reproduce the original signal at a different location or time. In speech communication, the reproduced sound usually allows some acceptable level of distortion to achieve low bit rate. The goal of source coding is to reduce the number of bits necessary to transmit or store data, subject to a distortion or fidelity criterion, or equivalently, to achieve the minimum possible distortion for a prescribed bit rate. *Vector quantization* (VQ) is one of the most efficient source-coding techniques.

Quantization is the process of approximating continuous amplitude signals by discrete symbols. The quantization of a single signal value or parameter is referred to as scalar quantization. In contrast, joint quantization of multiple signal values or parameters is referred to as vector quantization. Conventional pattern recognition techniques have been used effectively to solve the quantization or data compression problem with successful application to speech coding, image coding, and speech recognition [36, 85]. In both speech recognition and synthesis systems, vector quantization serves an important role in many aspects of the systems, ranging from discrete acoustic prototypes of speech signals for the discrete HMM, to robust signal processing and data compression.

A vector quantizer is described by a codebook, which is a set of fixed *prototype vectors* or reproduction vectors. Each of these prototype vectors is also referred to as a codeword. To perform the quantization process, the input vector is matched against each codeword in the codebook using some *distortion measure*. The input vector is then replaced by the index of the codeword with the smallest distortion. Therefore, a description of the vector quantization process includes:

1. the distortion measure;
2. the generation of each codeword in the codebook.

4.4.1.1. Distortion Measures

Since vectors are replaced by the index of the codeword with smallest distortion, the transmitted data can be recovered only by replacing the code index sequence with the corresponding codeword sequence. This inevitably causes distortion between the original data and the transmitted data. How to minimize the distortion is thus the central goal of vector quantization. This section describes a couple of the most common distortion measures.

Assume that $\underline{\mathbf{x}} = (x_1, x_2, \ldots, x_d)^t \in R^d$ is a d-dimensional vector whose components $\{x_k, 1 \le k \le d\}$ are real-valued, continuous-amplitude random variables. After vector quantization, the vector $\mathbf{x}$ is mapped (quantized) to another discrete-amplitude d-dimensional vector $\mathbf{z}$.

$$\mathbf{z} = q(\mathbf{x}) \tag{4.74}$$

In Eq. (4.74) $q()$ is the quantization operator. Typically, $\mathbf{z}$ is a vector from a finite set $\mathbf{Z} = \{\mathbf{z}_j, 1 \le j \le M\}$, where $\mathbf{z}_j$ is also a d-dimensional vector. The set $\mathbf{Z}$ is referred to as the codebook, M is the size of the codebook, and $\mathbf{z}_j$ is j^{th} *codeword*. The size M of the codebook is also called the number of partitions (or levels) in the codebook. To design a codebook, the

d-dimensional space of the original random vector $\mathbf{x}$ can be partitioned into M regions or cells $\{C_i, 1 \le i \le M\}$, and each cell C_i is associated with a codeword vector $\mathbf{z}_i$. VQ then maps (quantizes) the input vector $\mathbf{x}$ to codeword $\mathbf{z}_i$ if $\mathbf{x}$ lies in C_i. That is,

$$q(\mathbf{x}) = \mathbf{z}_i \text{ if } \mathbf{x} \in C_i \tag{4.75}$$

An example of partitioning of a two-dimensional space ($d = 2$) for the purpose of vector quantization is shown in Figure 4.12. The shaded region enclosed by the dashed lines is the cell C_i. Any input vector $\mathbf{x}$ that lies in the cell C_i is quantized as $\mathbf{z}_i$. The shapes of the various cells can be different. The positions of the codewords within each cell are shown by dots in Figure 4.12. The codeword $\mathbf{z}_i$ is also referred to as the *centroid* of the cell C_i because it can be viewed as the central point of the cell C_i.

