

- [4] A. V. Oppenheim and A. S. Willsky with I. T. Young. *Signals and Systems*. Prentice Hall, Englewood Cliffs, NJ, 1983.
- [5] R. D. Strum and D. E. Kirk. *First Principles of Discrete Systems and Digital Signal Processing*. Addison-Wesley, Reading, MA, 1988.
- [6] R. A. Roberts and C. T. Mullis. *Digital Signal Processing*. Addison-Wesley, Reading, MA, 1987.
- [7] J. G. Proakis and D. G. Manolakis. *Digital Signal Processing: Principles, Algorithms and Applications*. Macmillan, New York, second edition, 1992.

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 numerical 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, e.g., at $-N$. 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 not a problem until it comes time to plot the signal, in which case the horizontal axis must be labeled properly.

A final point is the use of MATLAB's vector notation to generate signals. A significant 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 treat the common signals encountered in digital signal processing: impulses, impulse trains, exponentials, and sinusoids.

BACKGROUND READING

Oppenheim and Schaffer (1989), Chapter 2, Sections 2.0 and 2.1.

PROJECT 1: BASIC SIGNALS

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.¹ The following MATLAB code will create 31 points of a discrete-time sinusoid.

¹In MATLAB version 4, this function has been renamed to `stem`.

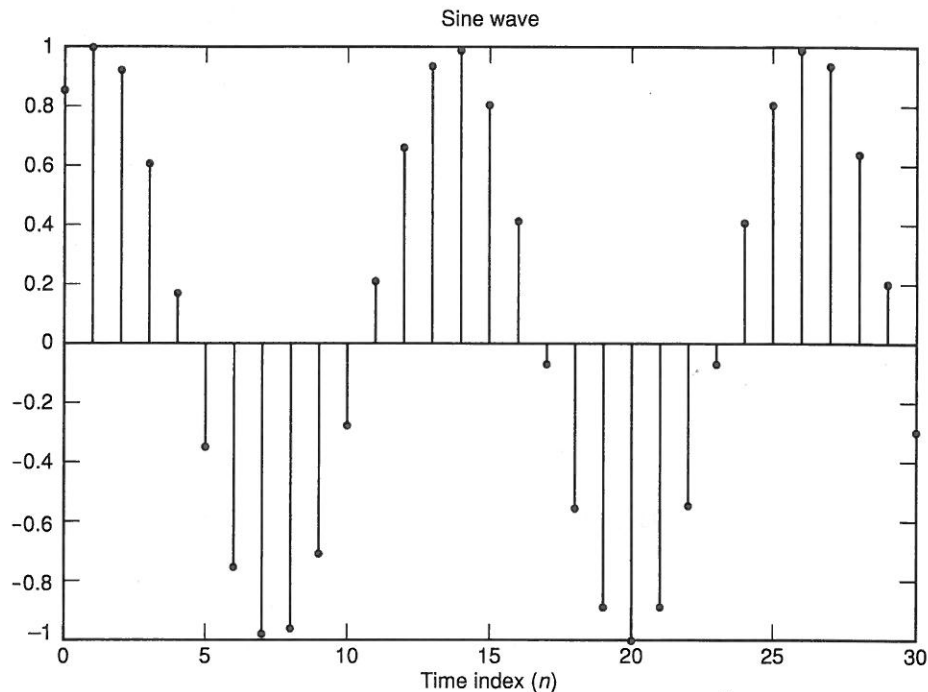
```
nn = 0:30;    %-- vector of time indices
sinus = sin(nn/2+1);
```

