EE323 Digital Signal Processing

Laboratory Manuals

Dr. Yu Yajun,
Associate Professor
Department of Electrical & Electronic Engineering
Southern University of Science & Technology

Contents

Lab 1 – Discrete and Continuous-Time Signals

1.1 Introduction

The purpose of this lab is to illustrate the properties of continuous and discrete-time signals using digital computers and the Matlab software environment. A continuous-time signal takes on a value at every point in time, whereas a discrete-time signal is only defined at integer values of the "time" variable. However, while discrete-time signals can be easily stored and processed on a computer, it is impossible to store the values of a continuous-time signal for all points along a segment of the real line. In later labs, we will see that digital computers are actually restricted to the storage of quantized discrete-time signals. Such signals are appropriately known as digital signals.

How then do we process continuous-time signals? In this lab, we will show that continuous-time signals may be processed by first approximating them by discrete-time signals using a process known as sampling. We will see that proper selection of the spacing between samples is crucial for an efficient and accurate approximation of a continuous-time signal. Excessively close spacing will lead to too much data, whereas excessively distant spacing will lead to a poor approximation of the continuous-time signal. Sampling will be an important topic in future labs, but for now we will use sampling to approximately compute some simple attributes of both real and synthetic signals.

In the following sections, we will illustrate the process of sampling, and demonstrate the importance of the sampling interval to the precision of numerical computations.

1.2 Review of Matlab

Practically all lab tasks in the EE323 lab will be performed using Matlab. Matlab (MATrix LABoratory) is a technical computing environment for numerical analysis, matrix computation, signal processing, and graphics. In this section, we will review some of its basic functions.

1.2.1 Starting Matlab and Getting Help

You can start Matlab on your computer by clicking the icon

on your desktop. After starting up, you will get a Matlab window. To get help on any specific command, such as "plot", you can type the following

help plot

in the "Command Window" portion of the Matlab window. You can do a keyword search for commands related to a topic by using the following.

lookfor topic

Here, replace "topic" by a word of topic that you are interested.

You can get an interactive help window using the function

doc

or by following the Help menu near the top of the window.

1.2.2. Matrices and Operations

Every element in Matlab is a matrix. So, for example, the Matlab command

a = [1 2 3]

creates a matrix named "*a*" with dimensions of 1 × 3. The variable "*a*" is stored in what is called the Matlab workspace. The operation

b = *a*.'

stores the transpose of "*a*" into the vector "*b*". In this case, "*b*" is a 3 × 1 vector.

Since each element in Matlab is a matrix, the operation

c = *a* ∗ *b*

computes the matrix product of "*a*" and "*b*" to generate a scalar value for "*c*" of 14 = 1∗1 + 2∗2 + 3∗3 .

Often, you may want to apply an operation to each element of a vector. For example, you may want to square each value of "*a*". In this case, you may use the following command.

c = *a*. ∗ *a*

The dot before the * tells Matlab that the multiplication should be applied to each corresponding element of "*a*". Therefore the .* operation is **not** a matrix operation. The dot convention works with many other Matlab commands such as divide ./ , and power .^. An error results if you try to perform element-wise operations on matrices that aren't the same size.

Note also that while the operation *a*.' performs a transpose on the matrix "*a*", the operation *a*' performs a **conjugate** transpose on "*a*" (transposes the matrix and conjugates each number in the matrix).

In addition, you may also add a ";" after a command, e.g.,

a = [1 2 3];
b = *a*.';
c = *a* ∗ *b*;
c
c = *a*. ∗ *a*;
c

Compare the difference between the commands with or without ";".

1.2.3 Matlab Scripts and Functions

Matlab has two methods for saving sequences of commands as standard files. These two methods are called **scripts** and **functions**. Scripts execute a sequence of Matlab commands just as if you typed them directly into the Matlab command window. Functions differ from scripts in that they accept inputs and return outputs, and variables defined within a function are generally local to that function.

A script-file is a text file with the filename extension ".m". The file should contain a sequence of Matlab commands. The script-file can be run by typing its name at the Matlab prompt without the .m extension. This is equivalent to typing in the commands at the prompt. Within the script-file, you can access variables you defined earlier in Matlab. All variables in the script-file are global, i.e. after the execution of the script-file, you can access its variables at the Matlab prompt. For more help on

scripts, see script.pdf.

To create a function nameed **func**, you first create a text file named func.m. The first line of the file must be

function output = func(input)

where input designates the set of input variables, and output are your output variables. The rest of the function file then contains the desired operations. All variables within the function are local; that means the function cannot access Matlab workspace variables that you don't pass as inputs. After the execution of the function, you cannot access internal variables of the function. For more help on functions see function.pdf.

1.3 Continuous-Time Vs. Discrete-Time

The "Introduction" (Section 1.1: Introduction) mentioned the important issue of representing continuous-time signals on a computer. In the following sections, we will illustrate the process of sampling, and demonstrate the importance of the sampling interval to the precision of numerical computations.

1.3.1 Analytical Calculation

Compute these two integrals. Do the computation manually.

1.

$$\int_0^{2\pi} (\sin 5t)^2 \, dt$$

2.

$$\int_0^1 e^t \, dt$$

INLAB REPORT: Hand in your calculations of these two integrals. Show all work.

1.3.2 Displaying Continuous- and Discrete-Time Signals in Matlab

To get help on the following topics, see plot.pdf, stem.pdf, and subplot.pdf

It is common to graph a discrete-time signal as dots in a Cartesian coordinate system. This can be done in the Matlab environment by using the **stem** command. We will also use the **subplot** command to put multiple plots on a single figure.

Start Matlab on your workstation and type the following sequence of commands.

```
n = 0:2:60;
y = sin(n/6);
subplot(3,1,1)
stem(n,y)
```

This plot shows the discrete-time signal formed by computing the values of the function sin(*t*/6) at points which are uniformly spaced at intervals of size 2. Notice that while sin(*t*/6) is a continuous-

time function, the sampled version of the signal, sin(n/6), is a discrete-time function.

A digital computer cannot store all points of a continuous-time signal since this would require an infinite amount of memory. It is, however, possible to plot a signal which looks like a continuous-time signal, by computing the value of the signal at closely spaced points in time, and then connecting the plotted points with lines. The Matlab plot function may be used to generate such plots. Use the following sequence of commands to generate two continuous-time plots of the signal sin(t/6).

```
n1 = 0:2:60;
z = sin(n1/6);
subplot(3,1,2)
plot(n1,z)
n2 = 0:10:60;
w = sin(n2/6);
subplot(3,1,3)
plot(n2,w)
```

As you can see, it is important to have many points to make the signal appear smooth. But how many points are enough for numerical calculations? In the following sections we will examine the effect of the sampling interval on the accuracy of computations.

INLAB REPORT: Submit a hard copy of the plots of the discrete-time function and two continuous-time functions. Label them with the **title** command, and include your names. Comment on the accuracy of each of the continuous time plots.

1.3.3 Numerical Computation of Continuous-Time Signals

To get help on the following topics, see script.pdf, function.pdf, and subplot.pdf.

Background on Numerical Integration

One common calculation on continuous-time signals is integration. Figure 2.1 illustrates a method used for computing the widely used Riemann integral. The Riemann integral approximates the area under a curve by breaking the region into many rectangles and summing their areas. Each rectangle is chosen to have the same width Δt, and the height of each rectangle is the value of the function at the start of the rectangle's interval.

To see the effects of using a different number of points to represent a continuous-time signal, write a Matlab function for numerically computing the integral of the function $\sin^2(5t)$ over the interval $[0, 2\pi]$. The syntax of the function should be I = integ1(N) where I is the result and N is the number of rectangles used to approximate the integral. This function should use the **sum** command and it should not contain any for loops!

NOTE: Since Matlab is an interpreted language, for loops are relatively slow. Therefore, we will avoid using loops whenever possible.

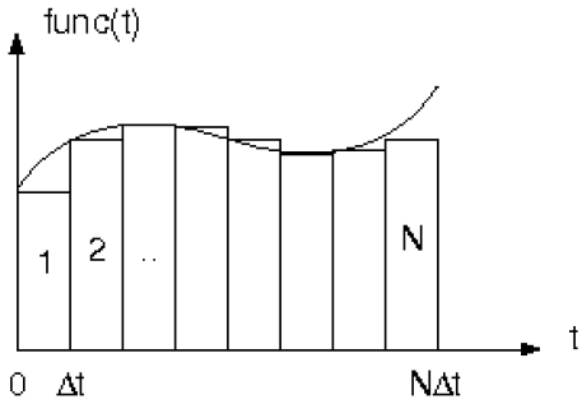


Figure 2.1: Illustration of the Riemann integral

Next write an m-file script that evaluates $I(N)$ for $1 \leq N \leq 100$, store the result in a vector and plot the resulting vector as a function of N. This m-file script may contain for loops.
Repeat this procedure for a second function J = integ2(N) which numerically computes the integral of
exp (t) on the interval [0; 1].

INLAB REPORT: Submit plots of $I(N)$ and $J(N)$ versus N. Use the **subplot** command to put both plots on a single sheet of paper. Also submit your Matlab code for each function. Compare your results to the analytical solutions from the "Analytical Calculation" (Section 2.1: Analytical Calculation) section. Explain why $I(5) = I(10) = 0$.

1.4 Processing of Speech Signals (Optional)

For this section download the speech.au file. For instructions on how to load and play audio signals see audio.pdf.

Digital signal processing is widely used in speech processing for applications ranging from speech compression and transmission, to speech recognition and speaker identification. This exercise will introduce the process of reading and manipulating a speech signal.

First download the speech audio file speech.au, and then do the following:
1. Use the **audioread** command to load the file **speech.au** into Matlab.
2. Plot the signal on the screen as if it were a continuous-time signal (i.e. use the **plot** command).
3. Play the signal via the digital-to-analog converter in your computer with the Matlab **sound** function.

INLAB REPORT: Submit your plot of the speech signal.

1.5 Special Functions

Plot the following two continuous-time functions over the specified intervals. Write separate script files if you prefer. Use the **subplot** command to put both plots in a single figure, and be sure to label the time axes.

- sinc (t) for t in [-10π, 10π]
- rect (t) for t in [−2, 2]

HINT: The function rect(t) may be computed in Matlab by using a Boolean expression. For example, if t = −10:0.1:10, then $y = rect(t)$ may be computed using the Matlab command **y=(abs(t)<=0.5)**.

Write an .m-script file to stem the following discrete-time function for a = 0.8, a = 1.0 and a = 1.5. Use the **subplot** command to put all three plots in a single figure. Issue the command **orient('tall')** just prior to printing to prevent crowding of the subplots.

- a^n (u [n] − u [n − 10]) for n in [−20, 20]

Repeat this procedure for the function

- cos(ωn) a^n u[n] for $\omega = \pi/4$, and n in [−1, 10]

HINT: The unit step function $y = u$ [n] may be computed in Matlab using the command **y = (n>=0)** ,
where n is a vector of time indices.

INLAB REPORT: Submit all three figures, for a total of 8 plots. Also submit the copies of your Matlab .m-files.

1.6 Sampling

The word **sampling** refers to the conversion of a continuous-time signal into a discrete-time signal. The signal is converted by taking its value, or sample, at uniformly spaced points in time. The time between two consecutive samples is called the sampling period. For example, a sampling period of 0.1 seconds implies that the value of the signal is stored every 0.1 seconds.

Consider the signal $f(t) = \sin(2\pi t)$. We may form a discrete-time signal, $x[n]$, by sampling this signal with a period of T_s. In this case,

$$x[n] = f(T_s n) = \sin(2\pi T_s n).$$

Use the **stem** command to plot the function $f(T_s n)$ defined above for the following values of T_s and n. Use the **subplot** command to put all the plots in a single figure, and scale the plots properly with the **axis** command.

1. T_s = 1/10, $0 \le n \le 100$; axis([0,100,−1,1])
2. T_s = 1/3, $0 \le n \le 30$; axis([0,30,−1,1])

3. $T_s = 1/2$, $0 \le n \le 20$; axis([0,20,–1,1])
4. $T_s = 10/9$, $0 \le n \le 9$; axis([0,9,–1,1])

INLAB REPORT: Submit a copy of the figure containing all four subplots. Discuss your results. How does the sampled version of the signal with $T_s = 1/10$ compare to those with $T_s = 1/3$, $T_s = 1/2$ and $T_s = 10/9$?

1.7 Random Signals

For help on the Matlab random function, see **random.pdf**.

The objective of this section is to show how two signals that "look" similar can be distinguished by computing their average over a large interval. This type of technique is used in signal demodulators to distinguish between the digits "1" and "0".

Generate two discrete-time signals called "sig1" and "sig2" of length 1,000. The samples of "sig1" should be independent, Gaussian random variables with mean 0 and variance 1. The samples of "sig2" should be independent, Gaussian random variables with mean 0.2 and variance 1. Use the Matlab command **random** or **randn** to generate these signals, and then plot them on a single figure using the **subplot** command. (Recall that an alternative name for a Gaussian random variable is a normal random variable.)

Next form a new signal "ave1(n)" of length 1,000 such that "ave1(n)" is the average of the vector "sig1(1:n)" (the expression sig1(1:n) returns a vector containing the first n elements of "sig1"). Similarly, compute "ave2(n)" as the average of "sig2(1:n)". Plot the signals "ave1(n)" and "ave2(n)" versus "n" on a single plot. Refer to help on the Matlab **plot** command for information on plotting multiple signals.

INLAB REPORT: Submit your plot of the two signals "sig1" and "sig2". Also submit your plot of the two signals "ave1" and "ave2". Comment on how the average values changes with n. Also comment on how the average values can be used to distinguish between random noise with different means.

1.8 2-D Signals (Optional)

For help on the following topics, see meshgrid.pdf, mesh.pdf and image.pdf.

So far we have only considered 1-D signals such as speech signals. However, 2-D signals are also very important in digital signal processing. For example, the elevation at each point on a map, or the color at each point on a photograph are examples of important 2-D signals. As in the 1-D case, we may distinguish between continuous-space and discrete-space signals. However in this section, we will restrict attention to discrete-space 2-D signals.

When working with 2-D signals, we may choose to visualize them as images or as 2-D surfaces in a 3-D space. To demonstrate the differences between these two approaches, we will use two different display techniques in Matlab. Do the following:

1. Use the **meshgrid** command to generate the discrete-space 2-D signal

$$f[m, n] = 255|\text{sinc}(0.2m)\sin(0.2n)|$$

for $-50 \le m \le 50$ and $-50 \le n \le 50$. See the help of meshgrid.pdf if you're unfamiliar with its usage.

2. Use the **mesh** command to display the signal as a surface plot.

3. Display the signal as an image. Use the command **colormap(gray(256))** just after issuing the **image** command to obtain a grayscale image. Read the help in image.pdf for more information.

INLAB REPORT: Hand in softcopies of your mesh plot and image. For which applications do you think the surface plot works better? When would you prefer the image?

Lab 2 - Discrete-Time Systems

2.1 Introduction

A discrete-time system is anything that takes a discrete-time signal as input and generates a discrete-time signal as output. The concept of a system is very general. It may be used to model the response of an audio equalizer or the performance of the US economy.

In electrical engineering, continuous-time signals are usually processed by electrical circuits described by differential equations. For example, any circuit of resistors, capacitors and inductors can be analyzed using mesh analysis to yield a system of differential equations. The voltages and currents in the circuit may then be computed by solving the equations.

The processing of discrete-time signals is performed by discrete-time systems. Similar to the continuous-time case, we may represent a discrete-time system either by a set of difference equations or by a block diagram of its implementation. For example, consider the following difference equation.

$$y(n) = y[n-1] + x[n] + x[n-1] + x[n-2] \tag{2.1}$$

This equation represents a discrete-time **system**. It operates on the input signal $x[n]$ to produce the output signal $y[n]$. This system may also be defined by a system diagram as in Figure 2.1.

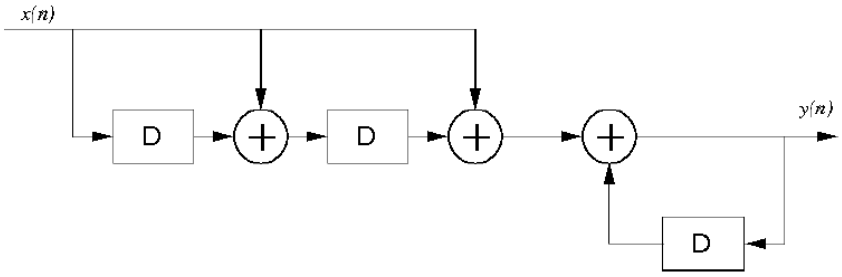


Figure 2.1: Diagram of a discrete-time system. (D = unit delay)

Mathematically, we use the notation $y = \boldsymbol{S}(x)$ to denote a discrete-time system $\boldsymbol{S}$ with input signal $x[n]$ and output signal $y[n]$. Notice that the input and output to the system are the complete signals for all time n. This is important since the output at a particular time can be a function of past, present and future values of $x[n]$.

It is usually quite straightforward to write a computer program to implement a discrete-time system from its difference equation. In fact, programmable computers are one of the easiest and most cost effective ways of implementing discrete-time systems.

While equation (2.1) is an example of a linear time-invariant system, other discrete-time systems may be nonlinear and/or time varying. In order to understand discrete-time systems, it is important to first understand their classification into categories of linear/nonlinear, time-invariant/ time-varying, causal/noncausal, memoryless/with-memory, and stable/unstable. Then it is possible to study the properties of restricted classes of systems, such as discrete-time systems which are linear, time-invariant and stable.

2.2 Background Exercises

INLAB REPORT: Submit these background exercises with the lab report.

2.2.1 Example Discrete-time Systems

Discrete-time digital systems are often used in place of analog processing systems. Common examples are the replacement of photographs with digital images, and conventional NTSC TV with direct broadcast digital TV. These digital systems can provide higher quality and/or lower cost through the use of standardized, high-volume digital processors.

The following two continuous-time systems are commonly used in electrical engineering:

$$\text{differentiator: } y(t) = \frac{d}{dt}x(t)$$

$$\text{integrator: } y(t) = \int_{-\infty}^{t} x(\tau)d\tau \tag{2.2}$$

For each of these two systems, do the following:

i. Formulate a discrete-time system that approximates the continuous-time function.
ii. Write down the difference equation that describes your discrete-time system. Your difference equation should be in closed form, i.e. no summations.
iii. Draw a block diagram of your discrete-time system as in Figure 2.1.

2.2.2 Stock Market Example

One reason that digital signal processing (DSP) techniques are so powerful is that they can be used for very different kinds of signals. While most continuous-time systems only process voltage and current signals, a computer can process discrete-time signals which are essentially just sequences of numbers. Therefore DSP may be used in a very wide range of applications. Let's look at an example.

A stockbroker wants to see whether the average value of a certain stock is increasing or decreasing. To do this, the daily fluctuations of the stock values must be eliminated. A popular business magazine recommends three possible methods for computing this average.

$$\text{avgvalue[today]} = \frac{1}{3}\ (\text{value[today]+value[yesterday]+value[2 days ago]}) \tag{2.3}$$

$$\text{avgvalue[today]} = 0.8 * \text{avgvalue[yesterday]} + 0.2 * (\text{value[today]}) \tag{2.4}$$

$$\text{avgvalue[today]} = \text{avgvalue[yesterday]} + \frac{1}{3}\ (\text{value[today]}–(\text{value[3 days ago]}) \tag{2.5}$$

Do the following:

- For each of these three methods: 1) write a difference equation, 2) draw a system diagram, and 3) calculate the impulse response.
- Explain why methods (2.3) and (2.5) are known as moving averages.

2.3 Example Discrete-Time Systems

Write two Matlab functions that will apply the differentiator and integrator systems, designed in the "Example Discrete-time Systems" (Section 2.2.1: Example Discrete-time Systems) section, to

arbitrary input signals. Then apply the differentiator and integrator to the following two signals for $-10 \leq n \leq 20$.

- $\delta[\mathrm{n}] - \delta[\mathrm{n}-5]$
- $u[\mathrm{n}] - u[n-(N+1)]$ with $N = 10$

HINT: To compute the function $u(n)$ for $-10 \leq n \leq 20$, first set **n = -10:20**, and then use the Boolean expression **u = (n>=0)**.

For each of the four cases, use the subplot and stem commands to plot each input and output signal on a single figure.

INLAB REPORT: Submit your Matlab code and softcopies containing the input and output signals. Discuss the stability of these systems.

2.4 Difference Equations

In this section, we will study the effect of two discrete-time filters. The first filter, $y = \boldsymbol{S}_1(x)$, obeys the difference equation

$$y[\mathrm{n}] = x[n] - x[n-1] \tag{2.6}$$

and the second filter, $y = \boldsymbol{S}_2(x)$, obeys the difference equation

$$y[\mathrm{n}] = \frac{1}{2}y[n-1] + x[n] \tag{2.7}$$

Write Matlab functions to implement each of these filters.

NOTE: In Matlab, when implementing a difference equation using a loop structure, it is very good practice to pre-define your output vector before entering into the loop. Otherwise, Matlab has to resize the output vector at each iteration. For example, say you are using a **for** loop to filter the signal $x[n]$, yielding an output $y[n]$. You can pre-define the output vector by issuing the command **y=zeros(1,N)** before entering the loop, where **N** is the final length of **y**. For long signals, this speeds up the computation dramatically.

Now use these functions to calculate the impulse response of each of the following 5 systems: $\boldsymbol{S}_1$, $\boldsymbol{S}_2$, $\boldsymbol{S}_1$ ($\boldsymbol{S}_2$) (i.e., the series connection with $\boldsymbol{S}_1$ following $\boldsymbol{S}_2$), $\boldsymbol{S}_2$ ($\boldsymbol{S}_1$) (i.e., the series connection with $\boldsymbol{S}_2$ following $\boldsymbol{S}_1$), and $\boldsymbol{S}_1 + \boldsymbol{S}_2$.

INLAB REPORT: For each of the five systems, draw and submit a system diagram (use only delays, multiplications and additions as in Figure 2.1). Also submit plots of each impulse response. Discuss your observations.

2.5 Audio Filtering

For this section download the music.au file. For help on how to play audio signals see audio.pdf.

Use the command **audioread** to load the file music.au into Matlab. Then use the Matlab

function sound to listen to the signal.

Next filter the audio signal with each of the two systems $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ from the previous section. Listen to the two filtered signals.