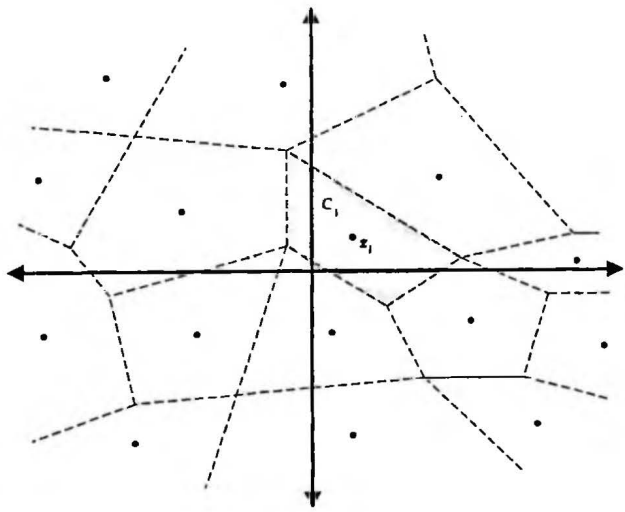


Figure 4.12 Partitioning of a two-dimensional space into 16 cells.

When $\mathbf{x}$ is quantized as $\mathbf{z}$, a quantization error results. A distortion measure $d(\mathbf{x}, \mathbf{z})$ can be defined between $\mathbf{x}$ and $\mathbf{z}$ to measure the quantization quality. Using this distortion measure, Eq. (4.75) can be reformulated as follows:

$$q(\mathbf{x}) = \mathbf{z}_i \text{ if and only if } i = \underset{k}{\operatorname{argmin}}\, d(\mathbf{x}, \mathbf{z}_k) \tag{4.76}$$

The distortion measure between $\mathbf{x}$ and $\mathbf{z}$ is also known as a distance measure in the speech context. The measure must be tractable in order to be computed and analyzed, and also must be subjectively relevant so that differences in distortion values can be used to indicate differences in original and transmitted signals. The most commonly used measure is the Euclidean distortion measure, which assumes that the distortions contributed by quantiz-

ing the different parameters are equal. Therefore, the distortion measure $d(\mathbf{x}, \mathbf{z})$ can be defined as follows:

$$d(\mathbf{x},\mathbf{z}) = (\mathbf{x}-\mathbf{z})'(\mathbf{x}-\mathbf{z}) = \sum_{i=1}^{d}(x_i - z_i)^2 \tag{4.77}$$

The distortion defined in Eq. (4.77) is also known as sum of squared error. In general, unequal weights can be introduced to weight certain contributions to the distortion more than others. One choice for weights that is popular in many practical applications is to use the inverse of the covariance matrix of $\mathbf{z}$.

$$d(\mathbf{x},\mathbf{z}) = (\mathbf{x}-\mathbf{z})'\Sigma^{-1}(\mathbf{x}-\mathbf{z}) \tag{4.78}$$

This distortion measure, known as the *Mahalanobis* distance, is actually the exponential term in a Gaussian density function.

Another way to weight the contributions to the distortion measure is to use *perceptually*-based distortion measures. Such distortion measures take advantage of subjective judgments of perceptual difference caused by two different signals. A perceptually-based distortion measure has the property that signal changes that make the sounds being perceived different should be associated with large distances. Similarly signal changes that keep the sound perceived the same should be associated with small distances. A number of perceptually based distortion measures have been used in speech coding [3, 75, 76].

4.4.1.2. The *K*-Means Algorithm

To design an M-level codebook, it is necessary to partition d-dimensional space into M cells and associate a quantized vector with each cell. Based on the source-coding principle, the criterion for optimization of the vector quantizer is to minimize overall average distortion over all M-levels of the VQ. The overall average distortion can be defined by

$$\begin{aligned} D &= E[d(\mathbf{x},\mathbf{z})] = \sum_{i=1}^{M} p(\mathbf{x}\in C_i)E[d(\mathbf{x},\mathbf{z}_i)|\,\mathbf{x}\in C_i] \\ &= \sum_{i=1}^{M} p(\mathbf{x}\in C_i)\int_{x\in C_i} d(\mathbf{x},\mathbf{z}_i)p(\mathbf{x}|\,\mathbf{x}\in C_i)d\mathbf{x} = \sum_{i=1}^{M} D_i \end{aligned} \tag{4.79}$$

where the integral is taken over all components of vector $\mathbf{x}$; $p(\mathbf{x}\in C_i)$ denotes the prior probability of codeword $\mathbf{z}_i$; $p(\mathbf{x}|\,\mathbf{x}\in C_i)$ denotes the multidimensional probability density function of $\mathbf{x}$ in cell C_i; and D_i is the average distortion in cell C_i. No analytic solution exists to guarantee global minimization of the average distortion measure for a given set of speech data. However, an iterative algorithm, which guarantees a local minimum, exists and works well in practice.

