over a very wide time span if the range coverage needs to be large. In order to avoid ambiguities, the range window must be limited to the time interval between successive pulses. Thus the maximum range is $R_{\max} = c\Delta/2$; the minimum range is dictated by the length of the pulse. In the simulation function `radar()`, a receive time window can be specified, so as to limit the number of returned samples that must be processed. (NOTE: This is the only way to make the data set a manageable size.)

NOISE

The data contains two forms of noise:

(a) Receiver noise that is modelled as white Gaussian noise. It is present in all the channels and is completely uncorrelated from one pulse to the next, and from one time sample to the next.

(b) Clutter which is really a distributed target. On a pulse to pulse basis, this sort of noise is *highly* correlated, and is usually removed by prefiltering with a canceller.

(c) For many of the pulses, the signal to noise ratio (SNR), and/or signal to clutter ratio is less than 1. Therefore, the uncompressed pulse is (well) below the noise and can only be identified after the pulse compression stage.

THEORY of RADAR RETURNS

The received signal in a radar can be expressed completely in terms of its complex envelope signal. If the transmitted signal has a complex envelope given by $s(t)$, which represents the phase (and amplitude) modulation used at the transmitter, then the actual RF signal would be:

$$\Re e\left\{ e^{j2\pi f_c t} \sum_{\ell=0}^{N_p-1} s(t-\ell\Delta) \right\}$$

where f_c is the center (RF) frequency of the radar.

For a moving target the range to the target varies as function of time (t). If we assume that the velocity is a constant (v), then the expression for the range is:

$$R(t) = R_\circ - vt$$

where $R_\circ$ is the range at a "reference" time $(t = 0)$. NOTE: the minus is due to the fact that a target traveling toward the receiver should have a positive velocity, and, therefore, a positive Doppler shift.

From the range $R(t)$, we can write the time-delay to the target as $2R(t)/c = 2(R_\circ - vt)/c$. Thus the received signal will be:

$$\Re e\left\{ e^{j2\pi f_c (t-2R(t)/c)} \sum_{\ell=0}^{N_p-1} s(t-2R(t)/c-\ell\Delta) \right\}$$

Since the carrier term can be extracted by a quadrature demodulator, the complex envelope of the received signal will be:

$$r(t) = e^{j2\pi f_c(-2R_\circ/c+2vt/c)} \sum_{\ell=0}^{N_p-1} s(t-2R_\circ/c+2vt/c-\ell\Delta)$$

In the delayed complex envelope $s(t)$, the delay term that depends on t can be dropped, because in $2vt/c$ the ratio v/c is extremely small. Thus the final result is:

$$r(t) = e^{j2\pi f_c(-2R_\circ/c)}\; e^{j2\pi f_c(2vt/c)} \sum_{\ell=0}^{N_p-1} s(t-\ell\Delta-2R_\circ/c)$$

An alternate form is possible if a new time variable $t' = t - \ell\Delta$ is defined.

$$r(t) = e^{j2\pi f_c(-2R_\circ/c)}\; e^{j2\pi f_c(2vt'/c)} \sum_{\ell=0}^{N_p-1} s(t'-2R_\circ/c)\; e^{j2\pi f_c(2v\ell\Delta/c)}$$

The time t' is referenced to the beginning of each pulse in the burst.

For a Linear FM signal, the complex envelope of the transmitted signal, $s(t)$, is actually made up of two parts: a pulse $p(t)$ that turns the signal on and off, and the LFM phase modulation. For $p(t)$, the delay term is just a shift, but in the LFM phase, the delay must be applied to the argument of the quadratic term. This is done by the `polyval()` function in the exponent of the last term that makes up the output signal `y` in `radar.m`. The quadratic phase was previously extracted by using a call to `polyfit()`, thus allowing the user to enter samples of the complex envelope rather than parameters describing the LFM modulation (i.e., T and W).

The data file that is given as `r100.mat` contains the weighted sum of 4–8 different targets, each with the form given above. Each pulse in the burst gives 201 data samples in range, so the data matrix is 201×11. The objective of the project is to extract the parameters of the various targets, by processing the returned waveform versus time-delay and frequency (Doppler). The targets have different amplitudes, so you should also try to extract information that measures the relative amplitudes.

CHIRP.M

```
function   x = chirp( T, W, p )
%CHIRP      generate a sampled chirp signal
%    X = chirp( T, W, <P> )
%      X:  N=pTW samples of a "chirp" signal
%            exp(j(W/T)pi*t^2)  -T/2 <= t < +T/2
%      T:  time duration from -T/2 to +T/2
%      W:  swept bandwidth from -W/2 to +W/2
%    optional:
%      P = samples at P times the Nyquist rate (W)
%          i.e., sampling interval is 1/(PW)
%          default is P = 1
%
if nargin < 3
  p = 1;
end
J = sqrt(-1);
%--------------
delta_t = 1/(p*W);
N = round( p*T*W );    %--same as T/delta_t
nn = [0:N-1]';
x = exp( J*pi*W/T * (delta_t*nn - T/2).^2 );
```

RADAR.M

```
function y = radar( x, fs, T_0, g, T_out, T_ref, fc, r, a, v )
%RADAR      simulate radar returns from a single pulse
%  usage:
%    R = radar( X, Fs, T_0, G, T_out, T_ref, Fc, R, A, V )
%      X:      input pulse (vector, use matrix for burst)
%      Fs:     sampling frequency of input pulse(s)      [in MHz]
%      T_0:    start time(s) of input pulse(s)         [microsec]
%      G:      complex gains; # pulses = length(g)
%      T_out:  2-vector [T_min,T_max] defines output
%                window delay times w.r.t. start of pulse
%      T_ref:  system "reference" time, needed to simulate
%                 burst returns. THIS IS THE "t=0" TIME !!!
%      Fc:     center freq. of the radar.              [in MHz]
%      R:      vector of ranges to target(s)        [kilometers]
%      A:      (complex) vector of target amplitudes
%      V:      vector of target velocities (optional)  [in m/sec]
%
%  note(1): VELOCITY in meters/sec !!!
%           distances in km, times in microsec, BW in MegaHz.

%  note(2): assumes each pulse is constant (complex) amplitude
%  note(3): will accommodate up to quadratic phase pulses
%  note(4): vector of ranges, R, allows DISTRIBUTED targets
%
%     (c) jMcClellan 7/28/90

J = sqrt(-1);
c = 0.3;         % velocity of light in km/microsec
r = r(:);   a = a(:);
if nargin < 7
  v = zeros(r);
end
[Mx, Nx] = size(x);
if Mx == 1
   old_Nx = Nx;   Mx = Nx;   Nx = 1;  x = x.';
end
if Nx ~= 1,    error('MATRIX x NOT ALLOWED !!!'),  end
g = g(:).';
delta_t = 1/fs;
T_p = Mx*delta_t;     % length of input pulse
t_x = (delta_t)*[0:(Mx-1)]';
%
%-------- extract the phase modulation of the input pulse ----
%
x_ph = unwrap(angle(x));
q = polyfit( t_x, x_ph, 2 );
xfit = polyval( q, t_x );
if (x_ph'*xfit)/norm(x_ph)/norm(xfit) < 0.99   % correlation coeff
   disp(' no quadratic phase match')
   keyboard
end
%
%---  output matrix ---
%
t_y = [ T_out(1):delta_t:T_out(2) ]';  % output sampling times
Mr = length(t_y);  Nr = length(g);     % output samples in a matrix
y = zeros(Mr,Nr);
for  i = 1:length(r)
   ri = r(i);   vi = v(i)/(10^9);  % convert m/sec to km/usec
   f_doppler = 2*(vi/c)*fc;
 for j = 1:length(g)
   r_at_T_0 = ri - vi*T_0(j);    %-- toward rcvr: f_doppler > 0
   tau = r_at_T_0./(c/2+vi);   tmax = tau + T_p;
   if tau >= T_out(2) | tmax <= T_out(1)
      disp('COMPLETELY OUT OF range window'),  ri=ri, i
   else
      t_vals = t_y - tau;
      n_out = find( t_vals >= 0  &  t_vals < T_p );
      if tau < T_out(1)
         disp('BEFORE range window'),  ri
      end
      if tmax > T_out(2)
         disp('AFTER range window'),  ri
      end
      if length(n_out) < 1
         disp('NO OVERLAP ???'),  keyboard
      else
%--------------------------------
  y(n_out,j) = y(n_out,j) + ...
          ( a(i) * g(j) * exp( -J*2*pi*fc*tau ) ) ...
          .* [ exp( J*2*pi*f_doppler*t_y(n_out) ) ]  ...
          .* [ exp( J*polyval(q,t_vals(n_out)) )  ];
%--------------------------------
      end
   end
 end
end
```

TEST_RADAR.M

```
%
%   EXAMPLE of calling the function radar()
%     make one radar return for a burst of LFM pulses
%
clear
format compact
T = 10     % microsec
W = 5      % MHz
fs = 10    % MHz, oversample by 2
s = chirp(T,W,fs/W);

Np = 7;               % 7 pulses
jkl = 0:(Np-1);
T_0 = 200*jkl;        % in usec
g = ones(1,Np);       % gains
T_out = [100 150];    % in usec
T_ref = 0;            % why use anything else?
fc = 10000;           % 10 GHz

Ntargets = 1;
ranges = 20;     % in km ???
amps = 1;
vels = 30;       % in m/sec

y = radar(s,fs,T_0,g,T_out,T_ref,fc,ranges,amps,vels);
```

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{Speech Modeling}}$

Speech Modeling

One of the most fruitful areas of application of digital signal processing is in the processing of speech signals. The basis for most digital speech processing algorithms is a discrete-time system model for the the production of the speech waveform. There are many useful models that have been used as the basis for speech synthesis, speech coding and speech recognition algorithms. One such model is depicted in Figure 1. The purpose of this set of projects is to show how such a model can be related to a specific speech waveform.

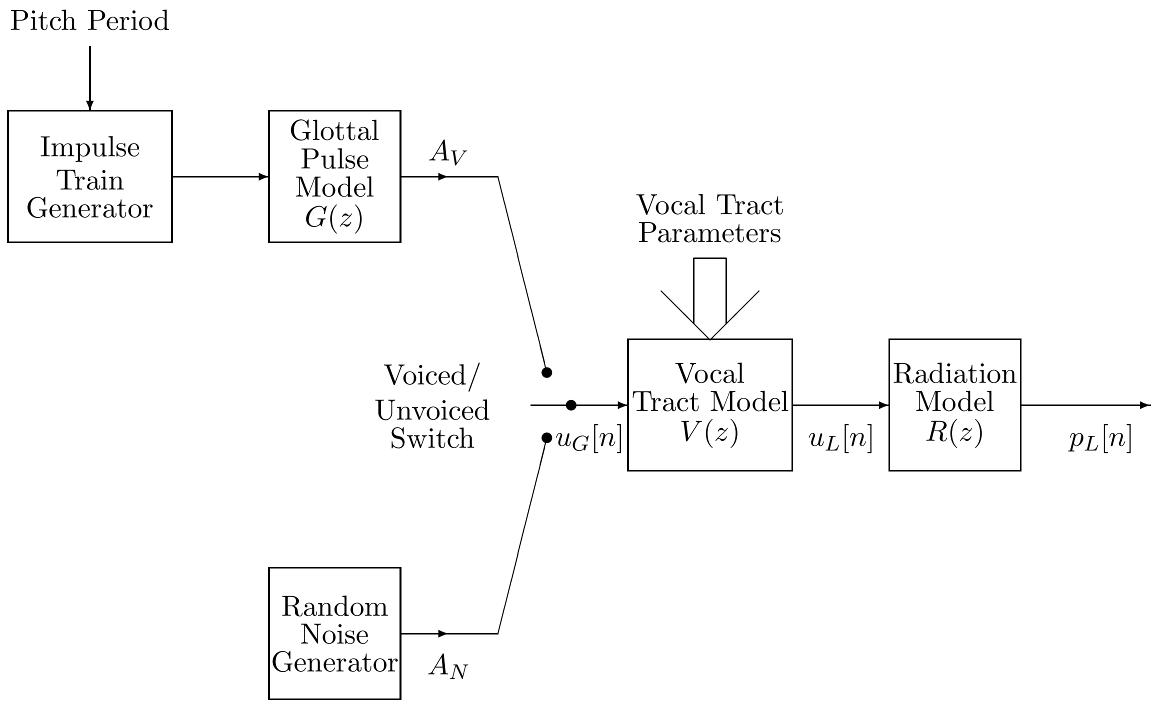


Figure 1: Discrete-time system model for speech production.

Background Reading

The following references provide appropriate background for this set of projects.

(a) G. Fant, *Acoustic Theory of Speech Production*, Mouton, The Hague, 1970.

(b) J. L. Flanagan, *Speech Analysis, Synthesis, and Perception* Second Edition, Springer-Verlag, New York, 1972.

(c) D. O'Shaugnessy, *Speech Communication Human and Machine*, Addison-Wesley Publishing Co., Reading, MA, 1987.

(d) T. Parsons, *Voice and Speech Processing*, McGraw-Hill Book Co., New York, 1986.

(e) L. R. Rabiner and R. W. Schafer, *Digital Processing of Speech Signals*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1978.

(f) A. E. Rosenberg, "Effect of glottal pulse shape on the quality of natural vowels", *Journal of Acoustical Society of America*, Vol. 49, No. 2, pp. 583-590, February, 1971.

Project 1: Speech Segmentation

Project Description

A fundamental problem in speech processing is the segmentation and phonetic labeling of speech waveforms. In general, this is very difficult to do automatically, and even computer-aided segmentation by humans requires a great deal of skill on the part of the analyst. While MATLAB is far from an ideal tool for this purpose, plotting functions such as `plot( )`, `subplot( )`, and `implot( )` can be used for plotting speech waveforms. Also useful in this project and in the other speech processing projects are two M-files called `readblock( )` and `writeblock( )`. These functions are used to read segments of speech from a disk file. They are essential, because, as you will quickly find, PC-MATLAB has a limitation of 8188 on the length of its vectors. Thus, in general you will not be able to read the entire speech file into the MATLAB environment, and you will have to work on it in pieces. (You will also find that the basic plotting function `plot( )` has a limitation of 4094.)

Hints

The problem in this part of the project is to segment and label the waveform in the file `S5.SP`, which is available in the speech data base that is provided with this book. This file is a sampled (sampling rate 8 kHz) waveform of an utterance of the sentence *Oak is strong and also gives shade.*

Exercise 1.1: Phonetic Representation of Text

First write out a phonetic representation of the utterance *Oak is strong and also gives shade* using the symbols defined in any major college level dictionary.

Exercise 1.2: Phonetic Labeling using Waveform Plots

Use the plotting features of MATLAB to examine the waveform in the file `S5.SP`, and make decisions on where each phoneme of the utterance begins and ends. Be alert for phonemes that are missing or barely realized in the waveform. There may be a period of "silence" or "noise" at the beginning and end of the file. Be sure to mark the beginning and end of these intervals too. Make a table showing the phonemes and the starting and ending samples for each.

Project 2: Glottal Pulse Models

Project Description

The model of Figure 1 often underlies our thinking about the speech waveform, and in some cases, such a system is explicitly used as an speech synthesizer. In this part, we will study the part labeled Glottal Pulse Model $G(z)$ in Figure 1.

Hints

In speech production, the excitation for voiced speech is a result of the quasi-periodic opening and closing of the opening between the vocal cords (the glottis). This is modeled in Figure 1 by the combination of the impulse train generator and the glottal pulse model filter. The shape of the pulse affects the magnitude and phase of the spectrum of the synthetic speech output of the the model.

