CHAPTER

8

The Discrete Fourier Transform

Fourier analysis is a family of mathematical techniques, all based on decomposing signals into sinusoids. The discrete Fourier transform (DFT) is the family member used with *digitized* signals. This is the first of four chapters on the **real DFT**, a version of the discrete Fourier transform that uses real numbers to represent the input and output signals. The **complex DFT**, a more advanced technique that uses complex numbers, will be discussed in Chapter 31. In this chapter we look at the mathematics and algorithms of the Fourier decomposition, the heart of the DFT.

The Family of Fourier Transform

Fourier analysis is named after **Jean Baptiste Joseph Fourier** (1768-1830), a French mathematician and physicist. (Fourier is pronounced: for·ē·ā, and is always capitalized). While many contributed to the field, Fourier is honored for his mathematical discoveries and insight into the practical usefulness of the techniques. Fourier was interested in heat propagation, and presented a paper in 1807 to the Institut de France on the use of sinusoids to represent temperature distributions. The paper contained the controversial claim that any continuous periodic signal could be represented as the sum of properly chosen sinusoidal waves. Among the reviewers were two of history's most famous mathematicians, Joseph Louis Lagrange (1736-1813), and Pierre Simon de Laplace (1749-1827).

While Laplace and the other reviewers voted to publish the paper, Lagrange adamantly protested. For nearly 50 years, Lagrange had insisted that such an approach could not be used to represent signals with *corners*, i.e., discontinuous slopes, such as in square waves. The Institut de France bowed to the prestige of Lagrange, and rejected Fourier's work. It was only after Lagrange died that the paper was finally published, some 15 years later. Luckily, Fourier had other things to keep him busy, political activities, expeditions to Egypt with Napoleon, and trying to avoid the guillotine after the French Revolution (literally!).

章

8 离散傅里叶变换

傅里叶分析是一系列数学技术的统称，其核心原理都是将信号分解为正弦波。其中离散傅里叶变换（DFT）是专门处理*数字化*信号的典型代表。作为**实数DFT**的入门章节，本章将系统讲解其数学原理与算法实现。需要特别说明的是，更高级的复数DFT技术将在第31章详细阐述。本章重点解析傅里叶分解的核心数学原理，这正是离散傅里叶变换的精髓所在。

傅里叶变换的家族

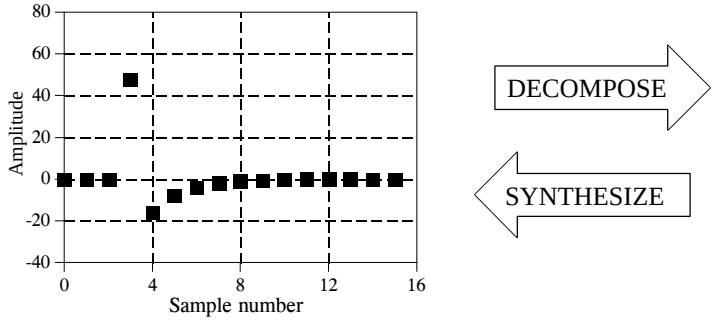
傅里叶分析是以法国数学家和物理学家**JeanBaptisteJosephFourier**（1768-1830）的名字命名的。（Fourier的发音是：for。ē·ā（且始终首字母大写）。尽管众多学者为该领域作出贡献，但傅里叶因其数学发现及对技术实用价值的深刻洞察而备受推崇。傅里叶对热传导现象深感兴趣，1807年向法兰西研究院提交论文，提出用正弦波表示温度分布的创新方法。该论文提出了颇具争议的观点：任何连续周期信号均可通过精心选择的正弦波叠加来表征。参与评审的学者中包括两位数学史上的泰斗——约瑟夫·路易·拉格朗日（1736-1813）与皮埃尔·西蒙·德·拉普拉斯（1749-1827）。

尽管拉普拉斯和其他评审专家投票支持发表论文，拉格朗日却坚决反对。近半个世纪以来，拉格朗日始终坚持认为这种分析方法无法表征具有*拐点*（即斜率不连续）的信号，比如方波信号。法兰西研究院最终屈从于拉格朗日的学术声望，拒绝了傅里叶的研究成果。直到拉格朗日去世约15年后，这篇论文才得以正式发表。值得庆幸的是，傅里叶当时正忙于其他事务：参与政治活动、随拿破仑远征埃及，以及在法国大革命后竭力躲避断头台（字面意思！）。

Who was right? It's a split decision. Lagrange was correct in his assertion that a summation of sinusoids cannot form a signal with a corner. However, you can get *very* close. So close that the difference between the two has *zero energy*. In this sense, Fourier was right, although 18th century science knew little about the concept of energy. This phenomenon now goes by the name: *Gibbs Effect*, and will be discussed in Chapter 11.

Figure 8-1 illustrates how a signal can be decomposed into sine and cosine waves. Figure (a) shows an example signal, 16 points long, running from sample number 0 to 15. Figure (b) shows the Fourier decomposition of this signal, nine cosine waves and nine sine waves, each with a different frequency and amplitude. Although far from obvious, these 18 sinusoids

FIGURE 8-1a
(see facing page)



add to produce the waveform in (a). It should be noted that the objection made by Lagrange only applies to *continuous* signals. For *discrete* signals, this decomposition is mathematically exact. There is no difference between the signal in (a) and the *sum* of the signals in (b), just as there is no difference between 7 and 3+4.

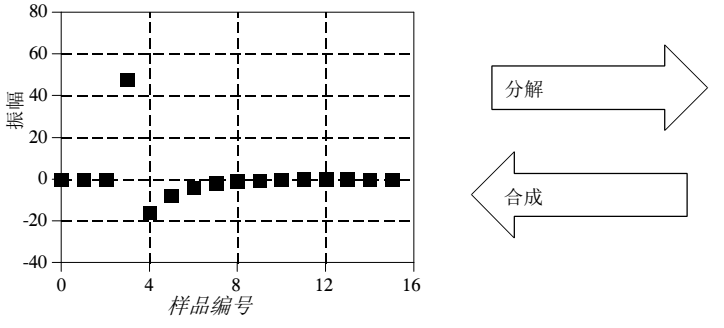
Why are sinusoids used instead of, for instance, square or triangular waves? Remember, there are an infinite number of ways that a signal can be decomposed. The goal of decomposition is to end up with something *easier* to deal with than the original signal. For example, impulse decomposition allows signals to be examined one point at a time, leading to the powerful technique of convolution. The component sine and cosine waves are simpler than the original signal because they have a property that the original signal does not have: *sinusoidal fidelity*. As discussed in Chapter 5, a sinusoidal input to a system is guaranteed to produce a sinusoidal output. Only the amplitude and phase of the signal can change; the frequency and wave shape must remain the same. Sinusoids are the only waveform that have this useful property. While square and triangular decompositions are *possible*, there is no general reason for them to be *useful*.

The general term: *Fourier transform*, can be broken into four categories, resulting from the four basic types of signals that can be encountered.

谁是对的？这个问题存在分歧。拉格朗日的断言是正确的——正弦波叠加无法形成具有拐角的信号。然而，你可以将叠加效果做到*非常*接近。接近到两者之间的差异*能量为零*。从这个意义上说，傅里叶是对的，尽管18世纪的科学界对能量概念知之甚少。这种现象现在被称为*吉布斯效应*，将在第11章中详细讨论。

图8-1展示了信号如何分解为正弦和余弦波。图(a)呈现了一个16个采样点的示例信号，采样点从0号到15号。图(b)显示了该信号的傅里叶分解，包含九个余弦波和九个正弦波，每个波形具有不同的频率和振幅。尽管这一分解过程并不直观，但这18个正弦波

图8-1a（参见对页）



相加得到(a)中的波形。需要注意的是，拉格朗日提出的反对意见只适用于*连续*信号。对于*离散*信号，这种分解在数学上是精确的。(a)中的信号与(b)中信号的*和*之间没有区别，就像7和3+4之间没有区别一样。

为何选择正弦波而非方波或三角波？要知道，信号分解的方式可谓五花八门。分解的终极目标，是让处理对象变得比原始信号更*易于掌控*。以脉冲分解为例，它能逐点解析信号，由此衍生出强大的卷积技术。正弦和余弦波之所以比原始信号更简单，是因为它们具备原始信号所不具备的特性——*正弦保真度*。正如第五章所述，正弦波输入系统时，输出必定是正弦波。信号的振幅和相位可以改变，但频率和波形必须保持不变。正弦波是唯一具备这种实用特性的波形。虽然方波和三角波的分解*也是可行的*，但它们缺乏通用性，无法满足实际应用需求。

通用术语*傅里叶变换*可依据信号的四种基本类型划分为四类。

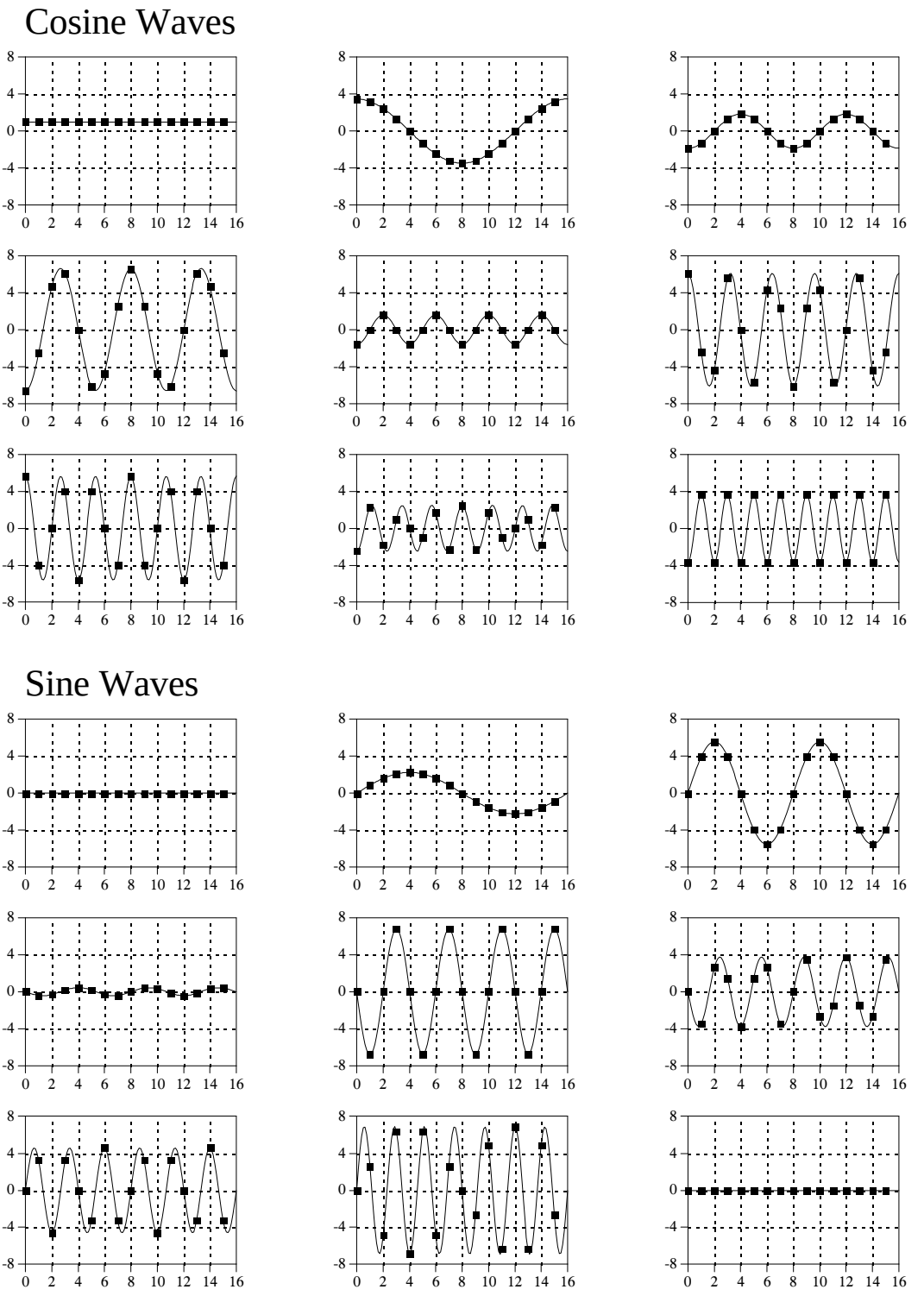
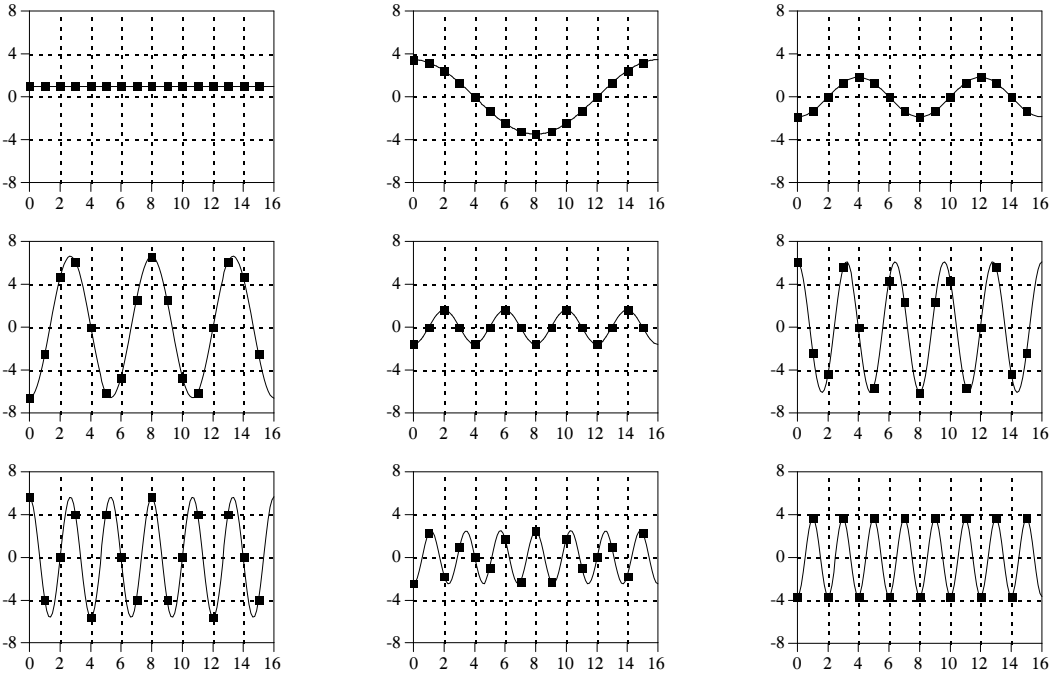


FIGURE 8-1b
Example of Fourier decomposition. A 16 point signal (opposite page) is decomposed into 9 cosine waves and 9 sine waves. The frequency of each sinusoid is fixed; only the amplitude is changed depending on the shape of the waveform being decomposed.

余弦波



正弦波

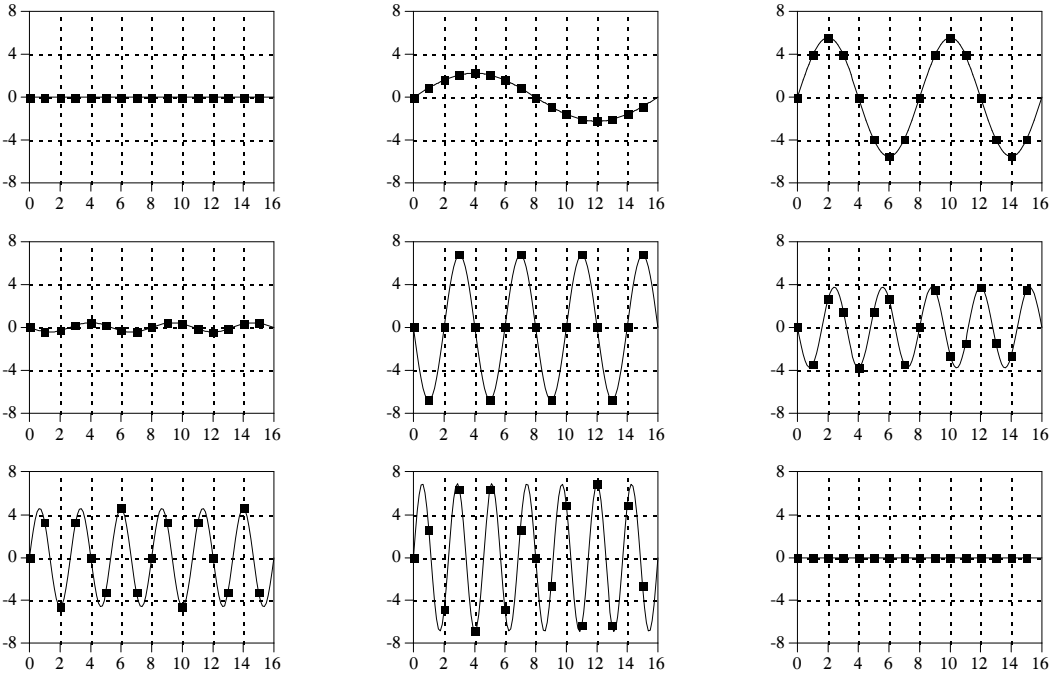


图8-1b

傅里叶分解示例。一个16点信号（见对页）被分解为9个余弦波和9个正弦波。每个正弦波的频率固定，仅根据待分解波形的形状改变其振幅。

A signal can be either *continuous* or *discrete*, and it can be either *periodic* or *aperiodic*. The combination of these two features generates the four categories, described below and illustrated in Fig. 8-2.

Aperiodic-Continuous
This includes, for example, decaying exponentials and the Gaussian curve. These signals extend to both positive and negative infinity *without* repeating in a periodic pattern. The Fourier Transform for this type of signal is simply called the **Fourier Transform**.

Periodic-Continuous
Here the examples include: sine waves, square waves, and any waveform that repeats itself in a regular pattern from negative to positive infinity. This version of the Fourier transform is called the **Fourier Series**.

Aperiodic-Discrete
These signals are only defined at discrete points between positive and negative infinity, and do not repeat themselves in a periodic fashion. This type of Fourier transform is called the **Discrete Time Fourier Transform.**

Periodic-Discrete
These are discrete signals that repeat themselves in a periodic fashion from negative to positive infinity. This class of Fourier Transform is sometimes called the Discrete Fourier Series, but is most often called the **Discrete Fourier Transform.**

You might be thinking that the names given to these four types of Fourier transforms are confusing and poorly organized. You're right; the names have evolved rather haphazardly over 200 years. There is nothing you can do but memorize them and move on.

These four classes of signals all extend to positive and negative *infinity*. Hold on, you say! What if you only have a finite number of samples stored in your computer, say a signal formed from 1024 points. Isn't there a version of the Fourier Transform that uses finite length signals? No, there isn't. Sine and cosine waves are *defined* as extending from negative infinity to positive infinity. You cannot use a group of infinitely long signals to synthesize something finite in length. The way around this dilemma is to make the finite data *look like* an infinite length signal. This is done by imagining that the signal has an infinite number of samples on the left and right of the actual points. If all these "imagined" samples have a value of zero, the signal looks *discrete* and *aperiodic*, and the Discrete Time Fourier Transform applies. As an alternative, the imagined samples can be a duplication of the actual 1024 points. In this case, the signal looks discrete and periodic, with a period of 1024 samples. This calls for the Discrete Fourier Transform to be used.