INLAB REPORT: How do the filters change the sound of the audio signals? Explain your observations.

2.6 Inverse Systems

Consider the system $y = \boldsymbol{S}_2(x)$ from the "Difference Equations" (Section 2.4: Difference Equations) section. Find a difference equation for a new system $y = \boldsymbol{S}_3(x)$ such that $\delta = \boldsymbol{S}_3(\boldsymbol{S}_2(x))$ where δ denotes the discrete-time impulse function $\delta[\mathrm{n}]$. Since both systems $\boldsymbol{S}_2$ and $\boldsymbol{S}_3$ are LTI, the time-invariance and superposition properties can be used to obtain $x = \boldsymbol{S}_3(\boldsymbol{S}_2(x))$ for any discrete-time signal x. We say that the systems $\boldsymbol{S}_3$ and $\boldsymbol{S}_2$ are inverse filters because they cancel out the effects of each other.

HINT: The system $y = \boldsymbol{S}_3(x)$ can be described by the difference equation

$$y[\mathrm{n}] = ax[n] + bx[n-1] \tag{2.8}$$

where a and b are constants.

Write a Matlab function $y = \boldsymbol{S}_3(x)$ which implements the system $\boldsymbol{S}_3$. Then obtain the impulse response of both $\boldsymbol{S}_3$ and $\boldsymbol{S}_3(\boldsymbol{S}_2(x))$.

INLAB REPORT: Draw a system diagram for the system $\boldsymbol{S}_3$, and submit plots of the impulse responses for $\boldsymbol{S}_3$ and $\boldsymbol{S}_3(\boldsymbol{S}_2)$.

2.7 System Tests

For this section download the zip file bbox.zip.

Often it is necessary to determine if a system is linear and/or time-invariant. If the inner workings of a system are not known, this task is impossible because the linearity and time-invariance properties must hold true for all possible inputs signals. However, it is possible to show that a system is non-linear or time-varying because only a single instance have to be found where the properties are violated. The zip file bbox.zip contains three "black-box" systems in the files bbox1.p, bbox2.p, and bbox3.p.

These files work as Matlab functions, with the syntax **y=bboxN(x)**, where **x** and **y** are the input and the output signals, and **N = 1, 2 or 3**. Exactly one of these systems is non-linear, and exactly one of them is time-varying. Your task is to find the non-linear system and the time-varying system.

HINTS:

1. You should try a variety of input signals until you find a counter-example.
2. When testing for time-invariance, you need to look at the responses to a signal and to its delayed version. Since all your signals in MATLAB have finite duration, you should be very careful about shifting signals. In particular, if you want to shift a signal x by M samples to the left, x should start with at least M zeros. If you want to shift x by M samples to the right, x should end with at least M zeros.
3. When testing for linearity, you may find that simple inputs such as the unit impulse do not

accomplish the task. In this case, you should try something more complicated like a sinusoid or a random signal generated with the random command.

INLAB REPORT: State which system is non-linear, and which system is time-varying. Submit plots of input/output signal pairs that support your conclusions. Indicate on the plots why they support your conclusions.

2.8 Stock Market Example

For this section download stockrates.mat. For help on loading Matlab files see load.pdf.

Load stockrates.mat into Matlab. This file contains a vector, called **rate**, of daily stock market exchange rates for a publicly traded stock. Apply filters (2.4) and (2.5) from the "Stock Market Example" (Section 2.2.2: Stock Market Example) section of the background exercises to smooth the stock values. When you apply the filter of (2.4) you will need to initialize the value of avgvalue(yesterday). Use an initial value of 0. Similarly, in (2.5), set the initial values of the "value" vector to 0 (for the days prior to the start of data collection). Use the subplot command to plot the original stock values, the result of filtering with (2.4), and the result of filtering with (2.5).

INLAB REPORT: Submit your plots of the original and filtered exchange-rates. Discuss the advantages and disadvantages of the two filters. Can you suggest a better method for initializing the filter outputs?

Lab 3 - Frequency Analysis

3.1 Introduction

In this experiment, we will use Fourier series and Fourier transforms to analyze continuous-time and discrete-time signals and systems. The Fourier representations of signals involve the decomposition of the signal in terms of complex exponential functions. These decompositions are very important in the analysis of linear time-invariant (LTI) systems, due to the property that the response of an LTI system to a complex exponential input is a complex exponential of the same frequency! Only the amplitude and phase of the input signal are changed. Therefore, studying the frequency response of an LTI system gives complete insight into its behavior.

In this experiment and others to follow, we will use the Simulink extension to Matlab. Simulink is an icon-driven dynamic simulation package that allows the user to represent a system or a process by a block diagram. Once the representation is completed, Simulink may be used to digitally simulate the behavior of the continuous or discrete-time system. Simulink inputs can be Matlab variables from the workspace, or waveforms or sequences generated by Simulink itself. These Simulink-generated inputs can represent continuous-time or discrete-time sources. The behavior of the simulated system can be monitored using Simulink's version of common lab instruments, such as scopes, spectrum analyzers and network analyzers.

3.2 Background Exercises

INLAB REPORT: Submit these background exercises with the lab report.

3.2.1 Synthesis of Periodic Signals

Each signal given below represents one period of a periodic signal with period T_0.

1. Period $T_0 = 2$. For $t \in [0,2]$:

$$s(t) = \text{rect}\left(t - \frac{1}{2}\right)$$

2.Period $T_0 = 1$. For $t \in \left[-\frac{1}{2}, \frac{1}{2}\right]$:

$$\text{s}(t) = \text{rect}(2t) - \frac{1}{2}$$

For each of these two signals, do the following:

i: Compute the Fourier series expansion in the form

$$\text{s}(t) = a_0 + \sum_{k=1}^{\infty} A_k \sin(2\pi k f_0 t + \theta_k)$$

Where $f_0 = 1/T_0$.

HINT: You may want to use the following reference: Sec. 3.3 of "Signals and Systems", Oppenheim and Willsky, 1997. Note that in the expression above, the function in the summation is $\sin(2\pi k f_0 t + \theta_k)$, rather than a complex sinusoid. The formulas in the above references must be modified to accommodate this. You can compute the cos/sin version of the Fourier series, then convert the coefficients.

ii: Sketch the signal on the interval $[0, T_0]$.

3.3 Getting Started with Simulink

In this section, we will learn the basics of Simulink and build a simple system.

For help on "Simulink" see simulink.pdf. For the following sections download the file Lab3Utilities.zip.

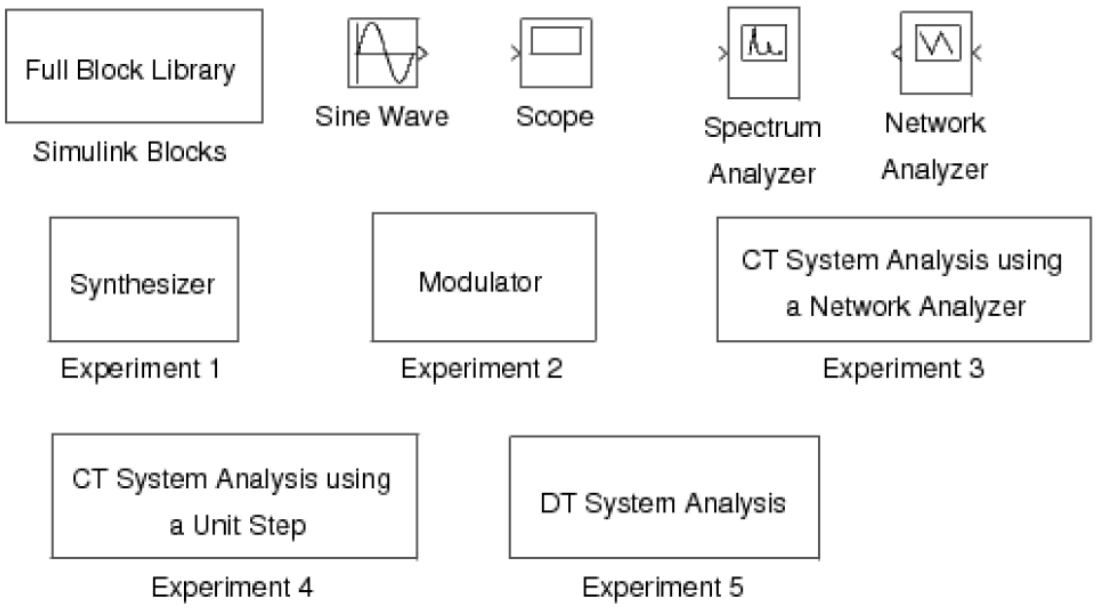


Figure 3.1: Simulink utilities for lab 3.

To get the library of Simulink functions for this laboratory, download the leLab3Utilities.zip. Once Matlab is started, type"Lab3" to bring up the library of Simulink components shown in Figure 3.1. This library contains a full library of Simulink blocks, a spectrum analyzer and network analyzer designed for this laboratory, a sine wave generator, a scope, and pre-design systems for each of the experiments that you will be running.

In order to familiarize yourself with Simulink, you will first build the system shown in Figure 3.2. This system consists of a sine wave generator that feeds a scope and a spectrum analyzer.

1. Open a window for a new system by using the **New** option from the **File** pull-down menu, and select **Simulink Model**.

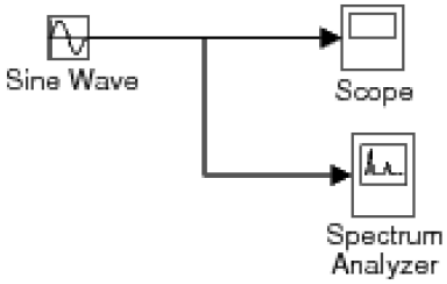


Figure 3.2: Simulink model for the introductory example.

2. Drag the **Sine Wave**, **Scope**, and **Spectrum Analyzer** blocks from the **Lab3** window into the new window you created.

3. Now you need to connect these three blocks. With the left mouse button, click on the output of the **Sine Wave** and drag it to the input of the **Scope**. Now use the right button to click on the line you just created, and drag to the input of the **Spectrum Analyzer** block. Your system should now look like Figure 3.2.

4. Double click on the **Scope** block to make the plotting window for the scope appear.

5. Set the simulation parameters by selecting **Model Configuration Parameters** from the **Simulation** pull-down menu. Under the **Solver** tab, set the **Stop time** to 50, and the **Max step size** to 0.02. Then select **OK**. This will allow the **Spectrum Analyzer** to make a more accurate calculation.

6. Start the simulation by using the **Run** option from the **Simulation** pull-down menu. A standard Matlab figure window will pop up showing the output of the **Spectrum Analyzer**.

7. Change the frequency of the sine wave to **5*pi rad/sec** by double clicking on the **Sine Wave** icon and changing the number in the **Frequency** field. Restart the simulation. Observe the change in the waveform and its spectral density. If you want to change the time scaling in the plot generated by the spectrum analyzer, from the Matlab prompt use the **subplot(2,1,1)** and **axis()** commands.

8. When you are done, close the system window you created by using the **Close** option from the **File** pull-down menu.

3.4 Continuous-Time Frequency Analysis

For help on the following topics see the corresponding files: simulink.pdf or printing.pdf.

In this section, we will study the use and properties of the continuous-time Fourier transform with Simulink. The Simulink package is especially useful for continuous-time systems because it allows the simulation of their behavior on a digital computer.

3.4.1 Synthesis of Periodic Signals

Double click the icon labeled **Synthesizer** in Lab3 window to bring up a model as shown in Figure 3.3. This system may be used to synthesize periodic signals by adding together the harmonic

components of a Fourier series expansion. Each Sin Wave block can be set to a specific frequency, amplitude and phase. The initial settings of the **Sin Wave** blocks are set to generate the Fourier series expansion

$$x(t) = 0 + \sum_{\substack{k=1 \\ k \text{ odd}}}^{13} \frac{4}{k\pi} \sin(2\pi k t)$$

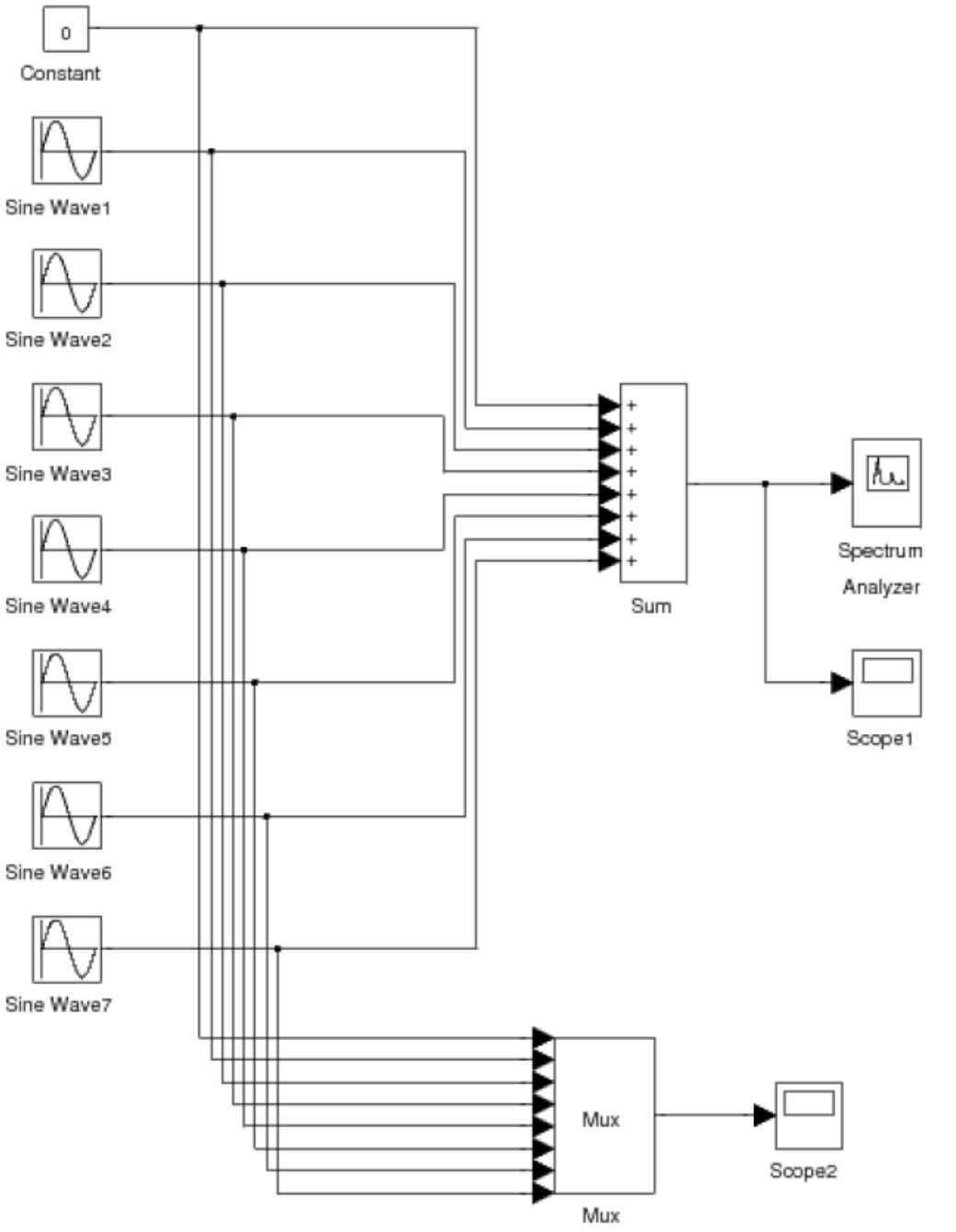


Figure 3.3: Simulink model for the synthesizer experiment.

These are the first 8 terms in the Fourier series of the periodic square wave shown in Figure 3.4.

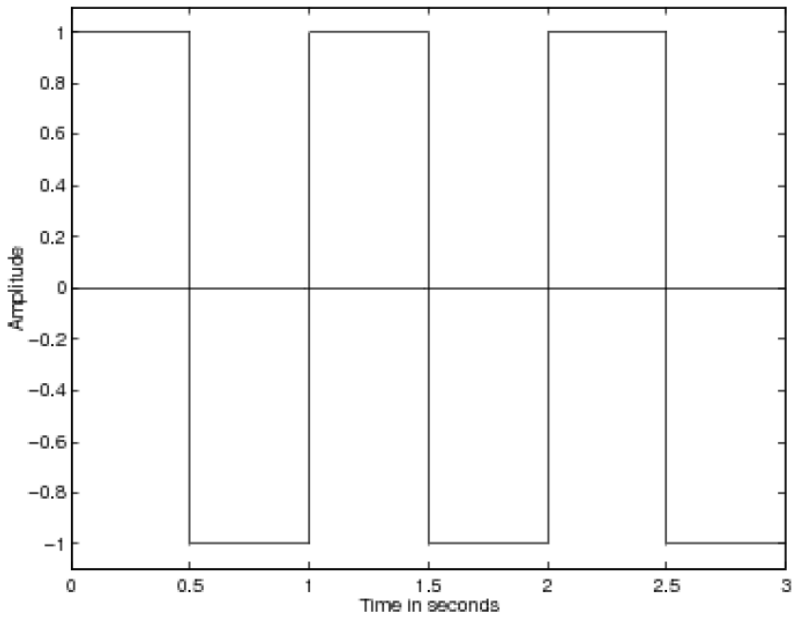


Figure 3.4: The desired waveform for the synthesizer experiment.

Run the model by selecting **Run** under the **Simulation** menu. A graph will pop up that shows the synthesized square wave signal and its spectrum. This is the output of the **Spectrum Analyzer**. After the simulation runs for a while, the **Spectrum Analyzer** element will update the plot of the spectral energy and the incoming waveform. Notice that the energy is concentrated in peaks corresponding to the individual sine waves. Print the output of the **Spectrum Analyzer**.

You may have a closer look at the synthesized signal by double clicking on the **Scope1** icon. You can also see a plot of all the individual sine waves by double clicking on the **Scope2** icon.

Synthesize the two periodic waveforms defined in the "Synthesis of Periodic Signals" (Section 3.2.1: Synthesis of Periodic Signals) section of the background exercises. Do this by setting the frequency, amplitude, and phase of each sine wave generator to the proper values. For each case, print the output of the **Spectrum Analyzer**.

INLAB REPORT: Hand in plots of the **Spectrum Analyzer** output for each of the three synthesized waveforms. For each case, comment on how the synthesized waveform differs from the desired signal, and on the structure of the spectral density.

3.4.2 Modulation Property

Double click the icon labeled **Modulator** in Lab3 window to bring up a system as shown in Figure 3.5. This system modulates a triangular pulse signal with a sine wave. You can control the duration and duty cycle of the triangular envelope and the frequency of the modulating sine wave. The system also contains a spectrum analyzer which plots the modulated signal and its spectrum.

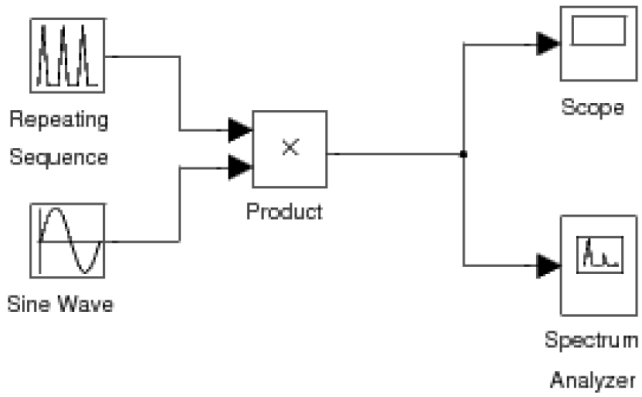


Figure 3.5: Simulink model for the modulation experiment.

Generate the following signals by adjusting the **Time values** and **Output values** of the **Repeating Sequence** block and the **Frequency** of the **Sine Wave**. The **Time values** vector contains entries spanning one period of the repeating signal. The **Output values** vector contains the values of the repeating signal at the times specified in the **Time values** vector. Note that the **Repeating Sequence** block does NOT create a discrete time signal. It creates a continuous time signal by connecting the output values with line segments.
Print the output of the **Spectrum Analyzer** for each signal.

1. Triangular pulse duration of 1 sec; period of 2 sec; modulating frequency of 10 Hz (initial settings of the experiment).
2. Triangular pulse duration of 1 sec; period of 2 sec; modulating frequency of 15 Hz.
3. Triangular pulse duration of 1 sec; period of 3 sec; modulating frequency of 10 Hz.
4. Triangular pulse duration of 1 sec; period of 6 sec; modulating frequency of 10 Hz.

Notice that the spectrum of the modulated signal consists of a comb of impulses in the frequency domain, arranged around a center frequency.

INLAB REPORT: Hand in plots of the output of the **Spectrum Analyzer** for each signal. Answer following questions: 1) What effect does changing the modulating frequency have on the spectral density? 2) Why does the spectrum have a comb structure and what is the spectral distance between impulses? Why? 3) What would happen to the spectral density if the period of the triangle pulse were to increase toward infinity? (in the limit)

3.5 Discrete-Time Frequency Analysis

In this section of the laboratory, we will study the use of the discrete-time Fourier transform.

3.5.1 Discrete-Time Fourier Transform

The DTFT (Discrete-Time Fourier Transform) is the Fourier representation used for finite energy discrete-time signals. For a discrete-time signal,$x(n)$, we denote the DTFT as the function $X(e^{j\omega})$given by the expression

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

Since $X(e^{j\omega})$ is a periodic function of ω with a period of 2π, we need only to compute $X(e^{j\omega})$ for $-\pi < \omega < \pi$.

Write a Matlab function X=DTFT(x, n0, dw) that computes the DTFT of the discrete-time signal **x**. Here **n0** is the time index corresponding to the 1st element of the **x** vector, and **dw** is the spacing between the samples of the Matlab vector **X**. For example, if **x** is a vector of length **N**, then its DTFT is computed by

$$X(\omega) = \sum_{n=1}^{N} x[n]e^{-j\omega(n+n0-1)}$$

Where ω is a vector of values formed by **w=(–pi:dw:pi)** .

HINT: In Matlab, *j* or *i* is defined as $\sqrt{-1}$. However, you may also compute this value using the Matlab expression i=sqrt(–1).

For the following signals use your DTFT function to

i: Compute $X(e^{j\omega})$

ii: Plot the magnitude and the phase of $X(e^{j\omega})$ in a single plot using the subplot command.

HINT : Use the abs() and angle() commands.