Exercise 2.1: The Exponential Model

A simple model that we will call the *exponential model* is represented by

$$G(z) = \frac{az^{-1}}{(1 - az^{-1})^2} \tag{2.1}$$

Write an M-file to generate `Npts` samples of the corresponding glottal pulse waveform $g[n]$ and also compute the frequency response of the glottal pulse model. The calling sequence for this function should be

```
[gE,GE,W]=glottalE(a,Npts,Nfreq)
```

where `gE` is the exponential glottal waveform vector of length `Npts`, `GE` is the frequency response of the exponential glottal model at the `Nfreq` frequencies `W` between 0 and π radians. You will use this function later.

Exercise 2.2: The Rosenberg Model

Rosenberg [6] used inverse filtering to extract the glottal waveform from speech. Based on his experimental results, he devised a model for use in speech synthesis, which is given by the equation

$$g_R[n] = \begin{cases} \frac{1}{2}[1 - \cos(\pi n/N_1)] & 0 \le n \le N_1 \\ \cos[\pi(n - N_1)/(2N_2)] & N_1 \le n \le N_1 + N_2 \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

This model incorporates most of the important features of the time waveform of glottal waves estimated by inverse filtering and by high-speed motion pictures [2].

Write an M-file to generate all $N_1 + N_2 + 1$ samples of a Rosenberg glottal pulse with parameters N_1 and N_2, and compute the frequency response of the Rosenberg glottal pulse model. The calling sequence for this function should be

```
[gR,GR,W]=glottalR(N1,N2,Nfreq)
```

where `gR` is the Rosenberg glottal waveform vector of length `N1+N2+1`, `GR` is the frequency response of the glottal model at the `Nfreq` frequencies `W` between 0 and π radians.

Exercise 2.3: Comparison of Glottal Pulse Models

In this exercise you will compare three glottal pulse models.

(a) First, use the M-files from Exercises 2.1 and 2.2 to compute `Npts=51` samples of the exponential glottal pulse `g` for `a=0.91` and compute the Rosenberg pulse `gR` for the parameters `N1=40` and `N2=10`.

(b) Also compute a new pulse `gRflip` by time reversing `gR` using the MATLAB function `fliplr( )` for row vectors or `flipud( )` for column vectors. This has the effect of creating a new causal pulse of the form

$$g_{Rflip}[n] = g_R[-(n - N_1 - N_2)] \tag{2.3}$$

Determine the relationship between $G_{Rflip}(e^{j\omega})$, the Fourier transform of $g_{Rflip}[n]$, and $G_R(e^{j\omega})$, the Fourier transform of $g_R[n]$.

(c) Now plot all three of these 51-point vectors on the same graph using `plot( )`. Normalize the exponential glottal pulse by dividing by its maximum value before plotting. Also plot the frequency response magnitude in dB for all three pulses on the same graph.

Experiment with the parameters of the models to see how the time-domain wave shapes affect the frequency response.

(d) The exponential model has a zero at $z = 0$ and a double pole at $z = a$. For the parameters `N1=40` and `N2=10`, use the MATLAB function `roots( )` to find the zeros of the z-transform of the Rosenberg model and also the zeros of the flipped Rosenberg model. Plot them using the MATLAB function `zplane( )`. Note that the Rosenberg model has all its zeros outside the unit circle (except one at $z = 0$). Such a system is called a *maximum-phase* system. The flipped Rosenberg model, however, should be found to have all its zeros inside the unit circle, and thus, it is a *minimum-phase* system. Show that in general, if a signal is maximum-phase, then flipping it as in Eq. (2.3) produces a minimum-phase signal and vice-versa.

Project 3: Lossless Tube Vocal Tract Models

Project Description

One approach to modeling sound transmission in the vocal tract is through the use of concatenated lossless acoustic tubes as depicted in Figure 2.

Using the acoustic theory of speech production [1,2,5], it can be shown that the lossless assumption and the regular structure leads to wave simple equations and simple boundary conditions at the tube junctions so that a solution for the transmission properties of the model is relatively straightforward, and can be interpreted as in Figure 3(a) where $\tau = \Delta x/c$ is the one-way propagation delay of the sections. For sampled signals with sampling period $T = 2\tau$, the structure of Figure 3(a) (or equivalently Fig. 2) implies a corresponding discrete-time lattice filter as shown in Figure 3(b) or 3(c).[5]

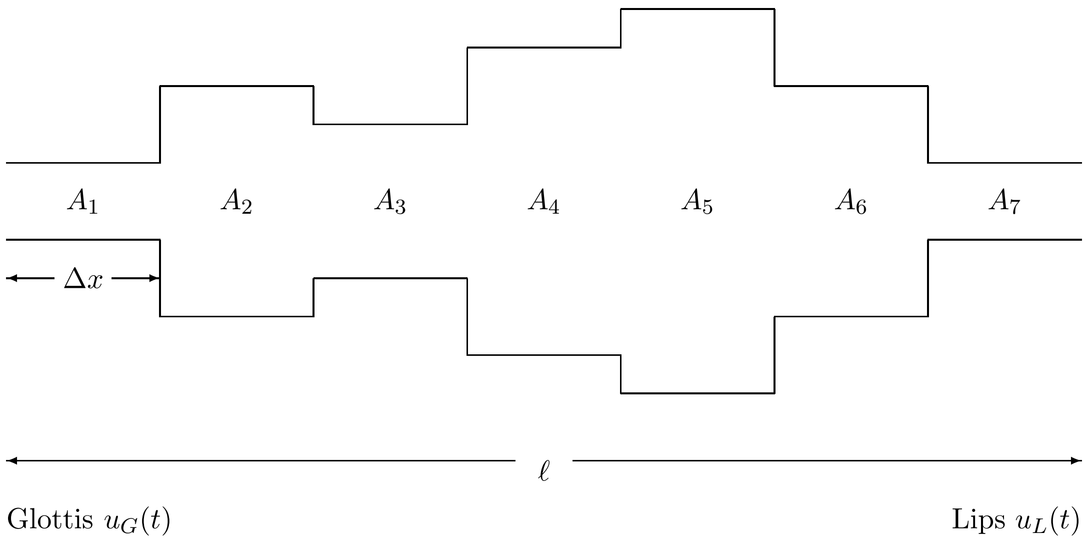


Figure 2: Concatenation of (N=7) lossless acoustic tubes of equal length as a model of sound transmission in the vocal tract.

Hints

Lossless tube models are useful for gaining insight into the acoustic theory of speech production, and they are also useful for implementing speech synthesis systems. It is shown in [5] that if $r_G = 1$, the discrete-time vocal tract model consisting of a concatenation of N lossless tubes of equal length has system function

$$V(z) = \frac{\prod_{k=1}^{N}(1 + r_k)z^{-N/2}}{D(z)} \tag{3.1}$$

The denominator polynomial $D(z)$ in Eq. (3.1) satisfies the polynomial recursion [5]

$$\begin{aligned} D_0(z) &= 1 \\ D_k(z) &= D_{k-1}(z) + r_k z^{-k} D_{k-1}(z^{-1}) \qquad k = 1, 2, \ldots, N \\ D(z) &= D_N(z) \end{aligned} \tag{3.2}$$

where the r_k's in Eq. (3.2) are the reflection coefficients at the tube junctions,

$$r_k = \frac{A_{k+1} - A_k}{A_{k+1} + A_k} \tag{3.3}$$

In deriving the recursion in Eq. (3.2), it was assumed that there were no losses at the glottal end ($r_G = 1$) and that all the losses are introduced at the lip end through the reflection coefficient

$$r_N = r_L = \frac{A_{N+1} - A_N}{A_{N+1} + A_N} \tag{3.4}$$

Figure 3: (a) Signal flow graph for lossless tube model ($N = 3$); (b) equivalent discrete-time system; (c) equivalent discrete-time system using only integer delays.

where A_{N+1} is the area of an impedance-matched tube that can be chosen to introduce a loss in the system.

Suppose that we have a set of areas for a lossless tube model, and we wish to obtain the system function for the system so that we can use the MATLAB `filter( )` function to implement the model; i.e., we want to obtain the system function of Eq. (3.1) in the form

$$V(z) = \frac{G}{D(z)} = \frac{G}{1 - \sum_{k=1}^{N} \alpha_k z^{-k}} \tag{3.5}$$

(Note that we have dropped the delay of $N/2$ samples, which is inconsequential for use in synthesis.) The following MATLAB M-file implements Eqs. (3.2) and (3.3); i.e., it takes an array of tube areas and a reflection coefficient at the lip end and finds the parameters of Eq. (3.5) along with the reflection coefficients.

```
function        [r,D,G]=AtoV(A,rN)
%       function to find reflection coefficients
%       and system function denominator for
%       lossless tube models.
%               [r,D,G]=AtoV(A,rN)
%               rN = reflection coefficient at lips (abs value < 1)
%               A = array of areas
%               D = array of denominator coefficients
%               G = numerator of transfer function
%               r = corresponding reflection coefficients
%       assumes no losses at the glottis end (rG=1).
[M,N]=size(A);
if(M~=1) A=A'; end      %make row vector
N=length(A);
r=[];
for m=1:N-1
        r=[r (A(m+1)-A(m))/(A(m+1)+A(m))];
end
r=[r rN];
D=[1];
G=1;
for m=1:N
        G=G*(1+r(m));
        D=[D 0] + r(m).*[0 fliplr(D)];
end
```

As test data for this project, the following area functions were estimated from data obtained by Fant.[1]

Section	1	2	3	4	5	6	7	8	9	10
vowel /a/	1.6	2.6	.65	1.6	2.6	4	6.5	8	7	5
vowel /i/	2.6	8	10.5	10.5	8	4	.65	.65	1.3	3.2

Exercise 3.1: Frequency Response and Pole/Zero Plot

(a) Use the M-file `AtoV( )` to obtain the denominator $D(z)$ of the vocal tract system function, and make plots of the frequency response for both area functions for `rN=.71` and also for the totally lossless case $rN = 1$.

(b) Factor the polynomials $D(z)$ and plot the poles in the z-plane using `zplane( )`. Convert the angles of the roots to analog frequencies corresponding to a sampling rate of $1/T = 10000$ samples/sec, and compare to the formant frequencies expected for these vowels.[1,2,5] For this sampling rate, what is the effective length of the vocal tract in cm?

Exercise 3.2: Finding the Model from the System Function

The inverse problem arises when we want to obtain the areas and reflection coefficients for a lossless tube model given the system function in the form of Eq. (3.5). We know that the denominator of the system function, $D(z)$, satisfies Eqs. (3.2). In this part we will use Eqs. (3.2) to develop an algorithm for finding the reflection coefficients and the areas of a lossless tube model having a given system function.

(a) Show that r_N is equal to the coefficient of z^{-N} in the denominator of $V(z)$; i.e. $r_N = -\alpha_N$.

(b) Use Eqs. (3.2) to show that

$$D_{k-1}(z) = \frac{D_k(z) - r_k z^{-k} D_k(z^{-1})}{1 - r_k^2} \qquad k = N, N-1, \ldots, 2$$

(c) How would you find r_{k-1} from $D_{k-1}(z)$?

(d) Using the results of parts (a), (b), and (c), state an algorithm for finding all of the reflection coefficients $r_k,\ k = 1, 2, \ldots, N$ and all of the tube areas $A_k,\ k = 1, 2, \ldots, N$. Are the A_k's unique? Write a MATLAB function to implement your algorithm for converting from $D(z)$ to reflection coefficients and areas. This M-file should as defined by the following:

```
function        [r,A]=VtoA(D,A1)
%       function to find reflection coefficients
%       and tube areas for lossless tube models.
%        [r,A]=VtoA(D,A1)
%               A1 = arbitrary area of first section
%               D = array of denominator coefficients
%               A = array of areas for lossless tube model
%               r = corresponding reflection coefficients
%       assumes no losses at the glottis end (rG=1).
```

For the vowel /a/, the denominator of the 10th-order model should be (to 4 digit accuracy)

$$\begin{aligned} D(z) \;=\; & 1 - 0.0460z^{-1} - 0.6232z^{-2} + 0.3814z^{-3} + 0.2443z^{-4} + 0.1973z^{-5} \\ & + 0.2873z^{-6} + 0.3655z^{-7} - 0.4806z^{-8} - 0.1153z^{-9} + 0.7100z^{-10} \end{aligned}$$

Use your MATLAB program to find the corresponding reflection coefficients and tube areas and compare to the data for the vowel /a/ in the table above.

Project 4: Vowel Synthesis

Project Description

For voiced speech, the speech model of Fig. 1 can be simplified to the system of Fig. 4. The exitation signal $e[n]$ is a quasi-periodic impulse train and the glottal pulse model could be either the exponential or the Rosenberg pulse. The vocal tract model could be a lattice filter of the form of Fig. 3(c) or it could be an equivalent direct form difference equation as implemented by MATLAB.

Hints

In this project we will use the MATLAB `filter( )` and `conv( )` functions to implement the system of Fig. 4 and thereby synthesize periodic vowel sounds.

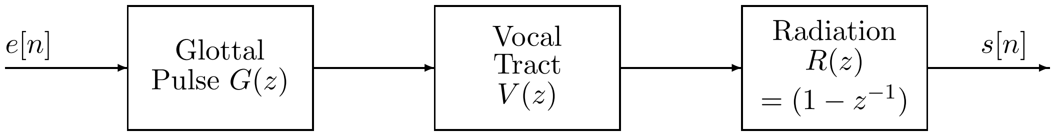


Figure 4: Simplified model for synthesizing voiced speech.

Exercise 4.1: Periodic Vowel Synthesis

Assume a sampling rate of 10000 samples/sec. Create a periodic impulse train vector `e` of length 1000 samples with period corresponding to a fundamental frequency of 100 Hz. You may find the functions `ones( )` and `zerofill( )` useful for this. Then use combinations of `filter( )` and `conv( )` to implement the system of Fig. 4.

Use the excitation `e` and radiation system $R(z) = (1 - z^{-1})$ to synthesize speech for both area functions given above, and for all three glottal pulses studied in Project 2. Use `subplot( )` and `plot( )` to make a plot comparing 1000 samples of the synthetic speech outputs for the exponential glottal pulse and the Rosenberg minimum-phase pulse. Make another plot comparing the outputs for the two Rosenberg pulses.

Exercise 4.2: Frequency Response of Vowel Synthesizer

Plot the frequency response of the overall system with system function $H(z) = G(z)V(z)R(z)$ for the case of the Rosenberg glottal pulse, $R(z) = (1 - z^{-1})$, and vocal tract response for the vowel /a/.

Exercise 4.3: Listening to the Output

If D/A facilities are available on your computer, use the m-file `writeblock()` to create a file of length corresponding to 0.5 sec. duration in the proper binary format and play it out through the D/A system. Be sure that you scale the samples appropriately and convert them to integers before writing the file. Does the synthetic speech sound like the desired vowels?

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{Speech Quantization}}$

Speech Quantization

Overview

Sampling and quantization (or A-to-D conversion) of speech waveforms is important in digital speech processing because it is the first step in any digital speech processing system, and because one of the basic problems of speech processing is digital coding of the speech signal for digital transmission and/or storage. Sampling and quantization of signals is generally implemented by a system of the form of Figure 1.

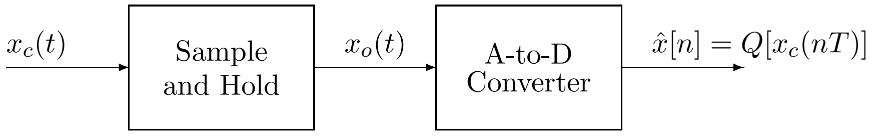


Figure 1: Representation of hardware for sampling and quantization of speech signals.