As it turns out, an *infinite* number of sinusoids are required to synthesize a signal that is *aperiodic*. This makes it impossible to calculate the Discrete Time Fourier Transform in a computer algorithm. By elimination, the only

信号可以是*连续的*或*离散的*，也可以是*周期的*或*非周期的*。这两种特征的组合产生了以下描述的四类信号，并在图8-2中进行了说明。

非周期连续的

这包括衰减指数和高斯曲线等。这些信号延伸到正负无穷大*而不*重复周期性模式。这种信号的傅里叶变换简单地称为**傅里叶变换**。

周期-连续

这里的例子包括：正弦波、方波和任何在负到正无穷大之间以规则模式重复的波形。这个版本的傅里叶变换被称为**傅里叶级数**。

非周期离散

这些信号只在正负无穷大之间的离散点上定义，且不以周期方式重复。这种傅里叶变换称为**离散时间傅里叶变换。**

周期离散的

这些是离散信号，它们以周期性的方式从负无穷到正无穷重复。这类傅里叶变换有时被称为离散傅里叶级数，但最常被称为**离散傅里叶变换。**

你或许认为这四种傅里叶变换的命名方式令人困惑且缺乏系统性。确实如此，这些名称在200年间的发展过程颇为随意。除了死记硬背外别无他法。

这四类信号都延伸至正负*无穷大*。等等，你可能会问！如果计算机中存储的样本数量有限，比如由1024个点组成的信号呢？难道不存在使用有限长度信号的傅里叶变换版本吗？答案是否定的。正弦和余弦波*的定义*是从负无穷延伸到正无穷。你无法用无限长的信号组来合成有限长度的信号。解决这个困境的方法是让有限数据*看起来像*无限长的信号。具体做法是想象信号在实际点的左右两侧有无限个样本。如果所有这些“想象”样本的值都为零，信号就会呈现*离散*且*非周期*的特性，此时就可以应用离散时间傅里叶变换。另一种方法是将想象样本复制为实际的1024个点。这样信号就呈现离散且周期性，周期为1024个样本，这时就需要使用离散傅里叶变换。

事实证明，合成一个*非周期性*的信号需要*无限*个正弦波。这使得在计算机算法中无法计算离散时间傅里叶变换。通过排除法，唯一

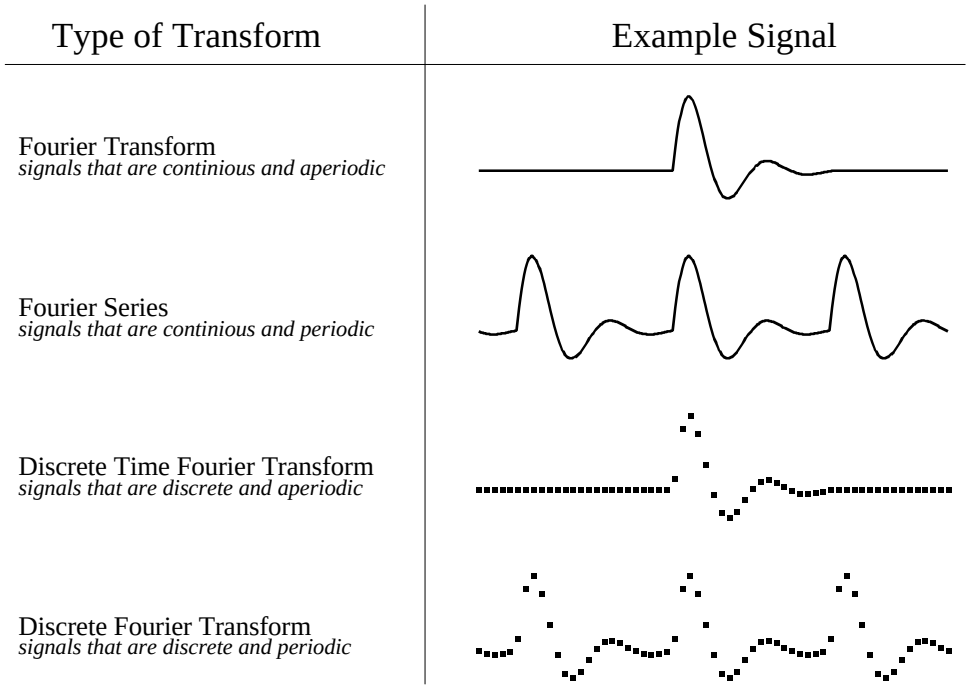


FIGURE 8-2
Illustration of the four Fourier transforms. A signal may be continuous or discrete, and it may be periodic or aperiodic. Together these define four possible combinations, each having its own version of the Fourier transform. The names are not well organized; simply memorize them.

type of Fourier transform that can be used in DSP is the DFT. In other words, digital computers can only work with information that is *discrete* and *finite* in length. When you struggle with theoretical issues, grapple with homework problems, and ponder mathematical mysteries, you may find yourself using the first three members of the Fourier transform family. When you sit down to your computer, you will only use the DFT. We will briefly look at these other Fourier transforms in future chapters. For now, concentrate on understanding the Discrete Fourier Transform.

Look back at the example DFT decomposition in Fig. 8-1. On the face of it, it appears to be a 16 point signal being decomposed into 18 sinusoids, each consisting of 16 points. In more formal terms, the 16 point signal, shown in (a), must be viewed as a single period of an infinitely long periodic signal. Likewise, each of the 18 sinusoids, shown in (b), represents a 16 point segment from an infinitely long sinusoid. Does it really matter if we view this as a 16 point signal being synthesized from 16 point sinusoids, or as an infinitely long periodic signal being synthesized from infinitely long sinusoids? The answer is: *usually no, but sometimes, yes*. In upcoming chapters we will encounter properties of the DFT that seem baffling if the signals are viewed as finite, but become obvious when the periodic nature is considered. The key point to understand is that this periodicity is invoked in order to use a *mathematical tool*, i.e., the DFT. It is usually meaningless in terms of where the signal originated or how it was acquired.

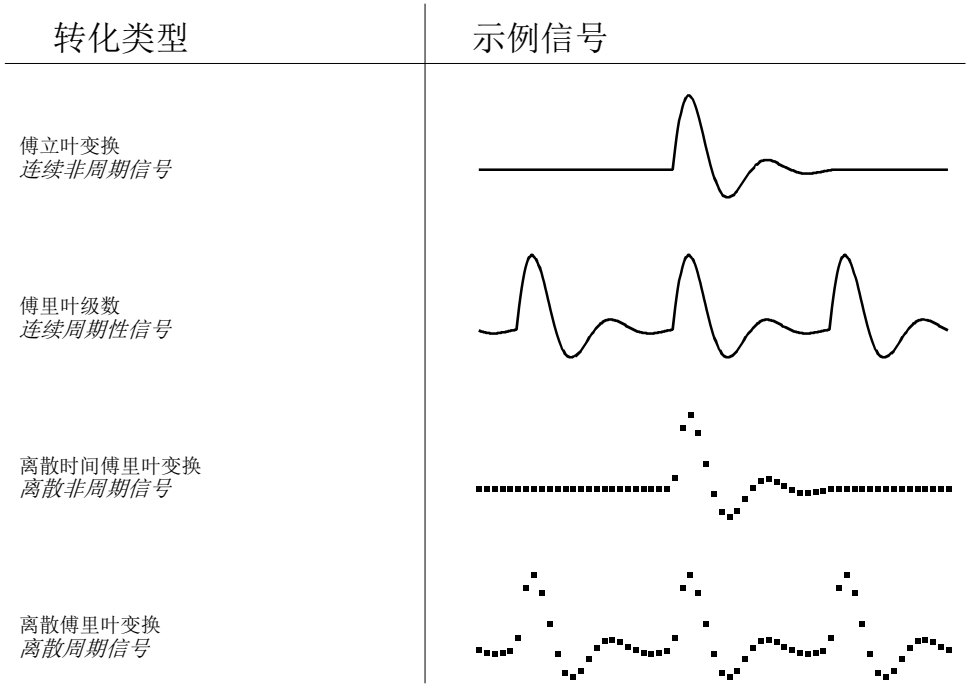


图8-2
图示四种傅里叶变换。信号可以是连续或离散的，也可以是周期性或非周期性的。这些定义了四种可能的组合，每种组合都有其对应的傅里叶变换版本。名称排列并不严谨，只需记忆即可。

在数字信号处理领域，最常用的傅里叶变换当属离散傅里叶变换（DFT）。说白了，数字计算机只能处理*离散*且长度有限的数据。当你钻研理论问题、攻克作业难题、探索数学奥秘时，很可能需要用到傅里叶变换家族的前三个成员。而当你对着电脑工作时，用到的永远是DFT。至于其他傅里叶变换，咱们在后续章节会详细讲解。现在咱们先来好好掌握离散傅里叶变换，这可是数字信号处理的入门必修课！

回顾图8-1中的DFT分解示例。乍看之下，这像是将一个16点信号分解为18个正弦波，每个正弦波由16个点组成。更正式地说，图(a)所示的16点信号必须被视为无限长周期信号的一个完整周期。同样，图(b)中显示的18个正弦波，每个都代表无限长正弦波中的16点片段。那么问题来了：我们是将这个16点信号视为由16个点的正弦波合成，还是将其视为由无限长正弦波合成的无限长周期信号？答案是：通常不是，但有时是。在后续章节中，我们将遇到DFT的一些特性——当信号被视为有限时这些特性令人费解，但一旦考虑其周期性特征就会变得显而易见。关键在于理解：这种周期性是为使用*数学工具*（即DFT）而引入的。通常而言，这些特性与信号的起源或获取方式并无直接关联。

Each of the four Fourier Transforms can be subdivided into **real** and **complex** versions. The real version is the simplest, using ordinary numbers and algebra for the synthesis and decomposition. For instance, Fig. 8-1 is an example of the **real DFT**. The complex versions of the four Fourier transforms are immensely more complicated, requiring the use of *complex numbers*. These are numbers such as: $3+4j$, where j is equal to $\sqrt{-1}$ (electrical engineers use the variable j, while mathematicians use the variable, i). Complex mathematics can quickly become overwhelming, even to those that specialize in DSP. In fact, a primary goal of this book is to present the fundamentals of DSP *without* the use of complex math, allowing the material to be understood by a wider range of scientists and engineers. The complex Fourier transforms are the realm of those that specialize in DSP, and are willing to sink to their necks in the swamp of mathematics. If you are so inclined, Chapters 30-33 will take you there.

The mathematical term: **transform**, is extensively used in Digital Signal Processing, such as: Fourier transform, Laplace transform, Z transform, Hilbert transform, Discrete Cosine transform, etc. Just what is a transform? To answer this question, remember what a *function* is. A function is an algorithm or procedure that changes one value into another value. For example, $y = 2x + 1$ is a function. You pick some value for x, plug it into the equation, and out pops a value for y. Functions can also change *several* values into a single value, such as: $y = 2a + 3b + 4c$, where a, b, and c are changed into y.

Transforms are a direct extension of this, allowing both the input and output to have *multiple* values. Suppose you have a signal composed of 100 samples. If you devise some equation, algorithm, or procedure for changing these 100 samples into another 100 samples, you have yourself a transform. If you think it is useful enough, you have the perfect right to attach your last name to it and expound its merits to your colleagues. (This works best if you are an eminent 18th century French mathematician). Transforms are not limited to any specific type or number of data. For example, you might have 100 samples of discrete data for the input and 200 samples of discrete data for the output. Likewise, you might have a continuous signal for the input and a continuous signal for the output. Mixed signals are also allowed, discrete in and continuous out, and vice versa. In short, a transform is any fixed procedure that changes one chunk of data into another chunk of data. Let's see how this applies to the topic at hand: the Discrete Fourier transform.

Notation and Format of the Real DFT

As shown in Fig. 8-3, the discrete Fourier transform changes an N point input signal into two $N/2+1$ point output signals. The input signal contains the signal being decomposed, while the two output signals contain the *amplitudes* of the component sine and cosine waves (scaled in a way we will discuss shortly). The input signal is said to be in the **time domain**. This is because the most common type of signal entering the DFT is composed of

四个傅里叶变换中的每一个都可以细分为**实数**和**复数**版本。实数版本是最简单的，使用普通数字和代数进行合成与分解。例如，图8-1是**实数DFT**的一个例子。这四个傅里叶变换的复数版本要复杂得多，需要使用*复数*。这些是诸如：3+4j之类的数，其中*j*等于$\sqrt{-1}$（电气工程师使用变量*j*，而数学家则使用变量*i*）。复数数学即使对数字信号处理（DSP）专家来说也可能让人望而生畏。事实上，本书的核心目标就是用*不涉及*复数数学的方式讲解DSP基础，让更广泛的科研人员和工程师都能轻松理解。复傅里叶变换属于DSP专家的专属领域，需要他们甘愿在数学泥潭中深陷其中。如果你有此意愿，第30-33章将带你深入这个专业领域。

数学术语**变换**在数字信号处理中被广泛应用。
处理，例如：傅里叶变换、拉普拉斯变换、Z变换、希尔伯特变换、离散余弦变换等。什么是变换？要回答这个问题，先记住*函数*是什么。函数是一种将一个值转换为另一个值的算法或过程。例如，*y*=2x+1就是一个函数。你选择某个*x*值代入方程，就会得到*y*的值。函数也可以将*多个*值转换为一个值，例如：*y*=2a+3b+4c，其中*a*、*b*和*c*被转换为*y*。

变换是这一概念的直接延伸，它允许输入和输出都具有*多个*数值。假设你有一个由100个样本组成的信号。如果你设计出某种方程、算法或程序，将这100个样本转换为另外100个样本，这就构成了一个变换。如果你认为它足够实用，你完全有权给它冠上自己的姓氏，并向同事阐述它的优点。（这在你是一位18世纪法国著名数学家时效果最佳）。变换并不局限于任何特定类型或数量的数据。例如，你可能有100个离散数据样本作为输入，200个离散数据样本作为输出。同样，你可能有连续信号作为输入，连续信号作为输出。混合信号也是允许的，即输入离散而输出连续，反之亦然。简而言之，变换是任何将一组数据转换为另一组数据的固定程序。让我们看看这如何应用于当前主题：离散傅里叶变换。

实DFT的记号与格式

如图8-3所示，离散傅里叶变换将一个*N*点输入信号转换为两个*N*/2+1点输出信号。输入信号包含待分解的信号，而两个输出信号则包含分量正弦和余弦波的*振幅*（其缩放方式我们稍后将讨论）。输入信号被称为处于**时域**。这是因为输入DFT的最常见信号类型由

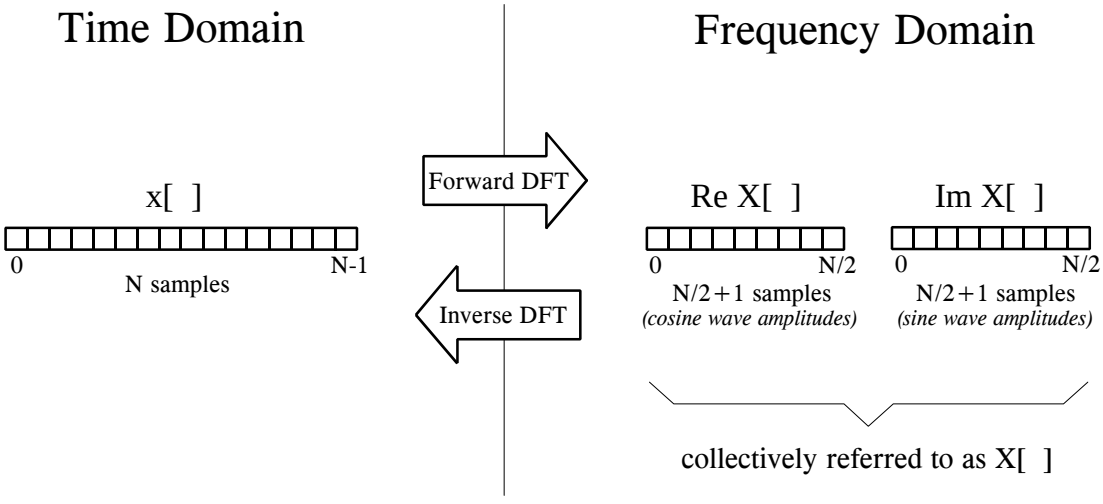


FIGURE 8-3
DFT terminology. In the time domain, $x[\ ]$ consists of N points running from 0 to $N-1$. In the frequency domain, the DFT produces two signals, the real part, written: $ReX[\ ]$, and the imaginary part, written: $ImX[\ ]$. Each of these frequency domain signals are $N/2+1$ points long, and run from 0 to $N/2$. The Forward DFT transforms from the time domain to the frequency domain, while the Inverse DFT transforms from the frequency domain to the time domain. (Take note: this figure describes the **real DFT**. The **complex DFT**, discussed in Chapter 31, changes N complex points into another set of N complex points).

samples taken at regular intervals of *time*. Of course, any kind of sampled data can be fed into the DFT, regardless of how it was acquired. When you see the term "time domain" in Fourier analysis, it may actually refer to samples taken over time, or it might be a general reference to any discrete signal that is being decomposed. The term **frequency domain** is used to describe the amplitudes of the sine and cosine waves (including the special scaling we promised to explain).

The frequency domain contains exactly the same information as the time domain, just in a different form. If you know one domain, you can calculate the other. Given the time domain signal, the process of calculating the frequency domain is called **decomposition**, **analysis,** the **forward DFT**, or simply, **the DFT**. If you know the frequency domain, calculation of the time domain is called **synthesis**, or the **inverse DFT**. Both synthesis and analysis can be represented in equation form and computer algorithms.

The number of samples in the time domain is usually represented by the **variable *N***. While N can be any positive integer, a power of two is usually chosen, i.e., 128, 256, 512, 1024, etc. There are two reasons for this. First, digital data storage uses binary addressing, making powers of two a natural signal length. Second, the most efficient algorithm for calculating the DFT, the Fast Fourier Transform (FFT), usually operates with N that is a power of two. Typically, N is selected between 32 and 4096. In most cases, the samples run from 0 to $N-1$, rather than 1 to N.

Standard DSP notation uses **lower case letters** to represent time domain signals, such as $x[\ ]$, $y[\ ]$, and $z[\ ]$. The corresponding **upper case letters** are

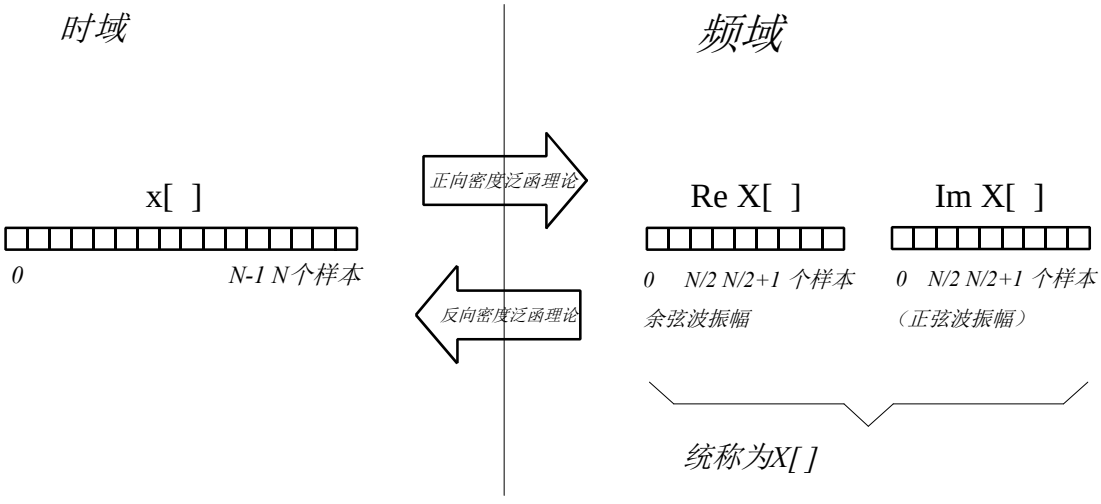


图8-3

DFT术语。在时域中，*x*[ ] 包含从0到*N*-1的N个点。在频域中，DFT生成两个信号：实部记为*ReX*[ ]，虚部记为*Im X*[ ]。这两个频域信号的长度均为*N*/2+1个点，范围从0到*N*/2。前向DFT将时域转换为频域，而逆DFT将频域转换为时域。（注意：此图描述的是**实DFT**。第31章讨论的**复DFT**将*N*个复点转换为另一组N个复点）。

*时间*的等间隔采样点。需要说明的是，任何采样数据都可以输入DFT（离散傅里叶变换），无论其采集方式如何。在傅里叶分析中出现的“时域”一词，既可能指代随时间采集的样本，也可能泛指任何被分解的离散信号。而**频域**则用于描述正弦和余弦波的振幅（包括我们承诺要解释的特殊缩放关系）。

频域所包含的信息与时域完全相同，只是形式不同。如果你知道一个域，就可以计算出另一个域。给定时域信号，计算频域的过程称为**分解**、**分析**、**正向DFT**，或简称为**DFT**。如果你知道频域，计算时域的过程称为**合成**，或**逆DFT**。合成与分析均可通过方程形式和计算机算法来表示。

时域中的样本数量通常用**变量*N***表示。虽然*N*可以是任意正整数，但通常选择2的幂次，即128、256、512、1024等。这有两个原因：首先，数字数据存储采用二进制寻址，使得2的幂次成为自然的信号长度；其次，计算DFT最高效的算法——快速傅里叶变换（FFT）通常使用*N*为2的幂次。通常，*N*的取值范围在32到4096之间。在大多数情况下，样本从0到*N*-1，而非从1到*N*。

标准DSP符号表示使用**小写字母**来表示时域信号，例如*x*[ ]、*y*[ ]和*z*[ ]。相应的**大写字母**是

used to represent their frequency domains, that is, $X[\ ]$, $Y[\ ]$, and $Z[\ ]$. For illustration, assume an N point time domain signal is contained in $x[\ ]$. The frequency domain of this signal is called $X[\ ]$, and consists of two parts, each an array of $N/2+1$ samples. These are called the **Real part of $X[\ ]$**, written as: $Re\,X[\ ]$, and the **Imaginary part of $X[\ ]$**, written as: $Im\,X[\ ]$. The values in $Re\,X[\ ]$ are the amplitudes of the cosine waves, while the values in $Im\,X[\ ]$ are the amplitudes of the sine waves (not worrying about the scaling factors for the moment). Just as the time domain runs from $x[0]$ to $x[N-1]$, the frequency domain signals run from $Re\,X[0]$ to $Re\,X[N/2]$, and from $Im\,X[0]$ to $Im\,X[N/2]$. Study these notations carefully; they are critical to understanding the equations in DSP. Unfortunately, some computer languages don't distinguish between lower and upper case, making the variable names up to the individual programmer. The programs in this book use the array XX[ ] to hold the time domain signal, and the arrays REX[ ] and IMX[ ] to hold the frequency domain signals.

The names *real part* and *imaginary part* originate from the complex DFT, where they are used to distinguish between *real* and *imaginary* numbers. Nothing so complicated is required for the real DFT. Until you get to Chapter 31, simply think that "real part" means the *cosine wave amplitudes*, while "imaginary part" means the *sine wave amplitudes*. Don't let these suggestive names mislead you; everything here uses ordinary numbers.

Likewise, don't be misled by the *lengths* of the frequency domain signals. It is common in the DSP literature to see statements such as: "The DFT changes an N point time domain signal into an N point frequency domain signal." This is referring to the *complex DFT*, where each "point" is a complex number (consisting of real and imaginary parts). For now, focus on learning the real DFT, the difficult math will come soon enough.