1. $$x(n) = \delta(n)$$
2. $$x(n) = \delta(n-5)$$
3. $$x(n) = (0.5)^n u(n)$$

INLAB REPORT: Hand in a copy of your Matlab function. Also hand in plots of the DTFT's magnitude and phase for each of the three signals.

Lab 4 - Discrete Fourier Transform

4.1 Introduction

This laboratory will introduce the Discrete Fourier Transform (DFT) and the associated sampling and windowing effects.

In previous laboratories, we have used the Discrete-Time Fourier Transform (DTFT) extensively for analyzing signals and linear time-invariant systems.

$$\text{(DTFT)} \qquad X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]\, e^{-j\omega n} \tag{4.1}$$

$$\text{(inverse DTFT)} \quad x[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X\left(e^{j\omega}\right) e^{j\omega n} d\omega. \tag{4.2}$$

While the DTFT is very useful **analytically**, it usually cannot be exactly evaluated on a computer because (4.1) requires an infinite sum and (4.2) requires the evaluation of an integral.

The discrete Fourier transform (DFT) is a sampled version of the DTFT, hence it is better suited for numerical evaluation on computers.

$$\text{(DFT)} \qquad X_N[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N} \tag{4.3}$$

$$\text{(inverse DFT)} \quad x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X_N[k] e^{j2\pi kn/N} \tag{4.4}$$

Here $X_N[k]$ is an N point DFT of $x[n]$. Note that $X_N[k]$ is a function of a discrete integer k, where k ranges from 0 to $N-1$.

In the following sections, we will study the derivation of the DFT from the DTFT, and several DFT implementations.

4.2 Deriving the DFT from the DTFT

4.2.1 Truncating the Time-domain Signal

The DTFT usually cannot be computed exactly because the sum in (4.1) is infinite. However, the DTFT may be approximately computed by truncating the sum to a finite window. Let $w[n]$ be a rectangular window of length N:

$$w[n] = \begin{cases} 1 & 0 \le n \le N-1 \\ 0 & \text{otherwise} \end{cases}. \tag{4.5}$$

Then we may define a truncated signal to be

$$x_{\text{tr}}[n] = w[n]x[n]$$

(4.6)

The DTFT of $x_{\text{tr}}[n]$ is given by:

$$X_{\text{tr}}(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x_{\text{tr}}[n]e^{-j\omega n} = \sum_{n=0}^{N-1} x[n]\, e^{-j\omega n}$$

(4.7)

We would like to compute $X(e^{j\omega})$, but the truncation window distorts the desired frequency characteristics; $X(e^{j\omega})$ and $X_{\text{tr}}(e^{j\omega})$ are generally not equal. To understand the relation between these two DTFT's, we need to convolve in the frequency domain:

$$X_{\text{tr}}\left(e^{j\omega}\right) = \frac{1}{2\pi}\int_{-\pi}^{\pi} X\left(e^{j\sigma}\right) W(e^{j(\omega-\sigma)})d\sigma$$

(4.8)

where $W\left(e^{j\omega}\right)$ is the DTFT of $w[n]$. Eq. (4.8) is the periodic convolution of $X\left(e^{j\omega}\right)$ and $W\left(e^{j\omega}\right)$. Hence the true DTFT, $X\left(e^{j\omega}\right)$, is smoothed via convolution with $W\left(e^{j\omega}\right)$ to produce the truncated DTFT, $X_{\text{tr}}\left(e^{j\omega}\right)$.

We can calculate $W\left(e^{j\omega}\right)$:

$$\begin{aligned} W\left(e^{j\omega}\right) &= \sum_{n=-\infty}^{\infty} w[n]\, e^{-j\omega n} = \sum_{n=0}^{N-1} e^{-j\omega n} \\ &= \begin{cases} \dfrac{1-e^{-j\omega N}}{1-e^{-j\omega}}, & \text{for } \omega \neq 0, \pm 2\pi, \ldots \\ N, & \text{for } \omega = 0, \pm 2\pi, \ldots \end{cases} \end{aligned}$$

(4.9)

For $\omega \neq 0, \pm 2\pi$, ..., we have:

$$W(e^{j\omega}) = \frac{e^{-j\omega N/2}}{e^{-j\omega/2}} \frac{e^{j\omega N/2} - e^{-j\omega N/2}}{e^{j\omega/2} - e^{-j\omega/2}} = e^{-j\omega(N-1)/2} \frac{sin\,(\omega N/2)}{sin\,(\omega/2)}$$

(4.10)

Notice that the magnitude of this function is similar to sinc $(\omega N/2)$ except that it is periodic in ω with period 2π.

4.2.2 Frequency Sampling

Eq. (4.7) contains a summation over a finite number of terms. However, we can only evaluate (4.7) for a finite set of frequencies, ω. We must sample in the frequency domain to compute the DTFT on a computer. We can pick any set of frequency points at which to evaluate (4.7), but it is particularly useful to uniformly sample ω at N points, in the range $[0,2\pi)$. If we substitute

$$\omega = 2\pi k/N$$

(4.11)

for $k = 0,1, \ldots (N-1)$ in (7.7), we find that

$$X_{\text{tr}}\left(e^{j\omega}\right) \mid_{\omega=\frac{2\pi k}{N}} = \sum_{n=0}^{N-1} x[n]e^{-j\omega n} \mid_{\omega=\frac{2\pi k}{N}} = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N} = X_N[k]$$

(4.12)

In short, the DFT values result from sampling the DTFT of the truncated signal.

$$X_N[k] = X_{\text{tr}}(e^{j2\pi k/N}) \tag{4.13}$$

4.2.3 Windowing Effects

Download DTFT.m for the following section.

We will next investigate the effect of windowing when computing the DFT of the signal $x[n] = \cos\left(\frac{\pi}{4}n\right)$ truncated with a window of size $N = 20$.

1. In the same figure, plot the phase and magnitude of $W(e^{j\omega})$, using equations (4.9) and (4.10).
2. Determine an expression for $X(e^{j\omega})$ (the DTFT of the non-truncated signal).
3. Truncate the signal $x[n]$ using a window of size $N = 20$ and then use DTFT.m to compute $X_{\text{tr}}(e^{j\omega})$.
   Make sure that the plot contains a least 512 points.
   HINT: Use the command [X ,w] = DTFT(x,512) .
4. Plot the magnitude of $X_{\text{tr}}(e^{j\omega})$.

INLAB REPORT:

1. Submit the plot of the phase and magnitude of $W(e^{j\omega})$.
2. Submit the analytical expression for $X(e^{j\omega})$.
3. Submit the magnitude plot of $X_{\text{tr}}(e^{j\omega})$.
4. Describe the difference between $|X_{\text{tr}}(e^{j\omega})|$ and $|X(e^{j\omega})|$. What is the reason for this difference?
5. Comment on the effects of using a different length of the window $w(n)$.

4.3 The Discrete Fourier Transform

4.3.1 Computing the DFT

We will now develop our own DFT functions to help our understanding of how the DFT comes from the DTFT. Write your own Matlab function to implement the DFT of equation (4.3). Use the syntax

X = DFTsum(x)

where **x** is an N point vector containing the values $x(0), \ldots, x(N-1)$ and **X** is the corresponding DFT. Your routine should implement the DFT exactly as specified by (4.3) using for-loops for n and k, and compute the exponentials as they appear. Note: In Matlab, "j" may be computed with the command

j=sqrt(-1) . If you use $j = \sqrt{-1}$, remember not to use *j* as an index in your for-loop.

Test your routine DFTsum by computing $X_N[k]$ for each of the following cases:

1. $x[n] = \delta[n]$ for $N = 10$.
2. $x[n] = 1$ for $N = 10$.

3. $x[n] = e^{j2\pi n/10}$ for $N = 10$.
4. $x[n] = \cos(2\pi n/10)$ for $N = 10$.

Plot the magnitude of each of the DFT's. In addition, derive simple closed-form analytical expressions for the DFT (not the DTFT!) of each signal.

INLAB REPORT:

1. Submit a listing of your code for **DFTsum**.
2. Submit the magnitude plots.
3. Submit the corresponding analytical expressions.

Write a second Matlab function for computing the inverse DFT of (4.4). Use the Syntax

x = IDFTsum(X)

where **X** is the N point vector containing the DFT and **x** is the corresponding time-domain signal. Use **IDFTsum** to invert each of the DFT's computed in the previous problem. Plot the magnitudes of the inverted DFT's, and verify that those time-domain signals match the original ones. Use **abs(x)** to eliminate any imaginary parts which roundoff error may produce.

INLAB REPORT:

1. Submit the listing of your code for **IDFTsum**.
2. Submit the four time-domain IDFT plots.

4.3.2 Matrix Representation of the DFT

The DFT of (4.3) can be implemented as a matrix-vector product. To see this, consider the equation

$$\boldsymbol{X} = \mathbf{A}\boldsymbol{x} \tag{4.14}$$

where $\mathbf{A}$ is an $N \times N$ matrix, and both $\boldsymbol{X}$ and x are $N \times 1$ column vectors. This matrix product is equivalent to the summation

$$\boldsymbol{X}_k = \sum_{n=1}^{N} \mathbf{A}_{kn}\, \boldsymbol{x}_n \tag{4.15}$$

where $\mathbf{A}_{kn}$ is the matrix element in the k^{th} row and n^{th} column of $\mathbf{A}$. By comparing (4.3) and (4.15) we see that for the DFT,

$$\mathbf{A}_{kn} = e^{-j2\pi(k-1)(n-1)/N} \tag{4.16}$$

The $-1'$s are in the exponent because Matlab indices start at 1, not 0. For this section, we need to:

- Write a Matlab function **A = DFTmatrix(N)** that returns the *N* x *N* DFT matrix **A**.
  NOTE: Remember that the symbol ' $*$ ' is used for matrix multiplication in Matlab, and that '**.'** ' performs a simple transpose on a vector or matrix. An apostrophe without the period is a conjugate transpose.
- Use the matrix $\mathbf{A}$ to compute the DFT of the following signals. Confirm that the results are the same as in the previous section.
  a. $x[n] = \delta[n]$ for $N = 10$

b. $x[n] = 1 \quad$ for $N = 10$
c. $x[n] = e^{j2\pi n/N}$ for $N = 10$

INLAB REPORT:

1. Print out the matrix $\mathbf{A}$ for $N = 5$.
2. Hand in the three magnitude plots of the DFT's.
3. How many multiplies are required to compute an N point DFT using the matrix method? (Consider a multiply as the multiplication of either complex or real numbers.)

As with the DFT, the inverse DFT may also be represented as a matrix-vector product.

$$\boldsymbol{x} = \mathbf{B}\boldsymbol{X} \tag{4.18}$$

For this section,

1. Write an analytical expression for the elements of the inverse DFT matrix $\mathbf{B}$, using the form of (4.16).
2. Write a Matlab function **B = IDFTmatrix(N)** that returns the *N* x *N* inverse DFT matrix $\mathbf{B}$.
3. Compute the matrices $\mathbf{A}$ and $\mathbf{B}$ for $N = 5$. Then compute the matrix product $\mathbf{C} = \mathbf{BA}$.

INLAB REPORT:

1. Hand in your analytical expression for the elements of $\mathbf{B}$.
2. Print out the matrix $\mathbf{B}$ for $N = 5$.
3. Print out the elements of $\mathbf{C} = \mathbf{BA}$. What form does $\mathbf{C}$ have? Why does it have this form?

4.3.3 Computation Time Comparison

Read cputime.pdf for help on the **cputime** function.

Although the operations performed by **DFTsum** are mathematically identical to a matrix product, the computation times for these two DFT's in Matlab are quite different. (This is despite the fact that the computational complexity of two procedures is of the same order!) This exercise will underscore why you should try to avoid using **for loops** in Matlab, and wherever possible, try to formulate your computations using matrix/vector products.

To see this, do the following:

1. Compute the signal $x[n] = \cos\left(\frac{2\pi n}{10}\right)$ for $N = 512$.
2. Compute the matrix $\mathbf{A}$ for $N = 512$.
3. Compare the computation time of X = DFTsum(x) with a matrix implementation **X = A*x** by using the **cputime** function before and after the program execution. Do not include the computation of **A** in your timing calculations.

INLAB REPORT: Report the CPU time required for each of the two implementations. Which method is faster? Which method requires less storage?

Lab 5 – Fast Fourier Transform

5.1 Introduction

The last laboratory covers the Discrete Fourier Transform (DFT). This laboratory will continue the discussion of the DFT and will introduce the FFT.

5.2 Continuation of DFT Analysis

This section continues the analysis of the DFT started in the previous week's laboratory.

$$\text{(DFT)} \quad X_N[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N}$$
$$\text{(inverse DFT)} \quad x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X_N[k]\, e^{j2\pi kn/N} \tag{5.1}$$

5.2.1 Shifting the Frequency Range

In this section, we will illustrate a representation for the DFT of (5.1) that is a bit more intuitive. First create a Hamming window x of length $N = 20$, using the Matlab command x = hamming(20) . Then use your Matlab function DFTsum to compute the 20 point DFT of x. Plot the magnitude of the DFT, $|X_{20}[k]|$, versus the index k. Remember that the DFT index k starts at 0 not 1!

> INLAB REPORT: Hand in the plot of the $|X_{20}[k]|$. Circle the regions of the plot corresponding to low frequency components.

Our plot of the DFT has two disadvantages. First, the DFT values are plotted against k rather then the frequency ω. Second, the arrangement of frequency samples in the DFT goes from 0 to 2π rather than from $-\pi$ to π, as is conventional with the DTFT. In order to plot the DFT values similar to a conventional DTFT plot, we must compute the vector of frequencies in radians per sample, and then "rotate" the plot to produce the more familiar range, $-\pi$ to π.

Let's first consider the vector w of frequencies in radians per sample. Each element of w should be the frequency of the corresponding DFT sample $X(k)$, which can be computed by

$$\omega = 2\pi k/N \quad k \in [0, \dots, N-1] \tag{5.2}$$

However, the frequencies should also lie in the range from $-\pi$ to π. Therefore, if $\omega \geq \pi$, then it should be set to $\omega - 2\pi$. An easy way of making this change in Matlab is (w>=pi) = w(w>=pi)-2*pi .

The resulting vectors X and w are correct, but out of order. To reorder them, we must swap the first and second halves of the vectors. Fortunately, Matlab provides a function specifically for this purpose, called **fftshift**.

Write a Matlab function to compute samples of the DTFT and their corresponding frequencies in the range $-\pi$ to π. Use the syntax

[X,w] = DTFTsamples(x)

where x is an N point vector, X is the length N vector of DTFT samples, and w is the length N vector of corresponding radial frequencies. Your function DTFT samples should call your function DFTsum and use the Matlab function fftshift.

Use your function DTFT samples to compute DTFT samples of the Hamming window of length $N = 20$. Plot the magnitude of these DTFT samples versus frequency in rad/sample.

INLAB REPORT: Hand in the code for your function DTFTsamples. Also hand in the plot of the magnitude of the DTFT samples.

5.2.2 Zero Padding

The spacing between samples of the DTFT is determined by the number of points in the DFT. This can lead to surprising results when the number of samples is too small. In order to illustrate this effect, consider the finite-duration signal

$$x[n] = \begin{cases} \sin(0.1\pi n) & 0 \le n \le 49 \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

In the following, you will compute the DTFT samples of $x(n)$ using both $N = 50$ and $N = 200$ point FT's. Notice that when $N = 200$, most of the samples of $x[n]$ will be zeros because $x[n] = 0$ for $n \ge 50$.This technique is known as "zero padding", and may be used to produce a finer sampling of the DTFT.

For $N = 50$ and $N = 200$, do the following:

1. Compute the vector x containing the values $x[0], \dots, x[N-1]$.
2. Compute the samples of $X[k]$ using your function DTFTsamples.
3. Plot the magnitude of the DTFT samples versus frequency in rad/sample.

INLAB REPORT: Submit your two plots of the DTFT samples for $N = 50$ and $N = 200$. Which plot looks more like the true DTFT? Explain why the plots look so different.

5.3 The Fast Fourier Transform Algorithm

We have seen in the preceding sections that the DFT is a very computationally intensive operation. In 1965, Cooley and Tukey ([1]) published an algorithm that could be used to compute the DFT much more efficiently. Various forms of their algorithm, which came to be known as the fast Fourier transform (FFT), had actually been developed much earlier by other mathematicians (even dating back to Gauss). It was their paper, however, which stimulated a revolution in the field of signal processing.

It is important to keep in mind at the outset that the FFT is **not** a new transform. It is simply a very efficient way to compute an existing transform, namely the DFT. As we saw, a straightforward implementation of the DFT can be computationally expensive because the number of multiplies grows as the square of the input length (i.e. N^2 for an N point DFT). The FFT reduces this computation using two simple but important concepts. The first concept, known as divide-and-conquer, splits the problem into two smaller problems. The second concept, known as

recursion, applies this divide-and-conquer method repeatedly until the problem is solved.

Consider the defining equation for the DFT and assume that N is even, so that $N/2$ is an integer:

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N} \tag{5.4}$$

Here we have dropped the subscript of N in the notation for $X[k]$. We will also use the notation

$$X[k] = \text{DFT}_N\{x[n]\} \tag{5.5}$$

to denote the N point DFT of the signal $x[n]$.

Suppose we break the sum in (5.4) into two sums, one containing all the terms for which n is even, and one containing all the terms for which n is odd:

$$X[k] = \sum_{\substack{n=0 \\ n \text{ even}}}^{N-1} x[n]e^{-j2\pi kn/N} + \sum_{\substack{n=0 \\ n \text{ odd}}}^{N-1} x[n]e^{-j2\pi kn/N} \tag{5.6}$$

We can eliminate the conditions "n even" and "n odd" in (5.6) by making a change of variable in each sum. In the first sum, we replace n by $2m$. Then as we sum m from 0 to $N/2-1, n = 2m$ will take on all even integer values between 0 and $N-2$. Similarly, in the second sum, we replace n by $2m+1$. Then as we sum m from 0 to $N/2-1, n = 2m+1$ will take on all odd integer values between 0 and $N-1$. Thus, we can write

$$X[k] = \sum_{m=0}^{N/2-1} x[2m]\, e^{-j2\pi k2m/N} + \sum_{m=0}^{N/2-1} x[2m+1]e^{-j2\pi k(2m+1)/N} \tag{5.7}$$

Next we rearrange the exponent of the complex exponential in the first sum, and split and rearrange the exponent in the second sum to yield

$$X[k] = \sum_{m=0}^{N/2-1} x[2m]\, e^{-j2\pi km/(N/2)} + e^{-j2\pi k/N} \sum_{m=0}^{N/2-1} x[2m+1]e^{-j2\pi km/(N/2)} \tag{5.8}$$

Now compare the first sum in (5.8) with the definition for the DFT given by (5.4). They have exactly the same form if we replace N everywhere in (5.4) by $N/2$. Thus the first sum in (5.8) is an $N/2$ point DFT of the even-numbered data points in the original sequence. Similarly, the second sum in (5.8) is an $N/2$ point DFT of the odd-numbered data points in the original sequence. To obtain the N point DFT of the complete sequence, we multiply the DFT of the odd-numbered data points by the complex exponential factor $e^{-j2\pi k/N}$, and then simply sum the two $N/2$ point DFTs.

To summarize, we will rewrite (5.8) according to this interpretation. First, we define two new $N/2$ point data sequences $x_0[n]$ and $x_1[n]$, which contain the even and odd-numbered data points from the original N point sequence, respectively:

$$x_0[n] = x[2n]$$
$$x_1[n] = x[2n+1]$$

(5.9)

where $n = 0, \ldots, N/2 - 1$. This separation of even and odd points is called **decimation in time**. The N point DFT of $x[n]$ is then given by

$$X[k] = X_0[k] + e^{-j2\pi k/N} X_1[k] \text{ for } k = 0, \ldots, N-1$$

(5.10)

where $X_0[k]$ and $X_1[k]$ are the $N/2$ point DFT's of the even and odd points.

$$X_0[k] = \text{DFT}_{N/2}\{x_0[n]\}$$

$$X_1(k) = \text{DFT}_{N/2}\{x_1[n]\}$$

(5.11)

While (5.10) requires less computation than the original *N* point DFT, it can still be further simplified. First, note that each $N/2$ point DFT is periodic with period $N/2$. This means that we need to only compute $X_0[k]$ and $X_1[k]$ for $N/2$ values of k rather than the N values shown in (5.10). Furthermore, the complex exponential factor $e^{-j2\pi k/N}$ has the property that

$$-e^{-j2\pi\frac{k}{N}} = e^{-j2\pi\frac{k+N/2}{N}}$$

(5.12)

These two facts may be combined to yield a simpler expression for the N point DFT:

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

(5.13)

where the complex constants defined by $W_N^k = e^{-j2\pi k/N}$ are commonly known as the **twiddle factors**.

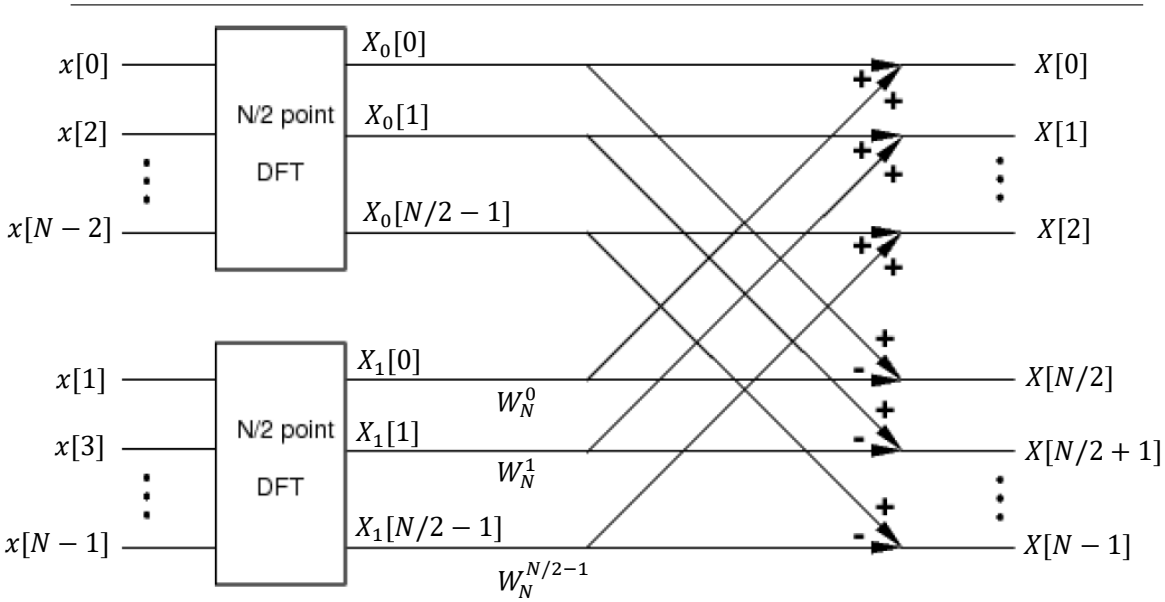


Figure 5.1: Divide and conquer DFT of equation (5.13). The *N*-point DFT is computed using the two $N/2$-point DFT's $X_0^{(N/2)}[k]$ and $X_1^{(N/2)}[k]$.