We say a quantizer is optimal if the overall average distortion is minimized over all M-levels of the quantizer. There are two necessary conditions for optimality. The first is that the optimal quantizer is realized by using a nearest-neighbor selection rule as specified by Eq. (4.76). Note that the average distortion for each cell C_i

$$E\left[d(\mathbf{x},\mathbf{z}_i)\,|\,\mathbf{x}\in C_i\right] \tag{4.80}$$

can be minimized when $\mathbf{z}_i$ is selected such that $d(\mathbf{x},\mathbf{z}_i)$ is minimized for $\mathbf{x}$. This means that the quantizer must choose the codeword that results in the minimum distortion with respect to $\mathbf{x}$. The second condition for optimality is that each codeword $\mathbf{z}_i$ is chosen to minimize the average distortion in cell C_i. That is, $\mathbf{z}_i$ is the vector that minimizes

$$D_i = p(\mathbf{z}_i)E\left[d(\mathbf{x},\mathbf{z})|\mathbf{x}\in C_i\right] \tag{4.81}$$

Since the overall average distortion D is a linear combination of average distortions in C_i, they can be independently computed after classification of $\mathbf{x}$. The vector $\mathbf{z}_i$ is called the centroid of the cell C_i and is written

$$\mathbf{z}_i = \text{cent}(C_i) \tag{4.82}$$

The centroid for a particular region (cell) depends on the definition of the distortion measure. In practice, given a set of training vectors $\{\mathbf{x}_t, 1\le t\le T\}$, a subset of K_i vectors will be located in cell C_i. In this case, $p(\mathbf{x}\,|\,\mathbf{z}_i)$ can be assumed to be $1/K_i$, and $p(\mathbf{z}_i)$ becomes K_i/T. The average distortion D_i in cell C_i can then be given by

$$D_i = \frac{1}{T}\sum_{\mathbf{x}\in C_i} d(\mathbf{x},\mathbf{z}_i) \tag{4.83}$$

The second condition for optimality can then be rewritten as follows:

$$\hat{\mathbf{z}}_i = \arg\min_{\mathbf{z}_i} D_i(\mathbf{z}_i) = \arg\min_{\mathbf{z}_i}\frac{1}{T}\sum_{\mathbf{x}\in C_i} d(\mathbf{x},\mathbf{z}_i) \tag{4.84}$$

When the sum of squared error in Eq. (4.77) is used for the distortion measure, the attempt to find such $\hat{\mathbf{z}}_i$ to minimize the sum of squared error is equivalent to least squared error estimation, which was described in Chapter 3. Minimization of D_i in Eq. (4.84) with respect to $\mathbf{z}_i$ is given by setting the derivative of D_i to zero:

$$\begin{aligned}\nabla_{\mathbf{z}_i} D_i &= \nabla_{\mathbf{z}_i}\frac{1}{T}\sum_{\mathbf{x}\in C_i}(\mathbf{x}-\mathbf{z}_i)'(\mathbf{x}-\mathbf{z}_i)\\ &= \frac{1}{T}\sum_{\mathbf{x}\in C_i}\nabla_{\mathbf{z}_i}(\mathbf{x}-\mathbf{z}_i)'(\mathbf{x}-\mathbf{z}_i)\\ &= \frac{-2}{T}\sum_{\mathbf{x}\in C_i}(\mathbf{x}-\mathbf{z}_i) = 0\end{aligned} \tag{4.85}$$

By solving Eq. (4.85), we obtain the least square error estimate of centroid $\hat{\mathbf{z}}_i$ simply as the sample mean of all the training vectors $\mathbf{x}$, quantized to cell C_i:

$$\hat{\mathbf{z}}_i = \frac{1}{K_i} \sum_{\mathbf{x} \in C_i} \mathbf{x} \tag{4.86}$$