The Frequency Domain's Independent Variable

Figure 8-4 shows an example DFT with $N = 128$. The time domain signal is contained in the array: $x[0]$ to $x[127]$. The frequency domain signals are contained in the two arrays: $Re\,X[0]$ to $Re\,X[64]$, and $Im\,X[0]$ to $Im\,X[64]$. Notice that 128 points in the time domain corresponds to 65 points in each of the frequency domain signals, with the frequency indexes running from 0 to 64. That is, N points in the time domain corresponds to $N/2+1$ points in the frequency domain (not $N/2$ points). Forgetting about this extra point is a common bug in DFT programs.

The horizontal axis of the frequency domain can be referred to in **four different ways**, all of which are common in DSP. In the first method, the horizontal axis is labeled from 0 to 64, corresponding to the 0 to $N/2$ samples in the arrays. When this labeling is used, the index for the frequency domain is an integer, for example, $Re\,X[k]$ and $Im\,X[k]$, where k runs from 0 to $N/2$ in steps of one. Programmers like this method because it is how they write code, using an index to access array locations. This notation is used in Fig. 8-4b.

用于表示它们的频域，即*X*[ ]、*Y*[ ]和*Z*[ ]。举例来说，假设一个包含*N*个点的时域信号被包含在*x*[ ]中。该信号的频域称为*X*[ ]，由两部分组成，每部分都是*N*/2+1个样本的数组。这些被称为**实部*X*[]**，记作：***ReX*[]**，以及**虚部*X*[]**，记作：***ImX*[]**。*ReX*[ ]中的值是余弦波的振幅，而*ImX*[ ]中的值是正弦波的振幅（暂且不考虑缩放因子）。正如时域从*x*[0]延伸到*x*[*N*&1]，频域信号则从*ReX*[0]延伸到*ReX*[*N*/2]，以及从*Im X*[0]延伸到*ImX*[*N*/2]。请仔细研究这些符号；它们对于理解DSP中的方程至关重要。遗憾的是，某些计算机语言不区分大小写，使得变量名称由程序员自行决定。本书中的程序使用数组XX[ ]来存储时域信号，而数组REX[ ]和 IMX [ ]则用于存储频域信号。

名称*实部*和*虚部*源自复数DFT，用于区分*实数*与*虚数*。实数DFT无需如此复杂的概念。直到你读到第
31，简单地认为“实部”指的是*余弦波的振幅*，而“虚部”指的是*正弦波的振幅*。不要让这些暗示性的名字误导你；这里使用的是普通的数字。

同样，切勿被频域信号的*长度*所迷惑。在数字信号处理文献中，常见这样的表述：“离散傅里叶变换将*N*点时域信号转换为*N*点频域信号。”这里指的是*复数离散傅里叶变换*，其中每个“点”都是由实部和虚部组成的复数。目前重点应放在学习实数离散傅里叶变换，复杂的数学推导很快就会展开。

频域的自变量

图8-4展示了一个*N*=128的DFT示例。时域信号包含在数组中：*x*[0]至*x*[127]。频域信号为
包含在两个数组中：*ReX*[0] 到*ReX*[64]，以及*Im X*[0] 到*ImX*[64]。注意时域中的128个点对应频域信号中每个的65个点，频率索引从0到64。也就是说，时域中的*N*个点对应频域中的*N*/2+1个点（而非*N*/2个点）。忽略这个额外点是DFT程序中常见的错误。

频域的横轴可以**以四种不同方式**引用，这些方式在DSP中都很常见。第一种方法中，横轴标记为0到64，对应数组中的0到*N*/2个样本。当使用这种标记时，频域的索引是一个整数，例如*ReX*[*k*] 和*Im X*[*k*]，其中*k*以1为步长从0到*N*/2运行。程序员喜欢这种方法，因为他们使用索引来访问数组位置，这正是他们编写代码的方式。这种表示法在图8-4b中使用。

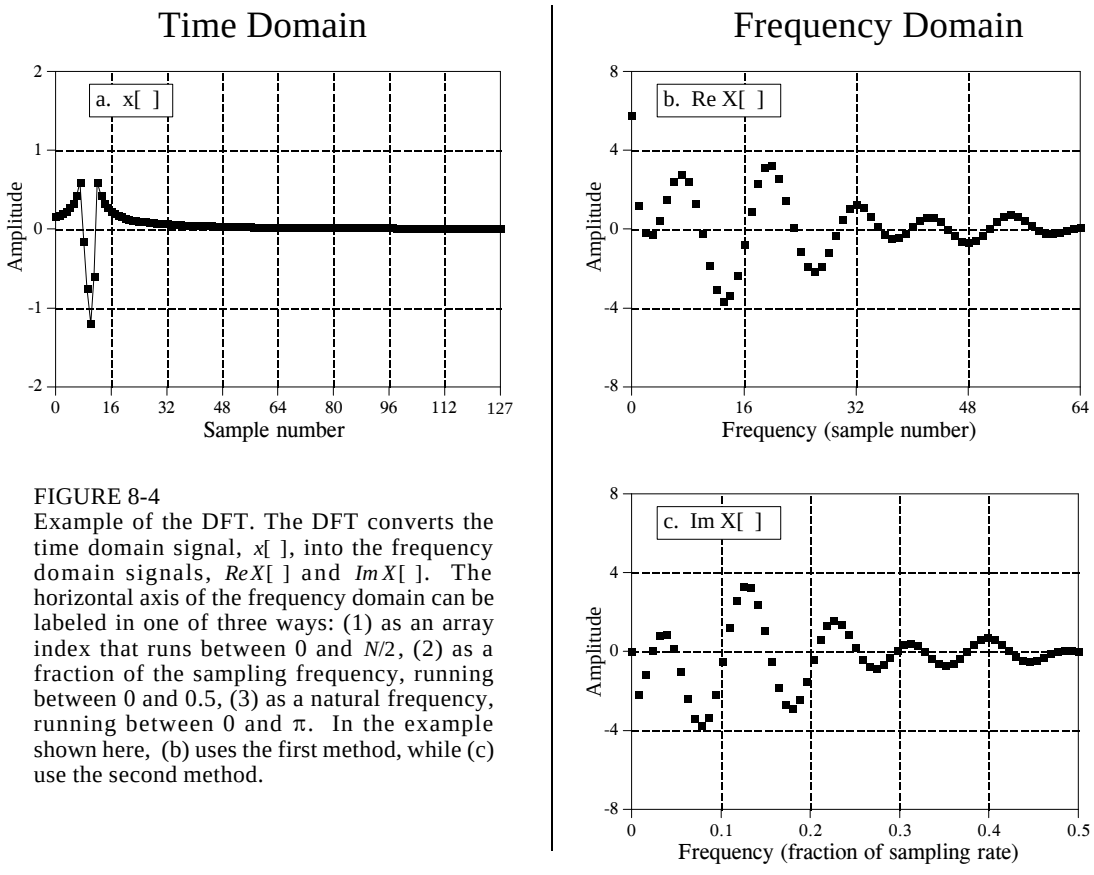


FIGURE 8-4
Example of the DFT. The DFT converts the time domain signal, $x[\ ]$, into the frequency domain signals, $ReX[\ ]$ and $ImX[\ ]$. The horizontal axis of the frequency domain can be labeled in one of three ways: (1) as an array index that runs between 0 and $N/2$, (2) as a fraction of the sampling frequency, running between 0 and 0.5, (3) as a natural frequency, running between 0 and π. In the example shown here, (b) uses the first method, while (c) use the second method.

In the second method, used in (c), the horizontal axis is labeled as a *fraction of the sampling rate*. This means that the values along the horizonal axis always run between 0 and 0.5, since discrete data can only contain frequencies between DC and one-half the sampling rate. The index used with this notation is f, for frequency. The real and imaginary parts are written: $ReX[f]$ and $ImX[f]$, where f takes on $N/2+1$ equally spaced values between 0 and 0.5. To convert from the first notation, k, to the second notation, f, divide the horizontal axis by N. That is, $f = k/N$. Most of the graphs in this book use this second method, reinforcing that discrete signals only contain frequencies between 0 and 0.5 of the sampling rate.

The third style is similar to the second, except the horizontal axis is multiplied by 2π. The index used with this labeling is ω, a lower case Greek *omega*. In this notation, the real and imaginary parts are written: $ReX[\omega]$ and $ImX[\omega]$, where ω takes on $N/2+1$ equally spaced values between 0 and π. The parameter, ω, is called the **natural frequency**, and has the units of **radians**. This is based on the idea that there are 2π radians in a circle. Mathematicians like this method because it makes the equations shorter. For instance, consider how a cosine wave is written in each of these first three notations: using k: $c[n] = \cos(2\pi kn/N)$, using f: $c[n] = \cos(2\pi fn)$, and using ω: $c[n] = \cos(\omega n)$.

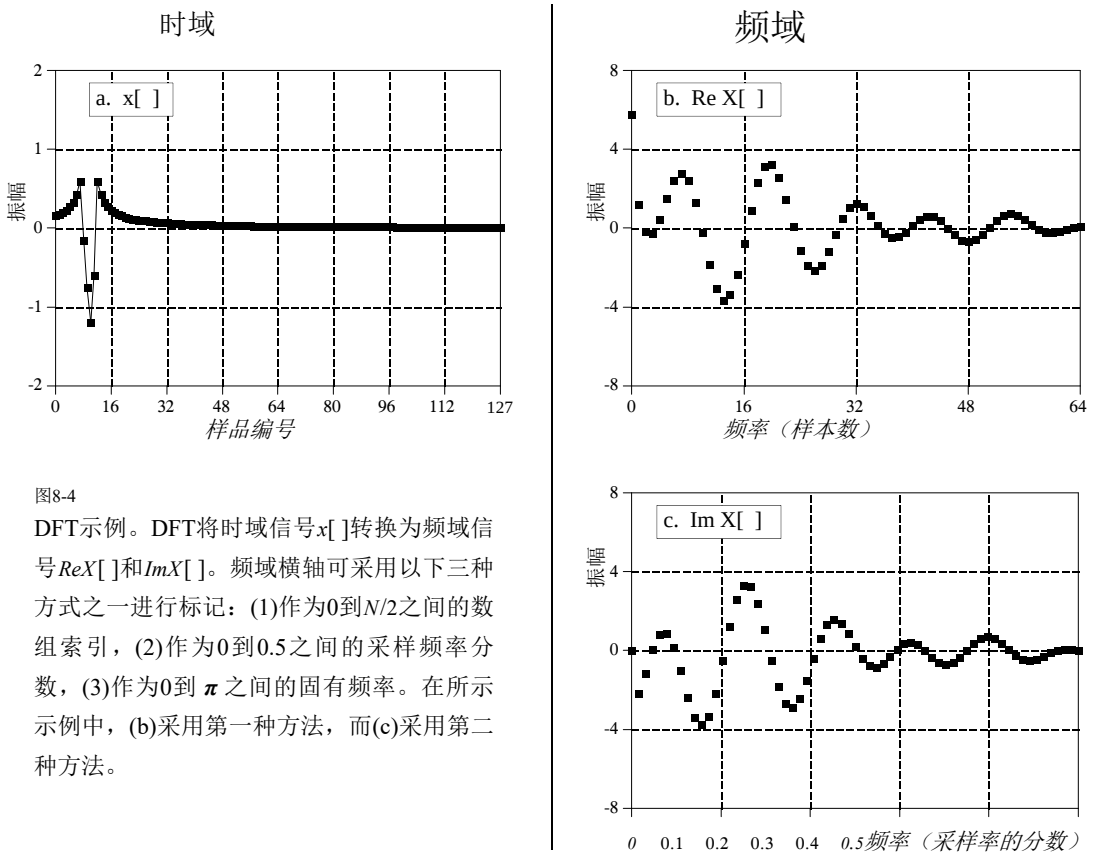


图8-4
DFT示例。DFT将时域信号$x[\ ]$转换为频域信号$ReX[\ ]$和$ImX[\ ]$。频域横轴可采用以下三种方式之一进行标记：(1)作为0到$N/2$之间的数组索引，(2)作为0到0.5之间的采样频率分数，(3)作为0到π之间的固有频率。在所示示例中，(b)采用第一种方法，而(c)采用第二种方法。

在第二种方法（用于(c)中）中，横轴被标记为*采样率的分数*。这意味着横轴上的数值始终在0到0.5之间，因为离散数据只能包含从直流到采样率一半之间的频率。这种记法使用的索引是f，表示频率。实部和虚部分别写作：$ReX[f]$和$Im\,X[f]$，其中f在0到0.5之间取$N/2+1$个等距值。要将第一种记法k转换为第二种记法f，需将横轴除以N，即$f=k/N$。本书中大多数图表采用第二种方法，这强化了离散信号仅包含采样率0到0.5之间频率的结论。

第三种风格与第二种类似，只是横坐标轴被放大了2倍π。这种标记使用的索引是T，即小写的希腊字母ω。在此记法中，实部和虚部分别表示为：$ReX[T]$和$Im\,X[T]$，其中T在0到π之间取$N/2+1$个等间距值。参数T被称为**固有频率**，其单位为**弧度**。这一定义基于圆周包含2π弧度的原理。数学家们青睐这种方法，因为它能简化方程。例如，考虑余弦波在以下三种记法中的表示：使用k：$c[n]=\cos(2\pi\, kn/N)$，使用f：$c[n]=\cos(2\pi\, fn)$，以及使用T：$c[n]=\cos(Tn)$。

The fourth method is to label the horizontal axis in terms of the analog frequencies used in a *particular* application. For instance, if the system being examined has a sampling rate of 10 kHz (i.e., 10,000 samples per second), graphs of the frequency domain would run from 0 to 5 kHz. This method has the advantage of presenting the frequency data in terms of a *real world* meaning. The disadvantage is that it is tied to a particular sampling rate, and is therefore not applicable to general DSP algorithm development, such as designing digital filters.

All of these four notations are used in DSP, and you need to become comfortable with converting between them. This includes both graphs and mathematical equations. To find which notation is being used, look at the independent variable and its range of values. You should find one of four notations: k (or some other integer index), running from 0 to $N/2$; f, running from 0 to 0.5; ω, running from 0 to π; or a frequency expressed in hertz, running from DC to one-half of an actual sampling rate.

DFT Basis Functions

The sine and cosine waves used in the DFT are commonly called the DFT **basis functions**. In other words, the output of the DFT is a set of numbers that represent amplitudes. The basis functions are a set of sine and cosine waves with *unity* amplitude. If you assign each amplitude (the frequency domain) to the proper sine or cosine wave (the basis functions), the result is a set of *scaled* sine and cosine waves that can be added to form the time domain signal.

The DFT basis functions are generated from the equations:

EQUATION 8-1
Equations for the DFT basis functions. In these equations, $c_k[i]$ and $s_k[i]$ are the cosine and sine waves, each N points in length, running from $i = 0$ to $N-1$. The parameter, k, determines the frequency of the wave. In an N point DFT, k takes on values between 0 and $N/2$.

$$c_k[i] = \cos(2\pi ki/N)$$

$$s_k[i] = \sin(2\pi ki/N)$$

where: $c_k[\ ]$ is the cosine wave for the amplitude held in $ReX[k]$, and $s_k[\ ]$ is the sine wave for the amplitude held in $ImX[k]$. For example, Fig. 8-5 shows some of the 17 sine and 17 cosine waves used in an $N = 32$ point DFT. Since these sinusoids add to form the input signal, they must be the same *length* as the input signal. In this case, each has 32 points running from $i = 0$ to 31. The parameter, k, sets the frequency of each sinusoid. In particular, $c_1[\ ]$ is the cosine wave that makes *one* complete cycle in N points, $c_5[\ ]$ is the cosine wave that makes *five* complete cycles in N points, etc. This is an important concept in understanding the basis functions; the frequency parameter, k, is equal to the number of complete cycles that occur over the N points of the signal.

第四种方法是根据*具体应用*中使用的模拟频率来标注横坐标轴。例如，若被测系统的采样率为10千赫兹（即每秒10,000个采样点），频域图的横坐标范围将从0到5千赫兹。这种方法的优势在于能以*实际应用*的意义呈现频率数据。但其缺点是受限于特定采样率，因此不适用于通用数字信号处理算法开发，例如数字滤波器设计。

这四种表示法在数字信号处理中都使用，你需要熟练掌握它们之间的转换。这包括图形和数学方程。要确定使用的是哪种表示法，可以观察自变量及其取值范围。通常会发现以下四种表示法之一：k（或其他整数索引），取值范围从0到$N/2$；f，取值范围从0到0.5；T，取值范围从0到π；或是以赫兹表示的频率，取值范围从直流到实际采样率的一半。

密度泛函理论基函数

在离散傅里叶变换（DFT）中使用的正弦和余弦波通常被称为DFT**基函数**。换句话说，DFT的输出是一组表示振幅的数值。基函数是一组振幅为*1*的正弦和余弦波。如果你将每个振幅（即频域信号）分配给对应的正弦或余弦波（即基函数），最终得到的是一组*标度化*的正弦和余弦波，这些波形相加后就能还原出时域信号。

DFT基函数由以下方程生成：

方程8-1
DFT基函数的方程。在这些方程中，$c_k[i]$ 和 $s_k[i]$ 分别是余弦波和正弦波，每个波长为N个点，从i=0到N-1。参数k决定了波的频率。在N点DFT中，k的取值范围为0到N/2。

$$c_k[\text{i }] = \cos（2\pi \text{ ki /N}）s$$

$$_k[\text{i }] = \sin（2\pi \text{ ki /N}）$$

【补充】k的意思是：输入信号的采样率是N，那么被记录的最高频率就只能是N/2，傅里叶转换后我们最高也只能取N/2频率

其中：$c_k[\,]$ 是$ReX[k]$ 中保持的振幅的余弦波，$s_k[\,]$ 是$ImX[k]$ 中保持的振幅的正弦波。例如，图8-5展示了在N=32点DFT中使用的17个正弦波和17个余弦波中的部分。由于这些正弦波相加形成输入信号，它们必须与输入信号具有相同的*长度*。在这种情况下，每个波形从i=0到31共有32个点。参数k设置了每个正弦波的频率。特别地，$c_1[\,]$ 是在N个点上完成*一个*完整周期的余弦波，$c_5[\,]$ 是在N个点上完成*五个*完整周期的余弦波，依此类推。这是理解基函数的一个重要概念；频率参数k等于信号N个点上发生的完整周期数。

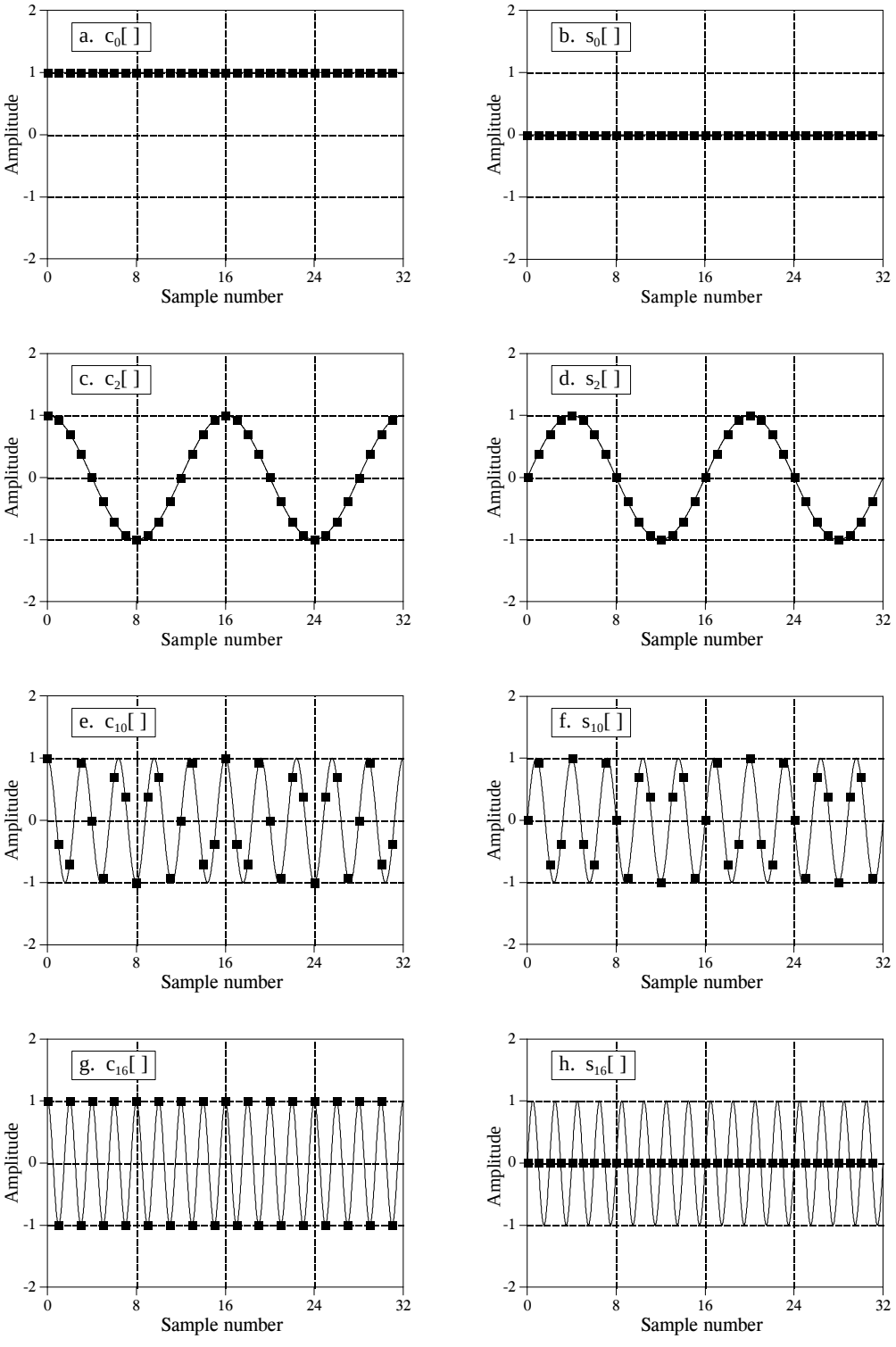


FIGURE 8-5
DFT basis functions. A 32 point DFT has 17 discrete cosine waves and 17 discrete sine waves for its basis functions. Eight of these are shown in this figure. These are discrete signals; the continuous lines are shown in these graphs only to help the reader's eye follow the waveforms.