Figure 5.1 shows a graphical interpretation of (5.13) which we will refer to as the "divide-and-conquer DFT". We start on the left side with the data separated into even and odd subsets. We perform an $N/2$ point DFT on each subset, and then multiply the output of the odd DFT by the required twiddle factors. The first half of the output is computed by adding the two branches, while the second half is formed by subtraction. This type of flow diagram is conventionally used to describe a fast Fourier transform algorithm.

5.3.1 Implementation of Divide-and-Conquer DFT

In this section, you will implement the DFT transformation using (5.13) and the illustration in Figure 5.1. Write a Matlab function with the syntax

X = dcDFT(x)

where x is a vector of even length N, and X is its DFT. Your function dcDFT should do the following:

1. Separate the samples of x into even and odd points.

HINT: The Matlab command x0 = x(1:2:N) can be used to obtain the "even" points.

2. Use your function DFTsum to compute the two $N/2$ point DFT's.
3. Multiply by the twiddle factors $W_N^k = e^{-j2\pi k/N}$.
4. Combine the two DFT's to form X.

Test your function dcDFT by using it to compute the DFT's of the following signals.

1. $x[n] = \delta[n]$ for $N = 10$.
2. $x[n] = 1$ for $N = 10$.
3. $x[n] = e^{j2\pi n/N}$ for $N = 10$.

INLAB REPORT

1. Submit the code for your function dcDFT.
2. Determine the number of multiplies that are required in this approach to computing an N point DFT. (Consider a multiply to be one multiplication of real or complex numbers.)

HINT: Refer to the diagram of Figure 5.1, and remember to consider the $N/2$ point DFTs.

5.3.2 Recursive Divide and Conquer

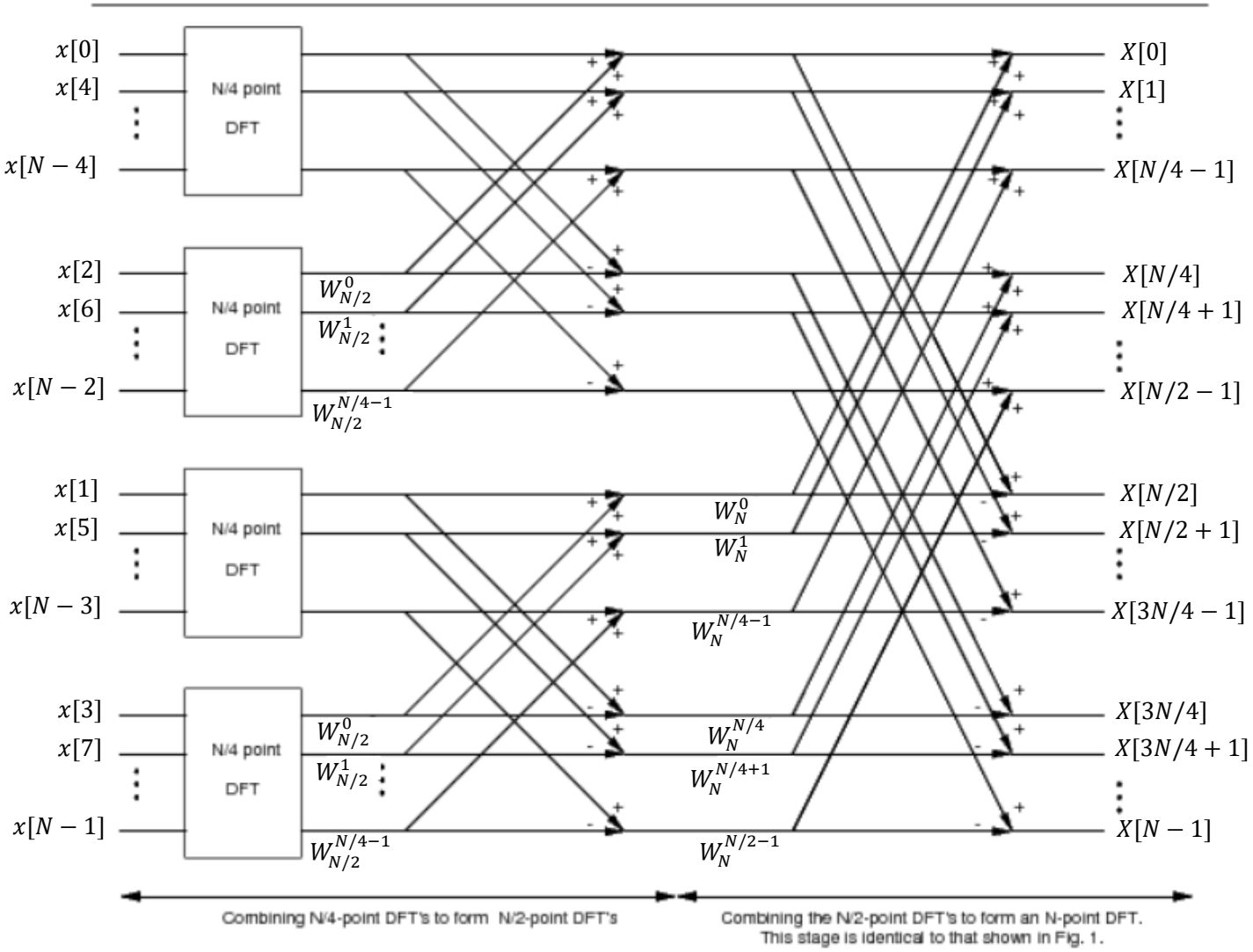


Figure 5.2: Recursion of the decimation-in-time process. Now each $N/2$-point is calculated by combining two $N/4$-point DFT's.

The second basic concept underlying the FFT algorithm is that of recursion. Suppose $N/2$ is also even. Then we may apply the same decimation-in-time idea to the computation of each of the $N/2$ point DFT's in Figure 5.1. This yields the process depicted in Figure 5.2. We now have two stages of twiddle factors instead of one.

How many times can we repeat the process of decimating the input sequence? Suppose N is a power of 2, i.e. $N = 2^p$ for some integer p. We can then repeatedly decimate the sequence until each subsequence contains only two points. It is easily seen from (5.4) that the 2 point DFT is a simple sum and difference of values.

$$X[0] = x[0] + x[1]$$
$$X[1] = x[0] - x[1] \tag{5.14}$$

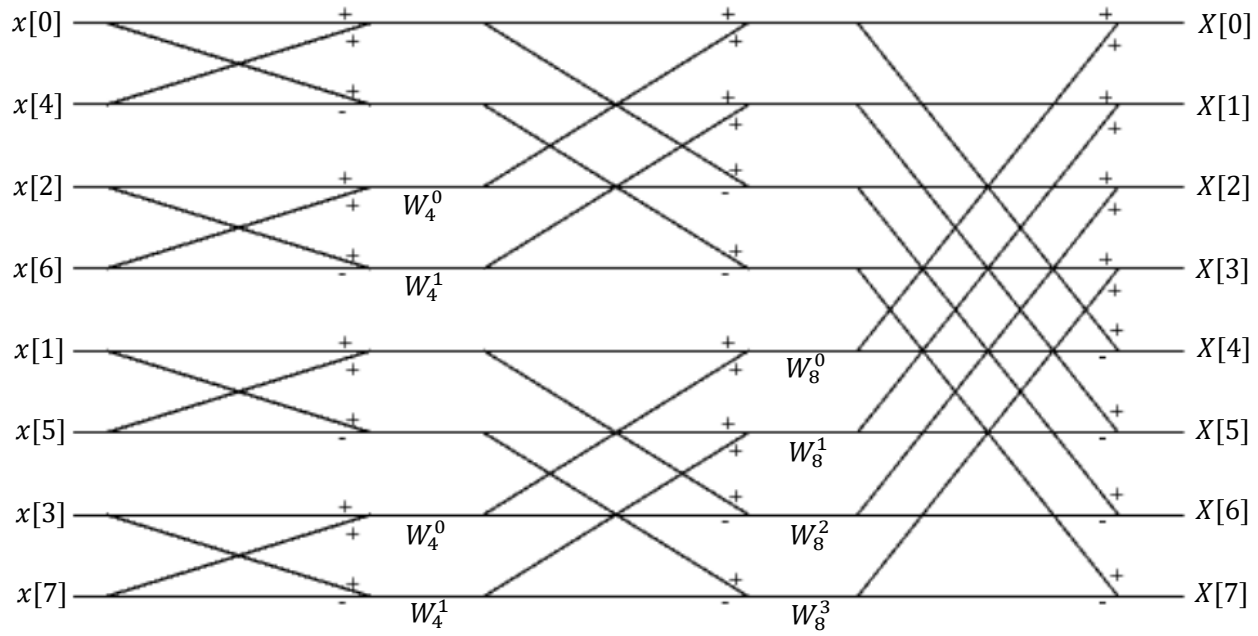


Figure 5.3: 8-Point FFT.

Figure 5.3 shows the flow diagram that results for an 8-point DFT when we decimate 3 times. Note that there are 3 stages of twiddle factors (in the first stage, the twiddle factors simplify to "1"). This is the flow diagram for the complete decimation-in-time 8-point FFT algorithm. How many multiplies are required to compute it?

Write three Matlab functions to compute the 2-, 4-, and 8-point FFT's using the syntax

X = FFT2(x)

X = FFT4(x)

X = FFT8(x)

The function FFT2 should directly compute the 2-point DFT using (5.14), but the functions FFT4 and FFT8 should compute their respective FFT's using the divide and conquer strategy. This means that FFT8 should call FFT4, and FFT4 should call FFT2.

Test your function FFT8 by using it to compute the DFT's of the following signals. Compare these results to the previous ones.

1. $x[n] = \delta[n]$ for $N = 8$.
2. $x[n] = 1$ for $N = 8$.
3. $x[n] = e^{\frac{j2\pi n}{8}}$ for $N = 8$.

INLAB REPORT

1. Submit the code for your functions FFT2, FFT4 and FFT8.
2. List the output of FFT8 for the case $x[n] = 1$ for $N = 8$.
3. Calculate the total number of multiplies **by twiddle factors** required for your 8-point FFT. (A multiply is a multiplication by a real or complex number.)
4. Determine a formula for the number of multiplies required for an $N = 2^p$ point FFT. Leave the expression in terms of N and p. How does this compare to the number of multiplies

required for direct implementation when p =10?

If you wrote the **FFT4** and **FFT8** functions properly, they should have almost the exact same form. The only difference between them is the length of the input signal, and the function called to compute the (N/2)-point DFTs. Obviously, it's redundant to write a separate function for each specific length DFT when they each have the same form. The preferred method is to write a **recursive** function, which means that the function calls itself within the body. It is imperative that a recursive function has a condition for exiting without calling itself, otherwise it would never terminate.

Write a recursive function X=fft_stage(x) that performs one stage of the FFT algorithm for a power-of-2 length signal. An outline of the function is as follows:

1. Determine the length of the input signal.
2. If N=2, then the function should just compute the 2-pt DFT as in (5.14), and then return.
3. If N>2, then the function should perform the FFT steps described previously (i.e. decimate, compute (N/2)-point DFTs, re-combine), calling **fft_stage** to compute the (N/2)-point DFTs.

Note that the body of this function should look very similar to previous functions written in this lab. Test **fft_stage** on the three 8-pt signals given above, and verify that it returns the same results as **FFT8**.

INLAB REPORT: Submit the code for your **fft_stage** function.

Lab 6 – z-transform and LTI system in Transformed Domain

6.1 Introduction

The z-transform is useful for the manipulation of discrete data sequences and has acquired a new significance in the formulation and analysis of discrete-time systems. It is used extensively today in the areas of applied mathematics, digital signal processing, control theory, population science, and economics. These discrete models are solved with difference equations in a manner that is analogous to solving continuous models with differential equations. The role played by the z-transform in the solution of difference equations corresponds to that played by the Laplace transforms in the solution of differential equations.

6.2 z-transform

The z-transform of a time domain sequence, $x[n]$, is given by

$$X(z) = Z\{x[n]\} = \sum_{n=-\infty}^{\infty} x[n]z^{-n} \tag{6.1}$$

where z is a complex variable. The set of *z* values for which $X(z)$ exists is called the region of convergence (ROC). If we express the complex variable z in polar form, $z = re^{j\omega}$, then (6.1) reduces to

$$X\left(re^{j\omega}\right) = \sum_{n=-\infty}^{\infty} x[n]r^{-n}e^{-j\omega n}, \tag{6.2}$$

which can be interpreted as the discrete-time Fourier transform of the modified sequence $x[n]r^{-n}$.

The inverse z-transform of a complex function $X(z)$ is given by:

$$x[n] = \frac{1}{2\pi j}\oint_C X(z)z^{n-1}\,dz$$

where *C* is a counterclockwise contour encircling the origin and lying in the ROC.

6.2.1 Solving difference equation using z-transform

The transfer function of a generalized difference equation (or a generally infinite impulse response (IIR) system)

$$\sum_{m=0}^{N} b_m y[n-m] = \sum_{m=0}^{M} a_m x[n-m]$$

is a rational form $A(z)/B(z)$ given by

$$\frac{Y(z)}{X(z)} = \frac{\sum_{m=0}^{M} a_m z^{-m}}{\sum_{m=0}^{N} b_m z^{-m}} \triangleq H(z)$$

The ratio of the z-transform of the input signal to that of the output signal, $H(z)$, is defined to be the z-transform transfer function of the system.

Since LTI systems are often realized by difference equations, the rational form is the most common and useful form of z-transforms, represented as a rational function of z^{-1}

$$H(z) = \frac{P(z)}{D(z)} = \frac{p_0 + p_1 z^{-1} + \cdots + p_{M-1} z^{-(M-1)} + p_M z^{-M}}{d_0 + d_1 z^{-1} + \cdots + d_{N-1} z^{-(N-1)} + d_N z^{-N}} \tag{6.3}$$

where the degree of $P(z)$ is M, and that of $D(z)$ is N. The degree of the system is the larger one of M and N. When b_0 is normalized to 1, and $b_m = 0$ for $m = 1$ ... N, the difference equation degenerates to an FIR system.

$$H(z) = \frac{Y(z)}{X(z)} = \sum_{m=0}^{M} a_m z^{-m}$$

It can still be represented by a rational form of the variable *z* as

$$H(z) = \frac{\sum_{m=0}^{M} a_m z^{(M-m)}}{z^M}$$

6.2.2 Poles and zeros of *z*-transform

The poles of $H(z)$ in rational form given in (6.3) are the roots of $D(z)$, and the zeros are the roots of $P(z)$, respectively. If we factorize $H(z)$ to the following form

$$H(z) = \frac{p_0}{d_0} \cdot \frac{\prod_{l=1}^{M}(1 - \xi_l z^{-1})}{\prod_{l=1}^{N}(1 - \lambda_l z^{-1})} = z^{(N-M)} \frac{p_0}{d_0} \cdot \frac{\prod_{l=1}^{M}(z - \xi_l)}{\prod_{l=1}^{N}(z - \lambda_l)},$$

the system contains *N* poles at $z = \lambda_l$, for $l = 1, 2, .., N$, and *M* zeros at $z = \xi_l$ for $l = 1, 2, .., M$.

Use the Matlab function **roots.m** to find the poles and zeros of the following z-transform

(a) $H_1(z) = -1 + 2z^{-1} - 3z^{-2} + 6z^{-3} - 3z^{-4} + 2z^{-5} - z^{-6}$

(b) $G_1(z) = \frac{3z^4 - 2.4z^3 + 15.36z^2 + 3.84z + 9}{5z^4 - 8.5z^3 + 17.6z^2 + 4.7z - 6}$

(c) $G_2(z) = \frac{2z^4 + 0.2z^3 + 6.4z^2 + 4.6z + 2.4}{5z^4 + z^3 + 6.6z^2 + 4.2z + 24}$

Use the Matlab function **zplane.m** to display the zero-pole plot. Determine all possible ROCs of each of the above z-transforms, and describe the type of their inverse z-transform (left-sided, right-sided, two sided sequences) associated with each of the ROCs.

INLAB REPORT: For each of the above z-transform, report the poles and zeros, hand in the zero-pole plots, show the all possible ROCs, and describe the type of their inverse z-transform

associated with each of the ROCs.

6.2.3 *z*-transform and frequency response

In (6.2), if $r = 1(\text{i.e.}, |z| = 1)$, the *z*-transform reduces to DTFT. The contour $|z| = 1$ is a circle in the *z* plan of unity radius, called unit circle**.** If the ROC of $H(z)$ includes the unit circle, the frequency response $H(e^{j\omega})$ of the LTI digital system can be obtained by evaluating $H(z)$ on the unit circle:

$$H(e^{j\omega}) = H(z)|_{z=e^{j\omega}}$$

For the digital filters characterized by the following transfer functions:

(a) $H(z) = z^{-4} + 2z^{-3} + 2z^{-1} + 1$, (b) $H(z) = \frac{z^2-1}{z^2-1.2z+0.95}$

Write a Matlab program to plot the magnitude (in both magnitude values and gain of dB) and phase of the frequency response of the digital filters with syntax [**mag**, **phase**]=**FreRes**(**num**, **den**), where **num**, and **den** are vectors specifying the coefficients of numerator and denominator polynomials in descending power of *z*, i.e., with **num**[1] and **den**[1] being the coefficient of z^0, **num**[2] and **den**[2] the coefficient of z^{-1}, etc.

INLAB REPORT: Submit your Matlab code and plots of the magnitude and phase responses of the transfer functions.

6.2.4 Inverse *z*-transform

In practice, inverse z-transforms are computed by avoiding contour integral if possible. One of the many methods is partial fraction expansion, in which, a fractional form z-transform

$$X(z) = \frac{P(z)}{D(z)} = \frac{\sum_{i=0}^{M} p_i\, z^{-i}}{\sum_{i=0}^{N} d_i z^{-i}}$$

is first written to a integer polynomial $K(z)$ plus a proper fraction, i.e.,

$$X(z) = K(z) + \frac{P_1(z)}{D(z)}$$

where the degree of $P_1(z)$ is less than *N*. Then, the proper fraction is expanded to partial fractions in the forms of

$$\frac{P_1(z)}{D(z)} = \sum_{l=1}^{N} \left(\frac{\rho_l}{1 - \lambda_l z^{-l}}\right)$$

where the $z = \lambda_k$, $1 \le k \le N$ are the poles of the proper fraction. In Matlab, the integer polynomial and the partial fractions together with the residues, ρ_l, maybe found by using function **residue.m**.

Read the help file of the function **residue.m**, and write a Matlab function to find the partial fraction expansion of the following two z-transforms:

(a) $X(z) = \frac{3-7.8z^{-1}}{(1-0.7z^{-1})(1+1.6z^{-1})}$, (b) $Y(z) = \frac{3z^2+1.8z+1.28}{(z-0.5)(z+0.4)}$

Each above z-transform has several ROCs. Evaluate their respective inverse z-transforms corresponding to each ROC.

INLAB REPORT: Submit the partial fraction expansion, all possible ROCs, and the corresponding inverse z-transforms.

6.2.5 Stability Conditions

A stable causal LTI system has all poles inside unit circle. This implies that a causal LTI FIR digital filter with bounded impulse response coefficients is always stable, as all its poles are at the origin in the *z*-plane. On the other hand, a causal LTI IIR digital filter may or may not be stable, dependent on its pole locations. In addition, an originally stable IIR filter characterized by infinite precision coefficients and with all poles inside the unit circle may become unstable after implementation due to the unavoidable quantization of all coefficients.

Slides 58 and 59 of L7 demonstrate an example where a stable IIR filter become unstable when the coefficients are quantized. Practice on the following:

1. Show the zero-pole plots of these two transforms, and observe the locations of the poles.
2. Matlab function **impz.m** can be used to compute a given number impulse response samples of digital filter with *z*-transform in rational form. Use **impz.m** to determine the first 100 samples of the impulse response of the two *z*-transforms.
3. Display and compare the two impulse response, and comment on their implications to the stabilities.

INLAB REPORT: Submit the zero-pole plots, and stem figure of the impulse responses, with your comments, if any.

6.3 Linear Phase FIR Filters

Linear Phase FIR filters are an important class of FIR filters. The frequency response of linear phase FIR filters is given by $e^{-j\left(\frac{N-1}{2}\omega-\beta\right)}R(\omega)$, where $R(\omega)$ is a real function, $\beta = 0\ or\ \pm 0.5\pi$ and N is the filter length, i.e., its phase function is in a linear form of $\theta(\omega) = \frac{N-1}{2}\omega - \beta$.

6.3.1 Four Types of Linear Phase FIR Filters

The linear phase of the FIR filter may be achieved by making the impulse response of the FIR filter either symmetric or anti-symmetric. Dependent on the parity of the filter length, there are 4 types of linear phase FIR filters.

Type I FIR filters: N is odd, and the impulse response is symmetric. Its frequency response is given by $H\left(e^{j\omega}\right) = e^{-j\omega\frac{N-1}{2}}\sum_{n=0}^{\frac{N-1}{2}} a[n]\cos(\omega n)$, where $a[0] = h\left[\frac{N-1}{2}\right]$, $a[n] = 2h\left[\frac{N-1}{2} - n\right]$, for $n = 1,\ 2,\ \ldots,\ \frac{N-3}{2}$. This type of filter has either an even number or no zeros at $z = 1$ and $z = -1$.

Type II FIR filters: N is even, and the impulse response is symmetric. Its frequency response is given by $H(e^{j\omega}) = e^{-j\omega\frac{N-1}{2}}\sum_{n=1}^{\frac{N}{2}} b[n]\cos(\omega(n-1/2))$, where $b[n] = 2h[N/2-n]$, for $n = 1, 2, \ldots, N/2$. This type of filter has either an even number or no zeros at $z = 1$ and an odd number of zeros at $z = -1$.