If the Mahalanobis distance measure [Eq. (4.78)] is used, minimization of D_i in Eq. (4.84) can be done similarly:

$$\begin{aligned} \nabla_{\mathbf{z}_i} D_i &= \nabla_{\mathbf{z}_i} \frac{1}{T} \sum_{\mathbf{x} \in C_i} (\mathbf{x} - \mathbf{z}_i)^t \Sigma^{-1} (\mathbf{x} - \mathbf{z}_i) \\ &= \frac{1}{T} \sum_{\mathbf{x} \in C_i} \nabla_{\mathbf{z}_i} (\mathbf{x} - \mathbf{z}_i)^t \Sigma^{-1} (\mathbf{x} - \mathbf{z}_i) \\ &= \frac{-2}{T} \sum_{\mathbf{x} \in C_i} \Sigma^{-1} (\mathbf{x} - \mathbf{z}_i) = 0 \end{aligned} \tag{4.87}$$

and centroid $\hat{\mathbf{z}}_i$ is obtained from

$$\hat{\mathbf{z}}_i = \frac{1}{K_i} \sum_{\mathbf{x} \in C_i} \mathbf{x} \tag{4.88}$$

One can see that $\hat{\mathbf{z}}_i$ is again the sample mean of all the training vectors $\mathbf{x}$, quantized to cell C_i. Although Eq. (4.88) is obtained based on the Mahalanobis distance measure, it also works with a large class of Euclidean-like distortion measures [61]. Since the Mahalanobis distance measure is actually the exponential term in a Gaussian density, minimization of the distance criterion can be easily translated into maximization of the logarithm of the Gaussian likelihood. Therefore, similar to the relationship between least square error estimation for the linear discrimination function and the Gaussian classifier described in Section 4.3.3.1, the distance minimization process (least square error estimation) above is in fact a *maximum likelihood estimation*.

According to these two conditions for VQ optimality, one can iteratively apply the nearest-neighbor selection rule and Eq. (4.88) to get the new centroid $\hat{\mathbf{z}}_i$ for each cell in order to minimize the average distortion measure. This procedure is known as the *k-means* algorithm or the *generalized Lloyd* algorithm [29, 34, 56]. In the *k*-means algorithm, the basic idea is to partition the set of training vectors into M clusters C_i $(1 \le i \le M)$ in such a way that the two necessary conditions for optimality described above are satisfied. The *k*-means algorithm can be described as in Algorithm 4.2.

ALGORITHM 4.2: *THE K-MEANS ALGORITHM*

Step 1: Initialization: Choose some adequate method to derive initial VQ codewords ($\mathbf{z}_i$, $1 \le i \le M$) in the codebook.

Step 2: Nearest-neighbor Classification: Classify each training vector $\{\mathbf{x}_k\}$ into one of the cells C_i by choosing the closest codeword $\mathbf{z}_i$ ($\mathbf{x} \in C_i$, i.f.f. $d(\mathbf{x},\mathbf{z}_i) \le d(\mathbf{x},\mathbf{z}_j)$ for all $j \ne i$). This classification is also called minimum-distance classifier.

Step 3: Codebook Updating: Update the codeword of every cell by computing the centroid of the training vectors in each cell according to Eq. (4.84) ($\hat{\mathbf{z}}_i = cent(C_i)$, $1 \le i \le M$).

Step 4: Iteration: Repeat steps 2 and 3 until the ratio of the new overall distortion D at the current iteration relative to the overall distortion at the previous iteration is above a preset threshold.

In the process of minimizing the average distortion measure, the k-means procedure actually breaks the minimization process into two steps. Assuming that the centroid $\mathbf{z}_i$ (or mean) for each cell C_i has been found, then the minimization process is found simply by partitioning all the training vectors into their corresponding cells according to the distortion measure. After all of the new partitions are obtained, the minimization process involves finding the new centroid within each cell to minimize its corresponding within-cell average distortion D_i based on Eq. (4.84). By iterating over these two steps, a new overall distortion D smaller than that of the previous step can be obtained.

Theoretically, the k-means algorithm can converge only to a local optimum [56]. Furthermore, any such solution is, in general, not unique [33]. Initialization is often critical to the quality of the eventual converged codebook. Global optimality may be approximated by repeating the k-means algorithm for several sets of codebook initialization values, and then one can choose the codebook that produces the minimum overall distortion. In the next subsection we will describe methods for finding a decent initial codebook.

4.4.1.3. The LBG Algorithm

Since the initial codebook is critical to the ultimate quality of the final codebook, it has been shown that it is advantageous to design an M-vector codebook in stages. This extended k-means algorithm is known as the LBG algorithm proposed by Linde, Buzo, and Gray [56]. The LBG algorithm first computes a 1-vector codebook, then uses a splitting algorithm on the codewords to obtain the initial 2-vector codebook, and continues the splitting process until the desired M-vector codebook is obtained. The procedure is formally implemented by Algorithm 4.3.