In a hardware realization, the sample-and-hold circuit samples the input continuous-time signal and holds the value constant during the sampling period T. This gives a constant signal at the input of the A-to-D converter, whose purpose is to decide which of its quantization levels is closest to the input sample value. The A-to-D converter emits a digital code corresponding to that level. Normally the digital code is assigned according to a convenient binary number system such as twos-complement so that the binary numbers can be taken as numerical representations of the sample values.

An equivalent representation of sampling and quantization is depicted in Fig. 2.

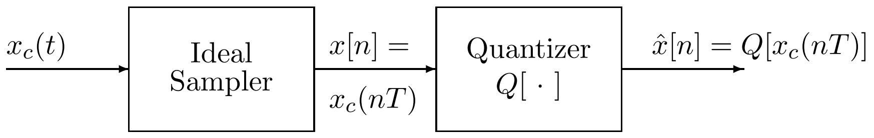


Figure 2: Representation of sampling and quantization that facilitates analysis and simulation.

This representation is convenient because it separates the sampling and quantization into two independent operations. The following projects focus solely on quantization. It will be assumed that the speech signal has been lowpass filtered and sampled at a high enough sampling rate to avoid significant aliasing distortion. These projects seek to explore some of the fundamentals of quantization of speech signal waveforms.

Background Reading

The following references provide appropriate background for this set of projects.

1. N. S. Jayant and P. Noll, *Digital Coding of Waveforms*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984.

2. L. R. Rabiner and R. W. Schafer, *Digital Processing of Speech Signals*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1978.

Project 1: Speech Properties

Project Description

In this project you will use MATLAB tools to examine a particular speech waveform and verify some fundamental statistical properties of speech signals that are important for quantization.

Hints

The speech files `S1.SP` – `S6.SP` are available on the distribution disk. The files were sampled with sampling rate 8000 samples per second and originally quantized to 12-bits. Subsequently, the samples were multiplied by 16 to raise the amplitude levels to just under 32767, the maximum value for a 16-bit integer. Thus, these files are 12-bit samples pretending to be 16-bit samples. This will generally not be a problem in this project.

Exercise 1.1: Speech Waveform Plotting

First, use the MATLAB function `readblock( )` to read 8000 samples, starting at sample number 1200, from the file S5.SP . Plot all 8000 samples with 2000 samples/line using the plotting function `striplot( )`.

Exercise 1.2: Statistical Analysis

Compute the minimum, maximum, average, and mean-squared value of the 8000 samples from the file `S5.SP`. Use the MATLAB function `hist( )` to plot a histogram of the 8000 samples. Experiment with the number and location of the histogram bins to obtain a useful plot. The histogram should show that the small samples are more probable than large samples in the speech waveform. Is this consistent with what you see in the waveform plot? See refs. [1,2] for discussions of continuous probability density function models for the distribution of speech amplitudes.

Exercise 1.3: Spectral Analysis

Use the m-file `pspect()` to compute an estimate of the long-time average power spectrum of speech using the 8000 samples from file `S5.SP`. Plot the spectrum in dB units labeling the frequency axis appropriately. Save this spectrum estimate for use in Project 2.

Project 2: Uniform Quantization

Project Description

Figure 3 shows the input/output relation for a 3-bit uniform quantizer in which the input samples are *rounded* to the nearest quantization level. In discussing the effects of

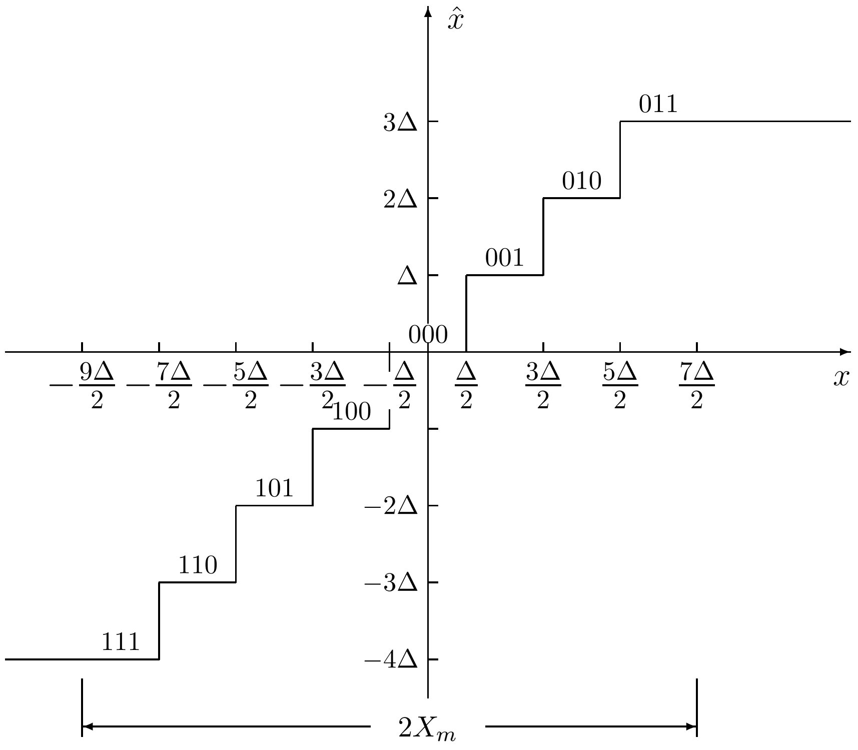


Figure 3: Input/output characteristic for a 3-bit uniform quantizer.

quantization it is useful to define the quantization error as

$$e[n] = \hat{x}[n] - x[n] \tag{2.1}$$

This definition leads to the additive noise model for quantization that is depicted in Fig. 4. If the signal sample $x[n]$ remains in the prescribed peak-to-peak range of the quantizer,

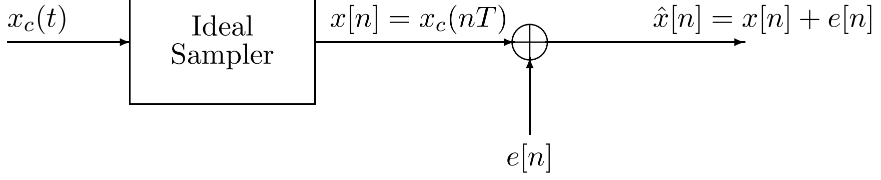


Figure 4: Additive noise model for sampling and quantization.

then it is clear that the quantization error samples satisfy

$$-\Delta/2 \leq e[n] < \Delta/2 \tag{2.2}$$

Furthermore, speech is a complicated signal that fluctuates rapidly among the quantization levels, and if Δ is small enough, the amplitude of the signal is likely to traverse many quantization steps in one sample time. Under these conditions, it is found that the quantization error sequence is well described by the following model:

1. The error sequence $e[n]$ is uncorrelated with the unquantized sequence $x[n]$.
2. The error sequence has the properties of white noise; i.e., it has a flat power spectrum, and the error samples are uncorrelated with one another.
3. The probability distribution of the error samples is uniform over the range of quantization error amplitudes.

These assumptions will be tested in this project.

Hints

In this project, you will write a MATLAB M-file to implement a uniform quantizer and perform several experiments with it. The implementation of the quantizer in MATLAB is quite simple. Assuming that the maximum absolute value of the signal samples is less than or equal to 32767, the following MATLAB code implements an 8-bit uniform quantizer with rounding.

```
s=readblock('s5.sp',1,8000);
sh=256*round(s/256);
```

Be sure you understand how this works, and think about how to generalize it. (Note that the above code does not simulate clipping; i.e., samples that are greater than 32767 are not assigned the largest level less than 32767.)

Exercise 2.1: Uniform Quantizer in MATLAB

Write an M-file for a uniform quantizer. It should have the following calling sequence and parameters:

```
function        xq=quant(x,nbits)
%               uniform quantizer
%       xq=quant(x,nbits)
%               x=input column vector, max value 32767
%               nbits=number of bits in quantizer
%                       quantizer has 2^nbits levels
%               xq=quantized output vector
```

Exercise 2.2: Quantization Experiments

Use your program to quantize the input speech samples. Experiment with different numbers of bits for the quantizer. Compute the quantization error sequences for 10-, 8-, and 4-bit quantization. Use the program `striplot( )` to plot these error sequences. What are the important differences between them? Do they look like they fit the white noise model? Make histograms of the quantization noise samples. Do they seem to fit the uniform amplitude distribution model?

Exercise 2.3: Spectral Analysis of Quantization Noise

Use the m-file `pspect()` to compute the power spectrum of the quantization noise sequences for 10-, 8-, and 4-bits. Plot these spectra on the same plot as the power spectrum of the speech samples. (Remember: the power spectrum in dB is $10\log_{10}(P)$.) Do the noise spectra support the white noise assumption? What is the approximate difference in dB between the noise spectra for 10- and 8-bits?

Exercise 2.4: Quantization by Truncation

MATLAB has other quantization functions called `ceil()` and `floor()`. Replace the `round()` function with either `ceil()` and `floor()` and repeat Exercises 2.1-2.3.

Project 3: μ-Law Companding

Project Description

One of the problems with uniform quantization is that the maximum size of the quantization errors is the same no matter how big or small the samples are. For a coarse quantizer, low level fricatives, etc. may disappear completely because their amplitude is below the minimum step-size. μ-law compression/expansion is a way to obtain quantization errors that are effectively proportional to the size of the sample.

Hints

A convenient way of describing μ-law quantization is depicted in Figure 5. In this representation, a μ-law compressor preceeds a uniform quantizer. The combination (inside the dotted box) is a μ-law quantizer.

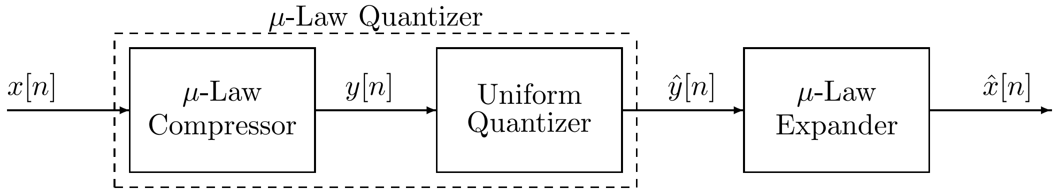


Figure 5: Representation of μ-law quantization.

The μ-law compressor is defined by the equation

$$y[n] = X_{max}\frac{\log\left[1+\mu\frac{|x[n]|}{X_{max}}\right]}{\log(1+\mu)}\cdot\text{sign}(x[n]) \tag{3.1}$$

The μ-law compressor is discussed in detail in [1,2].

The following M-file implements the μ-law compressor on a signal vector whose maximum value is assumed to be 32767:

```
function        y=mulaw(x,mu)
%               function for mu-law compression for
%               signals with maximum value of 32767
%       y=mulaw(x,mu)
%               x=input signal vector, column vector with max value 32767
%               mu=compression parameter (mu=255 used for telephony)
sign=ones(length(x),1);
sign(find(x<0))=-ones(find(x<0));
y=(32767/log(1+mu))*log(1+mu*abs(x)/32767).*sign;
```

Note the use of the `find( )` function to locate the negative samples.

Exercise 3.1: μ-Law Compressor

This exercise is concerned with the μ-law compressor and its inverse.

(a) Create a linearly increasing input vector (0:500:32000) and use it with the above function `mulaw( )` to plot the μ-law characteristic for $\mu = 100$, 255, and 500 all on the same plot. $\mu = 255$ is a standard value used in telephony.

(b) Using the segment of speech from file `S5.SP` and a value of $\mu = 255$, plot the output waveform $y[n]$ of the μ-law compressor. Observe how the low amplitude samples are increased in magnitude. Plot a histogram of the output samples.

(c) To implement the system of Figure 5, you will need to write an M-file for the inverse of the μ-law compressor. This M-file should have the following calling sequence and parameters:

```
function        x=mulawinv(y,mu)
%       function for inverse mulaw for Xmax=32767
%       x=mulawinv(y,mu)
%               y=input column vector
%               mu=mulaw compression parameter
%               x=expanded output vector
```

Use the technique used in `mulaw( )` to set the signs of the samples. Test the inverse system by applying it directly to the output of `mulaw()` without quantization.

Exercise 3.2: μ-Law Quantization

The MATLAB statement `yh=quant(mulaw(x,255),6);` implements a six-bit μ-law quantizer. That is, it is the compressed samples that would be represented by six bits. When the samples are used in a signal processing computation or when a continuous-time signal is reconstructed, the samples must be expanded. Hence, the quantization errors will likewise be expanded, so that to determine the quantization error, it is necessary to compare the output of the inverse system to the original samples. That is the quantization error would be `e=mulawinv(yh,255)-x;`. With this in mind, repeat all the exercises of Project 2 for the system of Figure 5.

Project 4: Signal-to-Noise Ratios

Project Description

A convenient way of comparing quantizers is to compute the ratio of signal power to quantization noise power. For experiments in MATLAB, a convenient definition of SNR is

$$SNR = 10\log\left(\frac{\sum_{n=0}^{L-1}(x[n])^2}{\sum_{n=0}^{L-1}(\hat{x}[n]-x[n])^2}\right) \tag{4.1}$$

Note that the division by L required for averaging cancels in the numerator and denominator.

Hints

Under the assumptions given in Project 3, it can be shown that the signal-to-noise ratio for a uniform quantizer with 2^{B+1} levels (B bits plus sign) has the form [1,2]

$$SNR = 6B + 10.8 - 20\log_{10}\left(\frac{X_m}{\sigma_x}\right) \tag{4.2}$$

where X_m is the clipping level of the quantizer (in our case 32767) and σ_x is the rms value of the input signal amplitude. Thus, Eq. (4.2) shows that the signal-to-noise ratio increases 6 dB per bit added to the quantizer word length. Furthermore, Eq. (4.2) shows that if the signal level is decreased by a factor of two, the signal-to-noise ratio decreases by 6 dB.

Exercise 4.1: Signal-to-Noise Computation

Write an M-file to compute the signal-to-noise ratio. Its calling sequence and parameters should be:

```
function        [s_n_r,e]=snr(xh,x);
%       function for computing signal-to-noise ratio
%       [s_n_r,e]=snr(xh,x)
%               xh=quantized signal
%               x=unquantized signal
%               e=quantization error signal (optional)
%               s_n_r=snr in dB
```

Use your SNR function to compute the SNRs for uniform quantization with 8- and 9-bits. Do the results differ by the expected amount?

Exercise 4.2: Comparison of Uniform and μ-Law Quantization

An important consideration in quantizing speech is that signal levels can vary with speakers and with transmission/recording conditions. The following M-file compares uniform and μ-law quantization for fixed quantizers with inputs of decreasing amplitude (by factors of 2).

Using the M-files that were written in Projects 2 and 3 and the following M-file, make plots for 10-, 8-, and 6-bits with $\mu = 255$ over a range of 10 factors of two. (By saving the arrays `snrunif` and `snrmu` for each quantization, the curves can all be plotted on the same graph using `plot()`.)

```
function        [snrunif,snrmu]=qplot(s,nbits,mu,ncases)
%       function for plotting signal-to-noise ratio of quantizers
%       [snrunif,snrmu]=qplot(s,nbits,mu,ncases)
%               s=input test signal
%               nbits=number of bits in quantizer
%               mu=mu-law compression parameter
%               ncases=number of cases to plot
%               snrunif=snr of uniform quantizer for the ncases
%               snrmu=snr of mu-law quantizer
%
snrunif=[];
snrmu=[];
x=s;
for i=1:ncases
sh=quant(x,nbits);
snrunif=[snrunif; snr(sh,x)];
y=mulaw(x,mu);
yh=quant(y,nbits);
xh=mulawinv(yh,mu);
snrmu=[snrmu; snr(xh,x)];
x=x/2;
end
plot(1:ncases,snrunif,1:ncases,snrmu)
title(['SNR for ',num2str(nbits),'-bit Uniform and ',num2str(mu),..
'-Law Quantizers'])
xlabel('power of 2 divisor');ylabel('SNR in dB')
pause
```

Your plots should show that the μ-law quantizer maintains a constant signal-to-noise ratio over a input amplitude range of about 64:1. How many bits are required for a uniform quantizer to maintain at least the same signal-to-noise ratio as does a 6-bit μ-law quantizer over the same range?