Type III FIR filters: N is odd, and the impulse response is anti-symmetric. Its frequency response is given by $H(e^{j\omega}) = je^{-j\omega\frac{N-1}{2}}\sum_{n=1}^{\frac{N-1}{2}} c[n]\sin(\omega n)$, where $c[n] = 2h\left[\frac{N-1}{2}-n\right]$, for $n = 1, 2, \ldots, \frac{N-1}{2}$. This type of filter has an odd number of zeros at $z = 1$ and $z = -1$.

Type IV FIR filters: N is even, and the impulse response is anti-symmetric. Its frequency response is given by $H(e^{j\omega}) = je^{-j\omega\frac{N-1}{2}}\sum_{n=1}^{\frac{N}{2}} d[n]\sin(\omega(n-1/2))$where $d[n] = 2h[N/2-n]$, for $n = 1, 2, \ldots, N/2$. This type of filter has an odd number of zeros at $z = 1$ and either an even number or no zeros at $z = -1$.

Besides the zeros at $z = 1$ and $z = -1$, all other zeros of real coefficient linear phase FIR filters exhibit mirror image symmetry with respect to unit circle.

An example of a type I linear phase FIR filter $H(z)$ has impulse response given by
$\{h[n]\}$ = {–0.0035, –0.0039, 0.0072, 0.0201, –0.0000, –0.0517, –0.0506, 0.0855, 0.2965, 0.4008,
0.2965, 0.0855, –0.0506, –0.0517, –0.0000, 0.0201, 0.0072, –0.0039, –0.0035}
for $0 \le n \le 18$.

Using the program developed in Section 6.2.3, plot the frequency response of the filter. What is the magnitude characteristic of the filter? Show as well the zero-pole plot of the filter.

INLAB REPORT: Submit the frequency response and zero-pole plots of the filter. Answer (1) the magnitude characteristic of the filter, and (2) if a type III filter may be designed to be a lowpass filter, and justify your answer.

The impulse response of the $h_1[n]$ and $h_2[n]$ is obtained by

$$h_1[n] = \begin{cases} h[n], & \text{for even } n \\ -h[n], & \text{for odd } n \end{cases}$$

$$h_2[n] = \begin{cases} h[n/5], & \text{for } n = 5k \text{ for interger } k \\ 0, & \text{otherwise} \end{cases}$$

INLAB REPORT: Express the z-transform of $h_1[n]$ and $h_2[n]$ in terms of $H(z)$, and express the frequency response of $H_1(e^{j\omega})$ and $H_2(e^{j\omega})$ in terms of $H(e^{j\omega})$. Verify the frequency response expressions by plotting the frequency response of $h_1[n]$ and $h_2[n]$. Describe the magnitude characteristic of the filter $H_1(z)$ and $H_2(z)$ with respect to that of $H(z)$.

6.3.2 Design of Simple FIR Filters

The simplest lowpass FIR filter is the first order moving average filter, with transfer function given by,

$$H_0(z) = \frac{1 + z^{-1}}{2}. \tag{6.4}$$

It is a type II filter with a zero at $z = -1$. The magnitude response of the filter has a maximum value one at $\omega = 0$, and a minimum value zero at $\omega = \pi$. The frequency response of the filter is given by,

$$H_0(e^{j\omega}) = e^{-j\omega/2}\cos(\omega/2).$$

The frequency $\omega = \omega_c$ at which $|H_0(e^{j\omega})| = \frac{1}{\sqrt{2}}|H(e^{j0})|$ is of practical interest as here the gain $\mathcal{G}(\omega_c)$ in dB is given by,

$$\mathcal{G}(\omega_c) = 20\log_{10}|H_0(e^{j\omega_c})| = 20\log_{10}|H_0(e^{j\omega_0})| - 20\log_{10}\sqrt{2} = -3.0103 \cong -3\text{dB}$$

Thus, the gain $\mathcal{G}(\omega_c)$ at $\omega = \omega_c$ is approximately 3-dB less than that at zero frequency. As a result, ω_c is called the 3-dB cutoff frequency. To determine the expression for ω_c, we set $|H_0(e^{j\omega_c})|^2 = 1/2$. For the filter in (6.4), it is $\cos^2(\omega_c/2) = 1/2$, yielding $\omega_c = \pi/2$.

A cascade of the simple filter of (6.4) results in an improved lowpass frequency response. The 3-dB cutoff frequency of a cascade of *M* sections of the lowpass filter of (6.4) is given by

$$\omega_c = 2\cos^{-1}\left(2^{-1/(2M)}\right). \tag{6.5}$$

Alternatively, improved lowpass frequency response can be obtained by a higher order moving average filter with transfer function given by

$$H(\mathrm{z}) = \frac{1}{M}\sum_{m=0}^{M-1} z^{-m}. \tag{6.6}$$

The frequency response of the moving average filter has been derived to

$$H(e^{j\omega}) = \frac{1}{M}\cdot\frac{\sin\left(\frac{M\omega}{2}\right)}{\sin\left(\frac{\omega}{2}\right)}e^{-\frac{j(M-1)\omega}{2}}.$$

Design a linear phase lowpass FIR filter with a 3-dB cutoff frequency at 0.3π using

(1) a cascade of first-order lowpass filters of (6.4). Determine the number of stages to be cascaded. Plot its gain response.

(2) a higher order moving average filter. Determine the order of the moving average filter. Plot its gain response.

INLAB REPORT: Submit your derivation of (6.5) and the number of stages in the cascade design. Show the steps to determine the order of the filter in the higher order moving average filter design. Plot the gain responses of both designed filters on the same figure. Indicate the actual 3-dB cutoff frequency. Give your comments if any.

The simplest highpass FIR filter is obtained by replacing z with $-z$ in (6.4), resulting in a transfer function given by

$$H_1(z) = \frac{1 - z^{-1}}{2} \tag{6.7}$$

The corresponding frequency response is given by

$$H_1(e^{j\omega}) = je^{-j\omega/2}\sin(\omega/2)$$

Improved highpass frequency response can be obtained by cascading several sections of the simple highpass filter of (6.7), or replacing z with $-z$ in (6.6).

6.4 IIR Filters

The first-order lowpass IIR filter with a zero at $z = -1$ is given by

$$H_3(z) = \frac{1-\alpha}{2}\frac{1 + z^{-1}}{1 - \alpha z^{-1}},\ 0 < |\alpha| < 1$$

The squared magnitude function can be derived as,

$$\left|H_{LP}(e^{j\omega})\right|^2 = \frac{(1-\alpha)^2(1+\cos\omega)}{2(1+\alpha^2 - 2\alpha\cos\omega)}$$

and the 3-dB cutoff frequency may be determined as

$$\cos\omega_c = \frac{2\alpha}{1+\alpha^2}.$$

Therefore, when $-1 < \alpha < 0$, the resultant filter is a wide band lowpass filter (with the passband width larger than $\pi/2$), while $0 < \alpha < 1$, a narrow band lowpass filter (with the passband width narrower than $\pi/2$). For a given cutoff frequency ω_c, α is solved to be

$$\alpha = \frac{1 - \sin\omega_c}{\cos\omega_c}$$

The first-order highpass IIR filter with a zero at $z = 1$ is given by

$$H_4(z) = \frac{1+\alpha}{2}\frac{1 - z^{-1}}{1 - \alpha z^{-1}},\ 0 < |\alpha| < 1 \tag{6.8}$$

Derive the squared magnitude response of the highpass IIR filter. Determine the 3-dB cutoff frequency, and for a given cutoff frequency ω_c, determine the value of α.

Design a first order highpass IIR filter with a 3-dB cutoff frequency of 0.2π. Plot the magnitude response of the filter.

INLAB REPORT: Submit your derivation of the squared magnitude response, the 3-dB cutoff

frequency, and α for a given cutoff frequency ω_c of the first order highpass IIR filter of (6.8). Design the highpass IIR filter, and plot the magnitude response of the resultant filter. Indicate the actual 3-dB cutoff frequency. Give your comments if any.

Lab 7 - Digital Filter Design (part 1)

7.1 Introduction

This laboratory covers some basic examples of FIR and IIR filters, and then introduces the concepts of FIR filter design. Systematic methods of designing both FIR and IIR filters are followed.

7.2 Background on Digital Filters

In digital signal processing applications, it is often necessary to change the relative amplitudes of frequency components or remove undesired frequencies of a signal. This process is called **filtering**. Digital filters are used in a variety of applications. Digital filters are used in audio systems in the previous laboratories that allow the listener to adjust the bass (low-frequency energy) and the treble (high frequency energy) of audio signals.

Digital filter design requires the use of both frequency domain and time domain techniques. This is because filter design specifications are often given in the frequency domain, but filters are usually implemented in the time domain with a difference equation. Typically, frequency domain analysis is done using the z-transform and the discrete-time Fourier Transform (DTFT).

In general, a linear and time-invariant causal digital filter with input $x[n]$ and output $y[n]$ may be specified by its difference equation

$$y[n] = \sum_{i=0}^{N-1} b_i\, x[n-i] - \sum_{k=1}^{M-1} a_k\, y[n-k] \tag{7.1}$$

where b_i and a_k are coefficients which parameterize the filter. This filter is said to have $N-1$ zeros and $M-1$ poles. Each new value of the output signal, $y[n]$, is determined by past values of the output, and by present and past values of the input. The impulse response, $h[n]$, is the response of the filter to an input of $\delta[n]$, and is therefore the solution to the recursive difference equation

$$h[n] = \sum_{i=0}^{N-1} b_i\, \delta[n-i] - \sum_{k=1}^{M-1} a_k\, h[n-k] \tag{7.2}$$

There are two general classes of digital filters: infinite impulse response (IIR) and finite impulse response (FIR). The FIR case occurs when $a_k = 0$, for all k. Such a filter is said to have no poles, only zeros. In this case, the difference equation in (7.2) becomes

$$h[n] = \sum_{i=0}^{N-1} b_i\, \delta[n-i] \tag{7.3}$$

Since (7.3) is no longer recursive, the impulse response has finite duration N.

In the case where $a_k \neq 0$, the difference equation usually represents an IIR filter. In this case, (7.2) will usually generate an impulse response which has non-zero values as $n \to \infty$.

The z-transform is the major tool used for analyzing the frequency response of filters and their difference equations. The z-transform of a discrete-time signal, $x[n]$, is given by

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]\, z^{-n} \tag{7.4}$$

where z is a complex variable. The DTFT may be thought of as a special case of the z-transform where z is evaluated on the unit circle in the complex plane.

$$X\left(e^{j\omega}\right) = X(z)|_{z=e^{j\omega}}$$
$$= \sum_{n=-\infty}^{\infty} x[n] e^{-j\omega n} \tag{7.5}$$

From the definition of the z-transform, a change of variable $m = n - K$ shows that a delay of K samples in the time domain is equivalent to multiplication by z^{-K} in the z-transform domain.

$$x[n-K] \xleftrightarrow{Z} \sum_{n=-\infty}^{\infty} x[n-K] z^{-n} = \sum_{m=-\infty}^{\infty} x[m] z^{-(m+K)}$$
$$= z^{-K} \sum_{m=-\infty}^{\infty} x[m] z^{-m} = z^{-K} X(z) \tag{7.6}$$

We may use this fact to re-write (7.1) in the z-transform domain, by taking z-transforms of both sides of the equation:

$$Y(z) = \sum_{i=0}^{N-1} b_i\, z^{-i} X(z) - \sum_{k=1}^{M-1} a_k\, z^{-k} Y(z)$$
$$Y(z)\left(1 + \sum_{k=1}^{M-1} a_k z^{-k}\right) = X(z) \sum_{i=0}^{N-1} b_i\, z^{-i}$$
$$H(z) \triangleq \frac{Y(z)}{X(z)} = \frac{\sum_{i=0}^{N-1} b_i z^{-i}}{1 + \sum_{k=1}^{M-1} a_k z^{-k}} \tag{7.7}$$

From this formula, we see that any filter which can be represented by a linear difference equation with constant coefficients has a rational transfer function (i.e. a transfer function which is a ratio of polynomials). From this result, we may compute the frequency response of the filter by evaluating $H(z)$ on the unit circle:

$$H\left(e^{j\omega}\right) = \frac{\sum_{i=0}^{N-1} b_i e^{-j\omega i}}{1 + \sum_{k=1}^{M-1} a_k e^{-j\omega k}} \tag{7.8}$$

There are many different methods for implementing a general recursive difference equation of the form (7.1). Depending on the application, some methods may be more robust to quantization error, require fewer multiplies or adds, or require less memory. Figure 7.1 shows a system diagram known as the direct form implementation; it works for any discrete-time filter described by the difference equation in (7.1). Note that the boxes containing the symbol z^{-1} represent unit delays, while a parameter written next to a signal path represents multiplication by that parameter.

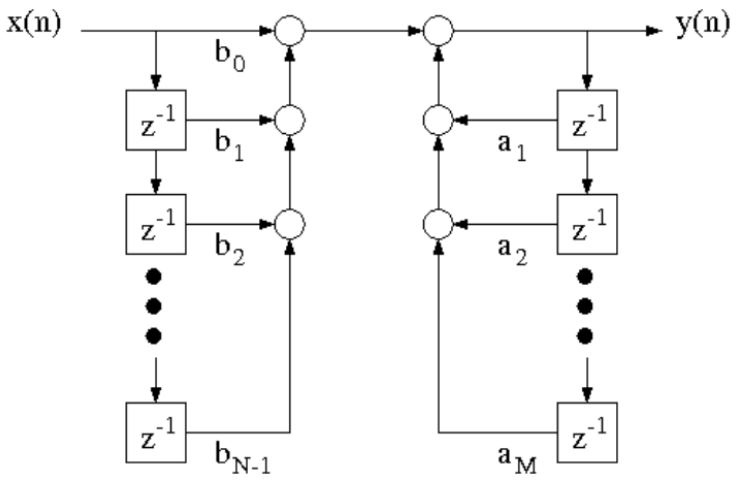


Figure 7.1: Direct form implementation for a discrete-time filter described by a linear recursive difference equation.

7.3 Design of a Simple FIR Filter

Download the files, **nspeech1.mat** and **DTFT.m** for the following section.

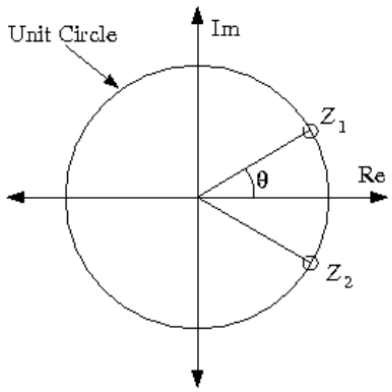


Figure 7.2: Location of two zeros for simple a FIR filter.

To illustrate the use of zeros in filter design, you will design a simple second order FIR filter with the two zeros on the unit circle as shown in Figure 7.2. In order for the filter's impulse response to be real-valued, the two zeros must be complex conjugates of one another:

$$z_1 = e^{j\theta}, \text{ and } z_2 = e^{-j\theta}$$

where θ is the angle of z_1 relative to the positive real axis. We will see later that $\theta \epsilon [0, \pi]$ may be interpreted as the location of the zeros in the frequency response.

The transfer function for this filter is given by

$$H_f(z) = (1 - z_1 z^{-1})(1 - z_2 z^{-1}) = (1 - e^{j\theta} z^{-1})(1 - e^{-j\theta} z^{-1}) = 1 - 2\cos\theta z^{-1} + z^{-2} \quad (7.9)$$

Use this transfer function to determine the difference equation for this filter. Then draw the corresponding system diagram and compute the filter's impulse response $h[n]$.

This filter is an FIR filter because it has impulse response $h[n]$ of finite duration. Any filter with only zeros and no poles other than those at 0 and $\pm\infty$ is an FIR filter. Zeros in the transfer function represent frequencies that are not passed through the filter. This can be useful for removing unwanted frequencies in a signal. The fact that $H_f(z)$ has zeros at $e^{\pm j\theta}$ implies that $H_f\left(e^{\pm j\theta}\right) = 0$. This means that the filter will not pass pure sine waves at a frequency of $\omega = \theta$.

Use Matlab to compute and plot the magnitude of the filter's frequency response $\left|H_f(e^{j\omega})\right|$ as a function of ω on the interval $-\pi < \omega < \pi$, for the following three values of θ:

i) $\theta = \pi/6$, ii) $\theta = \pi/3$ iii) $\theta = \pi/2$

INLAB REPORT: Submit the difference equation, system diagram, and the analytical expression of the impulse response for the filter $H_f(z)$. Also submit the plot of the magnitude response for the three values of θ. Explain how the value of θ affects the magnitude of the filter's frequency response.

In the next experiment, we will use the filter $H_f(z)$ to remove an undesirable sinusoidal interference from a speech signal. To run the experiment, first download the audio signal **nspeech1.mat**, and the M-file **DTFT.m.** Load **nspeech1.mat** into Matlab using the command **load nspeech1**. This will load the signal **nspeech1** into the workspace. Play **nspeech1** using the **sound** command, and then **plot** 101 samples of the signal for the time indices (100:200).

We will next use the **DTFT** command to compute samples of the DTFT of the audio signal. The DTFT command has the syntax

[X ,w]=DTFT(x ,M)

where x is a signal which is assumed to start at time $n = 0$, and M specifies the number of output points of the DTFT. The command [X ,w]=DTFT(x,0) will generate a DTFT that is the same duration as the input; if this is not sufficient, it may be increased by specifying M. The outputs of the function are a vector X containing the samples of the DTFT, and a vector w containing the corresponding frequencies of these samples.

Compute the magnitude of the DTFT of 1001 samples of the audio signal for the time indices (100:1100). Plot the magnitude of the DTFT samples versus frequency for $|\omega| < \pi$. Notice that there are two large peaks corresponding to the sinusoidal interference signal. Use the Matlab **max** command to determine the exact frequency of the peaks. This will be the value of θ that we will use for filtering with $H_f(z)$.

HINT: Use the command [Xmax , Imax]=**max**(**abs**(X)) to find the value and index of the maximum element in X . θ can be derived using this index.

Write a Matlab function **FIRfilter**(x) that implements the filter $H_f(z)$ with the measured value of θ and outputs the filtered signal (**HINT:** Use convolution). Apply the new function **FIRfilter** to the **nspeech1** vector to attenuate the sinusoidal interference. Listen to the filtered signal to hear the effects of the filter. Plot 101 samples of the signal for the time indices (100:200), and plot the

magnitude of the DTFT of 1001 samples of the filtered signal for the time indices (100:1100).

INLAB REPORT: For both the original audio signal and the filtered output, hand in the following:
- The time domain plot of 101 samples.
- The plot of the magnitude of the DTFT for 1001 samples.

Also hand in the code for the **FIRfilter** filtering function. Comment on how the frequency content of the signal changed after filtering. Is the filter we used a lowpass, highpass, bandpass, or a bandstop filter? Comment on how the filtering changed the quality of the audio signal.

7.4 Design of a Simple IIR Filter

Download the file **pcm.mat** for the following section.

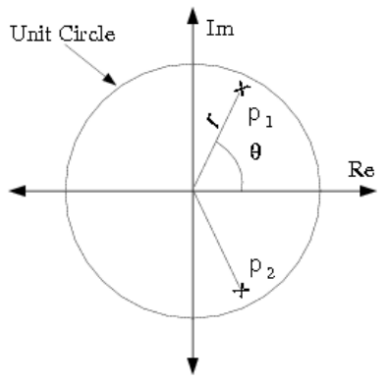


Figure 7.3: Location of two poles for a simple IIR filter.

While zeros attenuate a filtered signal, poles amplify signals that are near their frequency. In this section, we will illustrate how poles can be used to separate a narrow-band signal from adjacent noise. Such filters are commonly used to separate modulated signals from background noise in applications such as radio-frequency demodulation.

Consider the following transfer function for a second order IIR filter with complex-conjugate poles:

$$H_i(z) = \frac{1-r}{(1-re^{j\theta}z^{-1})(1-re^{-j\theta}z^{-1})} = \frac{1-r}{1-2r\cos(\theta)z^{-1}+r^2z^{-2}} \tag{7.10}$$

Figure 7.3 shows the locations of the two poles of this filter. The poles have the form

$$p_1 = re^{j\theta} \qquad p_2 = re^{-j\theta}$$

where r is the distance from the origin, and θ is the angle of p_1 relative to the positive real axis. From the theory of z-transforms, we know that a causal filter is stable if and only if its poles are located within the unit circle. This implies that this filter is stable if and only if $|r| < 1$. However, we will see that by locating the poles close to the unit circle, the filter's bandwidth may be made extremely narrow around θ.

This two-pole system is an example of an IIR filter because its impulse response has infinite duration. Any filter with nontrivial poles (not at $z = 0$ or $\pm\infty$) is an IIR filter unless the poles are canceled by zeros.

Calculate the magnitude of the filter's frequency response $\left|H_i(e^{j\omega})\right|$ on $|\omega| < \pi$ for $\theta = \pi/3$ and the following three values of r.

i) $r = 0.99$, ii) $r = 0.9$, iii) $r = 0.7$

Put all three plots on the same figure using the subplot command.

INLAB REPORT: Submit the difference equation, system diagram and the analytical expression of the impulse response for $H_i(z)$. Also submit the plot of the magnitude of the frequency response for each value of r. Explain how the value of r affects this magnitude.

In the following experiment, we will use the filter $H_i(z)$ to separate a modulated sinusoid from background noise. To run the experiment, first download the file **pcm.mat** and load it into the Matlab workspace using the command **load pcm**. Play **pcm** using the **sound** command. Plot 101 samples of the signal for indices (100:200), and then compute the magnitude of the DTFT of 1001 samples of **pcm** using the time indices (100:1100). Plot the magnitude of the DTFT samples versus angular frequency for $|\omega| < \pi$. The two peaks in the spectrum correspond to the center frequency of the modulated signal. The low amplitude wideband content is the background noise. In this exercise, you will use the IIR filter described above to amplify the desired signal, relative to the background noise.

The **pcm** signal is modulated at 3146Hz and sampled at 8kHz. Use these values to calculate the value of θ for the filter $H_i(z)$. Remember from the sampling theorem that an angular frequency of 2π corresponds to the sampling frequency.