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 in DSP books (see Fig. 1.1):

```
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 labeling.

Figure 1.1
Plotting a discrete-time
signal with `comb`.



EXERCISE 1.1

Basic Signals—Impulses

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

$$\delta[n - n_0] = \begin{cases} 1 & n = n_0 \\ 0 & n \neq n_0. \end{cases} \quad (1-1)$$

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 labeled accordingly. Each sequence should be displayed as a discrete-time signal using `comb`.

$$x_1[n] = 0.9\delta[n - 5] \quad 1 \leq n \leq 20$$

$$x_2[n] = 0.8\delta[n] \quad -15 \leq n \leq 15$$

$$x_3[n] = 1.5\delta[n - 333] \quad 300 \leq n \leq 350$$

$$x_4[n] = 4.5\delta[n + 7] \quad -10 \leq n \leq 0$$

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

$$s[n] = \sum_{\ell=0}^{M-1} A_\ell \delta[n - \ell P] \quad (1-2)$$

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 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 describe a real sinusoidal signal completely: amplitude (A), frequency (ω_0), and phase (ϕ).

$$x[n] = A \cos(\omega_0 n + \phi) \quad (1-3)$$

- 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 labeled accordingly. Each sequence should be displayed as a sequence using `comb`.

$$x_1[n] = \sin \frac{\pi}{17} n \quad 0 \leq n \leq 25$$

$$x_2[n] = \sin \frac{\pi}{17} n \quad -15 \leq n \leq 25$$

$$x_3[n] = \sin(3\pi n + \frac{\pi}{2}) \quad -10 \leq n \leq 10$$

$$x_4[n] = \cos\left(\frac{\pi}{\sqrt{23}} n\right) \quad 0 \leq n \leq 50$$

Give a simpler formula for $x_3[n]$ that does not use trigonometric 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 five input arguments: three for the parameters and two 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 $-20 \leq n \leq 20$.

- c. *Modification:* Rewrite the function in part (b) to return two arguments: a vector of indices over the range of n , and 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 relationship 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:

$$s(t) = A \cos(2\pi f_0 t + \phi) \quad (1-4)$$

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 $f_s = 1/T$, we get

$$s[n] = s(t)|_{t=nT} = A \cos(2\pi f_0 T n + \phi) = A \cos\left(2\pi \frac{f_0}{f_s} n + \phi\right) \quad (1-5)$$

Comparison with 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 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 require six inputs: three for the signal parameters, two 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 kHz
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 milliseconds), 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 of the sinusoid 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 part (a) 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 $n = 0, 1, 2, \dots, 20$.

```

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:

$$\sum_{n=0}^{L-1} a^n = \frac{1-a^L}{1-a} \quad \text{for } a \neq 1 \quad (1-6)$$

Use the function from part (a) to generate an exponential and then sum it up; compare the result to formula (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:

$$y[n] = ay[n-1] \quad \text{over the range } 1 \leq n \leq L-1 \quad (1-7)$$

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 endpoints 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:

$$y[n] - ay[n-1] = x[n] \quad \text{initial condition: } y[-1] = 0 \quad (1-8)$$

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

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 electrical engineering students are familiar with phasors in connection

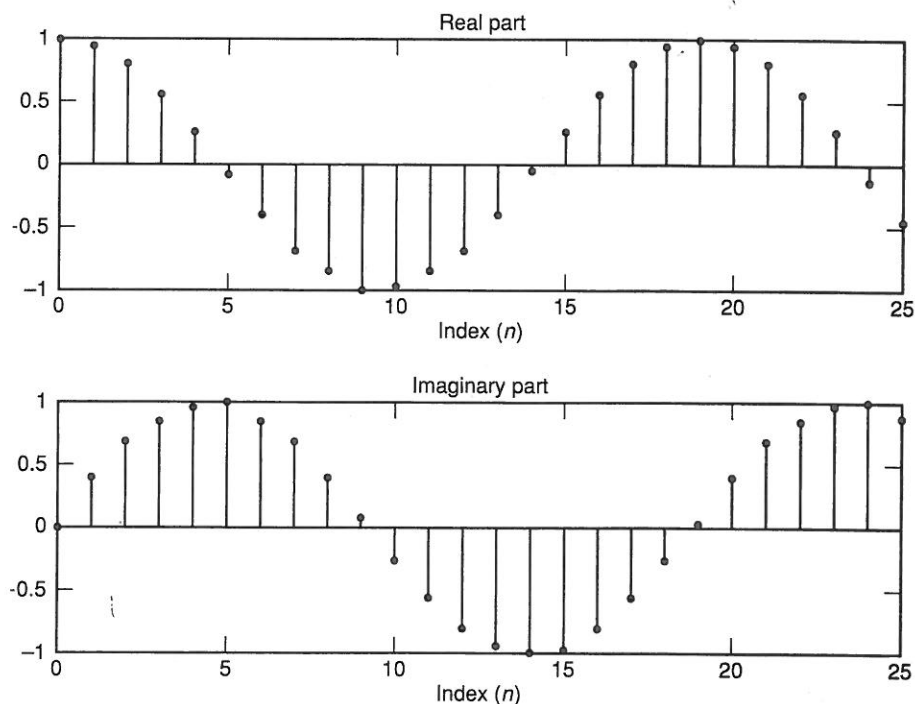
with ac circuits or power systems, but their use in radar, wave propagation, and Fourier analysis is just as significant (although the term *phasor* is not always used).