ALGORITHM 4.3: *THE LBG ALGORITHM*

Step 1: Initialization: Set M (number of partitions or cells) =1. Find the centroid of all the training data according to Eq. (4.84).
Step 2: Splitting: Split M into $2M$ partitions by splitting each current codeword by finding two points that are far apart in each partition using a heuristic method, and use these two points as the new centroids for the new $2M$ codebook. Now set $M = 2M$.
Step 3: K-means Stage: Now use the k-means iterative algorithm described in the previous section to reach the best set of centroids for the new codebook.
Step 4: Termination: If M equals the VQ codebook size required, STOP; otherwise go to Step 2.

4.4.2. The EM Algorithm

We introduce the EM algorithm that is important to hidden Markov models and other learning techniques. It discovers model parameters by maximizing the log-likelihood of incomplete data and by iteratively maximizing the expectation of log-likelihood from complete data. The EM algorithm is a generalization of the VQ algorithm described above.

The EM algorithm can also be viewed as a generalization of the MLE method, when the data observed is incomplete. Without loss of generality, we use scalar random variables here to describe the EM algorithm. Suppose we observe training data y. In order to determine the parameter vector Φ that maximizes $P(Y=y|\Phi)$, we would need to know some hidden data x (that is unobserved). For example, x may be a hidden number that refers to component densities of observable data y, or x may be the underlying hidden state sequence in *hidden Markov models* (as discussed in Chapter 8). Without knowing this hidden data x, we could not easily use the maximum likelihood estimation to estimate $\hat{\Phi}$, which maximizes $P(Y=y|\Phi)$. Instead, we assume a parameter vector Φ and estimate the probability that each x occurred in the generation of y. This way we can pretend that we had in fact observed a complete data pair (x, y), with frequency proportional to the probability $P(X=x,Y=y|\Phi)$, to compute a new $\bar{\Phi}$, the maximum likelihood estimate of Φ. We can then set the parameter vector Φ to be this new $\bar{\Phi}$ and repeat the process to iteratively improve our estimate.

The issue now is whether or not the process (EM algorithm) described above converges. Without loss of generality, we assume that both random variables X (unobserved) and Y (observed) are discrete random variables. According to Bayes' rule,

$$P(X=x,Y=y|\bar{\Phi}) = P(X=x|Y=y,\bar{\Phi})P(Y=y|\bar{\Phi}) \tag{4.89}$$

Our goal is to maximize the log-likelihood of the observable, real data y generated by parameter vector $\bar{\Phi}$. Based on Eq. (4.89), the log-likelihood can be expressed as follows:

$$\log P(Y=y|\bar{\Phi}) = \log P(X=x,Y=y|\bar{\Phi}) - \log P(X=x|Y=y,\bar{\Phi}) \tag{4.90}$$

Now, we take the conditional expectation of $\log P(Y=y|\bar{\Phi})$ over X computed with parameter vector Φ:

$$E_{\Phi}[\log P(Y=y|\bar{\Phi})]_{X|Y=y} = \sum_{x}\left(P(X=x|Y=y,\Phi)\log P(Y=y|\bar{\Phi})\right)$$
$$= \log P(Y=y|\bar{\Phi}) \tag{4.91}$$

where we denote $E_{\Phi}[f]_{X|Y=y}$ as the expectation of function f over X computed with parameter vector Φ. Then using Eq. (4.90) and (4.91), the following expression is obtained:

$$\log P(Y=y|\bar{\Phi}) = E_{\Phi}[\log P(X,Y=y|\bar{\Phi})]_{X|Y=y} - E_{\Phi}[\log P(X|Y=y,\bar{\Phi})]_{X|Y=y}$$
$$= Q(\Phi,\bar{\Phi}) - H(\Phi,\bar{\Phi}) \tag{4.92}$$

where

$$Q(\Phi,\bar{\Phi}) = E_{\Phi}[\log P(X,Y=y|\bar{\Phi})]_{X|Y=y}$$
$$= \sum_{x}\left(P(X=x|Y=y,\Phi)\log P(X=x,Y=y|\bar{\Phi})\right) \tag{4.93}$$

and

$$H(\Phi,\bar{\Phi}) = E_{\Phi}[\log P(X|Y=y,\bar{\Phi})]_{X|Y=y}$$
$$= \sum_{x}\left(P(X=x|Y=y,\bar{\Phi})\log P(X=x|Y=y,\bar{\Phi})\right) \tag{4.94}$$