Project 5: Listening to Quantized Speech

If your computer has D-to-A capability, you can listen to the quantized speech. This will be a little awkward since PC-MATLAB will only allow you to make variables of length 8188 or less. You will have to bring in the file in pieces and quantize it that way. With all variables cleared out the following should work:

```
s=readblock('s5.sp',1,8000);
writeblock('s5h.sp',s)
```

```
s1h=quant(s,4);
s=readblock('s5.sp',8001,8000);
writeblock('s5h.sp',s,'a')
s2h=quant(s,4);
s=readblock('s5.sp',16001,8000);
writeblock('s5h.sp',s,'a')
s3h=quant(s,4);
s=zeros(8000,1);
writeblock('s5h.sp',s,'a')
writeblock('s5h.sp',s1h,'a')
writeblock('s5h.sp',s2h,'a')
writeblock('s5h.sp',s3h,'a')
```

This may take quite a while, so be patient. It will give you the original unquantized sentence, followed by 1 sec of silence, followed by the quantized speech. You can substitute any quantizer for the 4-bit uniform quantizer used above. You might also wish to output the quantization error sequence for listening.

Computer-Based Exercises

for

Signal Processing

Linear Prediction of Speech

Linear Prediction of Speech

Overview

In this project you will study various aspects of the use of linear prediction in speech processing. This project follows closely the notation and point of view of [**?**], where speech is assumed to be the output of the linear system model shown in Fig. 1.

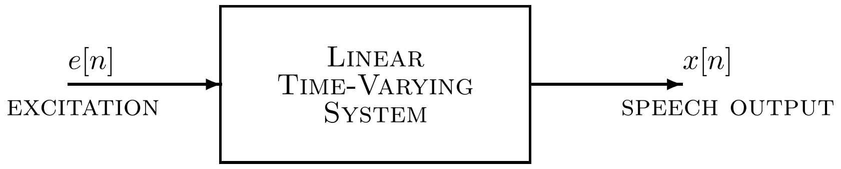


Figure 1: Speech model as time-varying linear system.

In this figure, the input $e[n]$ is ideally either white noise or a quasi-periodic train of impulses. The linear system in Fig. 1 is assumed to be slowly time-varying such that over short time intervals it can be described by the all-pole system function

$$H(z) = \frac{G}{1 + \sum_{k=1}^{P} \alpha_k z^{-k}} \tag{0.1}$$

It is easily seen that for such a system, the input and output are related by a difference equation of the form

$$x[n] = -\sum_{k=1}^{P} \alpha_k x[n-k] + Ge[n] \tag{0.2}$$

Note: the minus sign in front of the summation is consistent with the MATLAB `filter` function, but opposite from the notation in [**?**].

Linear predictive (LP) analysis is a set of techniques aimed at finding the set of prediction coefficients $\{a_k\}$ that minimize the mean-squared prediction error between a signal $x[n]$ and a predicted signal based on a linear combination of past samples; i.e.,

$$\left\langle (f[n])^2 \right\rangle = \left\langle \; (x[n] + \sum_{k=1}^{P} a_k x[n-k])^2 \; \right\rangle \tag{0.3}$$

where $\langle \cdot \rangle$ represents averaging over a finite range of values of n. It can be shown that using one method of averaging called the *autocorrelation method*, the optimum predictor coefficients $\{a_k\}$ satisfy a set of linear equations of the form

$$\mathbf{Ra} = -\mathbf{r} \tag{0.4}$$

where $\mathbf{R}$ is a $P \times P$ Toeplitz matrix made up of values of the autocorrelation sequence for $x[n]$, $\mathbf{a}$ is a $P \times 1$ vector of prediction coefficients, and $\mathbf{r}$ is a $P \times 1$ vector of autocorrelation values [**?**].

In using LP techniques for speech analysis, we make the assumption that the predictor coefficients $\{a_k\}$ are identical to the parameters $\{\alpha_k\}$ of the speech model. Then, by definition of the model, we see that the output of the prediction error filter with system function

$$A(z) = 1 + \sum_{k=1}^{P} a_k z^{-k} \tag{0.5}$$

is

$$f[n] = x[n] + \sum_{k=1}^{P} a_k x[n-k] \equiv Ge[n] \tag{0.6}$$

i.e., the excitation of the model is defined to be the input that produces the given output $x[n]$ for the prediction coefficients estimated from $x[n]$. The gain constant G is therefore simply the constant that is required so that $e[n]$ has unit mean-squared value, and is readily found from the autocorrelation values used in the computation of the prediction coefficients [?].

It can be shown that because of the special properties of the LP equations (0.4), an efficient method called the Levinson Recursion[?] exists for solving the equations for the predictor parameters. However, for purposes of these exercises it is most convenient to use the general MATLAB matrix functions. Specifically, the following help lines are from an M-file `autolpc( )` from Appendix A that implements the autocorrelation method of LP analysis:

```
function   [A, G, r, a] = autolpc(x, p)
%AUTOLPC      Autocorrelation Method for LPC
%   Usage: [A, G, r, a] = autolpc(x, p)
%       x : vector of input samples
%       p : LPC model order
%       A : prediction error filter, (A = [1; -a])
%       G : rms prediction error
%       r : autocorrelation coefficients: lag = 0:p
%       a : predictor coefficients (without minus sign)
%--- see also ATOK, KTOA
```

Project 1: Basic Linear Prediction

The file `S5.MAT` contains the utterance *Oak is strong and also gives shade* sampled at 8 kHz. The phoneme /sh/ in *shade* begins at about sample number 15500 and ends at about 16750, while the phoneme /a/ in *shade* begins at about 16750 and ends at about 18800.

Exercise 1.1: 12-th Order Predictor

Compute the predictor parameters of a 12th-order predictor for these two phonemes using a Hamming window of length 320 samples. For both phonemes, make a plot of the frequency response of the prediction error filter and the log magnitude response of the vocal tract model filter both on the same graph. Also use `zplane( )` to plot the zeros of the prediction error filter for both cases.
Hold onto the predictor information in both cases since you will need it for later exercises.

What do you observe about the relationship between the zeros of the prediction error filter and the following: (1) the poles of the vocal tract model filter; (2) the peaks in the frequency response of the vocal tract model filter; and (3) the dips in the frequency response of the prediction error filter?

Exercise 1.2: Frequency Response of Model

In the two cases, compute the Fourier transform of the windowed segment of speech, and plot its magnitude in dB on the same plot as the vocal tract model filter. Use the parameter `G` (available from the prediction analysis) in the numerator of the model filter system function to get the plots to line up.

What do you observe about the differences between the voiced /a/ and the unvoiced /sh/ phoneme?

Exercise 1.3: Vary the Model Order

If you have time it is instructive to look at other speech segments (frames) or to vary the window length and/or predictor order to observe the effects of these parameters. For example, compare the fit of the frequency response of the vocal tract model filter for $P = 8$, 10, and 12 to the short-time Fourier transform of the speech segment.

Exercise 1.4: Include Pre-Emphasis

Repeat Exercises 1.1 and 1.2 for the speech signal pre-emphasized with the 2-point FIR filter:

```
y = filter([1, -0.98], 1, s5)
```

Compare the results with and without pre-emphasis.

Exercise 1.5: Prediction Error Filtering

Now use the prediction error filters to compute the prediction error sequence $f[n]$ for both phonemes. Use `subplot(21*)` to make a two-panel subplot of the (unwindowed) speech segment on the top and the prediction error on the bottom part of the plot.

What do you observe about the differences in the two phonemes? Where do the peaks of the prediction error occur in the two cases?

Project 2: Line Spectrum Pair Representations

A useful transformation of the LP coefficients is the *line spectrum pair* (LSP) representation [?]. The line spectrum pair polynomials are defined by the equations

$$\begin{aligned} P(z) &= A(z) + z^{-(p+1)}A(z^{-1}) \\ Q(z) &= A(z) - z^{-(p+1)}A(z^{-1}) \end{aligned}$$

The LSP parameters are defined to be the angles of the roots of these two polynomials.

Exercise 2.1: M-file for Line Spectrum Pair

Write an M-file that converts the prediction error filter $A(z)$ to the LSP polynomials $P(z)$ and $Q(z)$. Its calling sequence should be as follows:

```
function [P, Q] = atolsp(A)
%ATOLSP   convert from prediction error filter to
%         line spectral pair (LSP) coefficients
%   Usage:
%          [P, Q] = atolsp(A)
%               A : column vector of prediction error filter
%         P and Q : column vectors of LSP polynomials
```

Exercise 2.2: Roots of LSP Polynomials

Use your M-file for both phonemes and use `zplane( )` to plot the roots of the two LSP polynomials with the roots of $P(z)$ plotted as x's and the roots of $Q(z)$ plotted as o's. Compare your plots to the plots of the zeros of the corresponding prediction error filters.

Observe the relationships among the roots of the two polynomials for each phoneme. In particular, note where all the roots lie radially, and note how the roots of the two polynomials interlace.

Project 3: Quantization of Parameters

In using linear prediction in speech coding, it is necessary to quantize the predictor parameters for digital coding. One possibility is to quantize the predictor parameters; i.e., the coefficients of the predictor polynomial $A(z)$. It is well known that these parameters are very sensitive to quantization. However, certain invertible nonlinear transformations of the predictor coefficients result in equivalent sets of parameters that are much more robust to quantization. One such set is the PARCOR parameters (or k-parameters), which are a by-product of the Levinson recursion method of solution of the LP equations [?]. Appendix A gives a pair of M-files called `atok( )` and `ktoa( )` which implement the transformation from predictor coefficients to PARCOR coefficients and the inverse respectively. The help lines for these two M-files are given below.

```
function k = atok(a)
%ATOK      converts AR polynomial to reflection coefficients
%   Usage:  K = atok(A)
%         where each column of A contains polynomial coeffs
%           and   "    "    of K contains PARCOR coeffs
%
%     If A is matrix, each column is processed separately.

function a = ktoa(k)
%KTOA    converts reflection coefficients to AR polynomial
%   Usage:  A = ktoa(K)
%         where each column of A contains polynomial coefficients
%           and   "    "    of K contains PARCOR coefficients
```

In the following exercises you will compare the effects of quantization on the predictor parameters and the PARCOR parameters. If you look at the coefficients of the polynomial $A(z)$, you will find that they are mostly less than or equal to one. The coefficients can be quantized to a fixed number of bits using the quantizer M-file `fxquant( )` given in Appendix A. For coefficients that are less than one, `fxquant( )` can be used directly; i.e., the statement

```
Ah = fxquant(A, 5, 'round', 'sat')
```

would quantize a coefficient `A` to five bits using rounding and saturation. (Of course, if the coefficient is less than one, no additional error results from the saturation mode.) If the coefficient is greater than one, but less than two, the statement

```
Ah = 2*fxquant(A/2, 7, 'round', 'sat')
```

would be used for 7-bit quantization, where the location of the binary point would have to be specified as one bit to the right of the sign bit.

Exercise 3.1: Round Prediction Coefficients

Round the coefficients of $A(z)$ for the phoneme /a/ to seven and four bits, respectively, using the method suggested above. Then make a plot of the the frequency responses of the original $1/A(z)$ and the two quantized vocal tract model filters all on the same graph. Also check the stability of the quantized vocal tract filters by finding the roots of the polynomial $A(z)$.

Exercise 3.2: Round PARCOR Coefficients

Now take the polynomial $A(z)$ for the phoneme /a/ and convert it to PARCOR parameters using the function `atok( )`.

(a) Round the PARCOR coefficients to seven and four bits as in Exercise 3.1. Then convert the quantized PARCOR coefficients back to prediction coefficients and make a plot of the frequency responses of the original and the two PARCOR-quantized vocal tract filters as in Exercise 3.1. Also check the stability of the quantized vocal tract filters.

(b) Compare the results of rounding $A\{a_k\}$ versus rounding the PARCOR coefficients. Which quantized filters show the most deviation from the original frequency response? Were any of the resulting filters unstable?

Project 4: Formant Tracking

In interpreting the prediction error filter, it is common to assume that the roots of $A(z)$ (i.e., the poles of the vocal tract filter) are representative of the formant frequencies for the segment of speech (frame) from which the predictor coefficients are computed. Thus, the angles of the roots expressed in terms of analog frequency are sometimes used as an estimate of the formant frequencies. For example consider Fig. 2 which is a plot of all the pole angles as a function of speech frame index.

Figure 2 was obtained by the following algorithm:

1. Read as many samples of speech starting at sample number `nbeg` as you can comfortably work with into an array `x`. Set `n = 1`
2. Compute the prediction coefficients for a Hamming windowed speech segment of length `nwin` samples, starting at sample `n`.
3. Find the magnitudes of the angles of the roots of the prediction error filter, and convert them to frequencies in Hz (assuming a sampling rate of speech is 8 kHz). Store the vector of frequencies as columns in a two-dimensional array (matrix) `F` where each column of `F` is the frequency vector of a "frame" of speech.
4. Set `n = n+ninc` where `ninc` is the increment in samples between frames. While `n+nwin <= length(x)` return to step 2 and repeat. Otherwise quit.

After the matrix `F` is computed you can make a plot like Fig. 2 with `plot(F','*w')`.

Exercise 4.1: M-file for Formant Tracking

Write an M-file to implement the above procedure. Use the following calling sequence:

```
function    F = formants(x, ninc, nwin, p)
%FORMANTS    Function to plot angles of roots of prediction
%             error filter as a function of time.
%   Usage:  F = formants(x, ninc, nwin, p)
%                x : input speech signal
%             ninc : number of samples between windows
%             nwin : window length
%                p : order of prediction error filter
%                F : array of frequencies
```

Figure 2 shows an analysis of the samples `s5(1200:17200)`. In this case, 99 frames are separated by 160 samples with a window length of 320 samples. In this case, the speech was pre-emphasized with

```
y = filter([1, -0.98], 1, s5(1200:17200))
```

prior to the formant analysis. This pre-emphasis tends to remove the effect of the glottal wave.

Since this involves 16000 speech samples, which is longer than can be read into a single array in PC-MATLAB, the processing might have to be done on a smaller section. A plot like Fig. 2 could be constructed by doing the analysis in pieces, it is not essential for understanding the method. Simply test your program on speech segments of convenient length for your computing environment.

Exercise 4.2: Editing the Formant Tracks

You will note from Fig. 2 that the above algorithm plots the angles of all the roots including the real roots which lie at $\omega = 0$ and $\omega = \pi$. Also it might plot the angles of complex roots twice because these roots occur in complex conjugate pairs. It is reasonable to eliminate these redundant roots, as well as any real roots, because they are obviously not formant frequencies. It is also quite likely that roots whose magnitude is less than about 0.8 are not formants, so they should be eliminated also. Figure 3 shows that a much cleaner formant track is obtained when these extraneous roots are not included. Modify your M-file in Exercise 4.1 to perform this editing feature. In doing so, you should use the `find( )` function to locate the roots to be eliminated. Also, you will find that simply eliminating these roots from the frequency vectors would result in vectors of different lengths from frame-to-frame, and this would cause problems in making up the matrix `F`. A neat way to eliminate the desired roots from the plot is to replace them with MATLAB's object `NaN` (not a number). This would keep all the vectors the same length (`p`), but the `plot( )` function will automatically ignore the `NaN` values.