Write a Matlab function **IIRfilter**(x) that implements the filter $H_i(z)$. In this case, you need to use a **for** loop to implement the recursive difference equation. Use your calculated value of θ and $r = 0.995$. You can assume that $y[n]$ is equal to 0 for negative values of n. Apply the new command **IIRfilter** to the signal **pcm** to separate the desired signal from the background noise, and listen to the filtered signal to hear the effects. Plot the filtered signal for indices (100:200), and then compute the DTFT of 1001 samples of the filtered signal using the time indices (100:1100). Plot the magnitude of this DTFT. In order to see the DTFT around $\omega = \theta$ more clearly, plot also the portion of this DTFT for the values of ω in the range $[\theta - 0.02, \theta + 0.02]$. Use your calculated value of θ.

INLAB REPORT: For both the pcm signal and the filtered output, submit the following:
- The time domain plot of the signal for 101 points.
- The plot of the magnitude of the DTFT computed from 1001 samples of the signal.
- The plot of the magnitude of the DTFT for ω in the range $[\theta - 0.02, \theta + 0.02]$.

Also hand in the code for the IIRfilter filtering function. Comment on how the signal looks and sounds before and after filtering. How would you expect changes in r to change the filtered output? Would a value of $r = 0.9999999$ be effective for this application? Why might such a value for r be ill-advised? (Consider the spectrum of the desired signal around $\omega = \theta$.)

7.5 Lowpass Filter Design Parameters

Download the file **nspeech2.mat** for the following section.

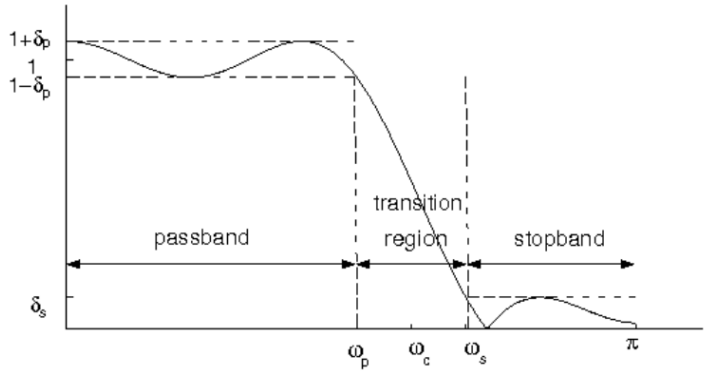


Figure 7.4: Tolerance specifications for the frequency response of a filter.

Oftentimes it is necessary to design a good approximation to an ideal lowpass, highpass or bandpass filter. Figure 7.4 illustrates the typical characteristics of a real lowpass filter. The frequencies $|\omega| < \omega_p$ are known as the passband, and the frequencies $\omega_s < |\omega| \leq \pi$ are the stopband. For any real lowpass filter, $\omega_p < \omega_s$. The range of frequencies $\omega_p \leq \omega \leq \omega_s$ is known as the transition band. The magnitude of the filter response, $H(e^{j\omega})$, is constrained in the passband and stopband by the following two equations

$$\left|H\left(e^{j\omega}\right) - 1\right| \leq \delta_p \quad \text{for} \quad |\omega| \leq \omega_p$$

$$\left|H\left(e^{j\omega}\right)\right| \leq \delta_s \quad \text{for} \quad \omega_s \leq |\omega| \leq \pi \tag{7.11}$$

where δ_p and δ_s are known as the passband and stopband ripple respectively. Most lowpass filter design techniques depend on the specification of these four parameters: ω_p, ω_s, δ_p, and δ_s.

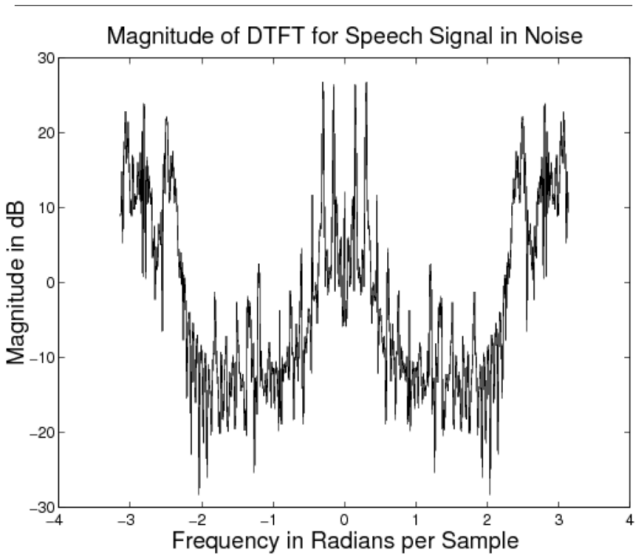


Figure 7.5: DTFT of a section of noisy speech.

To illustrate the selection of these parameters consider the problem of filtering out background noise from a speech signal. Figure 7.5 shows the magnitude of the DTFT over a window of such a signal, called **nspeech2**. Notice that there are two main components in **nspeech2**: one at the low

frequencies and one at the high. The high frequency signal is noise, and it is band limited to $|\omega| > 2.2$. The low frequency signal is speech and it is band limited to $|\omega| < 1.8$. Download the file **nspeech2.mat** and load it into the Matlab workspace. It contains the signal **nspeech2** from Figure 7.5. Play the **nspeech2** using the **sound** command and note the quality of the speech and background noise.

In the following sections, we will compute lowpass filters for separating the speech and noise using a number of different methods.

7.6 Filter Design Using Truncation

Ideally, a lowpass filter with cutoff frequency ω_c should have a frequency response of

$$H_{ideal}(e^{j\omega}) = \begin{cases} 1 & |\omega| \le \omega_c \\ 0 & \omega_c < |\omega| \le \pi \end{cases} \tag{7.12}$$

and a corresponding impulse response of

$$h_{ideal}[n] = \frac{\omega_c}{\pi}\text{sinc}\left(\frac{\omega_c n}{\pi}\right) \quad \text{for} \quad -\infty < n < \infty \tag{7.13}$$

However, no real filter can have this frequency response because $h_{ideal}(n)$ is infinite in duration.

One method for creating a realizable approximation to an ideal filter is to truncate this impulse response outside of $n \in [-M, M]$.

$$h_{trunc}[n] = \begin{cases} \frac{\omega_c}{\pi}\text{sinc}\left(\frac{\omega_c}{\pi}n\right) & n = -M, \ldots, 0, 1, \ldots M \\ 0 & \text{otherwise} \end{cases} \tag{7.14}$$

Figure 7.6 shows the magnitude response of the lowpass filter with cutoff frequency $\omega_c = 2.0$, with the impulse response truncated to $n \in [-10, 10]$. Notice the oscillatory behavior of the magnitude response near the cutoff frequency and the large amount of ripple in the stopband.

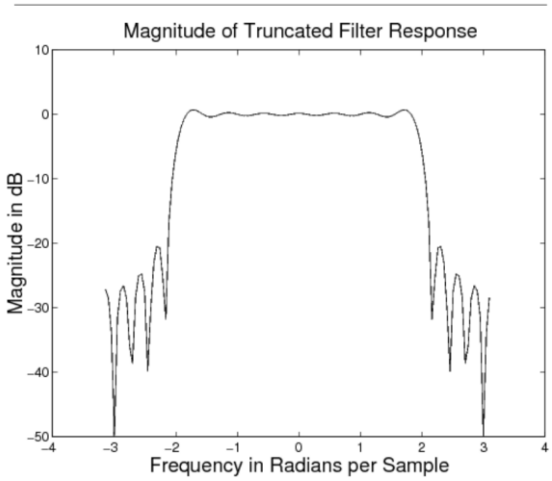


Figure 7.6: Frequency response of lowpass filter designed using the truncation method.

Due to the modulation property of the DTFT, the frequency response of the truncated filter is the result of convolving the magnitude response of the ideal filter (a rect) with the DTFT of the truncating window. The DTFT of the truncating window, shown in Figure 7.7, is similar to a sinc

function since it is the DTFT of a sampled rectangular window. Notice that this DTFT has very large sidelobes, which lead to large stopband ripple in the final filter.

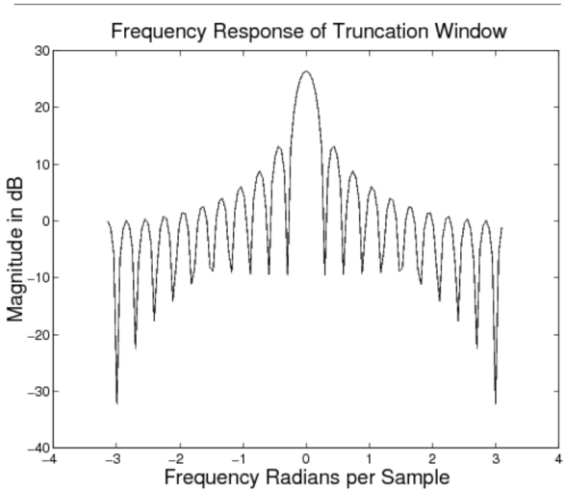


Figure 7.7: DTFT of a rectangular window of size 21.

A truncated impulse response is of finite duration, yet the filter is still non causal. In order to make the FIR filter causal, it must be shifted to the right by M units. For a filter of size $N = 2M + 1$ this shifted and truncated filter is given by

$$h[n] = \begin{cases} \frac{\omega_c}{\pi}\text{sinc}\left(\frac{\omega_c}{\pi}\left(n - \frac{N-1}{2}\right)\right) & n = 0,1,\ldots,N-1 \\ 0 & \text{otherwise} \end{cases} \tag{7.15}$$

This time shift of $(N-1)/2$ units to the right corresponds to multiplying the frequency response by $e^{-j\omega(N-1)/2}$. It does not affect the magnitude response of the filter, but adds a factor of $-j\omega(N-1)/2$ to the phase response. Such a filter is called linear phase because the phase is a linear function of ω.

It is interesting to see that the filter formula of (7.15) is valid for N both even and odd. While both of these filters are linear phase, they have different characteristics in the time domain. When N is odd, then the value at $n = (N-1)/2$ is sampled at the peak of the sinc function, but when *N* is even, then the two values at $n = N/2$ and $n = (N/2) - 1$ straddle the peak.

To examine the effect of filter size on the frequency characteristics of the filter, write a Matlab function **LPFtrunc**(N) that computes the truncated and shifted impulse response of size N for a lowpass filter with a cutoff frequency of $\omega_c = 2.0$. For each of the following filter sizes, plot the magnitude of the filter's DTFT in decibels.

HINTS: The magnitude of the response in decibels is given by $\left|H_{dB}(e^{j\omega})\right| = 20\log_{10}\left|H(e^{j\omega})\right|$.

Note that the **log** command in Matlab computes the natural logarithm. Therefore, use the **log10** command to compute decibels. To get an accurate representation of the DTFT make sure that you compute at least 512 sample points using the command [X ,w]=DTFT(filter_response,512) .

i) $N = 21$, and ii) $N = 101$ (7.16)

Now download the noisy speech signal **nspeech2.mat**, and load it into the Matlab workspace.

Apply the two filters with the sizes specified in (7.16) to this signal. Since these are FIR filters, you can simply convolve them with the audio signal. Listen carefully to the unfiltered and filtered signals, and note the result. Can you hear a difference between the two filtered signals? In order to hear the filtered signals better, you may want to multiply each of them by 2 or 3 before using sound.

INLAB REPORT

- Submit the plots of the magnitude response for the two filters (not in decibels). On each of the plots, mark the passband, the transition band and the stopband.
- Submit the plots of the magnitude response in decibels for the two filters.
- Explain how the filter size effects the stopband ripple. Why does it have this effect?
- Comment on the quality of the filtered signals. Does the filter size have a noticeable effect on the audio quality?

Lab 8 - Digital Filter Design (part 2)

8.1 Introduction

This is the second part of a three week laboratory in digital filter design. The two weeks of the laboratory covered some basic examples of FIR and IIR filters, and then introduced the concepts of filter design. In this week we will cover more systematic methods of designing FIR filters.

8.2 Filter Design Using Standard Windows

Download **DTFT.m** for the following section.

We can generalize the idea of truncation by using different windowing functions to truncate an ideal filter's impulse response. Note that by simply truncating the ideal filter's impulse response, we are actually multiplying (or "windowing") the impulse response by a shifted $rect(\,)$ function. This particular type of window is called a **rectangular** window. In general, the impulse response $h[n]$ of the designed filter is related to the impulse response $h_{ideal}[n]$ of the ideal filter by the relation

$$h[n] = w[n]h_{ideal}[n] \tag{8.1}$$

where $w[n]$ is an *N*-point window. We assume that

$$h_{ideal}[n] = \frac{\omega_c}{\pi}\text{sinc}\left(\frac{\omega_c}{\pi}\left(n - \frac{N-1}{2}\right)\right) \tag{8.2}$$

where ω_c is the cutoff frequency and N is the desired window length.

The rectangular window is defined as

$$w[n] = \begin{cases} 1 & n = 0,1,\dots,N-1 \\ 0 & \text{otherwise} \end{cases} \tag{8.3}$$

As we have seen in the last lab, the DTFT of $w[n]$ for $N = 21$ is shown in Figure 7.7. The rectangular window is usually not preferred because it leads to the large stopband and passband ripple as shown in Figure 7.6.

More desirable frequency characteristics can be obtained by making a better selection for the window, $w[n]$. In fact, a variety of raised cosine windows are widely used for this purpose. Some popular windows are listed below.

1. Hanning window (as defined in Matlab, command hann(N)):

$$w[n] = \begin{cases} 0.5 - 0.5\cos\dfrac{2\pi n}{N-1} & n = 0,1,\dots,N-1 \\ 0 & \text{otherwise} \end{cases} \tag{8.4}$$

2. Hamming window

$$\omega[n] = \begin{cases} 0.54 - 0.46\cos\dfrac{2\pi n}{N-1} & n = 0,1,\dots,N-1 \\ 0 & \text{otherwise} \end{cases} \tag{8.5}$$

3. Blackman window

$$\omega[n] = \begin{cases} 0.42 - 0.5\cos\dfrac{2\pi n}{N-1} + 0.08\cos\dfrac{4\pi n}{N-1} & n = 0,1,\dots,N-1 \\ 0 & \text{otherwise} \end{cases} \tag{8.6}$$

In filter design using different truncation windows, there are two key frequency domain effects that are important to the design: the transition band **roll-off**, and the passband and stopband **ripple** (see Figure 7.4 in last section). There are two corresponding parameters in the spectrum of each type of window that influence these filter parameters. The filter's roll-off is related to the **width** of center lobe of the window's magnitude spectrum. The **ripple** is influenced by the ratio of the mainlobe amplitude to the first sidelobe amplitude (or difference if using a dB scale). These two window spectrum parameters are **not** independent, and you should see a trend as you examine the spectra for different windows. The theoretical values for the **mainlobe width** and the **peak-to-sidelobe amplitude** are shown in Table 8.1.

Table 8.1: Approximate spectral parameters of truncation windows. See reference [1].

Window (length N)	Mainlobe width	Peak-to-sidelobe amplitude (dB)
Rectangular	$4\pi/N$	$-13dB$
Hanning	$8\pi/N$	$-32dB$
Hamming	$8\pi/N$	$-43dB$
Blackman	$12\pi/N$	$-58dB$

Plot the rectangular, Hamming, Hanning, and Blackman window functions of length 21 on a single figure using the subplot command. You may use the Matlab commands **hamming**, **hann**, and **blackman**. Then compute and plot the DTFT magnitude of each of the four windows. Plot the magnitudes on a decibel scale (i.e., plot $20\log_{10}\left|W(e^{j\omega})\right|$). Download and use the function **DTFT.m** to compute the DTFT.

NOTE: Use at least 512 sample points in computing the DTFT by typing the command DTFT(window,512).

Measure the null-to-null mainlobe width (in rad/sample) and the peak-to-sidelobe amplitude (in dB) from the logarithmic magnitude response plot for each window type. The Matlab command **zoom** is helpful for this. Make a table with these values and the theoretical ones.

Now use a Hamming window to design a lowpass filter $h[n]$ with a cutoff frequency of $\omega_c = 2.0$ and length 21. Note: You need to use (8.1) and (8.2) for this design. In the same figure, use

subplots to plot the filter's impulse response, and the magnitude of the filter's DTFT in decibels.

INLAB REPORT:

1. Submit the figure containing the time domain plots of the four windows.
2. Submit the figure containing the DTFT (in decibels) of the four windows.
3. Submit the table of the measured and theoretical window spectrum parameters. Comment on how close the experimental results matched the ideal values. Also comment on the relation between the width of the mainlobe and the peak-to-sidelobe amplitude.
4. Submit the plots of your designed filter's impulse response and the magnitude of the filter's DTFT.

8.3 Filter Design Using the Kaiser Window

Download **nspeech2.mat** for the following section.

The standard windows of the "Filter Design Using Standard Windows" section are an improvement over simple truncation, but these windows still do not allow for arbitrary choices of transition bandwidth and ripple. In 1964, James Kaiser derived a family of near-optimal windows that can be used to design filters which meet or exceed any filter specification. The Kaiser window depends on two parameters: the window length N, and a parameter β which controls the shape of the window. Large values of β reduce the window side lobes and therefore result in reduced passband and stopband ripple. The only restriction in the Kaiser filter design method is that the passband and stopband ripple must be equal in magnitude. Therefore, the Kaiser filter must be designed to meet the smaller of the two ripple constraints:

$$\delta = \min\{\delta_p, \delta_s\} \tag{8.7}$$

The Kaiser window function of length N is given by

$$\omega[n] = \begin{cases} \dfrac{I_0\left(\beta\sqrt{\dfrac{n(N-1-n)}{N-1}}\right)}{I_0(\beta)} & n = 0,1,\dots,N-1 \\ 0 & \text{otherwise} \end{cases} \tag{8.8}$$

Where $I_0(\cdot)$ is the zero-th order modified Bessel function of the first kind, N is the length of the window, and β is the shape parameter.

Kaiser found that values of β and N could be chosen to meet any set of design parameters, $(\delta, \omega_P, \omega_S)$, by defining $A = -20\log_{10}\delta$ and using the following two equations:

$$\beta = \begin{cases} 0.1102(A-8.7) & A > 50 \\ \{0.5842(A-21)^{0.4} + 0.07886(A-21) & 21 \le A \le 50 \\ 0.0 & A < 21 \end{cases} \tag{8.9}$$

$$N = \left\lceil 1 + \frac{A-8}{2.285(\omega_s - \omega_p)} \right\rceil$$

(8.10)

where $\lceil \cdot \rceil$ is the **ceiling** function, i.e. $\lceil x \rceil$ is the smallest integer which is greater than or equal to x.

To further investigate the Kaiser window, plot the Kaiser windows and their DTFT magnitudes (in dB) for $N = 21$ and the following values of β:

• $\beta = 0$ • $\beta = 1$ • $\beta = 5$

For each case use at least 512 points in the plot of the DTFT.

HINT: To create the Kaiser windows, use the Matlab command kaiser(N,beta) command where N is the length of the filter and beta is the shape parameter β. To insure at least 512 points in the plot use the command DTFT(window,512) when computing the DTFT.

INLAB REPORT: Submit the plots of the 3 Kaiser windows and the magnitude of their DTFT's in decibels. Comment on how the value β affects the shape of the window and the sidelobes of the DTFT.

Next use a Kaiser window to design a lowpass filter, $h(n)$, using equations (8.1) and (8.2), to remove the noise from the signal in **nspeech2.mat**. To do this, use equations (8.9) and (8.10) to compute the values of β and N that will yield the following design specifications:

$$\omega_p = 1.8,\ \omega_c = 2.0,\ \omega_s = 2.2,\ \delta_p = 0.05,\ \delta_s = 0.005$$

The lowpass filter designed with the Kaiser method will automatically have a cut-off frequency centered between ω_p and ω_s.

$$\omega_c = \frac{\omega_p + \omega_s}{2}$$

Plot the magnitude of the DTFT of $h[n]$ for $|\omega| < \pi$. Create three plots in the same figure: one that shows the entire frequency response, and ones that zoom in on the passband and stopband ripple, respectively. Mark ω_p, ω_s, δ_p, and δ_s on these plots where appropriate. Note: Since the ripple is measured on a magnitude scale, DO NOT use a decibel scale on this set of plots.

From the Matlab prompt, compute the stopband and passband ripple (do not do this graphically). Record the stopband and passband ripple to three decimal places.

HINT: To compute the passband ripple, find the value of the DTFT at frequencies corresponding to the passband using the command H(abs(w)<=1.8) where H is the DTFT of $h[n]$ and w is the corresponding vector of frequencies. Then use this vector to compute the passband ripple. Use a similar procedure for the stopband ripple.

Filter the noisy speech signal in **nspeech2.mat** using the filter you have designed. Then compute the DTFT of 400 samples of the filtered signal starting at time $n = 20000$ (i.e. 20001:20400). Plot the magnitude of the DTFT samples in decibels versus frequency in radians for $|\omega| < \pi$. Compare this with the spectrum of the noisy speech signal shown in Figure 7.5. Play the noisy and filtered speech signals back using **sound** and listen to them carefully.

INLAB REPORT: Do the following:

1. Submit the values of β and N that you computed.
2. Submit the three plots of the filter's magnitude response. Make sure the plots are labeled.
3. Submit the values of the passband and stopband ripple. Does this filter meet the design specifications?
4. Submit the magnitude plot of the DTFT in dB for the filtered signal. Compare this plot to the plot of Figure 7.5.
5. Comment on how the frequency content and the audio quality of the filtered signal have changed after filtering.

8.4 FIR Filter Design Using Parks-McClellan Algorithm

Click nspeech2.mat for help on the **firpm** function for Parks-McClellan filter design. Download the data file nspeech2.mat for the following section.

Kaiser windows are versatile since they allow the design of arbitrary filters which meet specific design constraints. However, filters designed with Kaiser windows still have a number of disadvantages. For example,

- Kaiser filters are not guaranteed to be the minimum length filter which meets the design constraints.
- Kaiser filters do not allow passband and stopband ripple to be varied independently.

Minimizing filter length is important because in many applications the length of the filter determines the amount of computation. For example, an FIR filter of length N may be directly implemented in the time domain by evaluating the expression

$$y(n) = \sum_{k=0}^{N-1} x(n-k)h(k) \tag{8.11}$$

For each output value $y(n)$ this expression requires N multiplies and $N-1$ additions.