The convergence of the EM algorithm lies in the fact that if we choose $\bar{\Phi}$ so that

$$Q(\Phi,\bar{\Phi}) \geq Q(\Phi,\Phi) \tag{4.95}$$

then

$$\log P(Y=y|\bar{\Phi}) \geq \log P(Y=y|\Phi) \tag{4.96}$$

since it follows from Jensen's inequality that $H(\Phi,\bar{\Phi}) \leq H(\Phi,\Phi)$ [21]. The function $Q(\Phi,\bar{\Phi})$ is known as the Q-function or auxiliary function. This fact implies that we can maximize the Q-function, which is the expectation of log-likelihood from complete data pair (x, y), to update parameter vector from Φ to $\bar{\Phi}$, so that the incomplete log-likelihood $L(x,\Phi)$ increases monotonically. Eventually, the likelihood will converge to a local maximum if we iterate the process.

The name of the EM algorithm comes from E for expectation and M for maximization. The implementation of the EM algorithm includes the E (expectation) step, which calculates the auxiliary Q-function $Q(\Phi,\bar{\Phi})$ and the M (maximization) step, which maximizes $Q(\Phi,\bar{\Phi})$ over $\bar{\Phi}$ to obtain $\hat{\Phi}$. The general EM algorithm can be described in the following way.

ALGORITHM 4.4: *THE EM ALGORITHM*

Step 1: Initialization: Choose an initial estimate Φ.
Step 2: E-Step: Compute auxiliary Q-function $Q(\Phi, \bar{\Phi})$ (which is also the expectation of log-likelihood from complete data) based on Φ.
Step 3: M-Step: Compute $\hat{\Phi} = \arg\max_{\Phi} Q(\Phi, \bar{\Phi})$ to maximize the auxiliary Q-function.
Step 4: Iteration: Set $\Phi = \hat{\Phi}$, repeat from Step 2 until convergence.

The M-step of the EM algorithm is actually a maximum likelihood estimation of complete data (assuming we know the unobserved data x based on observed data y and initial parameter vector Φ). The EM algorithm is usually used in applications where no analytic solution exists for maximization of log-likelihood of incomplete data. Instead, the Q-function is iteratively maximized to obtain the estimation of parameter vector.

4.4.3. Multivariate Gaussian Mixture Density Estimation

The vector quantization process described in Section 4.4.1 partitions the data space into separate regions based on some distance measure regardless of the probability distributions of the original data. This process may introduce errors in partitions that could potentially destroy the original structure of data. An alternative way for modeling a VQ codebook is to use a family of Gaussian probability density functions, such that each cell will be represented by a (Gaussian) probability density function as shown in Figure 4.13. These probability density functions can then overlap, rather than partition, in order to represent the entire data space. The objective for a mixture Gaussian VQ is to maximize the likelihood of the observed data (represented by the product of the Gaussian mixture scores) instead of minimizing the overall distortion. The centroid of each cell (the mean vectors of each Gaussian pdf) obtained via such a representation may be quite different from that obtained using the traditional k-means algorithm, since the distribution properties of the data are taken into account.

There should be an obvious analogy between the EM algorithm and the k-means algorithm described in the Section 4.4.1.2. In the k-means algorithm, the class information for the observed data samples is hidden and unobserved, so an EM-like algorithm instead of maximum likelihood estimate needs to be used. Therefore, instead of a single process of maximum likelihood estimation, the k-means algorithm first uses the old codebook to find the nearest neighbor for each data sample followed by maximum likelihood estimation of the new codebook and iterates the process until the distortion stabilizes. The steps 2 and 3 in the k-means algorithm are actually the E and M steps in the EM algorithm respectively.