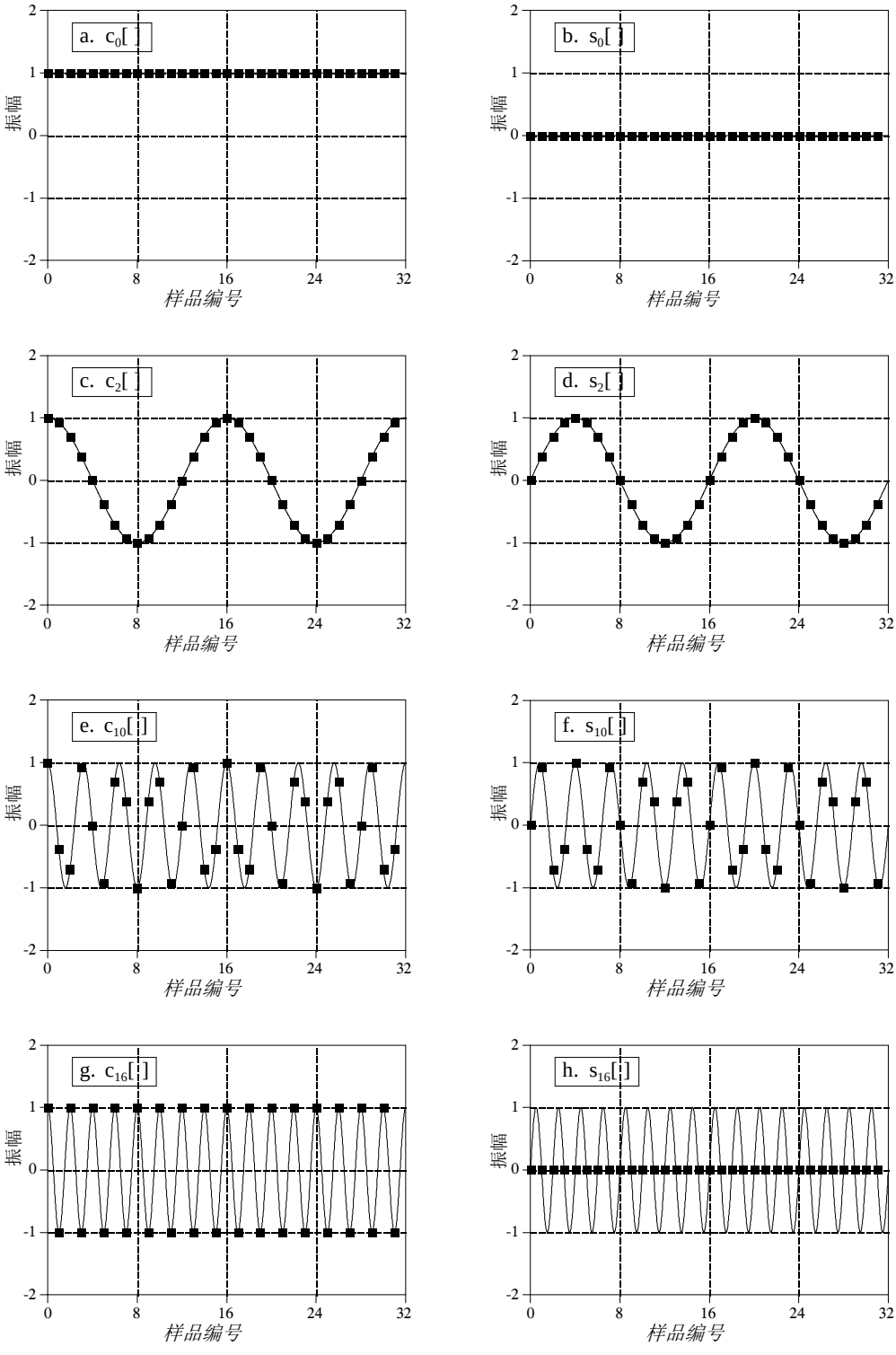


图8-5

DFT基函数。32点DFT的基函数包含17个离散余弦波和17个离散正弦波。本图展示了其中的8个基函数。这些是离散信号；图中仅用连续线表示，以便读者直观追踪波形。

Let's look at several of these basis functions in detail. Figure (a) shows the cosine wave $c_0[\,]$. This is a cosine wave of zero frequency, which is a constant value of one. This means that $ReX[0]$ holds the average value of all the points in the time domain signal. In electronics, it would be said that $ReX[0]$ holds the **DC offset**. The sine wave of zero frequency, $s_0[\,]$, is shown in (b), a signal composed of all *zeros*. Since this can not affect the time domain signal being synthesized, the value of $ImX[0]$ is *irrelevant*, and always set to zero. More about this shortly.

Figures (c) & (d) show $c_2[\,]$ & $s_2[\,]$, the sinusoids that complete *two* cycles in the N points. These correspond to $ReX[2]$ & $ImX[2]$, respectively. Likewise, (e) & (f) show $c_{10}[\,]$ & $s_{10}[\,]$, the sinusoids that complete *ten* cycles in the N points. These sinusoids correspond to the amplitudes held in the arrays $ReX[10]$ & $ImX[10]$. The problem is, the samples in (e) and (f) no longer *look* like sine and cosine waves. If the continuous curves were not present in these graphs, you would have a difficult time even detecting the pattern of the waveforms. This may make you a little uneasy, but don't worry about it. From a mathematical point of view, these samples do form discrete sinusoids, even if your eye cannot follow the pattern.

The highest frequencies in the basis functions are shown in (g) and (h). These are $c_{N/2}[\,]$ & $s_{N/2}[\,]$, or in this example, $c_{16}[\,]$ & $s_{16}[\,]$. The discrete cosine wave alternates in value between 1 and -1, which can be interpreted as sampling a continuous sinusoid at the *peaks*. In contrast, the discrete sine wave contains all zeros, resulting from sampling at the *zero crossings*. This makes the value of $ImX[N/2]$ the same as $ImX[0]$, always equal to zero, and not affecting the synthesis of the time domain signal.

Here's a puzzle: If there are N samples entering the DFT, and $N+2$ samples exiting, where did the extra information come from? The answer: two of the output samples contain *no* information, allowing the other N samples to be fully independent. As you might have guessed, the points that carry no information are $ImX[0]$ and $ImX[N/2]$, the samples that always have a value of zero.

Synthesis, Calculating the Inverse DFT

Pulling together everything said so far, we can write the **synthesis equation**:

$$x[i] = \sum_{k=0}^{N/2} Re\bar{X}[k]\cos(2\pi ki/N) + \sum_{k=0}^{N/2} Im\bar{X}[k]\sin(2\pi ki/N)$$

EQUATION 8-2
The synthesis equation. In this relation, $x[i]$ is the signal being synthesized, with the index, i, running from 0 to $N-1$. $Re\bar{X}[k]$ and $Im\bar{X}[k]$ hold the amplitudes of the cosine and sine waves, respectively, with k running from 0 to $N/2$. Equation 8-3 provides the normalization to change this equation into the inverse DFT.

让我们详细看看其中几个基函数。图(a)展示了余弦波$c_0[\,]$。这是一个零频率的余弦波，其值恒定为1。这意味着*ReX*[0]表示时域信号中所有点的平均值。在电子学中，可以说*ReX*[0]代表**直流偏移**。零频率的正弦波$s_0[\,]$如(b)所示，是一个由所有*零值*组成的信号。由于这不会影响被合成的时域信号，*ImX*[0]的值*无关紧要*，并始终设为零。稍后会详细说明这一点。

图(c)和(d)展示了$c_2[\,]$ & $s_2[\,]$，即在*N*个点上完成*两个*周期的正弦波。它们分别对应*ReX*[2] &*Im X*[2]。同样地，(e)和(f)展示了$c_{10}[\,]$ & $s_{10}[\,]$，即在*N*个点上完成*十个*周期的正弦波。这些正弦波对应阵列中保持的振幅*ReX*[10] &*Im X*[10]。问题在于，(e)和(f)中的样本不再*看起来*像正弦和余弦波。如果这些图中没有连续曲线，你甚至很难检测波形的模式。这可能会让你有点不安，但不必担心。从数学角度来看，这些样本确实形成了离散的正弦波，即使你的眼睛无法跟随其模式。

基函数中最高频率的值如(g)和(h)所示。这些值为$c_{N/2}[\,]$ &$s_{N/2}[\,]$，或在此示例中为$c_{16}[\,]$ &$s_{16}[\,]$。离散余弦波的数值在1和-1之间交替变化，可解释为在*峰值*处对连续正弦波进行采样。相比之下，离散正弦波包含所有零点，这是在*零交叉点*处采样所致。这使得*ImX*[*N*/2]的值始终等于零，且与*ImX*[0]相同，不会影响时域信号的合成。

这里有个谜题：如果有*N*个样本进入DFT，而*N*+2个样本退出，那么额外信息从何而来？答案是：两个输出样本包含*无*信息，使得其他*N*个样本完全独立。正如你可能猜到的，携带无信息的点是*Im X*[0]和*Im X*[*N*/2]，即始终为零的样本。

合成与逆DFT的计算

综合以上所述，我们可以写出**综合方程：**

$$x[i] = \sum_{k=0}^{N/2} Re\bar{X}[k]\cos(2\pi ki/N) + \sum_{k=0}^{N/2} Im\bar{X}[k]\sin(2\pi ki/N)$$

方程8-2
合成方程。在此关系中，*x*[*i*] 是被合成的信号，索引*i*从 0 到*N*-1运行。$Re\bar{X}[k]$ 和$Im\bar{X}[k]$ 分别保持余弦波和正弦波的振幅，其中*k*从 0 到*N*/2。公式 8-3 提供了归一化处理，将此方程转换为逆 DFT。

In words, any N point signal, $x[i]$, can be created by adding $N/2 + 1$ cosine waves and $N/2 + 1$ sine waves. The amplitudes of the cosine and sine waves are held in the arrays $Im\,\bar{X}[k]$ and $Re\,\bar{X}[k]$, respectively. The synthesis equation multiplies these amplitudes by the basis functions to create a set of scaled sine and cosine waves. Adding the scaled sine and cosine waves produces the time domain signal, $x[i]$.

In Eq. 8-2, the arrays are called $Im\,\bar{X}[k]$ and $Re\,\bar{X}[k]$, rather than $Im\,X[k]$ and $Re\,X[k]$. This is because the *amplitudes needed for synthesis* (called in this discussion: $Im\,\bar{X}[k]$ and $Re\,\bar{X}[k]$), are slightly different from the *frequency domain of a signal* (denoted by: $Im\,X[k]$ and $Re\,X[k]$). This is the scaling factor issue we referred to earlier. Although the conversion is only a simple normalization, it is a common bug in computer programs. Look out for it! In equation form, the conversion between the two is given by:

EQUATIONS 8-3
Conversion between the sinusoidal amplitudes and the frequency domain values. In these equations, $Re\,X[k]$ and $Im\,\bar{X}[k]$ hold the amplitudes of the cosine and sine waves needed for synthesis, while $Re\,X[k]$ and $Im\,X[k]$ hold the real and imaginary parts of the frequency domain. As usual, N is the number of points in the time domain signal, and k is an index that runs from 0 to $N/2$.

$$Re\,\bar{X}[k] = \frac{Re\,X[k]}{N/2}$$

$$Im\,\bar{X}[k] = -\frac{Im\,X[k]}{N/2}$$

except for two special cases:

$$Re\,\bar{X}[0] = \frac{Re\,X[0]}{N}$$

$$Re\,\bar{X}[N/2] = \frac{Re\,X[N/2]}{N}$$

Suppose you are given a frequency domain representation, and asked to synthesize the corresponding time domain signal. To start, you must find the amplitudes of the sine and cosine waves. In other words, given $Im\,X[k]$ and $Re\,X[k]$, you must find $Im\,\bar{X}[k]$ and $Re\,\bar{X}[k]$. Equation 8-3 shows this in a mathematical form. To do this in a computer program, three actions must be taken. First, divide all the values in the frequency domain by $N/2$. Second, change the sign of all the imaginary values. Third, divide the first and last samples in the real part, $Re\,X[0]$ and $Re\,X[N/2]$, by two. This provides the amplitudes needed for the synthesis described by Eq. 8-2. Taken together, Eqs. 8-2 and 8-3 *define* the inverse DFT.

The entire Inverse DFT is shown in the computer program listed in Table 8-1. There are two ways that the synthesis (Eq. 8-2) can be programmed, and both are shown. In the first method, each of the scaled sinusoids are generated one at a time and added to an accumulation array, which ends up becoming the time domain signal. In the second method, each sample in the time domain signal is calculated one at a time, as the sum of all the

具体来说，任何N点信号$x[i]$都可以通过叠加$N/2+1$个余弦波和$N/2+1$个正弦波来生成。余弦波和正弦波的振幅分别存储在数组Im中。$\bar{X}[k]$ 和$Re\bar{X}[k]$，分别。合成方程将这些振幅乘以基函数，生成一组缩放后的正弦和余弦波。将这些缩放后的正弦和余弦波相加，即可得到时域信号$x[i]$。

在等式8-2中，数组被称为$Im\bar{X}[k]$ 和$Re\bar{X}[k]$，而非$Im\,X[k]$和$ReX[k]$。这是因为*合成所需的振幅*（在本讨论中称为：$Im\bar{X}[k]$ 和$Re\bar{X}[k]$），*与信号的频域*（记为：$Im\,X[k]$ 和$ReX[k]$）略有不同。这就是我们之前提到的缩放因子问题。虽然这种转换仅是简单的归一化，但却是计算机程序中常见的错误。注意防范！用方程形式表示，两者之间的转换由以下公式给出：

方程8-3
正弦波振幅与频域值之间的转换。在这些方程中，$Re\bar{X}[k]$ 和$Im\bar{X}[k]$ 保存合成所需的余弦波和正弦波的振幅，而$ReX[k]$ 和$Im\,X[k]$ 保存频域中的实部和虚部。通常，N是时域信号中的点数，k是从 0 到$N/2$ 的索引。

$$Re\bar{X}[k] = \frac{ReX[k]}{N/2}$$

$$Im\bar{X}[k] = -\frac{ImX[k]}{N/2}$$

除以下两种特殊情况外：

$$Re\bar{X}[0] = \frac{ReX[0]}{N}$$

$$Re\bar{X}[N/2] = \frac{ReX[N/2]}{N}$$

【补充】：$Re\bar{X}[k] = ReX[k] \div N/2$解释：
$ReX[k]$是对N个(x[i] * cos$(2\pi ki/N)$)进行求和的结果
我们可以假设x[i]和cos$(2\pi ki/N)$是两个一模一样的余弦函数，他们的频率均为k，则：

当k==0 || k ==N/2时：
$ReX[k]$为x[i]振幅的N倍当 0 < k < N/2 时：
$ReX[k]$为x[i]振幅的N/2倍

详情见笔记：DFT分析算法中的振幅缩放

假设你得到一个频域表示，并被要求合成相应的时间域信号。首先，你必须找到正弦和余弦波的振幅。换句话说，给定$Im\,X[k]$ 和$ReX[k]$，你必须找到$Im\bar{X}[k]$ 和$Re\bar{X}[k]$. 公式8-3以数学形式展示了这一点。要在计算机程序中实现这一点，必须执行三个操作。首先，将频域中的所有值除以$N/2$。其次，改变所有虚部值的符号。第三，将实部中的第一个和最后一个样本$ReX[0]$ 和$ReX[N/2]$ 除以二。这为等式8-2描述的合成提供了所需的幅度。综合来看，公式8-2和8-3*定义*了逆DFT。

整个逆DFT过程在表8-1列出的计算机程序中展示。合成（等式8-2）有两种编程方式，两者均有所展示。在第一种方法中，每个缩放后的正弦波依次生成并添加到累加数组中，最终形成时域信号。在第二种方法中，时域信号中的每个样本依次计算，作为所有信号的总和。

```
100 'THE INVERSE DISCRETE FOURIER TRANSFORM
110 'The time domain signal, held in XX[ ], is calculated from the frequency domain signals,
120 'held in REX[ ] and IMX[ ].
130 '
140 DIM XX[511]                 'XX[ ]  holds the time domain signal
150 DIM REX[256]                'REX[ ] holds the real part of the frequency domain
160 DIM IMX[256]                'IMX[ ] holds the imaginary part of the frequency domain
170 '
180 PI = 3.14159265             'Set the constant, PI
190 N% = 512                    'N% is the number of points in XX[ ]
200 '
210 GOSUB XXXX                  'Mythical subroutine to load data into REX[ ] and IMX[ ]
220 '
230
240 '                           'Find the cosine and sine wave amplitudes using Eq. 8-3
250 FOR K% = 0 TO 256
260   REX[K%] =  REX[K%] / (N%/2)
270   IMX[K%]  = -IMX[K%] / (N%/2)
280 NEXT K%
290 '
300 REX[0] = REX[0] / 2
310 REX[256] = REX[256] / 2
320 '
330 '
340 FOR I% = 0 TO 511           'Zero XX[ ] so it can be used as an accumulator
350   XX[I%] = 0
360 NEXT I%
370 '
380 '                           Eq. 8-2  SYNTHESIS METHOD #1.  Loop through each
390 '                           frequency generating the entire length of the sine and cosine
400 '                           waves, and add them to the accumulator signal, XX[ ]
410 '
420 FOR K% = 0 TO 256           'K% loops through each sample in REX[ ] and IMX[ ]
430   FOR I% = 0 TO 511    'I% loops through each sample in XX[ ]
440     '
450     XX[I%] = XX[I%] + REX[K%] * COS(2*PI*K%*I%/N%)
460     XX[I%] = XX[I%] + IMX[K%] * SIN(2*PI*K%*I%/N%)
470     '
480   NEXT I%
490 NEXT K%
500 '
510 END
```

Alternate code for lines 380 to 510

```
380 '                           Eq. 8-2  SYNTHESIS METHOD #2.  Loop through each
390 '                           sample in the time domain, and sum the corresponding
400 '                           samples from each cosine and sine wave
410 '
420 FOR I% = 0 TO 511           'I% loops through each sample in XX[ ]
430   FOR K% = 0 TO 256         'K% loops through each sample in REX[ ] and IMX[ ]
440     '
450     XX[I%] = XX[I%] + REX[K%] * COS(2*PI*K%*I%/N%)
460     XX[I%] = XX[I%] + IMX[K%] * SIN(2*PI*K%*I%/N%)
470     '
480   NEXT K%
490 NEXT I%
500 '
510 END
```

TABLE 8-1

```
100 逆离散傅里叶变换
110 时域信号（存储于XX[ ]中）由频率域信号（存储于REX[ ]和 IMX [ ]中）计算得出。
130 '
140 DIM XX[511]                    XX[ ] 用于存储时域信号
150 DIM REX[256]                   'ReX[ ]' 表示频域中的实部
160 DIM IMX[256]                   IMX [ ] 用于存储频域170的虚部
180 PI = 3.14159265                设定常数PI
190 N% = 512                       N%表示XX[ ]中的点数
200 '
210 GOSUB XXXX                     用于将数据加载到REX[ ]和 IMX [ ]220中的特殊子程序
230
240 '                              使用等式8-3求出余弦和正弦波的振幅
250 K% = 0 至 256
260   REX[K%] =  REX[K%] / (N%/2)
270   IMX[K%]  = -IMX[K%] / (N%/2)
280 下一个K%
290 '
300 REX[0] = REX[0] / 2
310 REX[256] = REX[256] / 2
320 '
330 '
340 I% = 0 至 511                  Zero XX[ ] 可用作累加器
350   XX[I%] = 0
360 下 一 个  I%
370 '
380 '                              方程8-2合成方法#1。逐个循环
390 '                              正弦与余弦全周期信号发生器
400 '                              将波形信号累加至累加器信号XX[ ] 410 '
420 K% = 0 至 256                  K%遍历REX[ ]和 IMX [ ]中的每个样本
430   当I%=0至511时，  'I%' 遍历XX[ ]中的每个样本
440    '
450     XX[I%] = XX[I%] + REX[K%] * 硫化羰（2*PI*K%*I%/N%）
460     XX[I%] = XX[I%] + IMX[K%] * SIN（2*PI*K%*I%/N%）
470   '
480   下一次输入
490 下 一 个  K%
500 '
510 结束
```

第380至510行的替代代码

```
380 '                              式8-2合成方法#2。逐个循环
390 '                              时域采样，并求和相应
400 '                              从每个余弦波和正弦波中抽取样本 410 '
420 I% = 0 至 511                  I%循环遍历XX[ ]中的每个样本
430   K% = 0 至 256                K%遍历REX[ ]和 IMX [ ]中的每个样本
440    '
450     XX[I%] = XX[I%] + REX[K%] * 硫化羰（2*PI*K%*I%/N%）
460     XX[I%] = XX[I%] + IMX[K%] * SIN（2*PI*K%*I%/N%）
470   '
480   下一个K%
490 下 一 个  I%
500 '
510 结束
```