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 `comb` 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. 1.2, which was created using the following code:

```
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 1.2
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.

- In MATLAB a complex signal is a natural extension of the notation in Exercise 1.4. 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):

$$x[n] = (z_0)^n = e^{(ln z_0 + j\angle z_0)n} = r^n e^{j\theta n} = r^n (\cos \theta n + j \sin \theta n) \quad (2-1)$$

where $z_0 = r e^{j\theta} = r \angle \theta$. Use this relationship to generate a complex exponential with $z_0 = 0.9 \angle 45^\circ$. Plot the real and imaginary parts of $x[n]$ over the range $0 \leq n \leq 20$. 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. Equation (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*:

$$G \cdot z_0^n = A e^{j\phi} r^n e^{j\theta n} = A r^n e^{j(\theta n + \phi)} = A r^n [\cos(\theta n + \phi) + j \sin(\theta n + \phi)], \quad (2-2)$$

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, 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 labeled accordingly.

$$x_1[n] = 3 \sin\left(\frac{\pi}{7}n\right) + j4 \cos\left(\frac{\pi}{7}n\right) \quad 0 \leq n \leq 20$$

$$x_2[n] = \sin \frac{\pi}{17}n \quad -15 \leq n \leq 25$$

$$x_3[n] = 1.1^n \cos\left(\frac{\pi}{11}n + \frac{\pi}{4}\right) \quad 0 \leq n \leq 50$$

$$x_4[n] = 0.9^n \cos\left(\frac{\pi}{11}n\right) \quad -10 \leq n \leq 20$$

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`):

$$y[n] = z_0 y[n-1] + x[n]. \quad (2-3)$$

The filter coefficient, $z_0 = r e^{j\theta}$, 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 which drives the difference equation (i.e., $x[n] = G \delta[n]$). Use `filter` to create the same signals as in part (c). 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 (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 $y_R[n-1]$, $y_I[n-1]$, $x[n]$ 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 $\theta = \pi/4$. 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).

DIFFERENCE EQUATIONS

OVERVIEW

Of particular importance in digital signal processing is the class of systems that can be represented by linear constant-coefficient difference equations. In this set of projects we explore the characteristics of these systems in both the time and frequency domains. Specifically, in Project 1 we consider the impulse response of infinite impulse response 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

In Oppenheim and Schaffer (1989), Chapter 2, Sections 2.2 through 2.5, discrete-time systems, linear time-invariant systems, and linear constant-coefficient difference equations are discussed. Sections 2.6 through 2.9 cover frequency-domain representations of discrete-time signals and systems.

PROJECT 1: TIME-DOMAIN RESPONSE OF DIFFERENCE EQUATIONS

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:

$$\sum_{k=0}^{N_a} a_k y[n-k] = \sum_{\ell=0}^{N_b} b_{\ell} x[n-\ell] \quad (1-1)$$

In MATLAB, difference equations are represented by two vectors: one vector containing the feedforward 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` returns only 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

Simple Difference Equation

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

$$y[n] + 0.9y[n-2] = 0.3x[n] + 0.6x[n-1] + 0.3x[n-2]. \quad (1-2)$$

- Calculate $y[n]$ analytically for $x[n] = \delta[n]$.
- Now create a unit impulse vector, `imp`, of length 128. Generate the first 128 points of the impulse response of the filter in (1-2). Use `comb` to plot these values as a discrete-time signal versus time (see `help comb`).² It may help to plot just the first 10 or 20 points.

EXERCISE 1.2

Impulse Response with `filter`

- 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 $-10 \leq n \leq 100$.

$$y[n] - 1.8 \cos\left(\frac{\pi}{16}\right) y[n-1] + 0.81 y[n-2] = x[n] + \frac{1}{2} x[n-1] \quad (1-3)$$

- Also determine the impulse response analytically and confirm your results.

²In MATLAB version 4, this function has been renamed `stem`.