Oftentimes $h(n)$ is a symmetric filter so that $h(n) = h(N-1-n)$. If the filter $h(n)$ is symmetric and N is even, then (6.16) may be more efficiently computed as

$$y(n) = \sum_{k=0}^{N/2-1} \big(x(n-k) + x(n-N+1+k)\big)h(k) \tag{8.12}$$

This strategy reduces the computation to $N/2$ multiplies and $N-1$ adds for any value of N. Note that the computational effort is linearly proportional to the length of the filter.

The Kaiser filters do not guarantee the minimum possible filter length. Since the filter has equal passband and stopband ripple, it will usually exceed design requirements in one of the two bands; this results in an unnecessarily long filter. A better design would allow the stopband and passband constraints to be specified
separately.

In 1972, Parks and McClellan devised a methodology for designing symmetric filters that minimize filter length for a particular set of design constraints $\{\omega_p, \omega_s, \delta_p, \delta_s\}$. The resulting filters minimize the maximum error between the desired frequency response and the actual frequency

response by spreading the approximation error uniformly over each band. The Parks and McClellan algorithm makes use of the Remez exchange algorithm and Chebyshev approximation theory. Such filters that exhibit **equiripple** behavior in both the passband and the stopband, and are sometimes called equiripple filters.

As with Kaiser filters, designing a filter with the Parks and McClellan algorithm is a two-step process. First the length (i.e. order) of the filter must be computed based on the design constraints. Then the optimal filter for a specified length can be determined. As with Kaiser windows, the filter length computation is approximate so the resulting filter may exceed or violate the design constraints. This is generally not a
problem since the filter can be redesigned for different lengths until the constraints are just met.

The Matlab command for computing the approximate filter length is

[n, fo, mo, w] = firpmord(f, m, ripple, 2*pi) where the inputs are:

f - vector containing an even number of band edge frequencies. For a simple low pass filter, **f = [wp,ws]** ,where **wp** and **ws** are the passband and stopband frequencies, respectively.

m - vector containing the ideal filter magnitudes of the filter in each band. For a simple low pass filter **m = [1, 0]** .

ripple - vector containing the allowed ripple in each band. For a simple low pass filter **ripple = [delta_p, delta_s]**, where **delta_p** and **delta_s** are the passband and stopband ripples, respectively.

2*pi - value, in radians, that corresponds to the sampling frequency.

The outputs of the command are **n = filter length - 1**, and the vectors **fo**, **mo**, and **w** which are intermediate filter parameters.

Once the filter length, **n**, is obtained, the Matlab command for designing a Parks-McClellan filter is **b = firpm(n, fo, mo, w)** . The inputs **n**, **fo**, **mo**, and **w** are the corresponding outputs of **firpmord**, and the output **b** is a vector of FIR filter coefficients such that

$$H(z) = b(1) + b(2)z^{-1} + \cdots + b(n+1)z^{-n} \tag{8.13}$$

(What is the impulse response of this filter?)

For further information, read the help document on using Matlab to implement the Parks-McClellan algorithm.

Now design a symmetric FIR filter using **firpmord** and **firpm** in Matlab to meet the design specifications given in the "Filter Design Using the Kaiser Window" (Section 7.3: Filter Design Using the Kaiser Window) section. Compute the DTFT of the filter's response for at least 512 points, and use this result to compute the passband and stopband ripple of the filter that was designed. Adjust the filter length until the minimum order which meets the design constraints is found. Plot the magnitude of the DTFT in dB of the final filter design.

INLAB REPORT

1. Submit the final measured values of filter length, passband ripple, and stopband ripple. How accurate was the filter order computation using Matlab's firpmord? How does the length of this filter compare to the filter designed using a Kaiser window?
2. Submit the plot of the filter's DTFT. How does the frequency response of the Parks-McClellan filter compare to the filter designed using the Kaiser window? Comment on the shape of both the passband and stopband.

Use the filter you have designed to remove the noise from the signal nspeech2.mat . Play the noisy and filtered speech signals back using sound and listen to them carefully. Compute the DTFT of 400 samples of the filtered signal starting at time $n = 20{,}001$ (i.e. 20001:20400). Plot the magnitude of the DTFT in decibels versus frequency in radians for $|\omega| < \pi$. Compare this with the spectrum of the noisy speech signal shown in Figure 7.5, and also with the magnitude of the DTFT of the Kaiser filtered signal.

INLAB REPORT: Submit the plot of the DTFT magnitude for the filtered signal. Comment on how the audio quality of the signal changes after filtering. Also comment on any differences in audio quality between the Parks-McClellan filtered speech and the Kaiser filtered speech.

Lab 9 - Frequency Response Masking Filters (Optional)

For given passband ripple and stopband attenuation, the filter order, as well as the number of non-trivial coefficients, of a direct form FIR filter, is inversely proportional to the transition width. Therefore, sharp filters suffer from high computational complexity due to the high filter orders. Lim [1] proposed a frequency response masking (FRM) FIR filter structure in 1986 for the design of sharp FIR filters with low computational complexity. The structure was thereafter modified and improved by many researchers to achieve even further computation savings, and has been extended to the design of various types of filters, such as half-band filters, multirate filters, recursive filters and two-dimensional filters.

9.1 FRM Structure

Just as the name implies, FRM uses masking filters to obtain the desired frequency responses. The overall structure of an FRM is a subfilter network, consisting of three subfilters, as shown in Figure 9.1. The three subfilters are a prototype band edge shaping filter (or simply called prototype filter) with z-transform transfer function $F(z)$ and two masking filters with z-transform transfer functions $G_1(z)$ and $G_2(z)$, respectively; their corresponding frequency responses are denoted as $F(e^{j\omega})$, $G_1(e^{j\omega})$, and $G_2(e^{j\omega})$, respectively. A fourth filter function is realized by using a delay-complementary counterpart.

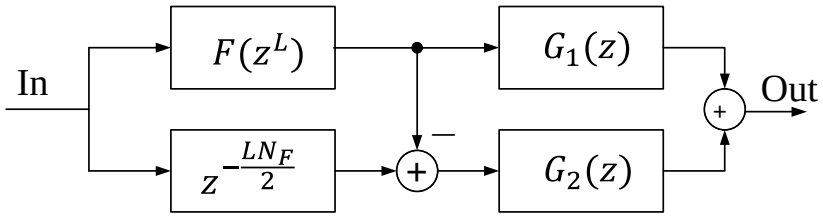


Figure 9.1: Structure of the frequency response masking filter.

In FRM structure, every delay of the prototype band edge shaping filter $F(z)$ is replaced by L delays. The frequency response of the resulting filter, $F(z^L)$, is a frequency compressed (by a factor of L) and periodic version of $F(z)$, as shown in Figures 9.2 (a) and (b). For this reason, the number of delays replacing one delay in the impulse response of the band edge shaping prototype filter, L, is referred to as the compression factor of the FRM filter.

As a result of the delay element replacement, the transition band of $F(z)$ is mapped to L transition bands with transition width shrunk by a factor of L. The complementary filter of $F(z^L)$ is obtained by subtracting $F(z^L)$ from a pure delay term, $z^{-LN_F/2}$, where N_F is the order of $F(z)$. To avoid a half delay, N_F is chosen to be even. Thus, the entire frequency region from DC to Nyquist frequency is decomposed into L bands; $F(z^L)$ and its complement hold alternate band respectively, as shown in Figures 9.2 (b).

Two masking filters $G_1(z)$ and $G_2(z)$ are used to filter (i.e., mask) the outputs of $F(z^L)$ and its

complement. The desired frequency bands of $F(z^L)$ and its complement are kept and then combined to obtain the final frequency response. The overall z-transform transfer function of the FRM structure, $H(z)$, is thus given by,

$$H(z) = F(z^L)G_1(z) + \left[z^{-\frac{LN_F}{2}} - F(z^L)\right]G_2(z) \tag{9.1}$$

In a synthesis problem, the passband and stopband edges of the overall filter $H(z)$, denoted ω_p and ω_s, respectively, are given, whereas the passband and stopband edges of the prototype filter $F(z)$, denoted ω_p^F and ω_s^F, have to be determined. From Figure 9.2 b, it can be seen that the transition band of the overall filter $H(z)$ may be generated either from $F(z^L)$ or from its complement.

We denote the two cases as Case A and Case B. The filter $F(z^L)$ (or its complement) is thus called periodic band edge shaping filter, or simply periodic filter or band edge shaping filter. We use lowpass filters as examples to illustrate the FRM technique. The extension of the technique to other frequency selective filters is straightforward.

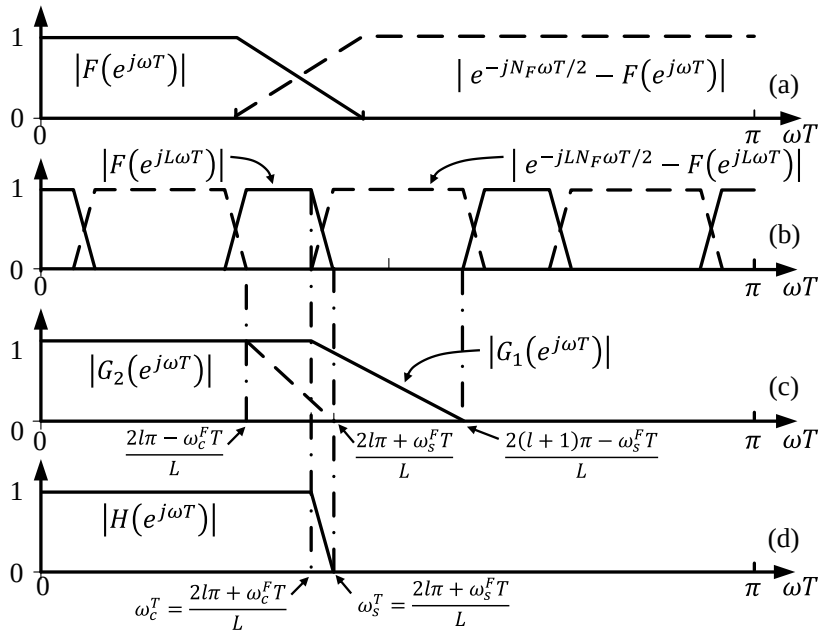


Figure 9.2: Frequency response of FRM subfilters of Case A.

9.1.1 Case A

In case A, the transition band of $H(z)$ is formed by the l-th transition band of $F(z^L)$, as shown in Figure 9.2, i.e.,

$$\omega_p = \frac{2l\pi + \omega_p^F}{L} \quad \text{and} \quad \omega_s = \frac{2l\pi + \omega_s^F}{L} \tag{9.2}$$

respectively. For a lowpass filter $H(z)$, the band edge of the prototype filter $F(z)$ must have $0 < \omega_p^F < \omega_s^F < \pi$, and meanwhile, we have $\omega_p < \omega_s$ and l must be an integer. According to (9.2), there is only one value of l that can meet these requirements if there is a solution, i.e., we must

select $l = \lfloor L\omega_p/2\pi \rfloor$, where $\lfloor x \rfloor$ is the largest integer less than or equal to x.

Thus, for Case A, we have

$$\omega_p^F = L\omega_p - 2l\pi, \;\; \omega_s^F = L\omega_s - 2l\pi \;\; \text{for} \;\; l = \lfloor L\omega_p/2\pi \rfloor \tag{9.3}$$

9.1.2 Case B

In Case B, the transition band of $H(z)$ is formed by the $(l-1)$-th transition band of $F(z^L)$'s complement, as shown in Figure 9.3, i.e.,

$$\omega_p = \frac{2l\pi - \omega_s^F}{L} \quad \text{and} \quad \omega_s = \frac{2l\pi - \omega_p^F}{L} \tag{9.4}$$

respectively. For the similar requirements as in Case A, the only value of l if there is a solution is $l = \lceil L\omega_s/2\pi \rceil$, where $\lceil x \rceil$ is the smallest integer larger than or equal to x. Thus, for Case B, we have

$$\omega_p^F = 2l\pi - L\omega_s, \;\; \omega_s^F = 2l\pi - L\omega_p \;\; \text{for} \;\; l = \lfloor L\omega_p/2\pi \rfloor \tag{9.5}$$

Once the band edges of the prototype filter $F(z)$ are determined, for a given L, the band edges of the masking filters $G_1(z)$ and $G_2(z)$ may be chosen to keep the desired bands to obtain the overall responses. The band edge values of the masking filters are shown in Figures 9.2 (c) and 9.3 (c), respectively, for Case A and Case B.

Overall, Figure 9.2 shows a Case A example, whereas Figure 9.3 shows a Case B example.

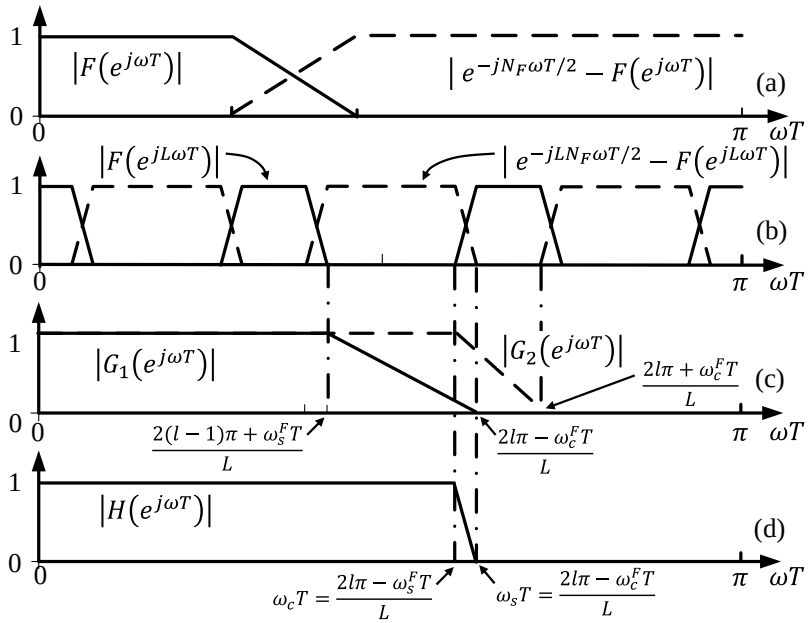


Figure 9.3: Frequency response of FRM subfilters of Case B.

9.2 Conditions for Feasible FRM Solutions

The conditions derived for Case A and Case B in sections 9.1.1 and 9.1.2 assume that a solution is available for a given set of ω_p, ω_s and L. It is possible that neither Case A nor Case B generates a feasible solution for a combination of ω_p, ω_s and L.

First, obviously only when $L > 2\pi/\omega_p$ and $L > 2\pi/(\pi - \omega_s)$, may FRM technique be considered; otherwise, the passband of the filter is too narrow or too wide to be obtained by combining bands from both band edge shaping filter and its complement. Besides the above condition, from (9.2), (9.4), and the fact that $0 < \omega_p^F < \omega_s^F < \pi$, and $\omega_p < \omega_s$, for a lowpass design, we can further find that L must satisfy

$$\left\lceil \frac{L\omega_p}{\pi} \right\rceil \neq \left\lfloor \frac{L\omega_s}{\pi} \right\rfloor, \tag{9.6}$$

so that either Case A or Case B, but not both, leads to a feasible solution. When (9.6) is violated, no feasible solution exists.

9.3 Selection of L

Since the transition width of the prototype filter $F(z)$ is $L(\omega_s - \omega_p)$ for given ω_p and ω_s, and the sum of the transition width of $G_1(z)$ and $G_2(z)$ is $2\pi/L$, the complexity of $F(z)$ decreases with increasing L, while the complexity of $G_1(z)$ and $G_2(z)$ increases with increasing L. It has shown that the minimum arithmetic complexity in terms of the number of multipliers is achieved by selecting

$$L = \frac{1}{\sqrt{2\beta(\omega_s - \omega_p)/\pi}} \tag{9.7}$$

where $\beta = 1$ if the subfilters are optimized separately, and $\beta = 0.6$ if the subfilters are optimized jointly.

9.4 Computational Complexity

The basic FRM structure utilizing the complementary periodic filter pairs and masking filters synthesizes sharp filter with low arithmetic complexity. When

$$\omega_s - \omega_p < 0.126\pi \text{ rad},$$

the arithmetic complexity saving is achieved by using FRM structure, compared with the direct form FIR filters. In a typical design with $\omega_s - \omega_p < 0.002$ rad, the FRM structure reduces the number of multipliers by a factor of 10, and the price paid for the arithmetic saving is a less than 5% increase in the overall filter order.

9.5 Design Procedure

For a given set of lowpass filter specifications, including passband edge, stopband edge, passband ripple and stopband ripple $\omega_p, \omega_s, \delta_p$ and δ_s, respectively, a general design procedure, where all the subfilters are optimized separately, is given as follows:

Step 1. Determine the optimum L that minimizes the filter complexity according to (9.7). If the selected L violates (9.6), increase or decrease L by 1.

Step 2. Determine the passband and stopband edges of the prototype band edge shaping filter according to (9.3) or (9.5).

Step 3. Determine the passband and stopband edges of the masking filters according to the parameters given in Figure 9.2 (c) or Figure 9.3 (c).

Step 4. Design the prototype band edge shaping filter and masking filters with 50% ripple margins.

Step 5. Verify the frequency response of the resulting FRM filter. If the specification has been satisfied, then the design is successful; otherwise, increase the ripple margin, and go to **Step 4**.

9.6 Design Exercise

Use FRM technique to design an FIR lowpass filter with passband edge, stopband edge, passband ripple and stopband ripple 0.4 rad, 0.402 rad, 0.01 and 0.001, respectively.

INLAB REPORT: Show the parameters of the design, for example, L, bandedges of the subfilters, filter lengths of the subfilters, etc. State if the design is a CASE A or CASE B design. Submit the plot of the frequency responses of the subfilters and the overall FRM filter. Comment on the complexity of the designed filter, compared with the direct form design.

Mini Project: 计算机生成和播放音乐

1. Objective

音乐文件有很多格式，但是其主要的数据（Data）部分，保存的都是一个 M * N 的矩阵（其中，N 为通道数）。因此，对于一个乐谱，我们可以用计算机程序生成这样一个矩阵，并用计算机播放出来。本项目的目标就是：

1. 理解音乐的基本要素，比如音调、节拍、音色、基频、和弦等，
2. 理解这些音乐要素如何反映在我们的数据中，
3. 通过 MATLAB 程序，将一张简谱转化成一段可以播放成音乐的数据，
4. 根据数据，产生可播放的音乐，
5. 根据数据，产生模仿某种乐器的音乐

2 数字简谱

数字简谱是简易的记谱法，用基本音符 1、2、3、4、5、6、7 代表音阶中的 7 个基本级，读音为 do、re、mi、fa、sol、la、ti（中国为 si），英文由 C、D、E、F、G、A、B 表示，休止以 0 表示。图 1 为《天空之城》数字简谱。简谱中主要包含了数字音符和一些其他符号，比如点和横线，这些数字配合上符号决定了每个音的频率和持续时间长度，我们需要通过编码将其输入到程序中，并用来控制音乐的播放。

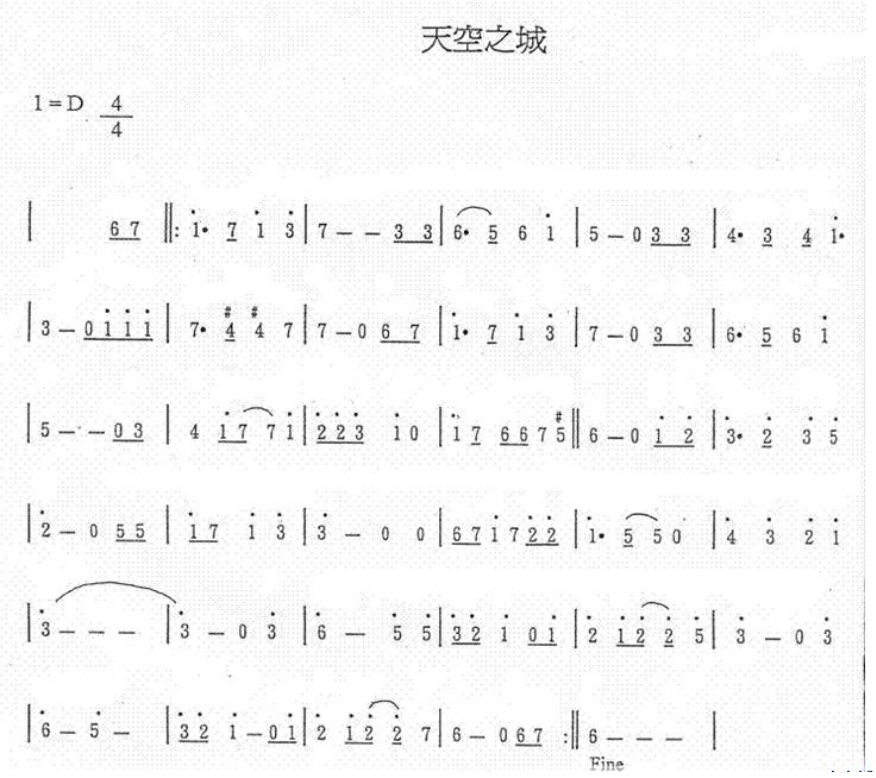


图 1.《天空之城》 数字简谱

2.1 音调

简谱中 1、2、3、4、5、6、7（C、D、E、F、G、A、B）表示七种高低不同的音，我们称之为音调。之所以会呈现不同音调，是因为他们分别对应不同频率的音波。表 1 为 C 调音符与频率对应表。

表 1.C 调中音符与频率(取整后)对应表，单位为 Hz

	1（C）	2(D)	3(E)	4(F)	5(G)	6(A)	7(B)
低音	262	294	330	349	392	440	494
中音	523	587	659	698	784	880	988
高音	1046	1175	1318	1397	1568	1760	1976
相邻音符频率比 $n+1/n$	1.12	1.12	1.057	1.12	1.12	1.12	1.058

从表 1 可以看出，**低音频率为同一音符中音频率的一半，同样的，中音频率为同一音符高音频率的一半。**也就是说，对应的中音音符和低音音符之间，频率减半，而音程相差一个八度。记在简谱基本音符号**下面**的小圆点，叫低音点，它表示将基本音符降低一个音组，即降低一个纯八度。记两个圆点表示将基本音符号降低两个音组，即降低两个纯八度。记在简谱基本音符号**上面**的小圆点，叫高音点，它表示将基本音符号升高一个音组，即升高一个纯八度。记两个圆点，表示升高两个音组，即升高两个纯八度。