表8-1

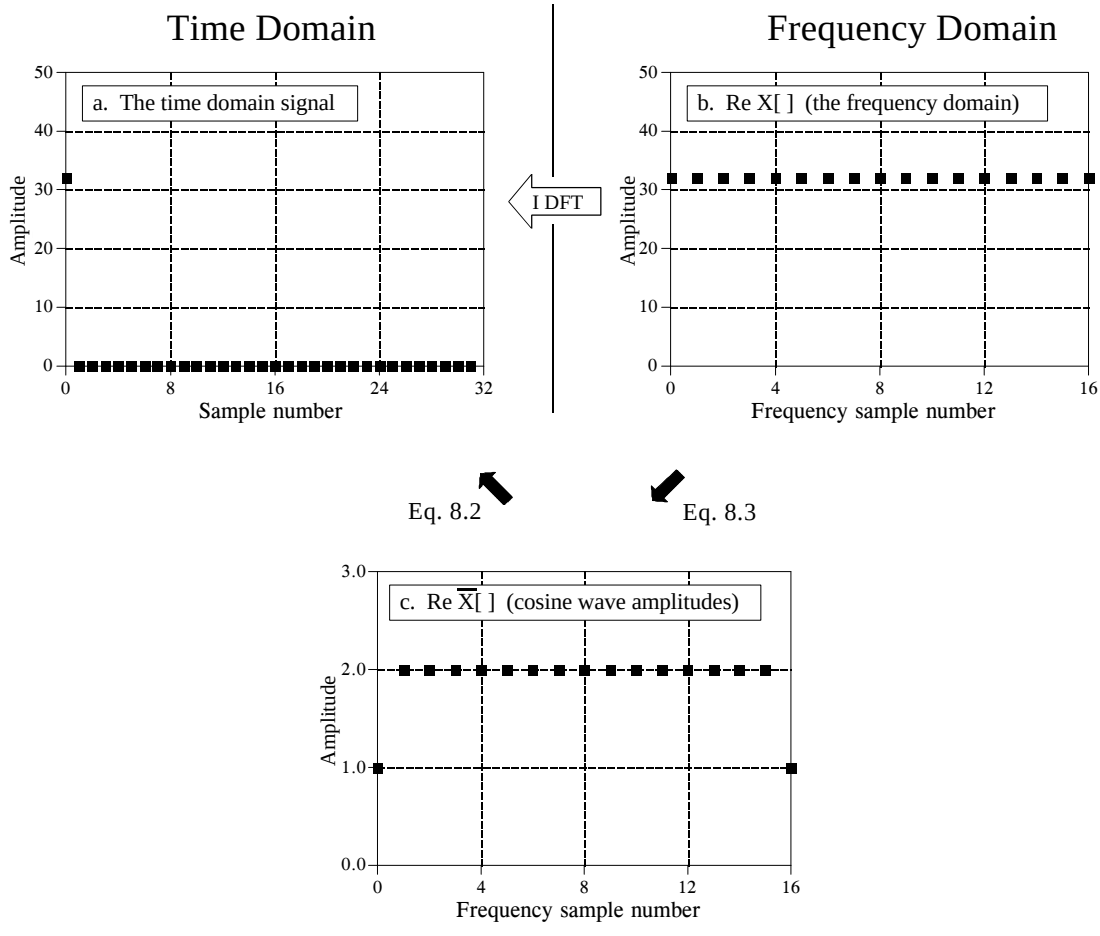


FIGURE 8-6
Example of the Inverse DFT. Figure (a) shows an example time domain signal, an impulse at sample zero with an amplitude of 32. Figure (b) shows the real part of the frequency domain of this signal, a constant value of 32. The imaginary part of the frequency domain (not shown) is composed of all zeros. Figure(c) shows the amplitudes of the cosine waves needed to reconstruct (a) using Eq. 8-2. The values in (c) are found from (b) by using Eq. 8-3.

corresponding samples in the cosine and sine waves. Both methods produce the same result. The difference between these two programs is very minor; the inner and outer loops are swapped during the synthesis.

Figure 8-6 illustrates the operation of the Inverse DFT, and the slight differences between the frequency domain and the amplitudes needed for synthesis. Figure 8-6a is an example signal we wish to synthesize, an impulse at sample zero with an amplitude of 32. Figure 8-6b shows the frequency domain representation of this signal. The real part of the frequency domain is a constant value of 32. The imaginary part (not shown) is composed of all zeros. As discussed in the next chapter, this is an important DFT *pair*: an impulse in the time domain corresponds to a constant value in the frequency domain. For now, the important point is that (b) is the *DFT* of (a), and (a) is the *Inverse DFT* of (b).

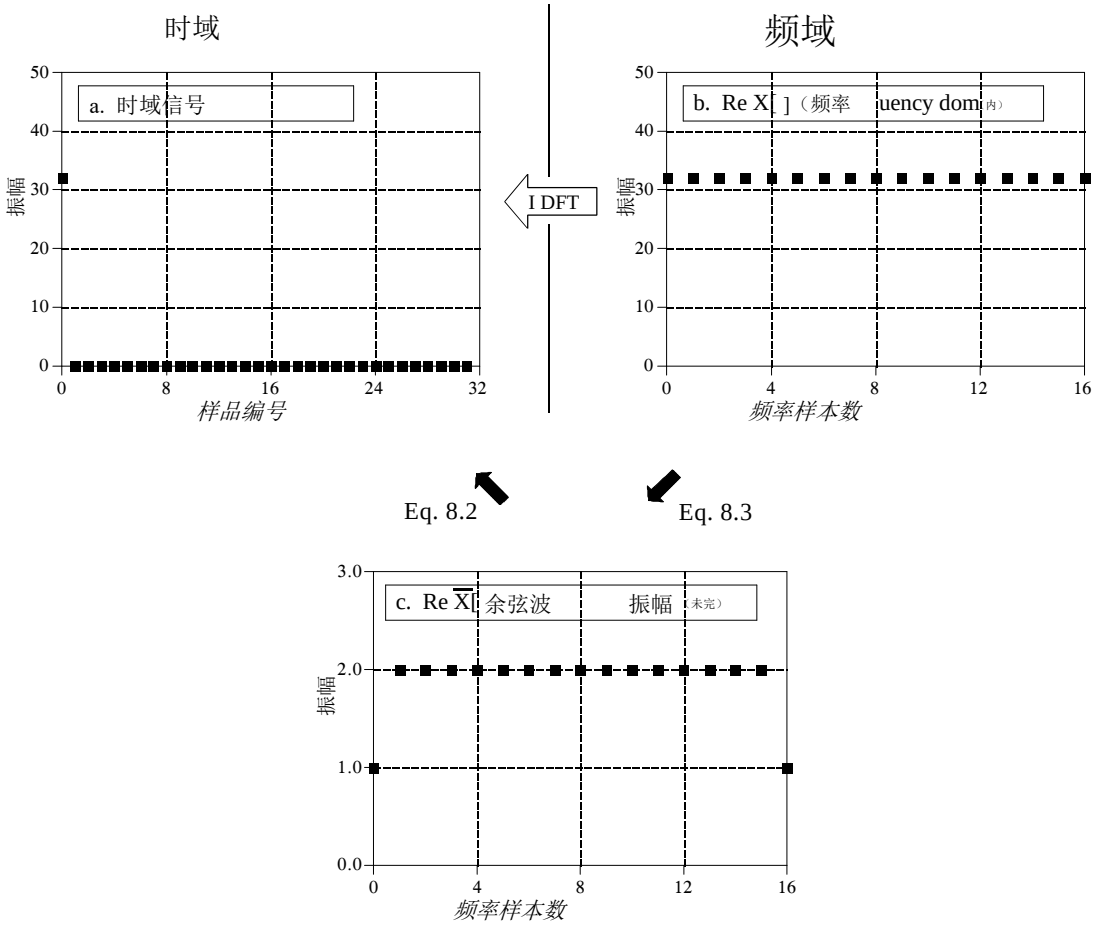


图8-6
逆DFT示例。图(a)展示了一个时域信号示例，即采样点零处的脉冲，其振幅为32。图(b)显示了该信号频域的实部，为一个恒定值。
32. 频域的虚部（未显示）由所有零点组成。图(c)展示了使用等式8-2重建(a)所需的余弦波振幅。图(c)中的数值是通过使用等式8-3从图(b)中得出的。

在余弦和正弦波中对应的样本。两种方法产生相同的结果。这两个程序之间的差异非常微小；在合成过程中内循环与外循环被互换。

图8-6展示了逆DFT的运算过程，以及频域与合成所需振幅之间的细微差异。图8-6a是我们想要合成的示例信号——采样点零处的脉冲信号，振幅为32。图8-6b显示了该信号的频域表示。频域的实部为恒定值32，虚部（未显示）则由全零点构成。正如下一章将讨论的，这是重要的DFT*对*：时域中的脉冲信号对应频域中的恒定值。目前关键在于：(b)是(a)的*DFT*，而(a)则是(b)的*逆DFT*。

Equation 8-3 is used to convert the frequency domain signal, (b), into the amplitudes of the cosine waves, (c). As shown, all of the cosine waves have an amplitude of *two*, except for samples 0 and 16, which have a value of *one*. The amplitudes of the sine waves are not shown in this example because they have a value of zero, and therefore provide no contribution. The synthesis equation, Eq. 8-2, is then used to convert the amplitudes of the cosine waves, (b), into the time domain signal, (a).

This describes *how* the frequency domain is different from the sinusoidal amplitudes, but it doesn't explain *why* it is different. The difference occurs because the frequency domain is defined as a **spectral density**. Figure 8-7 shows how this works. The example in this figure is the real part of the frequency domain of a 32 point signal. As you should expect, the samples run from 0 to 16, representing 17 frequencies equally spaced between 0 and 1/2 of the sampling rate. S*pectral density* describes how much signal (amplitude) is present *per unit of bandwidth*. To convert the sinusoidal amplitudes into a spectral density, divide each amplitude by the bandwidth represented by each amplitude. This brings up the next issue: how do we determine the bandwidth of each of the discrete frequencies in the frequency domain?

As shown in the figure, the bandwidth can be defined by drawing dividing lines between the samples. For instance, sample number 5 occurs in the band between 4.5 and 5.5; sample number 6 occurs in the band between 5.5 and 6.5, etc. Expressed as a fraction of the total bandwidth (i.e., $N/2$), the bandwidth of each sample is $2/N$. An exception to this is the samples on each end, which have one-half of this bandwidth, $1/N$. This accounts for the $2/N$ scaling factor between the sinusoidal amplitudes and frequency domain, as well as the additional factor of two needed for the first and last samples.

Why the negation of the imaginary part? This is done solely to make the *real DFT* consistent with its big brother, the *complex DFT*. In Chapter 29 we will show that it is necessary to make the mathematics of the complex DFT work. When dealing only with the real DFT, many authors do not include this negation. For that matter, many authors do not even include

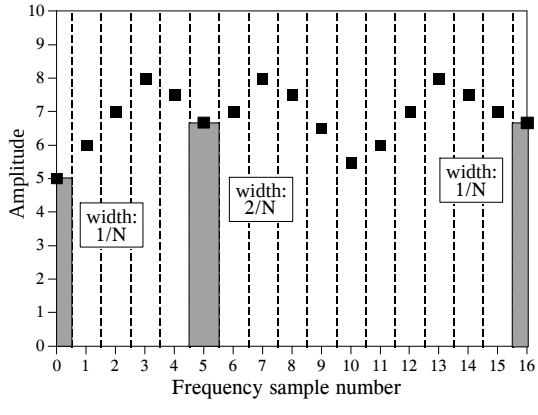


FIGURE 8-7
The bandwidth of frequency domain samples. Each sample in the frequency domain can be thought of as being contained in a frequency band of width $2/N$, expressed as a fraction of the total bandwidth. An exception to this is the first and last samples, which have a bandwidth only one-half this wide, $1/N$.

方程8-3用于将频域信号(b)转换为余弦波的振幅(c)。如图所示，所有余弦波的振幅均为二，除了第0和第16个样本，其值为一。正弦波的振幅在本示例中未显示，因为它们的值为零，因此不作贡献。随后使用合成方程等式8-2将余弦波振幅(b)转换为时域信号(a)。

这描述了*频域*与正弦波幅值的不同之处，但并未解释*其差异原因*。这种差异源于频域被定义为**频谱密度**。图8-7展示了其工作原理。该图示例为32点信号频域的实部。正如预期，采样点从0到16，代表在0到采样率1/2之间等距分布的17个频率。频谱*密度*描述了*每单位带宽*中存在的信号（幅值）量。要将正弦波幅值转换为频谱密度，需将每个幅值除以其对应的带宽。这就引出了下一个问题：如何确定频域中每个离散频率的带宽？

如图所示，带宽可通过在样本间绘制分界线来定义。例如，样本5位于4.5至5.5的频带内；样本6位于5.5至6.5的频带内，依此类推。若以总带宽的分数形式表示（即$N/2$），每个样本的带宽为$2/N$。例外情况是两端样本，其带宽为该值的一半，即$1/N$。这解释了正弦波幅值与频域之间$2/N$的缩放因子，以及首尾样本所需的额外两倍因子。

为什么要否定虚部？这样做纯粹是为了让*实部DFT*与其“大哥”——*复部DFT*保持一致。在第29章中我们将证明，这是确保复部DFT数学体系有效运行的必要条件。当仅处理实部DFT时，许多作者并未包含这一否定项。事实上，很多作者甚至完全省略了

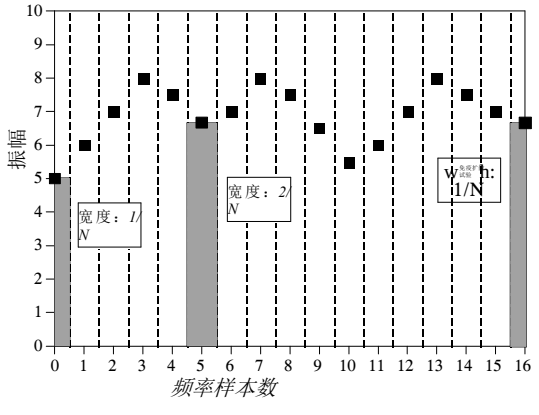


图8-7
频域样本的带宽。频域中的每个样本可以认为包含在带宽为$2/N$的频带中，表示为总带宽的分数。第一个和最后一个样本除外，它们的带宽只有这个宽度的一半，即$1/N$。

the 2/*N* scaling factor. Be prepared to find both of these missing in some discussions. They are included here for a tremendously important reason: The most efficient way to calculate the DFT is through the Fast Fourier Transform (FFT) algorithm, presented in Chapter 12. The FFT generates a frequency domain defined according to Eq. 8-2 and 8-3. If you start messing with these normalization factors, your programs containing the FFT are not going to work as expected.

Analysis, Calculating the DFT

The DFT can be calculated in three completely different ways. First, the problem can be approached as a set of *simultaneous equations*. This method is useful for understanding the DFT, but it is too inefficient to be of practical use. The second method brings in an idea from the last chapter: *correlation*. This is based on detecting a known waveform in another signal. The third method, called the Fast Fourier Transform (FFT), is an ingenious algorithm that decomposes a DFT with *N* points, into *N* DFTs each with a single point. The FFT is typically hundreds of times faster than the other methods. The first two methods are discussed here, while the FFT is the topic of Chapter 12. It is important to remember that all three of these methods produce an identical output. Which should you use? In actual practice, *correlation* is the preferred technique if the DFT has less than about 32 points, otherwise the *FFT* is used.

DFT by Simultaneous Equations

Think about the DFT calculation in the following way. You are given *N* values from the time domain, and asked to calculate the *N* values of the frequency domain (ignoring the two frequency domain values that you know must be zero). Basic algebra provides the answer: to solve for *N* unknowns, you must be able to write *N* linearly independent equations. To do this, take the first sample from each sinusoid and add them together. The sum must be equal to the first sample in the time domain signal, thus providing the first equation. Likewise, an equation can be written for each of the remaining points in the time domain signal, resulting in the required *N* equations. The solution can then be found by using established methods for solving simultaneous equations, such as Gauss Elimination. Unfortunately, this method requires a tremendous number of calculations, and is virtually never used in DSP. However, it is important for another reason, it shows *why* it is possible to decompose a signal into sinusoids, how *many* sinusoids are needed, and that the basis functions must be linearly independent (more about this shortly).

DFT by Correlation

Let's move on to a better way, the *standard way* of calculating the DFT. An example will show how this method works. Suppose we are trying to calculate the DFT of a 64 point signal. This means we need to calculate the 33 points in the real part, and the 33 points in the imaginary part of the frequency domain. In this example we will only show how to calculate a single sample, $Im\,X[3]$, i.e., the amplitude of the sine wave that makes three complete cycles

2/*N*缩放因子。请做好准备，在某些讨论中可能会发现这两个因子的缺失。它们被包含在这里有一个极其重要的原因：计算DFT最有效的方法是通过第12章介绍的快速傅里叶变换（FFT）算法。该 FFT 生成的频域定义依据等式8-2和8-3。如果你开始调整这些归一化因子，包含 FFT 的程序将无法按预期工作。

浅析DFT的计算

傅里叶变换（DFT）可以通过三种截然不同的方法进行计算。首先，我们可以将问题视为一组*同时方程*的组合。这种方法有助于理解DFT原理，但效率太低，实际应用中难以派上用场。第二种方法借鉴了上一章的核心思想——*相关性分析*，其原理是通过检测其他信号中的已知波形来实现。第三种方法称为快速傅里叶变换（FFT），这是一种精妙的算法，能将包含*N*个点的DFT分解为*N*个单点DFT。 FFT 的运算速度通常比其他方法快数百倍。本文重点讨论前两种方法，而 FFT 则属于第12章的内容。需要特别注意的是，这三种方法最终都会得到相同的计算结果。那么该如何选择呢？在实际应用中，当DFT点数少于约32个时，*相关性分析*是首选技术；若超过这个临界值，则应采用*FFT*。

用联立方程求解DFT

我们可以这样理解DFT计算：给定时域的*N*个样本值，需要计算频域的*N*个值（忽略已知必须为零的两个频域值）。基础代数原理给出了答案：要解*N*个未知数，必须建立*N*个线性独立方程。具体操作时，取每个正弦信号的第一个样本并求和，其和必须等于时域信号的第一个样本，这就构成了第一个方程。同理，对时域信号中其余每个点建立方程，最终得到所需的*N*个方程组。此时可采用高斯消元法等求解联立方程组的经典方法。但遗憾的是，这种方法需要进行海量计算，因此在数字信号处理领域几乎从未被采用。然而，它之所以重要还有另一个原因，它显示了*为什么*信号可以分解成正弦波，需要*多少*个正弦波，以及基函数必须是线性独立的（稍后会详细介绍）。

相关DFT

现在我们来探讨更优的计算方法——DFT的*标准计算方式*。通过具体示例，我们将直观理解该方法的运作原理。假设需要计算一个64点信号的DFT，这意味着需要分别计算频域实部的33个点和虚部的33个点。本例将重点演示单个采样点*Im X*[3] 的计算方法，即构成三个完整周期的正弦波振幅。

between point 0 and point 63. All of the other frequency domain values are calculated in a similar manner.

Figure 8-8 illustrates using correlation to calculate $ImX[3]$. Figures (a) and (b) show two example time domain signals, called: $x1[\ ]$ and $x2[\ ]$, respectively. The first signal, $x1[\ ]$, is composed of nothing but a sine wave that makes three cycles between points 0 and 63. In contrast, $x2[\ ]$ is composed of several sine and cosine waves, *none* of which make three cycles between points 0 and 63. These two signals illustrate what the algorithm for calculating $ImX[3]$ must do. When fed $x1[\ ]$, the algorithm must produce a value of 32, the amplitude of the sine wave present in the signal (modified by the scaling factors of Eq. 8-3). In comparison, when the algorithm is fed the other signal, $x2[\ ]$, a value of zero must be produced, indicating that this particular sine wave is not present in this signal.

The concept of correlation was introduced in Chapter 7. As you recall, to detect a known waveform contained in another signal, multiply the two signals and add all the points in the resulting signal. The single number that results from this procedure is a measure of how similar the two signals are. Figure 8-8 illustrates this approach. Figures (c) and (d) both display the signal we are looking for, a sine wave that makes 3 cycles between samples 0 and 63. Figure (e) shows the result of multiplying (a) and (c). Likewise, (f) shows the result of multiplying (b) and (d). The sum of all the points in (e) is 32, while the sum of all the points in (f) is zero, showing we have found the desired algorithm.

The other samples in the frequency domain are calculated in the same way. This procedure is formalized in the *analysis equation*, the mathematical way to calculate the frequency domain from the time domain:

EQUATION 8-4
The analysis equations for calculating the DFT. In these equations, $x[i]$ is the time domain signal being analyzed, and $ReX[k]$ & $ImX[k]$ are the frequency domain signals being calculated. The index i runs from 0 to $N-1$, while the index k runs from 0 to $N/2$.

$$ReX[k] = \sum_{i=0}^{N-1} x[i] \cos(2\pi k i / N)$$

$$ImX[k] = -\sum_{i=0}^{N-1} x[i] \sin(2\pi k i / N)$$

In words, each sample in the frequency domain is found by multiplying the time domain signal by the sine or cosine wave being looked for, and adding the resulting points. If someone asks you what you are doing, say with confidence: "I am correlating the input signal with each basis function." Table 8-2 shows a computer program for calculating the DFT in this way.

The analysis equation does *not* require special handling of the first and last points, as did the synthesis equation. There is, however, a negative sign in the imaginary part in Eq. 8-4. Just as before, this negative sign makes the *real DFT* consistent with the *complex DFT*, and is not always included.