Mixture density estimation [41] is a typical example of EM estimation. In the mixtures of Gaussian density, the probability density for observable data $\mathbf{y}$ is the weighted sum of each Gaussian component:

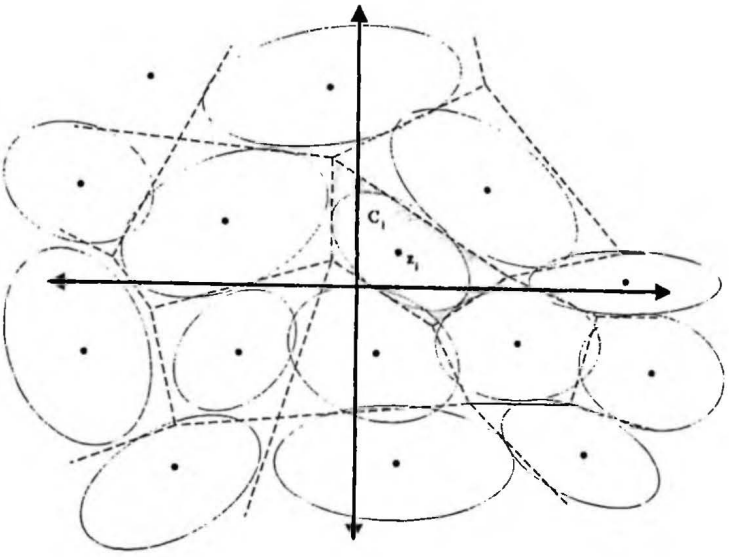


Figure 4.13 Partitioning of a two-dimensional space with 16 Gaussian density functions.

$$p(\mathbf{y} \mid \Phi) = \sum_{k=1}^{K} c_k p_k(\mathbf{y} \mid \Phi_k) = \sum_{k=1}^{K} c_k N_k(\mathbf{y} \mid \mu_k, \Sigma_k) \tag{4.97}$$

where $0 \le c_k \le 1$, for $1 \le k \le K$ and $\sum_{k=1}^{K} c_k = 1$.

Unlike the case of a single Gaussian estimation, we also need to estimate the mixture weight c_k. In order to do so, we can assume that observable data $\mathbf{y}$ come from one of the component densities $p_X(\mathbf{y} \mid \Phi_X)$, where X is a random variable taking value from $\{1, 2, \ldots K\}$ to indicate the Gaussian component. It is clear that x is unobserved and used to specify the pdf component Φ_X. Assuming that the probability density function for complete data $(x,\mathbf{y})$ is given by the joint probability:

$$p(\mathbf{y}, x \mid \Phi) = P(X = x) p_x(\mathbf{y} \mid \Phi_x) = P(X = x) N_x(\mathbf{y} \mid \mu_x, \Sigma_x) \tag{4.98}$$

$P(X = x)$ can be regarded as the probability of the unobserved data x used to specify the component density $p_x(\mathbf{y} \mid \Phi_x)$ from which the observed data $\mathbf{y}$ is drawn. If we assume the number of components is K and Φ is the vector of all probability parameters $(P(X), \Phi_1, \Phi_2, \ldots, \Phi_K)$, the probability density function of incomplete (observed) data $\mathbf{y}$ can be specified as the following marginal probability:

$$p(\mathbf{y} \mid \Phi) = \sum_{x} p(\mathbf{y}, x \mid \Phi) = \sum_{x} P(X = x) p_x(\mathbf{y} \mid \Phi_x) \tag{4.99}$$

By comparing Eq. (4.97) and (4.99), we can see that the mixture weight is represented as the probability function $P(X = x)$. That is,

$$c_k = P(X = k) \tag{4.100}$$

According to the EM algorithm, the maximization of the logarithm of the likelihood function $\log p(\mathbf{y} \mid \mathbf{\Phi})$ can be performed by iteratively maximizing the conditional expectation of the logarithm of Eq. (4.98), i.e., $\log p(\mathbf{y}, x \mid \mathbf{\Phi})$. Suppose we have observed N independent samples: $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\}$ with hidden unobserved data $\{x_1, x_2, \ldots, x_N\}$; the Q-function can then be written as follows:

$$\begin{aligned} Q(\mathbf{\Phi}, \bar{\mathbf{\Phi}}) &= \sum_{i=1}^{N} Q_i(\mathbf{\Phi}, \bar{\mathbf{\Phi}}) = \sum_{i=1}^{N} \sum_{x_i} P(x_i \mid \mathbf{y}_i, \mathbf{\Phi}) \log p(\mathbf{y}_i, x_i \mid \bar{\mathbf{\Phi}}) \\ &= \sum_{i=1}^{N} \sum_{x_i} \frac{p(\mathbf{y}_i, x_i \mid \mathbf{\Phi})}{p(\mathbf{y}_i \mid \mathbf{\Phi})} \log p(\mathbf{y}_i, x_i \mid \bar{\mathbf{\Phi}}) \end{aligned} \tag{4.101}$$