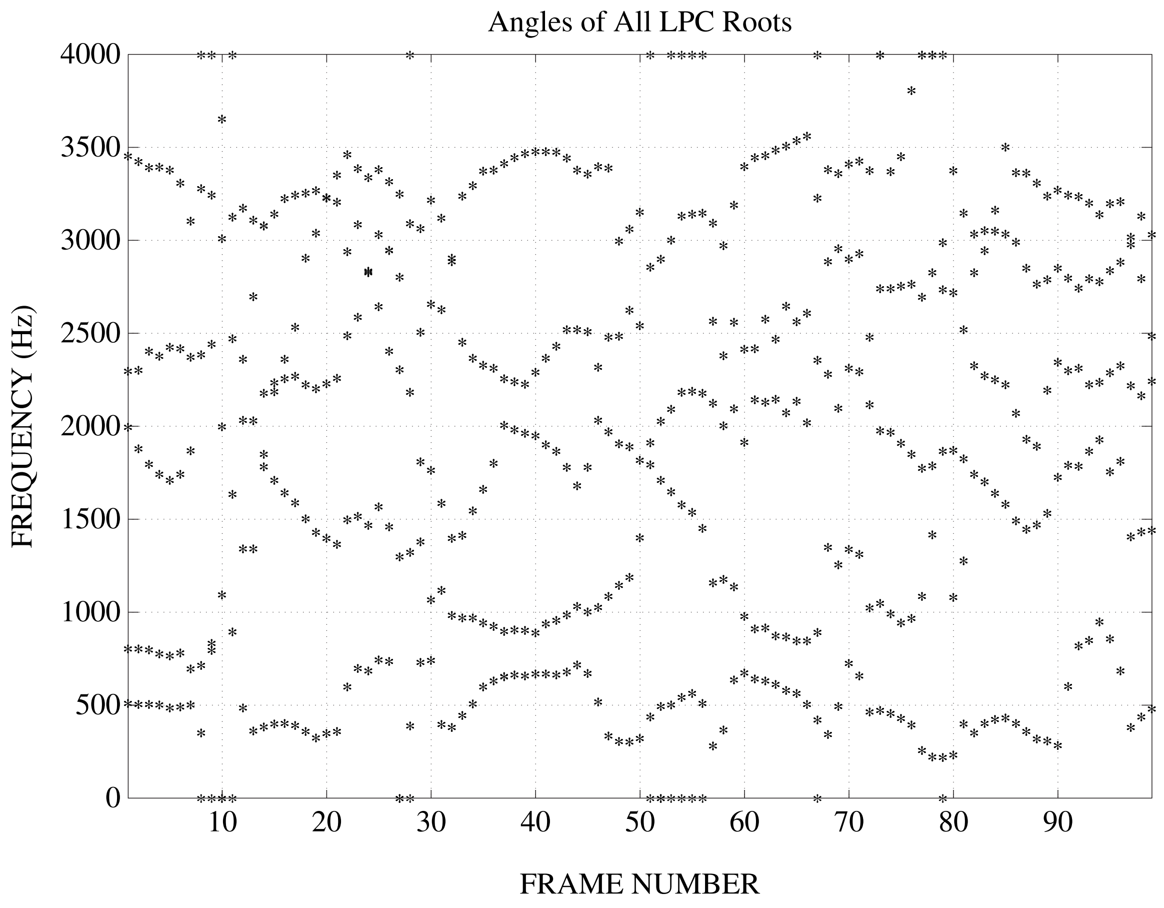


Figure 2: “Formant frequencies” estimated by linear prediction.

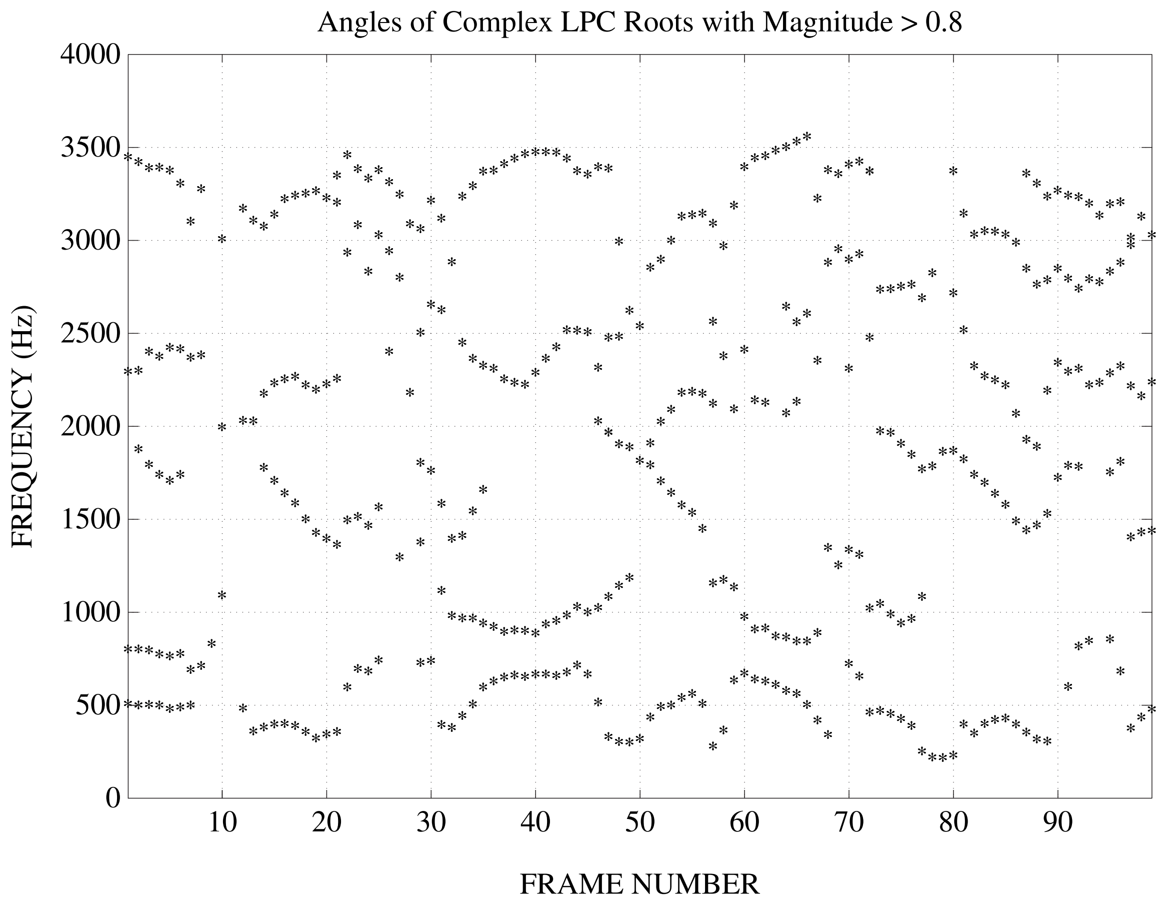


Figure 3: Cleaner "formant frequency" plot.

Chapter 11

Signal Modeling

January 17, 2007

Overview

This chapter presents a variety of special topics related to signal modeling and estimation. In the first packet, the widely used technique of linear prediction is studied. Several problems based on actual signals are posed, including one project on the prediction of stock market data. In the second packet, the application of linear prediction to speech modeling is presented. In this case, the linear predictor not only models the speech signal, but can also be used to resynthesize the signal from an all-pole model. In the third packet, the linear prediction methods are extended to the problem of exponential modeling. In this case, a signal is represented by a weighted sum of complex exponentials with unknown exponents. The determination of the unknown exponents is done via Prony's method which is amounts to rooting the linear prediction polynomial. In addition, problem of pole-zero modeling is studied using a variant of Prony's method called the Steiglitz-McBride iteration [?]. This algorithm also provides a superior time-domain match when fitting unknown exponentials to a signal.

The fourth packet examines the problem of interpolation from the viewpoint of least squares signal estimation. The general theory presented there is applicable to the estimation of any linear functionals of a signal; we focus, however, on the problem of estimating samples of a sub-sampled signal, i.e., interpolation. The last packet examines the problems of linear least squares inversion and of the solution of inaccurate, insufficient, and inconsistent linear equations. The problem of noisy data is considered and truncation of the Singular Value Decomposition (SVD) expansion is proposed as a way to reduce the effect of noise (at the expense of resolution). The trade-off between noise and resolution is explored.

Background Reading

A number of advanced topics are presented in the edited collection [20] and in the text by Marple [?]. Material on the application of linear prediction to speech can be found in the text by Rabiner and Schafer [?]. Methods of linear inversion are presented in the paper by Jackson [?] and Chapter 3 of the book by Lanczos [?].

Computer-Based Exercises for Signal Processing

$\boxed{\text{Exponential Modeling}}$

Exponential Modeling

Overview

This packet provides an introduction to exponential signal modeling. In this problem we seek to represent a signal $s[n]$ as a weighted sum of exponentials with unknown exponents:

$$s[n] \;\approx\; \sum_{k=1}^{N} c_k\, e^{\alpha_k n} \;=\; \sum_{k=1}^{N} c_k (\lambda_k)^n \tag{0.1}$$

where $\alpha_k = \log \lambda_k$ are the unknown exponents. If α_k is complex, then (0.1) can represent a decaying sinusoid; if α_k is real and negative, the exponential decays.

Computing the parameters of the model $\{c_k,\ \lambda_k\}$ from the data $s[n]$ is a difficult task, because the representation (0.1) is non-linear in the λ_k's. In fact, most algorithms for calculating the representation involve two steps: (1) determine the λ_k's, and (2) compute the c_k's, assuming the λ_k's are already known. The important simplification of the exponential modeling problem lies in its connection to linear prediction. By using the covariance method of linear prediction, the problem of finding the λ_k's is reduced to a polynomial factoring operation. This sort of technique is usually called *Prony's method* when applied to exponential modeling.

In the first project, the basic idea behind Prony's method will be illustrated by showing that any exponential signal can be perfectly predicted by a linear predictor. Since the z-transform of the representation in (0.1) is rational but not all-pole, a complete solution to the exponential modeling problem requires a discussion of pole-zero modeling. Therefore, the second project takes up the full pole-zero modeling problem and develops the Steiglitz-McBride algorithm [**?**] which is an iterative solution to the exponential modeling problem.

The z-transform of (0.1) is a partial fraction expansion:

$$H(z) = \sum_{k=1}^{N} \frac{c_k}{1 - \lambda_k z^{-1}} \tag{0.2}$$

assuming that all the λ_k's are different. The partial fraction form can be combined into a rational form where the coefficients of $A(z)$ and $B(z)$ are the parameters of the model.

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + \ldots + b_M z^{-M}}{1 + a_1 z^{-1} + \ldots + a_N z^{-N}} \tag{0.3}$$

The order of the numerator polynomial will be $M = N-1$ because it is produced from the partial fraction form. The λ_k's are the poles of the system, i.e., the roots of $A(z)$.

The exponential modeling problem is equivalent to representing $s[n]$ as the impulse response $h[n]$ of a pole-zero system. If we express the problem as one of signal approximation, we want to minimize the error

$$E(z) = S(z) - \frac{B(z)}{A(z)}$$

assuming that $X(z)$ is the z-transform of $s[n]$. In the time domain, this would require the minimization of the norm of the error signal $e[n] = s[n] - h[n]$

$$\min_{\{b_\ell, a_k\}} \| \, e[n] \, \|$$

where $h[n]$, the inverse transform of $H(z)$, is the impulse response of the pole-zero system in (0.3).

In general, the *direct* minimization of $\| e[n] \|$ requires solving a set of complicated non-linear equations. The two-step procedure suggested by Prony's method simplifies the problem but also changes it somewhat. In effect, an *indirect* modeling problem is solved, i.e., linear prediction, and then this solution is used to approximate the direct solution.

Background Reading

More details on this approach to the pole-zero modeling problem can be found in Chapter 1 of [20], or in Chapter 11 of [**?**].

Project 1: Prony's Method

The basic idea underlying Prony's method is that an exponential signal can be cancelled completely by a linear predictor. Thus, the zeros of the cancelling filter are the poles needed in (0.1).

Exercise 1.1: Cancel an Exponential

(a) Generate 25 points of the signal $s[n] = a^n u[n]$, with $a = -0.88$. Make a `comb` plot of $s[n]$.

(b) Process $s[n]$ through a 2-term FIR filter $G(z) = 1 + \gamma z^{-1}$. Compute and plot the output for the cases $\gamma = 0.9$ and -0.9.

(c) Determine the value of γ so that the output signal will be exactly zero for $n \geq 1$.

(d) Extend this idea to the 2nd-order case. Let $s[n] = \sin(\pi n/4)\, u[n]$. Process $s[n]$ through a 3-term FIR filter $G(z) = 1 + \gamma_1 z^{-1} + \gamma_2 z^{-2}$, with $\gamma_1 = 0$ and $\gamma_2 = 1$.

(e) Now select the coefficients γ_1 and γ_2 to make the output zero for $n \geq 2$.

(f) Determine the zeros of $G(z)$ found in the previous part. Explain the relationship between these zeros and the signal $s[n] = \sin(\pi n/4)\, u[n]$.

Exercise 1.2: Prony's Method

When the signal is composed of a large number of (complex) exponentials, a general approach is needed to design the FIR system that will cancel the signal. Refer to the pole-zero model given in (0.3). In the time-domain, the relationship in (0.3) is just a linear difference equation with coefficients a_k and b_ℓ.

$$-\sum_{k=1}^{N} a_k y[n-k] + \sum_{\ell=0}^{M} b_\ell x[n-\ell] = y[n] \qquad n = 0, 1, 2, \ldots, L-1 \tag{1.1}$$

where $x[n]$ is the input and $y[n]$ the output. If $s[n]$ were to satisfy this difference equation, then for $n \geq N$ we get:

$$-\sum_{k=1}^{N} a_k s[n-k] = s[n] \qquad n = N, N+1, \ldots, L-1 \tag{1.2}$$

which is a set of simultaneous linear equations in the N unknowns $\{a_k\}$.

(a) Rewrite equation (1.2) for $\{a_k\}$ in matrix form.

(b) The M-file below will generate a signal $s[n]$ that is a sum of exponentials as in (0.1). Use the following parameters to generate $s[n]$ for $0 \le n \le 30$.

```
lambda  =  [ 0.9 0.7+0.7j 0.7-0.7j -0.8 ]
     c  =  [ 3.3 4.2 -4.2 2.7 ]
```

Make a `comb` plot of $s[n]$.

```
function  ss = pronysyn( lam, c, nn )
%PRONYSYN   synthesize a sum of exponentials
%   usage:    ss = pronysyn( lam, c, nn )
%    lam = vector of exponents
%      c = vector of weights
%     nn = vector of time indices
%     ss = output signal
%
N = length(lam);
ss = 0*nn;
for k = 1:N
   ss = ss + c(k)*exp(lam(k)*nn);
end
```

(c) Form a polynomial $G(z)$ whose roots are given by `lambda` in the previous exercise (see `help poly`). Demonstrate that processing $s[n]$ from part (b) through $G(z)$ will give a zero output for $n \ge n_0$. Determine the value of n_0.

(d) The function `residuez` can be used to convert from a partial fraction representation to a rational $B(z)/A(z)$ form. Show that the same signal as in part (b) can be generated by using `filter` with the appropriate `b` and `a` vectors.

Exercise 1.3: Test Signal

The data file `EXPdata.mat` contains two signals that were generated via the exponential model in (0.1). The first, `sigclean`, is exactly in the form (0.1).