在0点至63点之间。其余所有频域值均以类似方式计算。

图8-8展示了使用相关性计算*Im X*[3] 的过程。图(a)和(b)分别显示了两个示例时域信号，称为：*x1*[ ] 和*x2*[ ] 。第一个信号*x1*[ ] 仅由一个正弦波组成，该波在0到63点之间完成三个周期。相比之下，*x2*[ ] 包含多个正弦和余弦波，*无*一个在0到63点之间完成三个周期。这两个信号说明了计算*Im X*[3] 的算法必须执行的操作。当输入*x1*[ ] 时，算法必须生成32的值，即信号中正弦波的振幅（经等式8-3的缩放因子调整）。相比之下，当算法输入另一个信号*x2*[ ] 时，必须生成零的值，表明该特定正弦波不存在于此信号中。

相关性概念在第七章中已有介绍。正如大家所知，要检测另一信号中包含的已知波形，需要将两个信号相乘，然后对结果信号中的所有点求和。通过这个过程得到的单一数值，就是衡量两个信号相似程度的指标。图8-8展示了这一方法的实现过程。图(c)和(d)都显示了我们要找的信号——一个在0到63采样点之间完成3个周期的正弦波。图(e)展示了(a)和(c)相乘的结果，图(f)则展示了(b)和(d)相乘的结果。图(e)中所有点的总和为32，而图(f)中所有点的总和为零，这表明我们找到了理想的算法。

频域中的其他样本以相同方式计算。该过程在*分析方程*中被形式化，这是从时域计算频域的数学方法：

方程8-4
用于计算DFT的分析方程。在这些方程中，*x*[*i*] 是被分析的时间域信号，而*ReX*[*k*] &*Im X*[*k*] 是被计算的频域信号。索引*i*从0到*N*-1运行，而索引*k*从0到N/2运行。

$$ReX[k] = \sum_{i=0}^{N-1} x[i] \cos(2\pi k i / N)$$

$$ImX[k] = -\sum_{i=0}^{N-1} x[i] \sin(2\pi k i / N)$$

具体而言，频域中的每个样本是通过将时域信号与目标正弦或余弦波相乘，再对所得点进行求和而获得的。若有人询问你的操作，可自信地回答：“我正在将输入信号与各基函数进行相关性分析。”表8-2展示了一个采用此方法计算离散傅里叶变换（DFT）的计算机程序。

分析方程*不需要*像合成方程那样对首尾点进行特殊处理。然而，在等式8-4的虚部中存在一个负号。与之前一样，这个负号使得*实DFT*与*复DFT*保持一致，并不总是被包含其中。

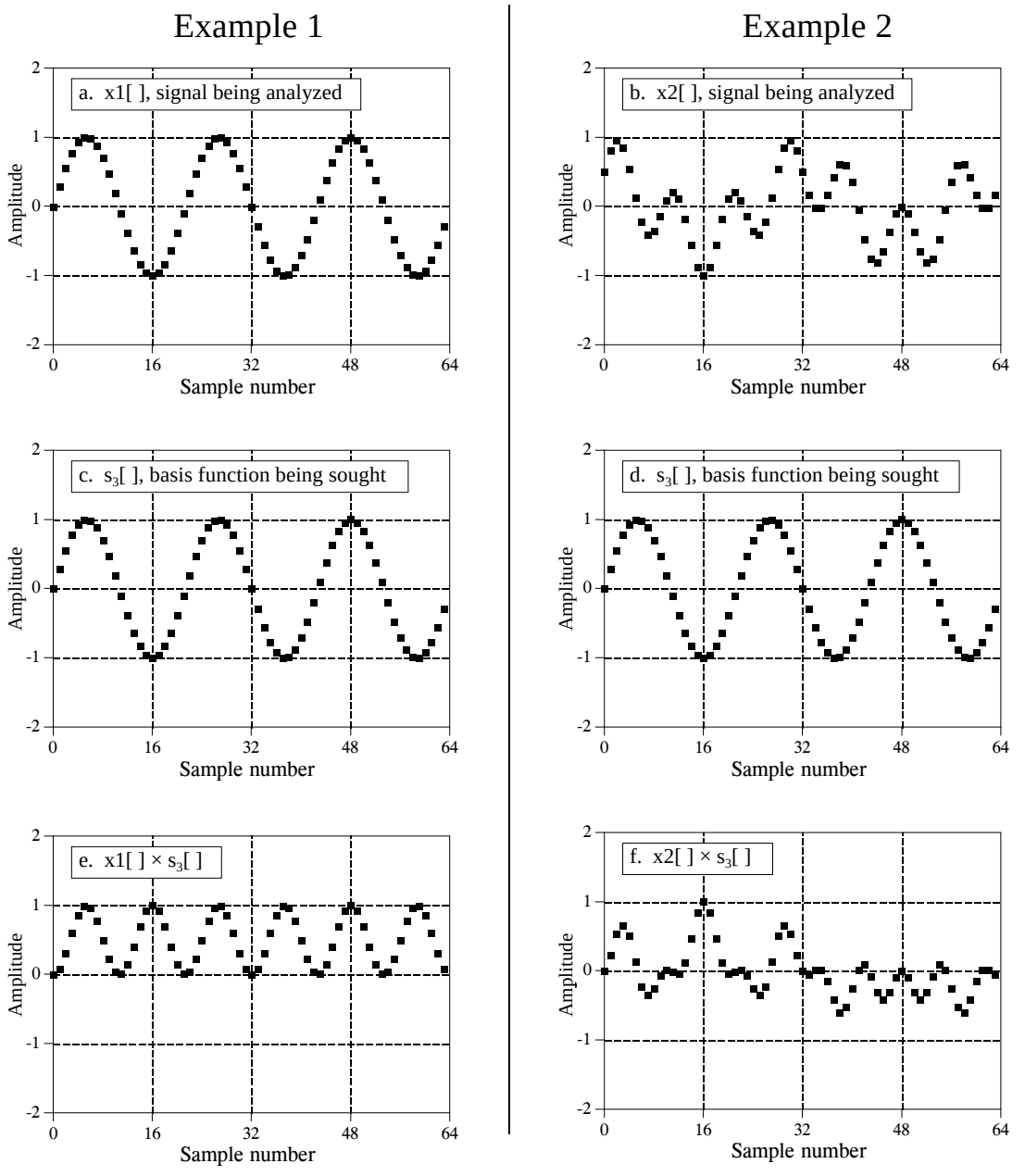


FIGURE 8-8
Two example signals, (a) and (b), are analyzed for containing the specific basis function shown in (c) and (d). Figures (e) and (f) show the result of multiplying each example signal by the basis function. Figure (e) has an average of 0.5, indicating that *x*1[ ]contains the basis function with an amplitude of 1.0. Conversely, (f) has a zero average, indicating that *x*2[ ] does not contain the basis function.

In order for this correlation algorithm to work, the basis functions must have an interesting property: each of them must be completely *uncorrelated* with all of the others. This means that if you multiply any two of the basis functions, the sum of the resulting points will be equal to zero. Basis functions that have this property are called **orthognal**. Many other

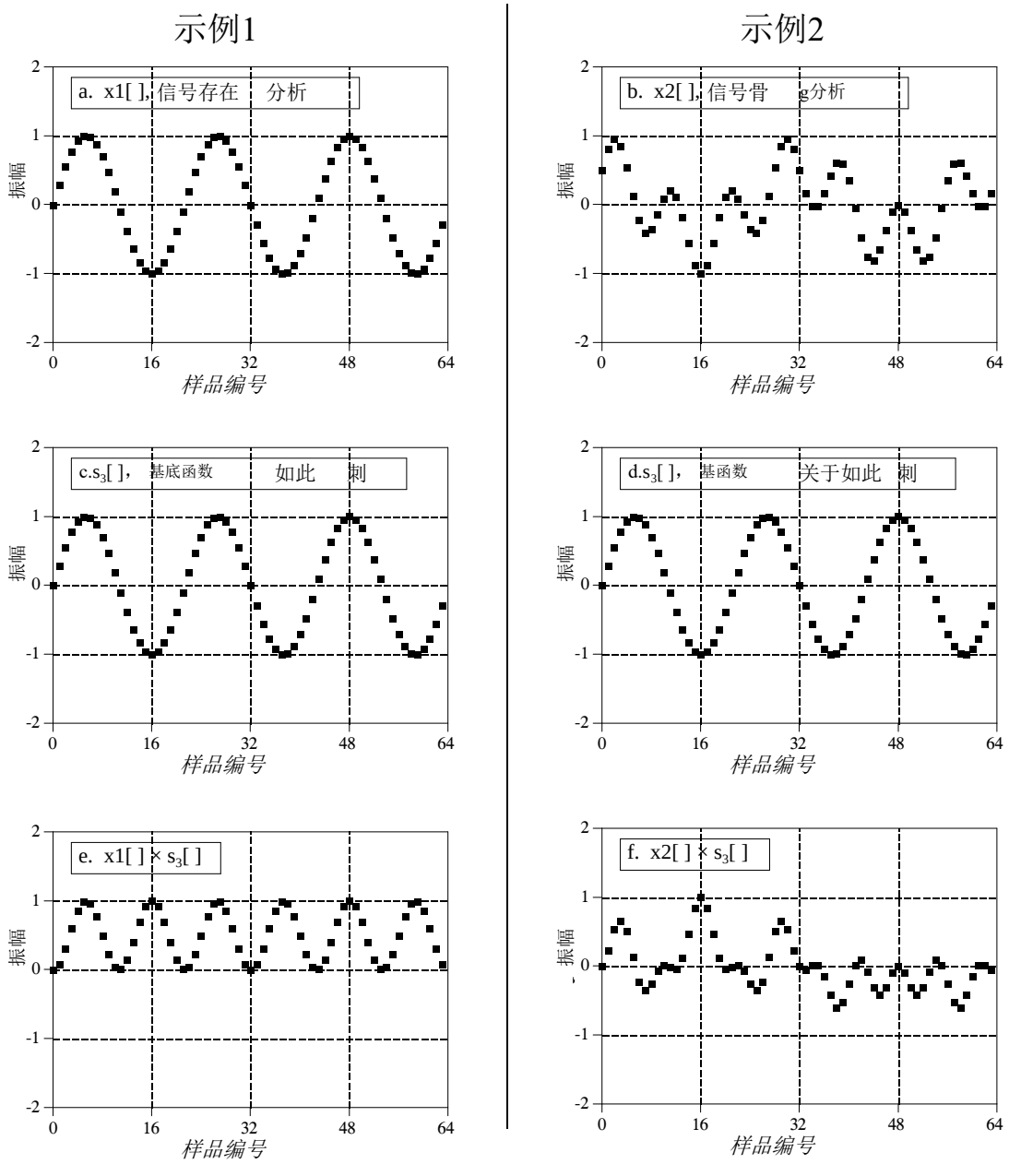


图8-8

针对(a)和(b)两个示例信号，我们分析其是否包含(c)和(d)中所示的特定基函数。图(e)和(f)展示了将每个示例信号与基函数相乘后的结果。图(e)的平均值为0.5，表明*x1*[ ]包含振幅为1.0的基函数。相反，图(f)的平均值为零，说明*x2*[ ]不包含该基函数。

为了使这种相关算法有效，基函数必须具有一个有趣的性质：每个基函数必须与其他所有基函数完全*不相关*。这意味着如果你将任意两个基函数相乘，结果点的总和将等于零。具有这种性质的基函数被称为**正交**。许多其他

```
100 'THE DISCRETE FOURIER TRANSFORM
110 'The frequency domain signals, held in REX[ ] and IMX[ ], are calculated from
120 'the time domain signal, held in XX[ ].
130 '
140 DIM XX[511]                    'XX[ ]  holds the time domain signal
150 DIM REX[256]                   'REX[ ] holds the real part of the frequency domain
160 DIM IMX[256]                   'IMX[ ] holds the imaginary part of the frequency domain
170 '
180 PI = 3.14159265                'Set the constant, PI
190 N% = 512                       'N% is the number of points in XX[ ]
200 '
210 GOSUB XXXX                     'Mythical subroutine to load data into XX[ ]
220 '
230 '
240 FOR K% = 0 TO 256    'Zero REX[ ] & IMX[ ] so they can be used as accumulators
250   REX[K%] = 0
260   IMX[K%] = 0
270 NEXT K%
280 '
290 '                              'Correlate XX[ ] with the cosine and sine waves, Eq. 8-4
300 '
310 FOR K% = 0 TO 256              'K% loops through each sample in REX[ ] and IMX[ ]
320   FOR I% = 0 TO 511    'I% loops through each sample in XX[ ]
330     '
340     REX[K%] = REX[K%] + XX[I%] * COS(2*PI*K%*I%/N%)
350     IMX[K%] = IMX[K%]  - XX[I%] * SIN(2*PI*K%*I%/N%)
360     '
370   NEXT I%
380 NEXT K%
390 '
400 END
```

TABLE 8-2

orthognal basis functions exist, including: square waves, triangle waves, impulses, etc. Signals can be decomposed into these other orthognal basis functions using correlation, just as done here with sinusoids. This is not to suggest that this is *useful*, only that it is *possible*.

As previously shown in Table 8-1, the *Inverse DFT* has two ways to be implemented in a computer program. This difference involves *swapping* the inner and outer loops during the synthesis. While this does not change the output of the program, it makes a difference in how you *view* what is being done. The *DFT* program in Table 8-2 can also be changed in this fashion, by swapping the inner and outer loops in lines 310 to 380. Just as before, the output of the program is the same, but the way you *think* about the calculation is different. (These two different ways of viewing the DFT and inverse DFT could be described as "input side" and "output side" algorithms, just as for convolution).

As the program in Table 8-2 is written, it describes how an individual sample in the frequency domain is affected by all of the samples in the time domain. That is, the program calculates each of the values in the frequency domain in succession, not as a group. When the inner and outer loops are exchanged, the program loops through each sample in the time domain, calculating the

```
100 离散傅里叶变换
110 存储在REX[]和 IMX []中的频域信号是通过以下方式计算得出的
120 时域信号，存储于XX[ ]中。
130 '
140 DIM XX[511]                    XX[ ] 用于存储时域信号
150 DIM REX[256]                   'ReX[ ]' 表示频域中的实部
160 DIM IMX[256]                   IMX [ ] 用于存储频域170的虚部
180 PI = 3.14159265                设定常数PI
190 N% = 512                       N%表示XX[ ]中的点数
200 '
210 GOSUB XXXX                     '将数据加载到XX[ ] 220中的神话子程序'
230 '
240 当 K% 为 0 至 256 时，  'Zero REX[ ] & IMX [ ] 可用作累加器
250   REX[K%] = 0
260   IMX[K%] = 0
270 下 一 个   K%
280  '
290 '                              将XX[ ]与余弦和正弦波相关联，等式 8-4 300
310 K% = 0 至 256                  K%遍历REX[ ]和 IMX [ ]中的每个样本
320   当I%=0至511时，  'I%'  遍历XX[ ]中的每个样本
330    '
340     REX[K%] = REX[K%] + XX[I%] * 硫化鞭（2*PI*K%*I%/N%）
350     IMX[K%] =  IMX[K%] -  XX[I%] *  SIN（2*PI*K%*I%/N%）
360  '
370   下一次输入
380 下 一 个   K%
390  '
400 结束
```

表8-2

正交基函数是存在的，包括：方波、三角波、脉冲等。信号可以分解成这些其他正交基函数，就像在这里用正弦波一样。这并不是说这是*有用的*，只是说它是*可能的*。

如表8-1所示，*逆DFT*在计算机程序中可通过两种方式实现。这种差异涉及在合成过程中*交换*内循环与外循环。虽然这不会改变程序的输出结果，但会影响你对计算过程的*理解*。表8-2中的*DFT*程序同样可以通过这种方式进行调整——只需交换第310至380行的内循环与外循环。与之前类似，程序输出结果保持不变，但你对计算过程的*思考方式*发生了变化。（这两种不同的DFT与逆DFT视角可被描述为“输入端”与“输出端”算法，就像卷积运算的处理方式一样）。

如表8-2所示程序的编写方式，其描述了频域中的单个样本如何受到时域中所有样本的影响。具体而言，该程序是按顺序计算频域中的每个值，而非整体处理。当内外循环顺序互换时，程序将遍历时域中的每个样本，进行计算。

contribution of that point to the frequency domain. The overall frequency domain is found by adding the contributions from the individual time domain points. This brings up our next question: what kind of contribution does an individual sample in the time domain provide to the frequency domain? The answer is contained in an interesting aspect of Fourier analysis called *duality*.

Duality

The synthesis and analysis equations (Eqs. 8-2 and 8-4) are strikingly similar. To move from one domain to the other, the known values are multiplied by the basis functions, and the resulting products added. The fact that the *DFT* and the *Inverse DFT* use this same mathematical approach is really quite remarkable, considering the totally different way we arrived at the two procedures. In fact, the only significant difference between the two equations is a result of the time domain being *one* signal of N points, while the frequency domain is *two* signals of $N/2 + 1$ points. As discussed in later chapters, the *complex DFT* expresses both the time and the frequency domains as complex signals of N points each. This makes the two domains completely symmetrical, and the equations for moving between them virtually *identical*.

This symmetry between the time and frequency domains is called **duality**, and gives rise to many interesting properties. For example, a single point in the frequency domain corresponds to a sinusoid in the time domain. By duality, the inverse is also true, a single point in the time domain corresponds to a sinusoid in the frequency domain. As another example, convolution in the time domain corresponds to multiplication in the frequency domain. By duality, the reverse is also true: convolution in the frequency domain corresponds to multiplication in the time domain. These and other duality relationships are discussed in more detail in Chapters 10 and 11.

Polar Notation

As it has been described so far, the frequency domain is a group of amplitudes of cosine and sine waves (with slight scaling modifications). This is called **rectangular** notation. Alternatively, the frequency domain can be expressed in **polar** form. In this notation, $ReX[\,]$ & $ImX[\,]$ are replaced with two other arrays, called the **Magnitude of $X[\,]$**, written in equations as: $MagX[\,]$, and the **Phase of $X[\,]$**, written as: $PhaseX[\,]$. The magnitude and phase are a pair-for-pair replacement for the real and imaginary parts. For example, $MagX[0]$ and $PhaseX[0]$ are calculated using only $ReX[0]$ and $ImX[0]$. Likewise, $MagX[14]$ and $PhaseX[14]$ are calculated using only $ReX[14]$ and $ImX[14]$, and so forth. To understand the conversion, consider what happens when you add a cosine wave and a sine wave of the same frequency. The result is a cosine wave of the same

该点对频域的贡献。通过将各个时域点的贡献相加，可以得到整体的频域。这就引出了我们的下一个问题：时域中的单个样本对频域提供了什么样的贡献？答案包含在傅里叶分析的一个有趣方面，称为*对偶性*。

二元性

合成与分析方程（方程8-2和8-4）惊人地相似。要从一个域转换到另一个域，已知值需乘以基函数，再将所得乘积相加。考虑到我们得出这两种方法的途径截然不同，*DFT*与*Inverse DFT*采用相同数学方法这一事实确实令人惊叹。实际上，两个方程之间唯一的显著差异在于：时域是*一个*信号的*N*个点，而频域是*两个*信号的*N*/2+1个点。如后续章节所述，*复DFT*将时域和频域分别表示为*N*个点的复信号。这使得两个域完全对称，且它们之间的转换方程几乎*完全相同*。

时域与频域之间的这种对称性被称为**对偶性**，它催生了许多有趣的特性。例如，频域中的单一点对应时域中的正弦波。根据对偶性原理，其逆关系同样成立：时域中的单一点对应频域中的正弦波。再比如，时域中的卷积运算对应频域中的乘法运算。根据对偶性原理，其逆关系同样成立：频域中的卷积运算对应时域中的乘法运算。这些及其他对偶关系将在第10章和第11章中进行更详细的讨论。

极坐标表示法

如前所述，频域是一组余弦和正弦波的振幅（经过轻微缩放调整）。这被称为**矩形**表示法。或者，频域也可以用**极坐标**形式表示。在这种表示法中，*ReX*[ ] &*Im X*[ ] 被替换为两个其他数组，称为**幅值*X*[]**，其方程形式为：***Mag X*[]**，以及**相位*X*[]**，其方程形式为：***Phase X*[]**。幅值和相位是对实部和虚部的一对一替换。例如，*Mag X*[0] 和*Phase X*[0] 只是使用*ReX*[0] 和*Im X*[0] 计算得出的。同样地，*Mag X*[14] 和*Phase X*[14] 仅使用*ReX*[14] 和*Im X*[14] 进行计算，依此类推。要理解这种转换，可以考虑当你将相同频率的余弦波和正弦波相加时会发生什么。结果是一个相同频率的余弦波

frequency, but with a new amplitude and a new phase shift. In equation form, the two representations are related:

EQUATION 8-5
The addition of a cosine and sine wave results in a cosine wave with a different amplitude and phase shift. The information contained in A & B is transferred to two other variables, M and θ.

$$A\cos(x) + B\sin(x) = M\cos(x + \theta)$$

The important point is that no information is lost in this process; given one representation you can calculate the other. In other words, the information contained in the amplitudes A and B, is also contained in the variables M and θ. Although this equation involves sine and cosine waves, it follows the same conversion equations as do simple vectors. Figure 8-9 shows the analogous vector representation of how the two variables, A and B, can be viewed in a rectangular coordinate system, while M and θ are parameters in polar coordinates.

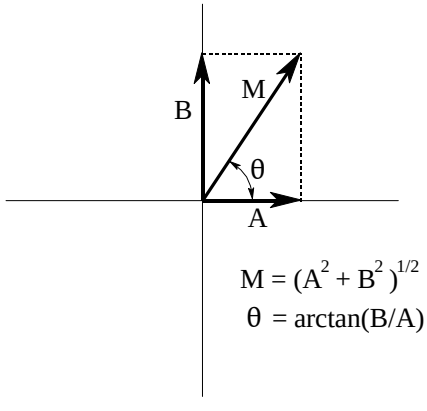


FIGURE 8-9
Rectangular-to-polar conversion. The addition of a cosine wave and a sine wave (of the same frequency) follows the same mathematics as the addition of simple vectors.