同时，稍加计算，也可以发现，相邻音符之间的频率比，2:1，3:2，5:4，6:5，7:6 的频率比值均为 1.12($2^{\frac{1}{6}}$)左右,而，4:3，1i:7 则为 1.057($2^{\frac{1}{12}}$)左右。由于相邻之间音符的频率比值不完全相等，要通过一个音符(比如 1)来计算其他音符的频率有点困难,不过,进一步分析,$\frac{1i}{1}=\frac{1i}{7}\times\frac{7}{6}\times\frac{6}{5}\times\frac{5}{4}\times\frac{4}{3}\times\frac{3}{2}\times\frac{2}{1}=2^{\frac{12}{12}}=2$，这也符合上面介绍的中音频率与低音频率之间的比值规律。因此，可以将一组音划分成十二等分，使得相邻音符的频率比值为相等，即$2^{\frac{1}{12}}$，这样间隔 12 个音符以后频率翻倍，这种音律叫十二平均律，亦称“十二等程律”，其将世界上通用的一组音（八度）分成十二个半音音程（$2^{\frac{1}{12}}$），因此，如果你知道了一组音中一个音符的频率（比如 1），那你就能计算出这一组音中其他音符的频率，在结合间隔八度相差一倍/一半的规律，你能计算出简谱中其他所有音符的频率。图 2 举例说明了如何用十二平均律来计算一组音中所有音阶的频率。图 2 的例子中，主音设定为 440Hz，以此为基准来计算 12 等份的其他音调。主音在乐理中有特别的含义，但在这里，我们当前只需要简单的把它理解为一个基准，更具体的我们将在下一部分解释。而对应到基本音级中，我们可以发现，3 - 4 和 7 - 1i 之间的间隔是一个半音，而其它相邻两个音之间的间隔都是全音（2 个半音），因此，我们也可以通过其中一个音符的频率来计算出其他音符的频率。

此外，简谱中还有升（降）调的符号#（b）号，升调表示比原音级高 1 个半音，降调同理，加上升降调的话，7 个音级就扩展到了十二平均律中的十二个音级，以 C 调为例，如图 3 所示。

总结一下，本节主要介绍在知道一个音符频率下，如何计算其他音符的频率。在十二平均律下，所有音符之间的频率比值相等，因此可以通过这个倍数来计算。在 1-7 的七个基本音级中，3 - 4、7 - 1i 的间隔是半音（$2^{\frac{1}{12}}$），而其它相邻两个音的间隔都是全音（2 个半音）($2^{\frac{1}{6}}$)，因此我们只需要知道其他音符对应于已知音符的位置关系，就可以通过这个频率间隔关系来计算。此外，如果相差了若干个八度，则应先

计算同一组七个音符的频率，再按照相差一个八度，频率翻倍或减半来计算。此外，简谱上出现的数字 1 – 7 加上升降调就有了 12 种音符，再加上上下的小圆点，就有了 12 x N（N 个八度）种音符。因此在对简谱编码时，就要考虑用一种合适的方式将这 12 x N 个音符输入进去。比如，用 3 个变量，分别控制 1 – 7 的基本音级、升降调、相差多少个八度；或者用 1 – 12 表示 1 – 7 的基本音级加上升降调，13 – 24 表示高一个八度，–12 – –1 表示低一个八度。此外还有考虑停顿（0）。对应不同的编码方式需要用不同的方式将简谱输入。每个人可以采用自己喜欢的方式进行编码。但是，在接下来的练习中，请务必在每个函数中提供注释介绍你的编码方式。

^ 十二平均律表

将主音设为a1(440Hz)，来计算所有音的频率，结果如下 *（为计算过程更清晰，分数不进行约分）*：

音程名称	间隔半音数	十二平均律的倍数	频率
纯一度 (A^1)	0	$2^0 = 1$	$440 \times 1 = 440$
增一度/小二度 ($A\#^1/B\flat^1$)	1	$\sqrt[12]{2} = 2^{\frac{1}{12}} \approx 1.059463094359295264561825294946 3$	$440 \times 2^{\frac{1}{12}} \approx 466.16376151808991640720312977 62$
大二度 (B^1)	2	$\sqrt[6]{2} = 2^{\frac{2}{12}} \approx 1.122462048309372981433533049679 2$	$440 \times 2^{\frac{2}{12}} \approx 493.88330125612411183075454185 86$
小三度 (C)	3	$\sqrt[4]{2} = 2^{\frac{3}{12}} \approx 1.189207115002721066717499970560 5$	$440 \times 2^{\frac{3}{12}} \approx 523.25113060119726935569998704 66$
大三度 (C#)	4	$\sqrt[3]{2} = 2^{\frac{4}{12}} \approx 1.259921049894873164767210607278 2$	$440 \times 2^{\frac{4}{12}} \approx 554.36526195374419249757266720 23$
纯四度 (D)	5	$\sqrt[12]{32} = 2^{\frac{5}{12}} \approx 1.334839854170034364830831881184 5$	$440 \times 2^{\frac{5}{12}} \approx 587.32953583481512052556602772 09$
增四度/减五度 (D#/Eb)	6	$\sqrt{2} = 2^{\frac{6}{12}} \approx 1.414213562373095048801688724209 7$	$440 \times 2^{\frac{6}{12}} \approx 622.25396744416182147274303865 22$
纯五度 (E)	7	$\sqrt[12]{128} = 2^{\frac{7}{12}} \approx 1.498307076876681498799280732029 8$	$440 \times 2^{\frac{7}{12}} \approx 659.25511382573985947168352209 30$
小六度 (F)	8	$\sqrt[3]{4} = 2^{\frac{8}{12}} \approx 1.587401051968199474751705639272 3$	$440 \times 2^{\frac{8}{12}} \approx 698.45646286600776889075048127 95$
大六度 (F#)	9	$\sqrt[4]{8} = 2^{\frac{9}{12}} \approx 1.681792830507429086062250952466 4$	$440 \times 2^{\frac{9}{12}} \approx 739.98884542326879786739041908 52$
小七度 (G)	10	$\sqrt[6]{32} = 2^{\frac{10}{12}} \approx 1.781797436280678609480452411181$	$440 \times 2^{\frac{10}{12}} \approx 783.99087196349858871371399060 9195$
大七度 (G#)	11	$\sqrt[12]{2048} = 2^{\frac{11}{12}} \approx 1.887748625363386993283826313335 1$	$440 \times 2^{\frac{11}{12}} \approx 830.60939515989027704488357786 70$
纯八度 (A)	12	$2^1 = 2$	$440 \times 2 = 880$

其中 $\sqrt[12]{2} = 2^{\frac{1}{12}} \approx 1.0594630943593$

$\approx \frac{18}{17} = 1.05882$ 99 音分

$\approx \frac{107}{101} = 1.05941$ 99.9 音分

$\approx \frac{11011}{10393} = 1.05946310$ 100 音分

图 2.十二平均律表

C调音符与频率对照表

音符	频率/Hz	音符	频率/Hz	音符	频率/Hz
低音1	262	中音1	523	高音1	1046
低音1#	277	中音1#	554	高音1#	1109
低音2	294	中音2	587	高音2	1175
低音2#	311	中音2#	622	高音2#	1245
低音3	330	中音3	659	高音3	1318
低音4	349	中音4	698	高音4	1397
低音4#	370	中音4#	740	高音4#	1480
低音5	392	中音5	784	高音5	1568
低音5#	415	中音5#	831	高音5#	1661
低音6	440	中音6	880	高音6	1760
低音6#	466	中音6#	932	高音6#	1865
低音7	494	中音7	988	高音7	1976

图 3 C 调音符与频率对应表

MATLAB 练习 1：编写函数

```
function freq = tone2freq(tone, noctave, rising)
% tone: 输入数字音符，数值范围1到7
% noctave: 高或低八度的数量，数值范围整数。0表示中音，正数表示高noctave个八度，负数为低noctave个八度
% rising: 升或降调。1为升，–1为降，0无升降调
% freq为输出的频率
```

以 1=440 Hz 为主音，编写函数，要求能够输出 1－7 音符，高若干八度，低若干八度，以及升降调的频率。

2.2 调号

2.1 中，我们介绍了在已知 1 个音符的频率的前提下计算其他音符的频率。因此，我们需要前提知道这个音符的频率。在简谱的开头，我们会看到 1=C 或 1=D 这样的记号，这表示该简谱是以 C 调或 D 调来记谱的，同时也规定了该简谱中 1 的频率（主音）。C 调或 D 调就是我们一般所谓的调号，不同调号之间 1 的频率（主音）有一定的对应关系，这个关系我们通过表 2 来介绍。表 2 中，第一行我们给出了 C 大调下音符 1-7 的频率，第二行是这七个音符的数字符号和英文符号，剩下的行给出了 C-B 调中的主音频率。你会发现，C 调（1=C）中 1 的频率就是 C 调中 1（C）的频率（近乎于废话），D 调（1=D）中 1 的频率就是 C 调中 2(D)的频率，E 调（1=E）中 1 的频率就是 C 调中 3(E)的频率，以此类推，也就是说，只要知道了 C 调的频率，同一组其他调的主音 1 的频率就可以按照这个音符对应 C 调中 1 的位置关系来计算，这样我们就能确定其他调的主音频率了。

总结，本节主要介绍调号。调号确定了该简谱中 1 的频率，再结合上 2.1 中介绍的规律，就能算出整张简谱中所有音符的频率。所以，我们首先需要计算 C 调中 7 个音符的频率，然后计算其他调中的主音频率（如果简谱用其他调写的），最后再计算剩下的其他音符的频率。

表 2 不同调号主音频率，单位 Hz

C 调频率	261.5	293.5	329.5	349	391.5	440	494
C 调音符（英文名）	1(C)	2(D)	3(E)	4(F)	5(G)	6(A)	7(B)
C 调	1=C (261.5)						
D 调		1=D (293.5)					
E 调			1=E (329.5)				
F 调				1=F (349)			
G 调					1=G (391.5)		
A 调						1=A (440)	
B 调							1=B (494)

MATLAB 练习 2：在 2.1 函数的基础上，编写函数

```
function freq = tone2freq(tone, scale, noctave, rising)
% tone为输入数字音符，scale 为调号，noctave 为高低八度数量，rising为升降调，freq为输出的频率,
```

编写函数，要求能够计算出一张简谱中任何一个音符的频率。

2.3 音符长短

简谱中，音的长短是在基本音符的基础上加短横线和/或附点表示的。

1）短横线的用法有两种：写在基本音符右边的短横线叫增时线。增时线越多，音的时值就越长。

不带增时线的基本音符叫四分音符，每增加一条增时线，表示延长一个四分音符的时间。

写在基本音符下面的短横线叫减时线。减时线越多，音就越短，每增加一条减时线，就表示缩短为原音符音长的一半。

2）写在音符右边的小圆点叫做附点，表示延长前面音符时值的一半。附点往往用于四分音符和少于四分音符的各种音符。带附点的音符叫附点音符。

在这里，我们也需要考虑编码的问题，如何将每个音符对应的持续时间输入，我们可以以 1s 的单个音符持续时长作为基准，加上一条增时线则时间变为 2，加上一条减时线则时间变为 0.5，小圆点同理。

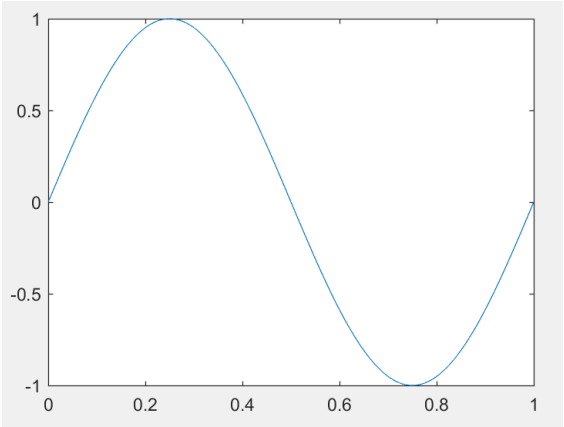
3 生成不同频率波形

在 MATLAB 中，生成特定频率的波形一般用三角函数。

例 1：

```
fs = 8192;  f = 1; T = 1/f;
t = linspace(0, T, fs);
y = sin(2*pi*f*t)
```

以上代码生成了一个频率为 1Hz 的正弦波。



使用 sound 函数播放（用 help 命令了解 sound 函数用法），在例 1 后面添加以下代码：

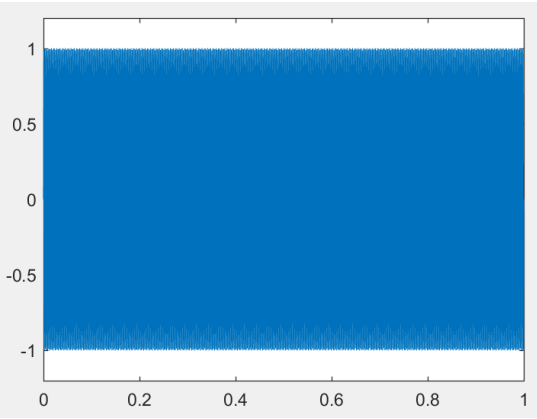
```
sound(y, fs);
```

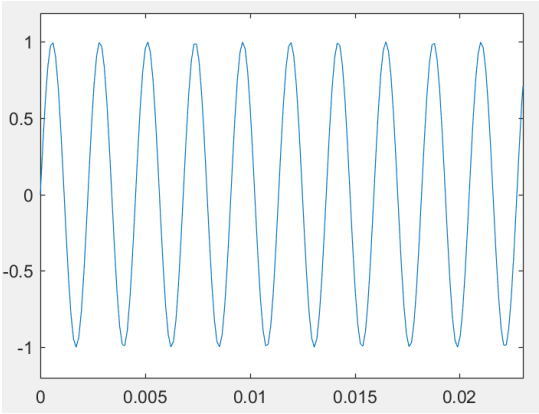
你会发现，什么都听不到，因为，人类的听力频率范围是 20-20k Hz. 上面产生的正弦波频率太低了。将例 1 代码改成：

例 2：

```
fs = 8192; f = 440; T = 1/f;
t = linspace(0, 1, fs);
y = sin(2*pi*f*t);
plot(t,y);
sound(y, fs);
```

你会听到一个持续 1 秒的‘du’的声音。波形如下图所示（右图为放大后）：





如果需要调整持续时长，则要增加 t 的长度，将例 2 第二行改为：

```
rhythm = 5;
t = linspace(0, rhythm, fs * rhythm);
```

使用 sound 函数播放，你会听到一个持续 5 秒的‘du’的声音。2.3 节中介绍的符号决定了音乐中某个音符持续的时长，对应在上面代码中的 **rhythm.**以 4/4 拍（以四分音符为一拍，每小节四拍）为例，每个音符持续时间设为 1/4 秒（可以自己定义），则延半拍（音符右边带小圆点）共持续 1/4+1/8 秒，一条减时线持续 1/8 秒，以此类推。
将数字简谱中的每个音符都转换成1 – 7表示的音级（频率），再结合每个音符持续的时长，在使用上面介绍到的代码，为每个音符生成对应的波形，将每个音符对应的波形连接在一起，你将得到一段简单的音乐，你可以使用sound函数播放，也可以使用audiowrite函数（使用help命令了解其用法）将其写入音乐文件中。

MATLAB 练习 3 编写函数

```
function waves = gen_wave(tone, scale, noctave, rising, rhythm, fs)
% tone为数字音符，scale为调号，noctave为高低八度数量，rising为升降调，rhythm为节拍，即每个音符持续时长，fs为采样频率，
```

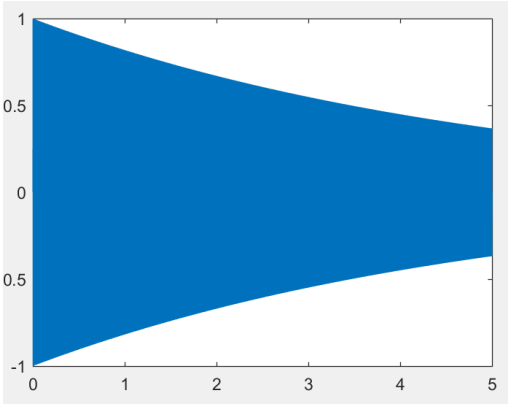
将图 1 中的《天空之城》 数字简谱转换成波形文件。

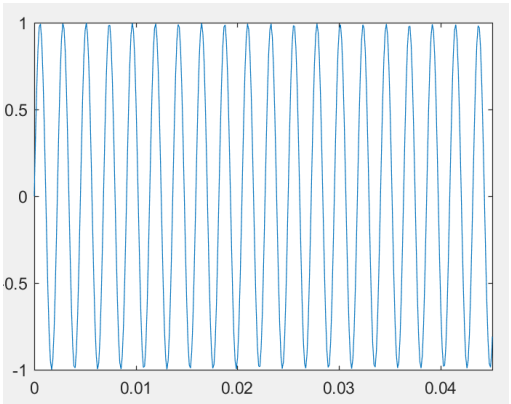
4 音量波动

考虑到乐器演奏时，振动会有衰减，不会以固定幅度持续振动，因此一个包络衰减函数能够更加真实的模拟音乐的产生：

```
waves = y.*exp(-x/rhythm));
```

在例 2 第三行后面增加这样一行代码，对单个音符的波形进行指数衰减，新的波形如下图所示：





当然，也有很多别的衰减函数或许能带来更加真实的听感，如线性衰减，平方衰减等，甚至可以尝试下其他的波动。同时，这种衰减也可以在一个节拍内的几个音符中进行，比如单个音符的波形加一个衰减函数，几个音符的波形连在一起后再加上一个衰减函数。

MATLAB 练习 4： 尝试使用不同的衰减函数对练习 2 中输出结果进行处理，使用 sound 函数播放并比较其听感，使用 plot 函数画出其波形并分析其播放效果，并选择其中你认为最好的。

5 泛音/不同乐器的音色区别

前面我们讲的乐谱中的音调都是指音乐的基频。而用乐器演奏音乐时，除了发出乐谱中的基频声音外，由于乐器的发声原理，还产生数量不等的驻波。驻波是指，当一根琴弦两端被固定时，我们拨动琴弦，琴弦振动部分的长度必然是半波长的整数倍，即，发出的声音频率包含基频以及基频的整数倍谐频。

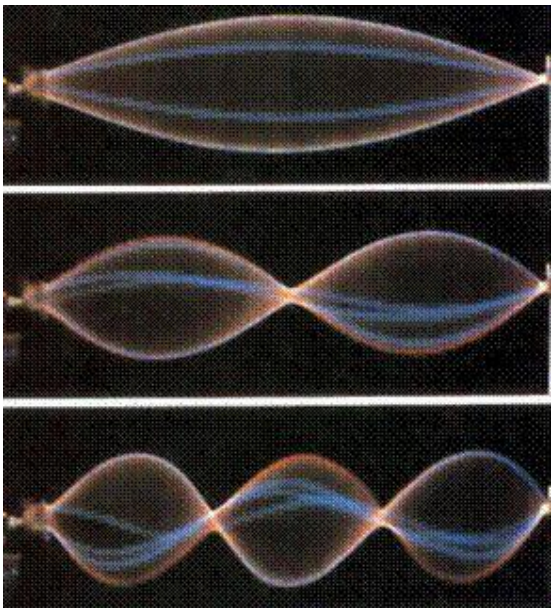
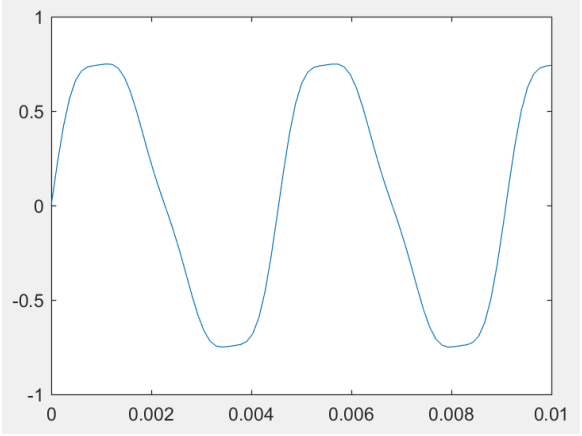


图 5. 驻波原理

所以乐器弹奏时会产生包括基频和若干整数倍频率的谐频，而主要的能量集中于基频。对于 2 倍、3 倍、4 倍、5 倍……的谐频，不同乐器这些谐频的能量比例各不相同。如果我们调整这个比例，将产生音色完全不同的声波：

```
fs = 8192; f = 220; T = 1/f;
rhythm = 1;
t = linspace(0, rhythm, fs * rhythm);
y = 0.8*sin(2*pi*f*t)+0.1*sin(2*pi*2*f*t)+ …
0.05*sin(2*pi*3*f*t)+0.05*sin(2*pi*4*f*t);
%waves = y.*exp(-x/rhythm);
plot(t,y);
axis([0 0.01 -1 1]);
```

如果用上面这段程序来产生某个音符的波形，得到的新的波形将如图所示，使用 sound 函数播放，你会发现，其音色明显与 2 中不同，将所有音符转化成波形（这里的包络衰减可以保留），再将这些音符连起来，试听以下整段音乐的音质，你会发现，跟单纯的基频生成的音乐有明显区别。



MATLAB 练习 5： 在练习 3 的基础上，尝试不同的谐波能量比例，用 plot 分析其波形以及频谱波形，并分析产生的音乐的音色差别。选择你喜欢的音色，将图 1 中的《天空之城》或者你喜欢的其他音乐转换成 wav 格式的音乐。

6 实验要求

1. 完成 MATLAB 练习 1-5，包括代码，结果，观察，分析等。
2. 完成完整的代码（包含音符频率计算（tone2freq），单个音符波形数据生成(gen_wave)，整个简谱波形数据生成(gen_music)共三个函数文件，以及将波形数据保存成音乐文件（建议 wav 格式）的脚本文件），从一份简谱生成一段音乐数据，尝试调整代码中的包络衰减、谐波能量比例等，并分析其对听感的影响，尝试模拟某种乐器的声音，并给出结果分析。
3. 提交文件清单：1）pdf 格式实验报告一份，包含代码，图片（非必须）以及分析过程。2）源代码压缩包，包含上文提到的三个函数和一个脚本文件。简谱数据文件以及最终保存下来的音乐文件。