(a) For this signal, determine an FIR system that will exactly cancel `sigclean` past a certain point. From this predictor calculate the exponents λ_k needed in the model for `sigclean`. Use the minimum number of equations from (1.2).

(b) Once the correct values of the λ_k's are determined, write a set of simultaneous linear equations for the unknown gains c_k. This can be done by considering (0.1) to be a linear equation for each n. Since there are N c_k's, N equations in N unknowns should be sufficient. Write the equations in matrix form, and solve using the backslash operator \ in MATLAB.

If you generate more equations than unknowns, do you still compute the same answer?

(c) Write an M-file that will compute both the exponents and the gains, thus implementing what would be considered an extended form of Prony's method.

Exercise 1.4: Noisy Signals

The second signal in `EXPdata.mat` is `signoisy`, which is just the signal `sigclean` plus a small amount of additive noise.

(a) Use the M-file from part (c) of the previous exercise, to calculate λ_k and c_k for the noisy signal. Use the minimum number of equations needed from (1.2) and (0.1). Comment on the differences that you observe.

(b) Redo the previous part with more equations than unknowns. In fact, use the maximum number of equations permitted by the length of the data set L. Comment on the difference between the λ_k's from this computation and those from part (a).

(c) As in Exercise 1.2, part (d), determine values for the filter coefficients and resynthesize the signal from the pole-zero model. Create an impulse response of the model $h[n]$ that is the same length as `signoisy` and then plot both on the same graph. Include the noise-free signal `sigclean` for comparison.

(d) What is the modeling error? Find the norm of the error ($\| e \|$) between the true signal $s[n]$ and your estimate $h[n]$.

Project 2: Pole-Zero Modeling

Complete modeling of an exponential signal requires a pole-zero model. Since Prony's method is unable to calculate the correct poles when the signal is noisy, the computation of the zeros will also be incorrect in Prony's method. However, in the technique known as *Iterative Pre-Filtering* [**?**], the denominator polynomial (i.e., poles) determined at one iteration is used to form a new problem in which a generalized form of the equations appearing in Prony's method are solved. The key feature of this method is that only linear equations have to be solved at any step. Furthermore, the method usually converges within 3–5 iterations, if the signal is well-matched by an exponential model.

The basic difficulty with Prony's method is that it does not minimize the true error between the given impulse response $s[n]$ and the model's impulse response $h[n]$. Instead, an "equation error" is minimized. The two errors can be described in the z-transform domain as:

$$\begin{aligned} E_{\text{eq}}(z) &= S(z)A(z) - B(z) && (2.1) \\ E_{\text{true}}(z) &= S(z) - \frac{B(z)}{A(z)} = \frac{1}{A(z)}\left[A(z)S(z) - B(z)\right] && (2.2) \end{aligned}$$

They are related via:

$$E_{\text{true}}(z) = E_{\text{eq}}(z)/A(z) \qquad (2.3)$$

So the basic idea is to develop a recursion in which the equation error is weighted so that it will be closer to the true error. This requires two distinct operations:

1. *Kalman's Method.* A method for finding the pole-zero model of a system when the input signal and the output signal are both known.
2. *Iterative Prefiltering.* Assuming that a computation of the poles has been done, a pseudo input-output problem is created; then Kalman's method is applied.

If the second step is carried out repeatedly, and the answers for $B_i(z)$ and $A_i(z)$ converge, then the error minimized is the true error.

Exercise 2.1: Kalman's Method

Assume that a rational system $H(z)$ has been tested such that both the input and output signals are known. The input does not have to be an impulse signal.

$$Y(z) = \frac{B(z)}{A(z)} X(z)$$

The rational system is to be approximated (maybe with zero error) by a pole-zero model (0.3). The number of poles and zeros must be fixed *a priori.* The objective is to determine the parameters $\{a_k\}$, and $\{b_\ell\}$; and to do so by solving only linear equations.

In (1.1), there are M+N+1 unknowns, and the number of equations depends on the length of the data available for $x[n]$ and $y[n]$. Usually, there will be more equations than unknowns (if the modeling is to work). These over-determined equations can then be solved in the least-squares sense (using the backslash operator in MATLAB). This is *Kalman's Method.*

(a) Write out *all* the equations in (1.1) for the specific case where $M = 1$ and $N = 2$ and the length of the data sequences is $L = 7$, i.e., $x[n] \neq 0$ and $y[n] \neq 0$ only when $0 \leq n \leq 6$.

(b) The equations (1.1) can be written in matrix form. Write a MATLAB function that will produce the coefficient matrix and the right-hand side for the equations in (1.1).

(c) To test the function, generate 20 points of the output signal when the input to the following system is a random signal with unit variance.

$$\frac{2 + 4z^{-1} + 2z^{-2}}{1 + 0.8z^{-1} + 0.81z^{-2}}$$

Solve the resulting equations using backslash. Compare the estimated parameters with the true values of $\{a_k,\ b_\ell\}$.

(d) To test the robustness of this solution, add some noise to the output $y[n]$. For the 20-point signal, add white Gaussian noise with a variance of 0.01; try it also with a variance of 0.1. Comment on the robustness of the answer.

(e) Try a 100-point signal with additive noise. Is the answer more robust with the longer signal?

NOTE: Kalman's method, when used for the impulse response modeling problem, amounts to Prony's method to find $A(z)$ followed by cross-multiplication of $A(z) \times S(z)$ to get the numerator $B(z)$. This is usually not very robust.

Exercise 2.2: Prefiltering Equations

The iterative prefiltering equations will be developed, assuming that an estimate of the denominator polynomial $A_i(z)$ has already been done at the i^{th} iteration. The all-pole filter $1/A_i(z)$ has an impulse response which we will call $h_i[n]$. We can also apply $1/A_i(z)$ to the given signal $s[n]$, and thus produce the the following signal: $s_i[n] = h[n] * h_i[n]$. If we now apply Kalman's method with $h_i[n]$ playing the role of the input, and $s_i[n]$ the output, we can compute a new pole-zero model so that the following is approximately true:

$$A_{i+1}(z)S_i(z) \approx B_{i+1}(z)H_i(z) \tag{2.4}$$

In other words, we must solve the following set of over-determined linear equations:

$$-\sum_{k=1}^{N} a_k s_i[n-k] + \sum_{\ell=0}^{M} b_\ell h_i[n-\ell] = s_i[n] \tag{2.5}$$

The improvement from step i to $i+1$ is based on the observation that the error being minimized in (2.4) can be written as:

$$E_K(z) = A_{i+1}(z)S_i(z) - B_{i+1}(z)H_i(z) = A_{i+1}(z)\frac{S(z)}{A_i(z)} - \frac{B_{i+1}(z)}{A_i(z)}$$

Therefore, if the Kalman error converges to zero, we get $A_i(z) \approx A_{i+1}(z) \to A(z)$ and $B_i(z)/A_i(z) \to S(z)$.

(a) Write a function that will produce the matrix equations described in (2.5). Omit those that have zero entries due to going beyond the length of the signal.

(b) Since the impulse response $h_i[n]$ can be produced for all $n \geq 0$, it is tempting to think that an arbitrary number of non-zero equations can be written. However, the convolution of $h_i[n]$ with $s[n]$ is not useful outside the finite range of the given data. Use this fact in limiting the number of equations for the least-squares solution.

Exercise 2.3: Steiglitz-McBride Iteration

Now comes the iterative part: since a computation of Kalman's method yields a new denominator polynomial, we can redo the whole process with the new $A(z)$.

(a) Write a MATLAB function `stmcbrid` that will implement the complete iteration, which is called the *Steiglitz-McBride* method.

(b) Generate the same example as in Exercise 2.1, part (c). Use the Prony solution as the starting point and apply the Steiglitz-McBride method [**?**].

(c) Use the function `pronysyn` to generate test signals that are weighted sums of exponentials. Verify that your function `stmcbrid` will compute the correct answer in the noise free case. It might be convenient to have `stmcbrid` return the c_k and λ_k parameters instead of the a_k's and b_ℓ's.

(d) Test the robustness of the technique by using `pronysyn` to generate a test signal and then add noise. Experiment with the SNR and the length of the signal L. Compare the pole positions λ_k with and without noise.

(e) Apply your function to the unknown signal `signoisy` from the file `EXPdata.mat` in Exercise 1.4. Comment on the improvement obtained with the Steiglitz-McBride iteration versus Prony's method.

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{Signal Estimation}}$

Signal Estimation

Overview

This packet examines the problem of interpolation from the viewpoint of least square signal estimation. The general theory used here is applicable to the estimation of any linear functionals of a signal; we focus, however, on the problem of estimating samples of a sub-sampled signal, i.e., interpolation.

This packet follows material in sections of the chapter by Golomb and Weinberger, "Optimal Approximation and Error Bounds", [**?**]. See especially pp. 132–135, pp. 140–143 and the Introduction, p. 117. The results we need are summarized below:

> Given signal measurements (linear functionals)
>
> $$F_i(\mathbf{u}) = f_i \quad i = 1, \ldots, N$$
>
> where the values of the linear functionals F_i are f_i for the signal $\mathbf{u}$, and given that $\mathbf{u}$ belongs to the signal class
>
> $$C = \Big\{ \mathbf{u} \in \mathcal{H} : \langle \mathbf{u}, \mathbf{u} \rangle \leq r^2, F_i(\mathbf{u}) = f_i \; i = 1, \ldots, N \Big\}$$
>
> the best estimate of the signal, $\hat{\mathbf{u}}$, is a linear combination of the representers ϕ_i of the linear functionals, F_i.
>
> $$\hat{\mathbf{u}} = \sum_{i=1}^{N} c_i \phi_i$$
>
> where the coefficients c_i are chosen so that $\hat{\mathbf{u}}$ has the given values f_i of the linear functionals,
>
> $$F_i(\hat{\mathbf{u}}) = f_i \quad i = 1, \ldots, N$$
>
> The signal estimate $\hat{\mathbf{u}}$ is best in the sense that
>
> $$\max_{\mathbf{u} \in C} |F(\mathbf{u}) - F(\hat{\mathbf{u}})|$$
>
> is minimized.

With this method we are not limited to interpolating an evenly decimated signal—we can interpolate non-uniformly sub-sampled signals, and we can extrapolate a signal in intervals in which we have no samples (such as in prediction of "future" samples). Additionally, we may obtain a bound on the error of our estimate. Throughout the packet we assume that the signal to be interpolated is the output of a known finite-dimensional linear transformation, with a known bound on the input norm. This is a special form of the class C above. More details on the application of signal estimation to the interpolation problem can be found in the paper by Shenoy and Parks [**?**].

Background Reading

The fundamental material on linear functionals and Hilbert space operators can be found in the text by Luenberger [**?**]. This should serve as a solid foundation for reading the chapter by Golomb and Weinberger [**?**]. For the application of least-squares to interpolation filter design see the paper by Oetken, Parks and Schüßler [**?**].

Project 1: Finding the Optimal Estimate in a Filter Class

In this project we obtain an optimal estimate of a signal with missing samples. We assume the sub-sampled signal is known to be in a certain class of signals, called a filter class.

A signal $\mathbf{a}$ is input to a linear transformation $\mathbf{C}$, which outputs $\mathbf{u}$:

$$\mathbf{u}_{n\times 1} = \mathbf{C}_{n\times m}\mathbf{a}_{m\times 1}.$$

If we let matrix multiplication by $\mathbf{C}$ represent a filtering operation of length l impulse response $\mathbf{h}$ with a finite length signal, we obtain:

$$\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} = \begin{pmatrix} h_l & h_{l-1} & \dots & h_1 & 0 & \dots & 0 \\ 0 & h_l & \dots & h_2 & h_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & h_l & h_{l-1} & \dots & h_1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{pmatrix}$$

Here we choose $\mathbf{h}$ to represent an averaging operation with an averaging length of l samples, given by:

$$h(n) = \begin{cases} \frac{1}{l} & \text{if } 1 \le n \le l \\ 0 & \text{otherwise} \end{cases}.$$

In this problem, we suppose that a number of samples of $\mathbf{u}$ are unknown and we would like to estimate them as accurately as possible. To do this we need to use all of the information about $\mathbf{u}$ that we have. One thing we know about $\mathbf{u}$ is that it is the result of a signal that has been averaged (i.e., it is the left hand side of the above matrix equation). This information, along with a suitable bound on the norm of the input signal $\mathbf{a}$, makes $\mathbf{u}$ a member of a *filter class*. Knowing that u is in a filter class, and given a certain set of known linear functionals of $\mathbf{u}$ (e.g., samples), we may estimate unknown linear functionals of $\mathbf{u}$ using the techniques of deterministic signal estimation, described in Golomb and Weinberger [?]. In this packet, *the unknown linear functionals are the unknown samples of* $\mathbf{u}$. You will estimate these linear functionals with the knowledge that $\mathbf{u}$ has been created by an averaging operation.

Hints

You may find the following MATLAB functions useful: `convmtx`, `norm` and backslash `\`. The backslash operator will solve a set of linear equations, and it will compute the least-squares solution for an over-determined set, and it will get a solution in the under-determined case.

Exercise 1.1: Create the averaging matrix

In MATLAB, create an $n \times m$ averaging matrix $\mathbf{C}$ as depicted above. Use $n = 31$ and $l = 5$.

Exercise 1.2: Create some test signals

Create a length m input signal $\mathbf{a}$ of normally distributed noise. Let $\mathbf{u} = \mathbf{Ca}$. Create a vector of sample indices, `xx`, at which we know the signal $\mathbf{u}$. For now, use

```
    xx = [ 3 10 14 18 21 29 ];
```

Let yy be the output signal sampled at xx, i.e.

```
yy = u(xx);
```

(*Note:* in real life, we will not know **a** or **u** of course, but for the purpose of evaluating our method this will be very useful.)

Exercise 1.3: Create the inner product matrix

Let

```
R = C*C';
Q = pinv(R);
```

R is the correlation matrix of the impulse response **h**. Its k^{th} column is the representor in the Q-inner-product space for the linear functional that extracts the k^{th} sample of a signal. That is, if $\phi_k = \texttt{R(:,k)}$, then

$$\langle \phi_k, \mathbf{u} \rangle_Q = \mathbf{u}_k$$

where $\mathbf{u}_k$ is the k^{th} sample of **u**. We want to form the Q inner product matrix of the representors corresponding to known samples, that is, the matrix $\mathbf{\Phi}$ with elements

$$\Phi_{ij} = \langle \phi_i, \phi_j \rangle_Q$$

where i and j are elements of xx. Call this matrix PHI.

Hint: Recall that $\langle \mathbf{x}, \mathbf{y} \rangle_Q = \texttt{x'*Q*y}$ for column vectors x and y. PHI can be created with one line of MATLAB code, using only the variables R, Q, and xx.

Exercise 1.4: Calculating ubar, the optimal estimate

The best estimate ubar is a linear combination of the representors of the known linear functionals, i.e.

```
ubar = R(:,xx)*c;
```

for some column c. We know that ubar goes through the known samples, i.e.

$$\langle \phi_k, \texttt{ubar} \rangle_Q = \texttt{yy(k)}$$

for all k in xx. With this information we can use PHI and yy to solve for c.

Calculate ubar and plot it along with the actual signal **u**. What is the norm of the error, $\|\texttt{ubar} - \mathbf{u}\|$?

Exercise 1.5: The Q-norm of ubar

For this exercise, add a constant to the filter input **a** on the order of twice the variance. For the resultant sub-sampled **u**, calculate ubar and plot it along with **u**. Is there an alarming property of the optimal estimate that you notice readily from this plot? Generate a number of inputs **a** (with non-zero mean) and for each, write down the Q-norm of the error, $\|\texttt{ubar} - \mathbf{u}\|_Q$, the Q-norm of ubar, $\|\texttt{ubar}\|_Q$ and the Q-norm of **u**, $\|\mathbf{u}\|_Q$. Compare $\|\texttt{ubar}\|_Q$ and $\|\mathbf{u}\|_Q$. Is one always larger than the other?