In polar notation, $Mag\,X[\ ]$ holds the amplitude of the cosine wave (M in Eq. 8-5 and Fig. 8-9), while $Phase\,X[\ ]$ holds the phase angle of the cosine wave (θ in Eq. 8-5 and Fig. 8-9). The following equations convert the frequency domain from rectangular to polar notation, and vice versa:

EQUATION 8-6
Rectangular-to-polar conversion. The rectangular representation of the frequency domain, $Re\,X[k]$ and $Im\,X[k]$, is changed into the polar form, $Mag\,X[k]$ and $Phase\,X[k]$.

$$Mag\,X[k] = (Re\,X[k]^2 + Im\,X[k]^2)^{1/2}$$

$$Phase\,X[k] = \arctan\left(\frac{Im\,X[k]}{Re\,X[k]}\right)$$

EQUATION 8-7
Polar-to-rectangular conversion. The two arrays, $Mag\,X[k]$ and $Phase\,X[k]$, are converted into $Re\,X[k]$ and $Im\,X[k]$.

$$Re\,X[k] = Mag\,X[k]\ \cos(Phase\,X[k])$$

$$Im\,X[k] = Mag\,X[k]\ \sin(Phase\,X[k])$$

频率不变，但具有新的振幅和相位偏移。在方程形式下，这两种表示方式相关联：

方程8-5
添加余弦和正弦波会产生一个具有不同振幅和相位偏移的余弦波。A和*B*中包含的信息被传递到另外两个变量*M*和θ中。

$$A\cos(x)+B\sin(x)=M\cos(x+\theta)$$

关键在于此过程不会丢失任何信息；给定一个表示形式，即可计算出另一个。换言之，振幅*A*和*B*所包含的信息，同样存在于变量*M*和θ中。尽管该方程涉及正弦和余弦波，但其转换公式与简单向量相同。图8-9展示了两个变量*A*和*B*在直角坐标系中的向量表示，而*M*和θ则是极坐标系中的参数。

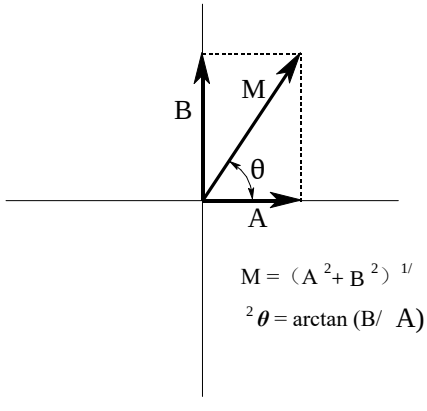


图8-9
矩形波到极坐标波的转换。同频的余弦波与正弦波叠加，其数学原理与简单矢量叠加相同。

在极坐标表示法中，*幅值X*[ ]表示余弦波的振幅（*M*见等式8-5和图8-9），而*相位X*[ ]表示余弦波的相位角（θ见等式8-5和图8-9）。以下方程将频域从直角坐标转换为极坐标表示法，反之亦然：

方程8-6
矩形到极坐标转换。将频域的矩形表示*ReX*[*k*] 和*Im X*[*k*] 转换为极坐标形式*Mag X*[*k*] 和*Phase X*[*k*]。

$$Mag\,X[k] = (ReX[k]^2 + Im\,X[k]^2)^{1/2}$$

$$Phase\,X[k] = \arctan\left(\frac{Im\,X[k]}{Re\,X[k]}\right)$$

方程8-7
极坐标到直角坐标转换。将两个数组*Mag X*[*k*] 和*Phase X*[*k*] 转换为*ReX*[*k*] 和*Im X*[*k*]。

$$ReX[k] = Mag\,X[k]\cos(\text{相位}\,X[k])$$

$$\text{在}X[k] = Mag\,X[k]\sin(Phase\,X[k])$$

Rectangular and polar notation allow you to think of the DFT in two different ways. With rectangular notation, the DFT decomposes an N point signal into $N/2 + 1$ cosine waves and $N/2 + 1$ sine waves, each with a specified *amplitude*. In polar notation, the DFT decomposes an N point signal into $N/2 + 1$ cosine waves, each with a specified *amplitude* (called the *magnitude*) and *phase shift*. Why does polar notation use cosine waves instead of sine waves? Sine waves cannot represent the DC component of a signal, since a sine wave of zero frequency is composed of all zeros (see Figs. 8-5 a&b).

Even though the polar and rectangular representations contain exactly the same information, there are many instances where one is easier to use that the other. For example, Fig. 8-10 shows a frequency domain signal in both rectangular and polar form. Warning: Don't try to understand the shape of the real and imaginary parts; your head will explode! In comparison, the polar curves are straightforward: only frequencies below about 0.25 are present, and the phase shift is approximately proportional to the frequency. This is the frequency response of a low-pass filter.

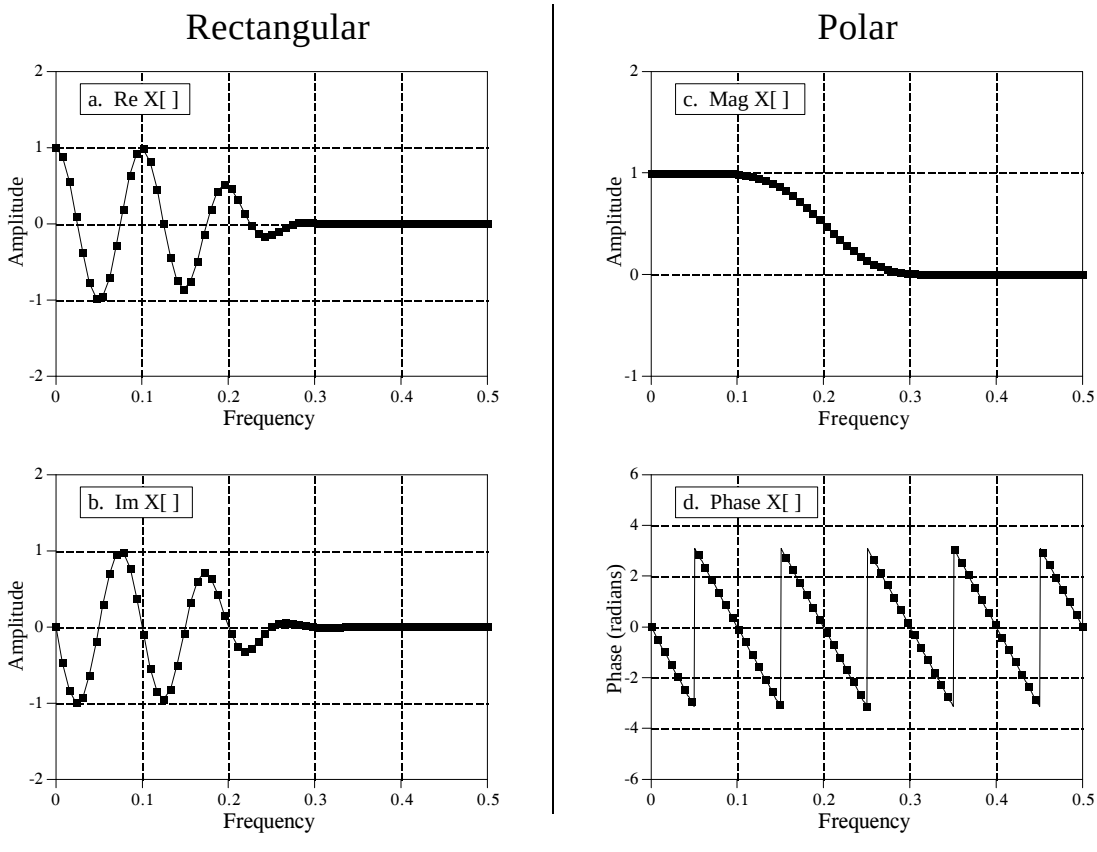


FIGURE 8-10
Example of rectangular and polar frequency domains. This example shows a frequency domain expressed in both rectangular and polar notation. As in this case, polar notation usually provides human observers with a better understanding of the characteristics of the signal. In comparison, the rectangular form is almost always used when math computations are required. Pay special notice to the fact that the first and last samples in the phase must be zero, just as they are in the imaginary part.

矩形表示法和极坐标表示法允许你以两种不同的方式理解DFT。在矩形表示法中，DFT将一个*N*点信号分解为*N*/2+1个余弦波和*N*/2+1个正弦波，每个波都有一个指定的*振幅*。在极坐标表示法中，DFT将一个*N*点信号分解为*N*/2+1个余弦波，每个波都有一个指定的*振幅*（称为*幅值*）和*相位偏移*。为什么极坐标表示法使用余弦波而不是正弦波？正弦波无法表示信号的直流分量，因为零频率的正弦波由所有零点组成（参见图8-5a和b）。

虽然极坐标和直角坐标两种表示法包含完全相同的信息，但在实际应用中，其中一种往往比另一种更便于使用。例如图8-10展示了频域信号的直角坐标与极坐标形式对比。注意：千万别试图理解实部和虚部的形状——这会让人头大！相比之下，极坐标曲线更直观：只有频率低于约0.25的信号成分存在，且相位偏移与频率大致成正比。这就是低通滤波器的频率响应特性。

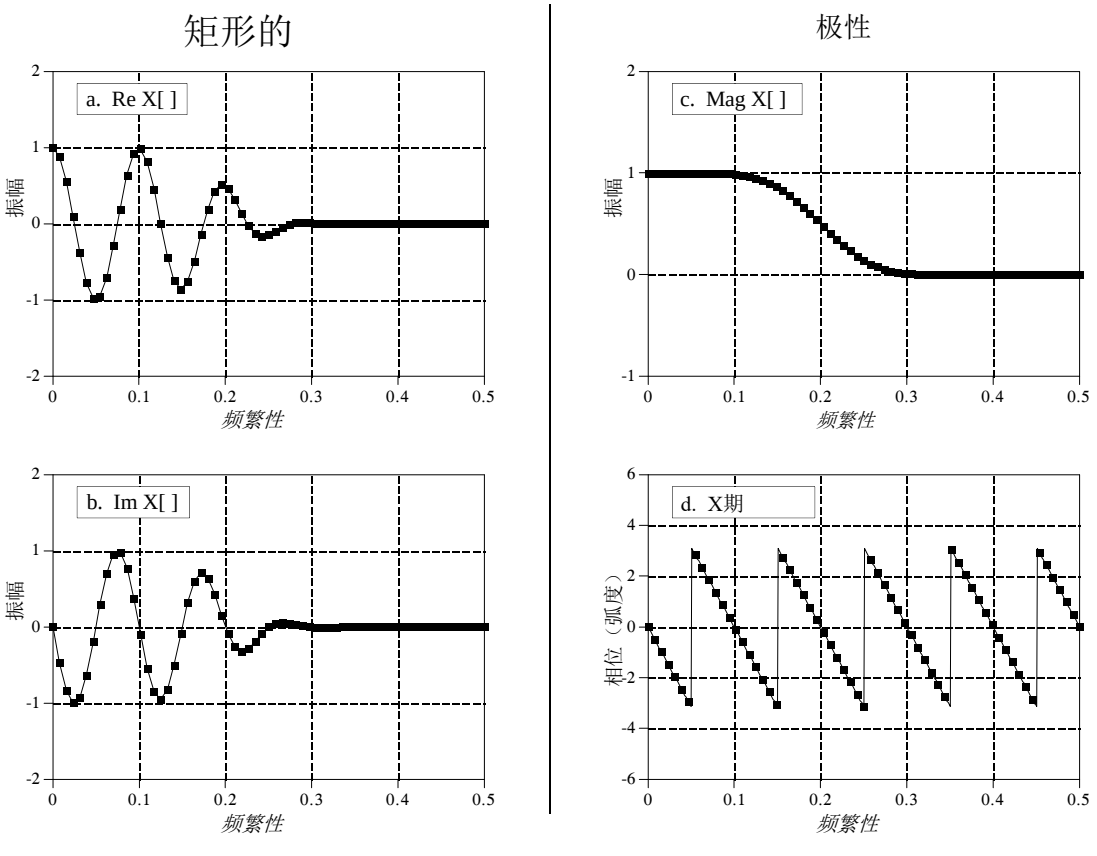


图8-10

矩形坐标与极坐标频域示例。本示例展示了以矩形坐标和极坐标两种形式呈现的频域图。与之相比，极坐标通常能帮助人类观察者更直观地理解信号特性，而矩形坐标则多用于数学计算场景。特别需要注意的是，相位域的首尾样本必须为零，这与虚部的处理方式完全一致。

When should you use rectangular notation and when should you use polar? Rectangular notation is usually the best choice for calculations, such as in equations and computer programs. In comparison, graphs are almost always in polar form. As shown by the previous example, it is nearly impossible for *humans* to understand the characteristics of a frequency domain signal by looking at the real and imaginary parts. In a typical program, the frequency domain signals are kept in rectangular notation until an observer needs to look at them, at which time a rectangular-to-polar conversion is done.

Why is it easier to understand the frequency domain in polar notation? This question goes to the heart of why decomposing a signal into sinusoids is *useful*. Recall the property of *sinusoidal fidelity* from Chapter 5: if a sinusoid enters a linear system, the output will also be a sinusoid, and at exactly the same frequency as the input. Only the amplitude and phase can change. Polar notation directly represents signals in terms of the amplitude and phase of the component cosine waves. In turn, systems can be represented by how they modify the amplitude and phase of each of these cosine waves.

Now consider what happens if rectangular notation is used with this scenario. A mixture of cosine and sine waves enter the linear system, resulting in a mixture of cosine and sine waves leaving the system. The problem is, a cosine wave on the input may result in both cosine and sine waves on the output. Likewise, a sine wave on the input can result in both cosine and sine waves on the output. While these cross-terms can be straightened out, the overall method doesn't match with why we wanted to use sinusoids in the first place.

Polar Nuisances

There are many nuisances associated with using polar notation. None of these are overwhelming, just really annoying! Table 8-3 shows a computer program for converting between rectangular and polar notation, and provides solutions for some of these pests.

Nuisance 1: Radians vs. Degrees
It is possible to express the phase in either *degrees* or *radians*. When expressed in degrees, the values in the phase signal are between -180 and 180. Using radians, each of the values will be between $-\pi$ and π, that is, between -3.141592 to 3.141592. Most computer languages require the use radians for their trigonometric functions, such as cosine, sine, arctangent, etc. It can be irritating to work with these long decimal numbers, and difficult to interpret the data you receive. For example, if you want to introduce a 90 degree phase shift into a signal, you need to add 1.570796 to the phase. While it isn't going to kill you to type this into your program, it does become tiresome. The best way to handle this problem is to define the constant, *P*I = 3.141592, at the beginning of your program. A 90 degree phase shift can then be written as *PI*/2. Degrees and radians are both widely used in DSP and you need to become comfortable with both.

矩形表示法和极坐标表示法各何时适用？在计算场景中，矩形表示法通常是最佳选择，例如在方程和计算机程序中。相比之下，图形几乎总是采用极坐标形式。正如前例所示，*人类*仅通过观察实部和虚部几乎无法理解频域信号的特性。在典型程序中，频域信号会以矩形表示法保存，直到观察者需要查看时，才会进行矩形到极坐标转换。

为何极坐标表示法能更直观地理解频域特性？这个问题揭示了将信号分解为正弦波的*重要价值*。回顾第五章*正弦波保真度*的特性：当正弦波输入线性系统时，输出信号仍为正弦波，且频率与输入完全一致，仅振幅和相位会发生变化。极坐标表示法通过正弦分量的振幅和相位直接描述信号，而系统特性则通过这些正弦分量的振幅和相位变化来体现。

现在让我们看看当采用矩形波表示法处理这种情况时会发生什么。当正弦波与余弦波的混合信号输入线性系统时，输出端也会产生正弦波与余弦波的混合信号。问题在于，输入端的余弦波可能会在输出端同时产生正弦波和余弦波。同理，输入端的正弦波同样可能在输出端同时产生正弦波和余弦波。虽然这些交叉项可以被消除，但整体方法与我们最初选择正弦波的初衷并不相符。

极性烦恼

使用极坐标系存在诸多不便之处，这些不便虽不令人困扰，却确实令人烦恼！表8-3展示了一个用于在直角坐标系与极坐标系之间进行转换的计算机程序，并为其中部分问题提供了解决方案。

干扰1：弧度与度

可以使用*度*或*弧度*来表示相位。当以度表示时，相位信号中的值介于-180和180之间。使用弧度时，每个值将介于π和π之间，即介于-3.141592到3.141592。大多数计算机语言在使用余弦、正弦、反正切等三角函数时，都要求采用弧度制。处理这些冗长的十进制数值既费时费力，又容易导致数据解读困难。举个例子，若要在信号中引入90度相位偏移，就需要将1.570796加到相位值上。虽然在程序中输入这个数值不会致命，但确实会让人抓狂。解决这个问题的最佳方案是在程序开头定义常量π=3.141592，这样90度相位偏移就能表示为π/2。在数字信号处理领域，度和弧度制都广泛应用，必须熟练掌握这两种单位。

```
100 'RECTANGULAR-TO-POLAR & POLAR-TO-RECTANGULAR CONVERSION
110 '
120 DIM REX[256]                    'REX[ ]     holds the real part
130 DIM IMX[256]                    'IMX[ ]     holds the imaginary part
140 DIM MAG[256]                    'MAG[ ]     holds the magnitude
150 DIM PHASE[256]                  'PHASE[ ]   holds the phase
160 '
170 PI = 3.14159265
180 '
190 GOSUB XXXX                      'Mythical subroutine to load data into REX[ ] and IMX[ ]
200 '
210 '
220 '                               'Rectangular-to-polar conversion, Eq. 8-6
230 FOR K% = 0 TO 256
240   MAG[K%] = SQR( REX[K%]^2 + IMX[K%]^2 )          'from Eq. 8-6
250   IF REX[K%] = 0 THEN REX[K%] = 1E-20             'prevent divide by 0 (nuisance 2)
260   PHASE[K%] = ATN( IMX[K%] / REX[K%] )            'from Eq. 8-6
270   '                                               'correct the arctan (nuisance 3)
280   IF REX[K%] < 0 AND IMX[K%] <   0 THEN PHASE[K%] = PHASE[K%]  - PI
290   IF REX[K%] < 0 AND IMX[K%] >= 0 THEN PHASE[K%] = PHASE[K%] + PI
300 NEXT K%
310 '
320 '
330 '                               'Polar-to-rectangular conversion, Eq. 8-7
340 FOR K% = 0 TO 256
350   REX[K%] = MAG[K%] * COS( PHASE[K%] )
360   IMX[K%] = MAG[K%] *  SIN( PHASE[K%] )
370 NEXT K%
380 '
390 END
```

TABLE 8-3

Nuisance 2: Divide by zero error

When converting from rectangular to polar notation, it is very common to find frequencies where the real part is zero and the imaginary part is some nonzero value. This simply means that the phase is exactly 90 or -90 degrees. Try to tell your computer this! When your program tries to calculate the phase from: $Phase\,X[k] = \arctan(Im\,X[k] / Re\,X[k])$, a *divide by zero error* occurs. Even if the program execution doesn't halt, the phase you obtain for this frequency won't be correct. To avoid this problem, the real part must be tested for being zero before the division. If it is zero, the imaginary part must be tested for being positive or negative, to determine whether to set the phase to $\pi/2$ or $-\pi/2$, respectively. Lastly, the division needs to be bypassed. Nothing difficult in all these steps, just the potential for aggravation. An alternative way to handle this problem is shown in line 250 of Table 8-3. If the real part is zero, change it to a negligibly small number to keep the math processor happy during the division.

Nuisance 3: Incorrect arctan

Consider a frequency domain sample where $Re\,X[k] = 1$ and $Im\,X[k] = 1$. Equation 8-6 provides the corresponding polar values of $Mag\,X[k] = 1.414$ and $Phase\,X[k] = 45°$. Now consider another sample where $Re\,X[k] = -1$ and

```
100 矩形到极坐标与极坐标到矩形的转换
110 '
120 DIM REX[256]                    'REX[ ]    取实部
130 DIM IMX[256]                    'IMX[ ]    取虚部
140 DIM MAG[256]                    'MAG[ ]    保持幅度
150 相位差[256]                  phase[ ] 用于存储第160个相位
170 PI = 3.14159265
180 '
190 GOSUB XXXX                      用于将数据加载到REX[ ]和 IMX [ ]200中的特殊子程序
210 '
220 '                               ‘矩形到极坐标转换，等式 8-6
230 K% = 0 至 256
240   MAG[K%] = SQR( REX[K%]^2 + IMX[K%]^2 )          ‘from 等式 8-6
250   若REX[K%] = 0，则REX[K%] = 1E-20                防止除以0（烦人问题2）
260   PHASE[K%] = ATN( IMX[K%] / REX[K%] )             ‘from 等式 8-6
270   '                                                纠正阿克坦（干扰项3）
280   若REX[K%] < 0且IMX[K%] < 0，则PHASE[K%] = PHASE[K%] - PI
290   若REX[K%] < 0且IMX[K%] >= 0，则PHASE[K%] = PHASE[K%] + PI
300 下 一 个  K%
310 ‘
320 '
330 '                               ‘极坐标到直角坐标转换，等式 8-7
340 K% = 0 至 256
350   REX[K%] = MAG[K%] * 硫化羰（相位[K%]）
360   IMX[K%] = MAG[K%] * SIN（PHASE[K%]）
370 下 一 个  K%
380 ‘
390 结束
```

表8-3

干扰2：除以零错误

在从直角坐标转换为极坐标时，经常会遇到实部为零而虚部为非零的频率。这仅仅意味着相位恰好是90度或-90度。试着告诉你的计算机这个情况！当程序尝试计算以下表达式的相位：*相位X*[*k*] =arctan（*ImX*[*k*]/*ReX*[*k*]）时，会出现*除以零错误*。即使程序执行不中断，你得到的相位也不正确。为避免此问题，必须在除法前检测实部是否为零。若实部为零，则需检测虚部是否为正或负，以确定相位应设为$\pi/2$或$-\pi/2$。最后，需要跳过除法运算。这些步骤本身并不复杂，只是存在潜在的麻烦。表8-3第250行展示了处理此问题的另一种方法。若实部为零，应将其调整为可忽略的极小数值，以确保除法运算时数学处理器的稳定运行。

干扰项3：阿替卡韦（arctan）使用错误

考虑一个频域样本，其中*ReX*[*k*] =1且*Im X*[*k*] =1。方程8-6给出了相应的极值：*Mag X*[*k*] =1.414以及*Phase X*[*k*] =45. . 现在考虑另一个样本，其中*ReX*[*k*] = -1且

$Im\,X[k] = -1$. Again, Eq. 8-6 provides the values of $Mag\,X[k] = 1.414$ and $Phase\,X[k] = 45°$. The problem is, the phase is wrong! It should be $-135°$. This error occurs whenever the real part is negative. This problem can be corrected by testing the real and imaginary parts after the phase has been calculated. If both the real and imaginary parts are negative, subtract 180° (or π radians) from the calculated phase. If the real part is negative and the imaginary part is positive, add 180° (or π radians). Lines 340 and 350 of the program in Table 8-3 show how this is done. If you fail to catch this problem, the calculated value of the phase will only run between -π/2 and π/2, rather than between -π and π. Drill this into your mind. If you see the phase only extending to ±1.5708, you have forgotten to correct the ambiguity in the arctangent calculation.

Nuisance 4: Phase of very small magnitudes

Imagine the following scenario. You are grinding away at some DSP task, and suddenly notice that part of the phase doesn't look right. It might be noisy, jumping all over, or just plain *wrong*. After spending the next hour looking through hundreds of lines of computer code, you find the answer. The corresponding values in the magnitude are so small that they are buried in round-off noise. If the magnitude is negligibly small, the phase doesn't have any meaning, and can assume unusual values. An example of this is shown in Fig. 8-11. It is usually obvious when an *amplitude* signal is lost in noise; the values are so small that you are forced to suspect that the values are meaningless. The phase is different. When a polar signal is contaminated with noise, the values in the phase are random numbers between -π and π. Unfortunately, this often *looks* like a real signal, rather than the nonsense it really is.

Nuisance 5: 2π ambiguity of the phase

Look again at Fig. 8-10d, and notice the several discontinuities in the data. Every time a point looks as if it is going to dip below -3.14592, it snaps back to 3.141592. This is a result of the periodic nature of sinusoids. For

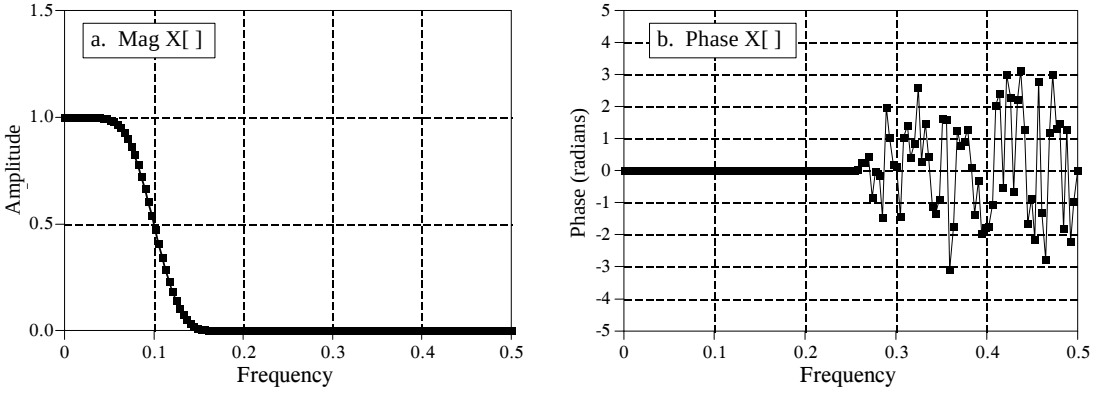


FIGURE 8-11
The phase of small magnitude signals. At frequencies where the magnitude drops to a very low value, round-off noise can cause wild excursions of the phase. Don't make the mistake of thinking this is a meaningful signal.

Im X[*k*] = -1。同样，等式 8-6 提供了*Mag X*[*k*] =1.414和*Phase X*[*k*] =45. . 的值。问题是，相位是错误的！它应该是 -135. . 这个错误发生在实部为负时。这个问题可以通过在相位确定后测试实部和虚部来纠正。
计算。如果实部和虚部都是负数，减去180
（或π弧度）与计算出的相位相差。若实部为负且虚部为正，则加180。（或π弧度）。表8-3中程序的第340行和第350行展示了如何实现这一点。若未能发现此问题，相位的计算值将仅在-π/2至π/2之间运行，而非-π至π之间。请牢记这一点。若仅看到相位延伸至±1.5708，说明你已忘记修正反正切计算中的歧义。

干扰4：极小震级阶段

想象这样一个场景：你正在处理某个数字信号处理（DSP）任务，突然发现某个相位部分存在异常。可能是噪声干扰、跳变波动，或是纯粹的*错误*。在花费一小时翻阅数百行代码后，你终于找到了答案——幅度对应的数值小到被舍入噪声掩盖。当幅度小到可以忽略时，相位就失去了意义，可能呈现异常值。图8-11展示了这种现象。通常当*幅值*信号被噪声淹没时，数值小到令人怀疑其意义，这种情况显而易见。但相位却不同。当极性信号被噪声污染时，相位值会变成介于-π和π之间的随机数。可惜的是，这些数值往往*看起来*像真实信号，而非其本质的无意义数据。

烦扰5：2 π相位的模糊性

再次观察图8-10d，注意数据中的多个不连续点。每当某个点看似即将跌破-3.14592时，便会突然反弹至3.141592。这是正弦波周期性特征所致。

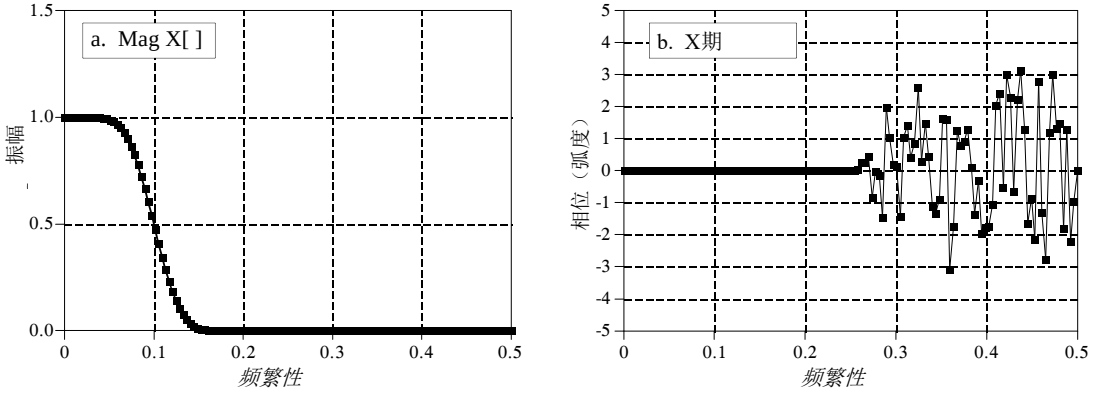


图8-11

小幅度信号的相位。当信号幅度降至极低值时，舍入噪声可能导致相位出现剧烈波动。切勿误以为这是有效信号。

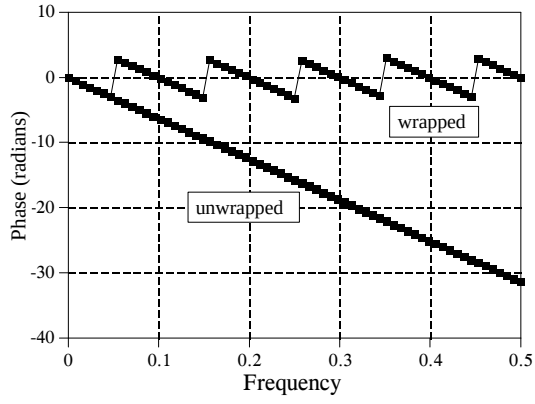


FIGURE 8-12
Example of phase unwrapping. The top curve shows a typical phase signal obtained from a rectangular-to-polar conversion routine. Each value in the signal must be between -π and π (i.e., -3.14159 and 3.14159). As shown in the lower curve, the phase can be *unwrapped* by adding or subtracting integer multiplies of 2π from each sample, where the integer is chosen to minimize the discontinuities between points.

example, a phase shift of q is exactly the same as a phase shift of $q + 2p$, $q + 4p$, $q + 6p$, etc. Any sinusoid is unchanged when you add an integer multiple of 2π to the phase. The apparent discontinuities in the signal are a result of the computer algorithm picking its favorite choice from an infinite number of equivalent possibilities. The smallest possible value is always chosen, keeping the phase between -π and π.

It is often easier to understand the phase if it does not have these discontinuities, even if it means that the phase extends above π, or below -π. This is called **unwrapping the phase**, and an example is shown in Fig. 8-12. As shown by the program in Table 8-4, a multiple of 2π is added or subtracted from each value of the phase. The exact value is determined by an algorithm that minimizes the difference between adjacent samples.

Nuisance 6: The magnitude is always positive (π ambiguity of the phase) Figure 8-13 shows a frequency domain signal in rectangular and polar form. The real part is smooth and quite easy to understand, while the imaginary part is entirely zero. In comparison, the polar signals contain abrupt

```
100 ' PHASE UNWRAPPING
110 '
120 DIM PHASE[256]              'PHASE[ ]   holds the original phase
130 DIM UWPHASE[256]            'UWPHASE[ ] holds the unwrapped phase
140 '
150 PI = 3.14159265
160 '
170 GOSUB XXXX                  'Mythical subroutine to load data into PHASE[ ]
180 '
190 UWPHASE[0] = 0              'The first point of all phase signals is zero
200 '
210 '                           'Go through the unwrapping algorithm
220 FOR K% = 1 TO 256
230   C% = CINT( (UWPHASE[K%-1] - PHASE[K%]) / (2 * PI) )
240   UWPHASE[K%] = PHASE[K%] + C%*2*PI
250 NEXT K%
260 '
270 END
```

TABLE 8-4

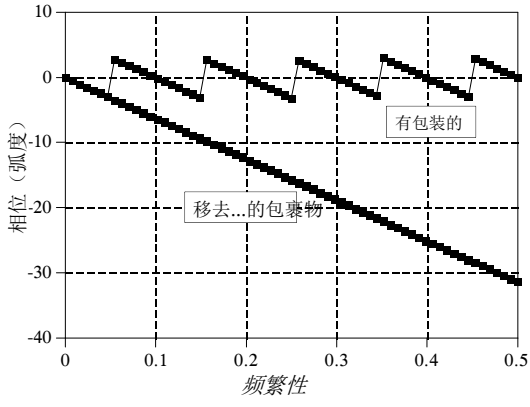


图8-12
相位展开的示例。顶部曲线显示了从矩形到极坐标转换例程中获得的典型相位信号。信号中的每个值必须介于-π和π之间（即-3.14159和3.14159）。如下曲线所示，通过从每个采样值中加减2π的整数倍，可以*展开*相位，其中选择的整数用于最小化点之间的不连续性。

例如，θ的相位偏移与θ+2π、θ+4π、θ+6π等等完全相同。当向相位添加2π的整数倍时，任何正弦波都不会改变。信号中出现的明显不连续性是计算机算法从无限多等效可能性中选择其最优方案的结果。系统始终选择最小可能值，使相位保持在-π到π之间。

如果相位没有这些不连续性，通常更容易理解，即使这意味着相位延伸到π之上或低于π之下。这被称为**解缠相位**，图8-12展示了示例。如表8-4中的程序所示，每个相位值都会加上或减去2的倍数π。具体数值由一个最小化相邻样本间差异的算法确定。

噪声6：幅值始终为正（相位π模糊）图8-13展示了频域信号的矩形图和极坐标图。实部平滑且易于理解，而虚部则完全为零。相比之下，极坐标信号包含突变

```
100  ‘解包阶段 110’
120 相位差[256]                phase[ ] 用于保存原始相位
130 双相超声波处理[256]          uwphase[ ] 用于存储未折叠的相位 140
150 PI = 3.14159265
160 '
170 GOSUB XXXX                  将数据加载至相位[ ] 180的神话式子程序
190 UWPHASE[0] = 0              所有相位信号的首点均为零点200
210 '                           执行解包算法
220 FOR K% = 1 TO 256
230   C% = CINT( (UWPHASE[K%-1] - PHASE[K%]) / (2 * PI) )
240   UWPHASE[K%] = PHASE[K%] + C%*2*PI
250 下 一 个  K%
260 ‘
270 结束
```

表8-4

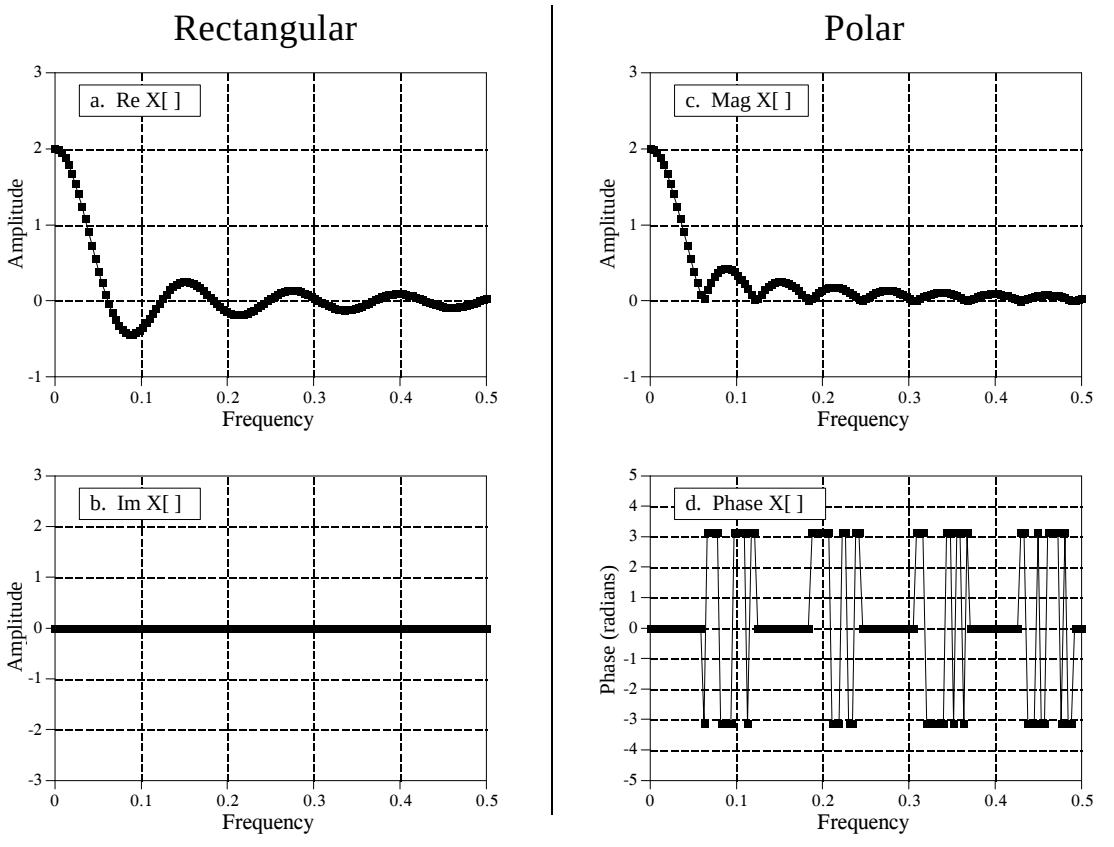


FIGURE 8-13
Example signals in rectangular and polar form. Since the magnitude must always be positive (by definition), the magnitude and phase may contain abrupt discontinuities and sharp corners. Figure (d) also shows another nuisance: random noise can cause the phase to rapidly oscillate between π or $-\pi$.

discontinuities and sharp corners. This is because the magnitude must always be positive, *by definition*. Whenever the real part dips below zero, the magnitude remains positive by changing the phase by π (or $-\pi$, which is the same thing). While this is not a problem for the mathematics, the irregular curves can be difficult to interpret.

One solution is to allow the magnitude to have *negative* values. In the example of Fig. 8-13, this would make the magnitude appear the same as the real part, while the phase would be entirely zero. There is nothing wrong with this if it helps your understanding. Just be careful not to call a signal with negative values the "magnitude" since this violates its formal definition. In this book we use the weasel words: *unwrapped magnitude* to indicate a "magnitude" that is allowed to have negative values.

Nuisance 7: Spikes between π and $-\pi$

Since π and $-\pi$ represent the same phase shift, round-off noise can cause adjacent points in the phase to rapidly switch between the two values. As shown in Fig. 8-13d, this can produce sharp breaks and spikes in an otherwise smooth curve. Don't be fooled, the phase isn't really this discontinuous.

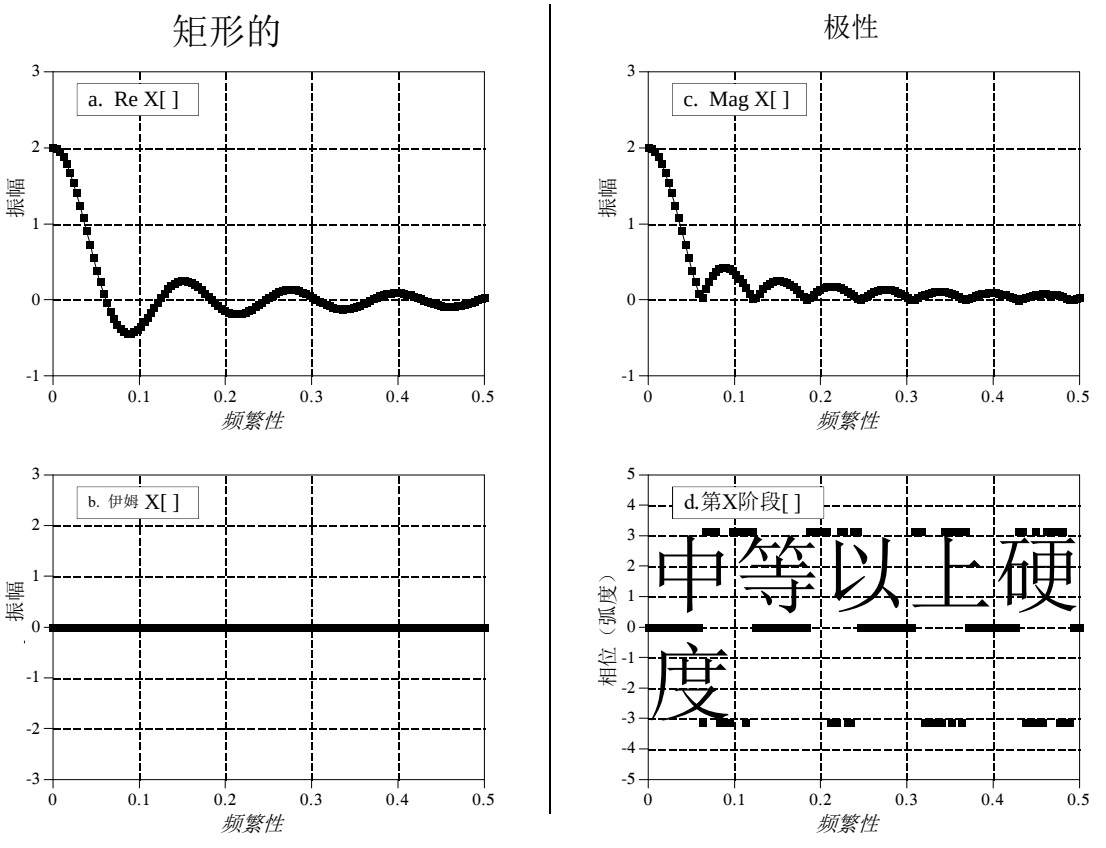


图8-13
矩形和极坐标形式的示例信号。由于幅值必须始终为正（根据定义），幅值和相位可能包含突变的不连续性和尖锐的拐角。图(d)还显示了另一个麻烦：随机噪声可能导致相位在π或$-\pi$之间快速振荡。

不连续性和尖角。这是因为幅值必须始终为正，*由定义*。当实部低于零时，通过改变相位π（或$-\pi$，这是同一件事）使幅值保持为正。虽然这对数学来说不是问题，但不规则的曲线可能难以解释。

一种解决方案是允许幅值取*负值*。以图8-13为例，这样会使幅值看起来与实部相同，而相位则完全为零。如果这有助于理解，这样做并无不妥。但需注意，切勿将负值信号称为“幅值”，因为这违背了其正式定义。本书采用模糊表述：用*未折返幅值*来指代允许负值的“幅值”。

干扰7：π与π之间的尖峰

由于π和π代表相同的相位偏移，舍入噪声会导致相邻相位点在两个值之间快速切换。如图8-13d所示，这会在原本平滑的曲线中产生尖锐的断点和尖峰。别被误导了，相位实际上并不那么不连续。