Exercise 1.6: Special case: the averaging operation

Find `ubar` for the following case: $n = 30$, $l = 6$, and `xx = [1:3:n]`. What sort of interpolation does this very closely approximate? In light of this, plot the representors corresponding to samples that we know (with `plot(R(:,xx))`), and consider taking a linear combination of these vectors. Note that the optimal interpolation in this case (equally spaced outputs of an averaging operator) results from a very simple interpolation scheme. Are there other spacings in the vector `xx` that lead to the same simple interpolation scheme? Are there other n and l that lead to the same simple interpolation scheme? Try to answer these two questions by finding a relationship between n, l, and p (where `xx = [1:p:n]`) that, when satisfied, results in this sort of simple interpolation.

Project 2: Error Bounds: *How Good is the Estimate?*

Now that we have calculated the estimate `ubar`, we would like to understand how well the actual signal has been approximated. In this project we will find the maximum error possible in each sample estimate, as well as worst case signals in our filter class that achieve the maximum error at some samples.

Exercise 2.1: Calculating `ybar` ($\bar{\mathbf{y}}$)

For each unknown sample, $\bar{\mathbf{y}}$ is a unit vector (in the `Q`-norm) that can be scaled and added to `ubar` in order to yield a signal in our filter class that is "furthest away" from `ubar`. As there is a different $\bar{\mathbf{y}}$ for each unknown sample of $\mathbf{u}$, we will create a matrix `ybar` whose k^{th} column `ybar(:,k)` is $\bar{\mathbf{y}}$ for the k^{th} sample of $\mathbf{u}$. Thus in MATLAB, `ybar` will be an $n \times n$ matrix (note that $\bar{\mathbf{y}}$ is undefined for samples that we know, so we will just set `ybar(:,xx)` to zero).

For an unknown sample k, `ybar(:,k)` is a linear combination of the representors of the known samples and of the representor of the k^{th} sample, i.e.

```
ybar(:,k) = R(:,[k xx])*cbar;
```

for some column `cbar`. We know that `ybar(:,k)` is orthogonal to the representors of the known samples, i.e.

$$\langle \phi_i, \texttt{ybar(:,k)} \rangle_Q = 0$$

for all i in `xx`. With this information we can use `PHI`, `R`, `xx`, `k` and `Q` to solve for `cbar` up to a constant factor. This constant factor is determined from the condition that `ybar(:,k)` is a unit vector in the `Q`-norm:

```
ybar(:,k) = ybar(:,k)/sqrt(ybar(:,k)'*Q*ybar(:,k));
```

Calculate `ybar(:,k)` for all k not in `xx`. For those k in `xx`, set `ybar(:,k)` to zero.

Hint: The most straight-forward way to form the `ybar` matrix is column-wise inside a `for` loop, although it is possible to do so (without normalization) with a "one-liner" (one line of MATLAB code). In order to save time, consider inverting `PHI` one time only prior to entering the `for` loop.

Exercise 2.2: Worst case signals and error bounds

The vector `ybar(:,k)` has the property that when scaled by the factor

```
    scale = sqrt(a'*a - ubar'*Q*ubar);
```

and added to or subtracted from `ubar`, the resultant signal is worst case in that it lies on the "edge" of our filter class. The factor `scale` is a measure of the distance from the center of our filter class to the boundary of our filter class. Here we use our knowledge of the energy of **a** to describe the boundary of our filter class; in an actual application this number is assumed known.

The signal

```
    uworst(:,k) = ubar + scale*ybar(:,k);
```

has maximum possible error at sample k and it lies on the boundary of our filter class. Create a matrix `uworst` whose columns are the worst case signals for different samples. What is the `Q`-norm of `uworst(:,k)`?

Create a vector of maximum errors by multiplying `scale` by the absolute value of the diagonal of the `ybar` matrix.

Exercise 2.3: Plotting the estimate with error bounds

Plot the estimate `ubar` in the following manner:

(a) Plot the upper and lower error bounds with a dotted line on the same plot. It is suggested that you use one plot statement so that MATLAB can scale the y-axis properly.

(b) With the above plot held, plot `ubar` with a solid black line between the two bounds.

(c) For comparison, plot the original signal **u** with plus or asterisk characters.

(d) Finally, plot the worst-case signals on the same plot. Your plot will be very messy, so focus your attention on the worst-case signal for one particular sample. Does any worst case signal achieve the maximum error at more than one sample index?

Computer-Based Exercises

for

Signal Processing

$\boxed{\text{Least Square Inversion}}$

Least Square Inversion

Overview

This packet examines the problems of linear least square inversion and of the solution of inaccurate, insufficient, and inconsistent linear equations. The Singular Value Decomposition (SVD) is used to produce a ''solution" to a set of equations which may not have an exact solution. If an exact solution does not exist, a solution is found which minimizes the sum of the squared errors in the equations. If the equations do not have a unique solution, then the minimum norm solution is used.

The problem of noisy data is considered and truncation of the SVD expansion is proposed as a way to reduce the effect of noise (at the expense of resolution). The trade-off between noise and resolution is explored.

Background Reading

This method of linear inversion is presented in the paper by Jackson [?] and Chapter 3 of the book by Lanczos [?]. Its application in geophysics is treated in Chapter 12 of the book by Aki and Richards [?]

Project 1: Least-Square Inversion

In this project you study the least square solution of a set of linear equations. Throughout this project, explicit reference is made to the paper by Jackson [?]. An effort has been made to keep the notation identical to that in the paper, whenever possible, to avoid undue confusion.

The system shown below is implemented in MATLAB:

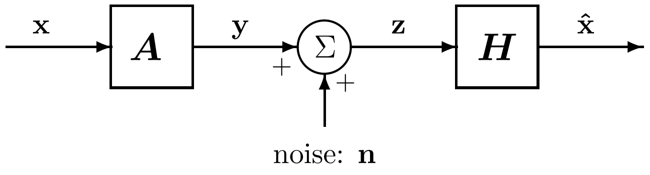


Figure 1: Linear system with additive measurement noise

A signal, $\mathbf{x}$ (the model), is input to a linear transformation, $\mathbf{A}$, which outputs $\mathbf{y}$ (the measurement). Note that, in general, $n \neq m$

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix}$$

or just

$$\mathbf{y}_{n\times 1} = \mathbf{A}_{n\times m}\mathbf{x}_{m\times 1}$$

The transformation, $\mathbf{A}$, is a sample-invariant averaging operator, where l is the number of samples averaged to produce each output sample:

$$y_k = \frac{1}{l} \sum_{j=k}^{k+l-1} x_j$$

equivalently, the entries of the matrix $\mathbf{A}$ are:

$$a_{ij} = \begin{cases} \frac{1}{l} & \text{if } 0 \leq (j-i) < l \\ 0 & \text{otherwise} \end{cases}$$

This project will examine the problem of inverting this operation, and the resulting tradeoffs in performance under different conditions.

The "true" measurement vector, $\mathbf{y}$, has a measurement error associated with it, modelled by the noise vector $\mathbf{n}$ in Fig. 1. The "observed" measurement, $\mathbf{z}$, is the signal available to analyze.

It is desirable to apply an inversion operation, $\mathbf{H}$, to $\mathbf{z}$ to produce an estimate $\hat{\mathbf{x}}$ of the model $\mathbf{x}$. The operator $\mathbf{H}$ may be designed by an SVD (Singular Value Decomposition) procedure on $\mathbf{A}$. One of the inverses possible to construct in this manner, the Lanczos inverse, has several properties that may be desirable in an inverse:

- it always exists (this is not trivial)
- it is a least squares solution (in the equation error $\mathbf{A}\hat{\mathbf{x}} - \mathbf{y} = \boldsymbol{\epsilon}$)
- it is a minimum norm solution
- resolution of the model parameters is optimized (in some sense)

Refer to [?] for a thorough explanation of these properties. In addition to the Lanczos inverse, the SVD procedure may be modified (often referred to as a *truncated* SVD) to create additional inverses, with different performance tradeoffs. In general, the optimality of the Lanczos inverse is traded off for lower variance in the model parameter estimates (see [?, p. 104]).

Hints

You will complete various MATLAB functions to implement the system shown in Fig. 1. Printouts of the partially completed functions are attached. Using test data and test noise provided for you, you will examine the performance of the inversion system under different conditions. Make sure you look ahead to the last exercises in Project 4 (which contain questions), so that you understand what concepts you are expected to learn.

Exercise 1.1: Averaging Matrix

Complete the MATLAB function `MAKEAVR.M`, which creates the matrix $\mathbf{A}$, so that it performs as advertised:

```
        A = makeavr(n,m)
```

Exercise 1.2: Least-Square Inverse

Complete the function `INVERT.M`, which performs the (full-rank or truncated) SVD-based inversion:

```
B = invert(A,q)
```

where $\mathbf{B}$ is a generalized inverse of $\mathbf{A}$, and q is the rank of $\mathbf{B}$. That is, for example, if $\text{rank}(\mathbf{A}) = p$ and $q = p$ then $\mathbf{B}$ is the Lanczos inverse of $\mathbf{A}$.

Exercise 1.3: Completing the MODLEST Function

Complete the function `MODLEST.M`. This calls the functions `INVERT.M` and `MAKEAVR.M` to implement the system in Fig. 1:

```
[H,A,xhat] = modlest(x,n,q)
```

Project 2: Testing with Noise-Free Signals

Exercise 2.1: Computing Test Signals

Load the test signals into the MATLAB workspace (these include two measurement noise vectors and four input signals):

```
load lslab
```

Exercise 2.2: Evaluation of the Least-Square Inverse

Run `MODLEST.M` to perform Lanczos inversion (i.e., full rank: $q = p$). Use the following pairs of test input and test noise signals, respectively:

`x1` and `n1`

`x2` and `n1`

`x3` and `n1`

(Note that the "noise" signal `n1` is a zero vector used to simulate zero measurement noise.)

Analyze these signal combinations as thoroughly as you can with the functions you have written, and any additional MATLAB (built-in) functions you think would be helpful. Be sure that you can answer the questions posed in the evaluation section of Project 4. Note: the plots in `MODLEST.M` may not be scaled the same.

Project 3: Inversion of Noisy Signals

Exercise 3.1: Noise Gain Calculatins

Complete the function `VARIANC.M` which calculates the noise gain of the matrix $\mathbf{H}$ (see [**?**, p. 98]). Notice that this will effectively compute the sensitivity of the inversion operation to measurement noise (why?):

```
[var] = varianc(H)
```

Exercise 3.2: Resolution Calculations

Complete the function `RESOLVE.M` which calculates the resolving error of the model estimating system, defined as follows:

$$r_k = \sum_{j=1}^{m} \left[\left(\sum_{i=1}^{n} h_{ki}\, a_{ij} \right) - \delta_{kj} \right]^2 \qquad k \in [1, m]$$

where

$$\delta_{kj} = \begin{cases} 1 & \text{if } k = j \\ 0 & \text{otherwise} \end{cases}$$

(This equation is a corrected version of equation 21 in [**?**].) This is only one of many ways that model estimate resolution may be defined (a different definition is used in Exercise 3.4). Note that a higher error means lower resolution and that $\max\{r_k\} \leq 1$ (why?).Your completed function should compute r_k for $k \in [1, m]$:

```
[r] = resolve(H,A)
```

Exercise 3.3: Evaluation with Noise

Using the following test input-test noise pair, examine the system performance for (at least) $q = 3, 5, 10, 15, 17, 20$:

`x4` and `n2`

Create $\mathbf{R} = \mathbf{H} * \mathbf{A}$, the "resolving" matrix, for each case. Plot it versus the identity matrix:

```
subplot(211);   mesh(R);   mesh(eye(R));
```

Using `RESOLVE.M` and `VARIANC.M`, calculate the resolution and variance for the index $k = 10$. Make a (hand) plot of these calculated values versus q. (i.e., plot `var(10)` vs. q and `r(10)` vs. q together on one plot.) This plot should have the general characteristics of Fig. 1 in [**?**].

Exercise 3.4: Resolution Definitions

Using a different measurement of resolution, make a similar (hand) plot to the one mentioned above:

resolution = "width of estimated pulse at amplitude = 0.3"

That is, measure the (amplitude = 0.3) crossings of the *estimated* pulse, $\hat{\mathbf{x}}$, for the same range of q. Note: don't be too concerned about getting very accurate results here, as you'll be estimating the width from a graph. To aid in this, try:

```
clg;  plot(xhat);  grid
```

Project 4: Evaluation of Least Square Inversion

Exercise 4.1: Type of Equations

Using the *Jackson* terminology [**?**], what type of system does **A** represent? (i.e., underdetermined, overconstrained, *strictly* overconstrained (overconstrained but not underdetermined), etc.?) How does this relate to the the quality of the estimated model parameters? That is, discuss uniqueness, exactness, etc.

Exercise 4.2: Comparison of Performance

Relate the performance of the system in Exercises 2.2 and 3.3, above, to the relationship between the test input and test noise and the various vector spaces associated with $\mathbf{A}$ (i.e., $\mathcal{V}, \mathcal{V}_{\mathbf{o}}, \mathcal{U}, \mathcal{U}_{\mathbf{o}}$, etc., where $\mathbf{A} = \mathbf{U\Lambda V}^t$ is the SVD of $\mathbf{A}$). How could you generate these test signals and others like them? (It is not necessary to write any MATLAB code for this.)

Exercise 4.3: Relationship Between $\mathbf{x}$ and $\hat{\mathbf{x}}$

What is the relationship between $\mathbf{x}$ and $\hat{\mathbf{x}}$ in Exercise 2.2? Why are they equal in some cases and not equal in others? Relate this to the discussion in Excercise 4.2, above.

Exercise 4.4: Relationship Between $\mathbf{x}1$ and $\mathbf{x}3$

What's the relationship between $\mathbf{x}1$ and $\mathbf{x}3$?

Exercise 4.5: Sample-Invariance

Is $\mathbf{H}$ a sample-invariant operator? Is it *almost* sample-invariant? Explain.

Exercise 4.6: The Effect of q

How does the difference between $\hat{\mathbf{x}}$ (with noise) and $\hat{\mathbf{x}}$ (without noise), in Exercise 3.3, change as q is varied? Explain.

Exercise 4.7: Choice of Test Signal

Note that $\mathbf{x}4$ is a pulse signal. Do you think this is an appropriate test signal for Exercise 3.3? Why or why not?

Matlab Script Files and Incomplete Functions

These Matlab shells are available on the distribution disk under the names: `invert.m`, `makeavr.m`, `modlest.m`, `resolve.m`, and `varianc.m`.

```
*********************************************  INVERT.M  ******
function B = invert(A, q)
%
% This routine finds the generalized inverse of the matrix A.
% The rank of the inverse is q, where q <= p and p = rank of A.
%
% The routine works the same as one computing the Lanczos
% inverse (which it will do if q = rank(A)), except that
% only those q largest (in absolute magnitude) singular values
% of A are used in creation of H.
%
[U,S,V] = svd(A);
%
% Check to make sure inversion of desired order is possible:
%  This code (from RANK.M) prevents having to do the SVD twice.
%
diag_S = diag(S);
tol = max(size(A)) * diag_S(1) * eps;
rank_A = sum(diag_S > tol);
if (q > rank_A)
rank_of_A = rank(A)
q = q
error('The rank of A is insufficient to produce an inverse of rank q.');
end
%
%  Now resize the results so that the matrix S is square.
%  This is the standard notational convention for SVD
%  MATLAB is not standard in that it forces U and V to be square.
%       That is, MATLAB returns the vectors associated with zeros
%       singular values.
%       Simultaneously, change the size of U,S and V to accomodate
%       the reduced order inversion.
%
%============>              <====================
%============> ADD CODE HERE <====================
%============>              <====================
%
% Now create the inverse:
%
B = V * inv(S) * U';
```

```
********************************************** MAKEAVR.M ******
function A = makeavr(n, m)
%
%       This function creates a matrix that describes a
%       sample-invariant averaging operator.
%
%       Note that the averaging period = l = (m-n+1),
%       where A is (n x m).
%
%       The entries are normalized so that the sum across rows = 1
%       I.e.,
%              A(i,j)   =    1/(m-n+1)   0 <= (j-i) < (m-n+1)
%                            0           otherwise
%
% Note: an input with n > m returns an error.
%
if(n > m),  error('n > m input to MAKEAVER is not allowed');  end
%
%============>              <===================
%============> ADD CODE HERE <===================
%============>              <===================

********************************************** MODLEST.M ******
function [H, A, xhat] = modlest(x, noise, q)
%
%   Inputs:
%        x : The input vector, length = m.
%    noise : The measurement noise vector, length = n.
%       q : The rank of the inverse.
%
% Outputs:
%       A : The input transformation matrix
%       H : A generalized inverse for A, rank = q
%   xhat : The model (estimate) of the input vector.
%
% Convert the inputs to column orientation:
%
[n,m] = size(x);
if (n == 1)   %--- I.e., if x is a row vector.
  x = x';
end
%
[n,m] = size(noise);
if (n == 1)     %--- I.e., if noise is a row vector.
  noise = noise';
end
```

```
%       Create the averaging matrix A:
%          The dimensions of A are variable, depending on the
%          dimensions of the signal and measurement noise
%
%============>               <====================
%============> ADD CODE HERE <====================
%============>               <====================
%
%  Create the measurement vector, y, and the inversion matrix, H:
%
%============>               <====================
%============> ADD CODE HERE <====================
%============>               <====================
%
%   Calculate the model estimate(s), xhat:
%       xhat_no_noise is a variable which is instructive to look at.
%       It is the xhat which would be produced if the measured
%       vector y had no noise on it. It is instuctive to plot,
%       but in practice, with a real system, you would not
%       have access to y (only to z) and thus could not look at
%       xhat_no_noise.
%
xhat_no_noise = H * y;
%
%============>               <====================
%============> ADD CODE HERE <====================
%============>               <====================
%
%  The following plots can be commented out:
%
clg
subplot(221), plot(x); title('model vector x');
plot(y);               title('measurement vector y');
plot(xhat_no_noise);   title('xhat (no noise)');
plot(xhat);            title('xhat');
pause
clg
```

```
*********************************************  RESOLVE.M  ******
function r = resolve(H, A)
%
%     Inputs:
%          A  : a matrix
%          H  : an inverse for A
%
% Outputs:
%       r  : a column vector, length = m, where m is the
%            dimension of the square matrix H*A = R
%            error(k) is the 2-norm of the error (squared)
%            in approximating an impulse with the kth row
%            of H*A = R.  If H is the Lanczos inverse for A,
%            this error is minimized for each k, over all
%            possible inverses for A.
%
%============>              <====================
%============> ADD CODE HERE <====================
%============>              <====================
%
```

```
*********************************************  VARIANC.M  ******
function var = varianc(H)
%
% Inputs:
%      H : a matrix
%
% Outputs:
%     var : a length m column vector where H is (m x n)
%           var(k) is the noise gain of the matrix
%         relative to the kth index inputs.
%      Refer to Jackson, "Interpretation of ...", pg. 98.
%
%============>              <====================
%============> ADD CODE HERE <====================
%============>              <====================
```

Computer-Based Exercises for Signal Processing

$\boxed{\text{Homomorphic Processing}}$

Homomorphic Processing

Background Reading

See Chapter 12 of

1. A. V. Oppenheim, R. W. Schafer: *Discrete-Time Signal Processing*, Prentice-Hall, Englewood Cliffs N.J., 1989.

Project 1: Cepstrum Computation

Project Description

This project explores some of the properties of cepstral analysis discussed in chapter 12 section 12.6. For a sequence $x[n]$, $\hat{x}[n]$ denotes the complex cepstrum as defined by equation 12.7 and $c_x[n]$ as the (real) cepstrum as defined by equation 12.8. $c_x[n]$ and $\hat{x}[n]$ are related by equation 12.9.

As discussed in section 12.6, if $x[n]$ is minimum phase then $\hat{x}[n]$ is causal. As a consequence of this property and equation 12.9 (see section 12.6.1), for a minimum phase sequence $\hat{x}[n]$ can be recovered from $c_x[n]$ (see figure 12.6). Equivalently, for a minimum phase sequence, $x[n]$ can be recovered from the magnitude of its Fourier transform.

If $x[n]$ is non-minimum phase, it can be represented as the convolution of a minimum-phase sequence $x_{mn}[n]$ and a maximum-phase sequence, $x_{mx}[n]$ (see section 12.6.4). Alternatively, it can be expressed as the convolution of a minimum-phase sequence $x_{min}[n]$ and an all-pass sequence $x_{ap}[n]$ (see section 12.6.2). In general, for a given $x[n]$, $x_{mn}[n]$ and $x_{min}[n]$ will be different. In parts (g) and (h) we illustrate the decomposition of a sequence into these components.

Hints

Tell the students that MATLAB *already has some functions for cepstral analysis.*

Exercise 1.1: Real cepstrum

The real cepstrum $c_x[n]$ is the inverse Fourier transform of the $\log_e$ of the magnitude of the Fourier transform of $x[n]$. Write a MATLAB function `realceps` that takes as its argument an input vector and a power of two length and returns the real cepstrum. Don't use the MATLAB function `rceps`.

Exercise 1.2: Complex cepstrum

The complex cepstrum $\hat{x}[n]$ is the inverse Fourier transform of the $\log_e |X(e^{j\omega})| + j\phi(\omega)$, where $\phi(\omega)$ is the unwrapped phase (in radians). Write a MATLAB function `compceps` that takes as its argument an input vector and a power of two length and returns the complex cepstrum. Don't use the MATLAB function `cceps`. However you should use the MATLAB function `unwrap` to generate the unwrapped phase from the Fourier transform of x[n].

Exercise 1.3:

The impulse response of the difference equation

$$y[n] - 1.86y[n-1] + .9y[n-2] = .x[n] - .9x[n-1] \tag{1.1}$$

which we will denote as $x[n]$, is minimum phase. Generate $x[n]$. Using the function `realceps` generate the real cepstrum $c_x[n]$ for $x[n]$. Use a DFT length of 512 to be sure that $X(e^{j\omega})$ is sufficiently well sampled so that there is no time aliasing in the cepstrum. Also, generate $c_x[n]$ using the MATLAB function `rceps` and compare this with what you obtained with `realceps`. (The results should be virtually identical).

Exercise 1.4:

Using the function `compceps` generate $\hat{x}[n]$ the complex cepstrum of $x[n]$. Also, generate $\hat{x}[n]$ using the MATLAB function `cceps`. Again, the results should be identical. Also, note that since $x[n]$ is minimum phase, $\hat{x}[n]$ should be causal.

Exercise 1.5:

Write a MATLAB function `icompceps` that takes as its argument an input vector $\hat{x}[n]$ and a power of two length and generates the sequence $x[n]$ whose complex cepstrum is $\hat{x}[n]$. Test the function by applying it to the complex cepstrum of the impulse response of equation (1). Compare your result to $x[n]$.

Project 2: Cepstral Decompositions

Project Description

We now want to demonstrate that the minimum phase impulse response $x[n]$ of equation (1) can be recovered from just the magnitude of its Fourier transform. The basic idea, as discussed in section 12.6.1, is that $\hat{x}[n]$ can easily be obtained from $c_x[n]$ for a minimum-phase signal. (see equation 12.62 and figure 12.6). Since $c_x[n]$ is computed from the Fourier transform magnitude and since $x[n]$ can be obtained from $\hat{x}[n]$, (see, for example, figure 12.9) then $x[n]$ can be reconstructed from the Fourier transform magnitude.

Hints

Exercise 2.1:

Using equation (12.60) obtain $\hat{x}[n]$ from $c_x[n]$ for the minimum phase impulse response $x[n]$. Apply `icompceps` to recover $x[n]$. Compare with the original input sequence $x[n]$.

Exercise 2.2: Min and Max Phase Components

We now want to demonstrate the use of the complex cepstrum to decompose a non-minimum-phase sequence into minimum-phase and maximum phase components (see section 12.6.4). The impulse $x[n]$ response of the following difference equation is non-minimum-phase.

$$y[n] - 1.86y[n-1] + .9y[n-2] = .98x[n] - 1x[n-1]$$

Generate $x[n]$ and using the approach outlined in section 12.6.4 and figure 12.8, obtain the minimum-phase and maximum-phase components $x_{mn}[n]$ and $x_{mx}[n]$. Recombine these components by convolution using the MATLAB function `conv` and compare the result with the original non-minimum-phase sequence $x[n]$.

Exercise 2.3: Min-phase plus all-pass

A non-minimum-phase sequence can alternatively be decomposed into the convolution of a minimum-phase sequence $x_{\min}[n]$ and an all-pass sequence $x_{\text{ap}}[n]$. Using the procedure outlined in section 12.6.2 and summarized in figure 12.7, generate $x_{\min}[n]$ and $x_{\text{ap}}[n]$ for the same non-minimum-phase sequence used in (g). Note that no phase unwrapping is needed. State specifically whether or not $x_{mn}[n]$ and $x_{\min}[n]$ are equal. Also, recombine $x_{\min}$ and $x_{\text{ap}}[n]$ by convolution and compare the result with the original non-minimum-phase sequence $x[n]$.

Project 3: Application: Homomorphic Deconvolution

Project Description

One of the applications of homomorphic filtering and cepstral analysis is echo detection and removal. Specifically, consider a signal $v[n]$ which is received together with an echo N_0 samples later so that the total signal received is

$$x[n] = v[n] + \beta v[n - N_0] \tag{3.1}$$

Equivalently,

$$x[n] = v[n] * p[n] \tag{3.2}$$

where $p[n]$ is made up of two impulses,

$$p[n] = \delta[n] + \beta\delta[n - N_0] \tag{3.3}$$

Let $\hat{x}[n]$, $\hat{v}[n]$ and $\hat{p}[n]$ denote the complex cepstrum of $x[n]$, $v[n]$ and $p[n]$. If $|\beta| < 1$ then $p[n]$ is minimum phase so that $\hat{p}[n]$ is causal. Also $\hat{p}[n]$ is only non-zero at integer multiples of N_0. If $\hat{v}[n]$ dies out rapidly with n, so that it is effectively zero for $n \geq N_0$, then $\hat{v}[n]$ can be obtained from $\hat{x}[n]$ by using a "lowpass frequency invariant filter" as discussed in section 12.8.4. Specifically, under the assumption stated above,

$$\hat{v}[n] \approx \begin{cases} \hat{x}[n] & n < N_0 \\ 0 & n > N_0 \end{cases}$$

Correspondingly, $\hat{p}[n]$ can be approximately obtained by using a "highpass frequency invariant filter." The separate components $v[n]$ and $p[n]$ can be obtained from $\hat{v}[n]$ and $\hat{p}[n]$.

Exercise 3.1: Signal containing an echo

The MATLAB vector `echoed` is of the form of equation (3.1) or (3.2), with $v[n]$ chosen so that $\hat{v}[n]$ decays rapidly. Type `load echoed` to load the sequence. Plot $x[n] =$ `echoed` and generate and plot $\hat{x}[n]$. From $\hat{x}[n]$, estimate as accurately as possible, the value of N_0

Exercise 3.2: Echo removal via deconvolution

Using the approach outlined in section 12.8.4, obtain from $\hat{x}[n]$ the approximations to the sequences $v[n]$ and $p[n]$. Plot the results. From $p[n]$, estimate the value of β.

References

[1] L. R. Rabiner and B. Gold. *Theory and Application of Digital Signal Processing.* Prentice-Hall, Englewood Cliffs, NJ, 1975.

[2] B. Gold and C. M. Rader. *Digital Processing of Signals.* McGraw-Hill, New York, NY, 1969.

[3] T. W. Parks and C. S. Burrus. *Digital Filter Design.* John Wiley and Sons, New York, 1987.

[4] F. J. Taylor. *Digital Filter Design Handbook.* Marcel Dekker, Inc., New York, NY, 1983.

[5] M.E. Van Valkenburg. *Analog Filter Design.* Holt, Rinehart, and Winston, New York, NY, 1982.

[6] C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems.* Prentice-Hall, Inglewood Cliffs, NJ, 1974.

[7] C. S. Burrus, A. W. Soewito, and R. A. Gopinath. Least squared error FIR filter design with transition bands. *IEEE Transactions on SP*, 40(6):1327–1340, June 1992.

[8] A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[9] Richard B. Darst. *Introduction to Linear Programming.* Marcel Dekker, New York, NY, 1991.

[10] P. P. Vaidyanathan. Design and implementation of digital FIR filters. In Douglas F. Elliott, editor, *Handbook of Digital Signal Processing*, chapter 2, pages 55–170. Academic Press, San Diego, CA, 1987.

[11] E. W. Cheney. *Introduction to Approximation Theory.* McGraw-Hill, New York, NY, 1966.

[12] Lawrence R. Rabiner. Linear program design of finite impulse response (FIR) digital filters. *IEEE Trans. on Audio and Electroacoustics*, AU-20(4):280–288, October 1972.

[13] Gilbert Strang. *Linear Algebra and Its Applications.* Academic Press, New York, NY, 1976.

[14] Andrew Grace. *Matlab Optimization Toolbox.* The MathWorks, Inc., Natick, MA, 1990.

[15] C. S. Burrus and T. W. Parks. Time domain design of recursive digital filters. *IEEE Trans. on Audio and Electroacoustics*, 18(2):137–141, June 1970.

[16] C. S. Burrus and T. W. Parks. *DFT/FFT and Convolution Algorithms.* John Wiley and Sons, New York, 1985.

[17] E. Oran Brigham. *The Fast Fourier Transform and Its Applications.* Prentice-Hall, Englewood Cliffs, NJ, 1988. Expansion of the 1974 book.

[18] Richard E. Blahut. *Fast Algorithms for Digital Signal Processing.* Addison-Wesley, Reading, Mass., 1985.

[19] J. H. McClellan and C. M. Rader. *Number Theory in Digital Signal Processing.* Prentice-Hall, Englewood Cliffs, NJ, 1979.

[20] J. S. Lim and A. V. Oppenheim. *Advanced Topics in Signal Processing.* Prentice-Hall, Englewood Cliffs, NJ, 1988.

[21] P. Duhamel and M. Vetterli. Fast Fourier transforms: A tutorial review and a state of the art. *Signal Processing*, 19(4):259–299, April 1990.

[22] DSP Committee, editor. *Programs for Digital Signal Processing.* IEEE Press, New York, NY, 1979.

[23] H. V. Sorensen, M. T. Heideman, and C. S. Burrus. On calculating the split-radix FFT. *IEEE Trans. on ASSP*, 34:152–156, February 1986.

[24] Ivan Niven and H. S. Zuckerman. *An Introduction to the Theory of Numbers.* John Wiley & Sons, New York, NY, fourth edition, 1980.

[25] Oystein Ore. *Number Theory and Its History.* McGraw-Hill, New York, NY, 1948.

[26] Donald E. Knuth. *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms.* Addison-Wesley, Reading, MA, second edition, 1981.