By replacing items in Eq. (4.101) with Eqs. (4.98) and (4.100), the following equation can be obtained:

$$Q(\mathbf{\Phi}, \bar{\mathbf{\Phi}}) = \sum_{k=1}^{K} \gamma_k \log \bar{c}_k + \sum_{k=1}^{K} Q_\lambda(\mathbf{\Phi}, \bar{\mathbf{\Phi}}_k) \tag{4.102}$$

where

$$\gamma_k^i = \frac{c_k p_k(\mathbf{y}_i \mid \mathbf{\Phi}_k)}{P(\mathbf{y}_i \mid \mathbf{\Phi})} \tag{4.103}$$

$$\gamma_k = \sum_{i=1}^{N} \gamma_k^i = \sum_{i=1}^{N} \frac{c_k p_k(\mathbf{y}_i \mid \mathbf{\Phi}_k)}{P(\mathbf{y}_i \mid \mathbf{\Phi})} \tag{4.104}$$

$$Q_\lambda(\mathbf{\Phi}, \bar{\mathbf{\Phi}}_k) = \sum_{i=1}^{N} \gamma_k^i \log p_k(\mathbf{y}_i \mid \bar{\mathbf{\Phi}}_k) = \sum_{i=1}^{N} \frac{c_k p_k(\mathbf{y}_i \mid \mathbf{\Phi}_k)}{P(\mathbf{y}_i \mid \mathbf{\Phi})} \log p_k(\mathbf{y}_i \mid \bar{\mathbf{\Phi}}_k) \tag{4.105}$$

Now we can perform a maximum likelihood estimation on the complete data $(x, \mathbf{y})$ during the M-step. By taking the derivative with respect to each parameter and setting it to zero, we obtain the following EM re-estimate of c_k, $\boldsymbol{\mu}_k$, and $\mathbf{\Sigma}_k$:

$$\hat{c}_k = \frac{\gamma_k}{\sum_{k=1}^{K} \gamma_k} = \frac{\gamma_k}{N} \tag{4.106}$$

$$\hat{\boldsymbol{\mu}}_k = \frac{\sum_{i=1}^{N} \gamma_k^i \mathbf{y}_i}{\sum_{i=1}^{N} \gamma_k^i} = \frac{\sum_{i=1}^{N} \frac{c_k p_k(\mathbf{y}_i \mid \boldsymbol{\Phi}_k)\mathbf{y}_i}{P(\mathbf{y}_i \mid \boldsymbol{\Phi})}}{\sum_{i=1}^{N} \frac{c_k p_k(\mathbf{y}_i \mid \boldsymbol{\Phi}_k)}{P(\mathbf{y}_i \mid \boldsymbol{\Phi})}} \tag{4.107}$$

$$\hat{\Sigma}_k = \frac{\sum_{i=1}^{N} \gamma_k^i (\mathbf{y}_i - \boldsymbol{\mu}_k)(\mathbf{y}_i - \boldsymbol{\mu}_k)'}{\sum_{i=1}^{N} \gamma_k^i} = \frac{\sum_{i=1}^{N} \frac{c_k p_k(\mathbf{y}_i \mid \boldsymbol{\Phi}_k)(\mathbf{y}_i - \boldsymbol{\mu}_k)(\mathbf{y}_i - \boldsymbol{\mu}_k)'}{P(\mathbf{y}_i \mid \boldsymbol{\Phi})}}{\sum_{i=1}^{N} \frac{c_k p_k(\mathbf{y}_i \mid \boldsymbol{\Phi}_k)}{P(\mathbf{y}_i \mid \boldsymbol{\Phi})}} \tag{4.108}$$

The quantity γ_k^i defined in Eq. (4.103) can be interpreted as the posterior probability that the observed data $\mathbf{y}_i$ belong to Gaussian component k ($N_k(\mathbf{y} \mid \boldsymbol{\mu}_k, \Sigma_k)$). This information as to whether the observed data $\mathbf{y}_i$ should belong to Gaussian component k is hidden and can only be observed through the hidden variable x (c_k). The EM algorithm described above is used to uncover how likely the observed data $\mathbf{y}_i$ are expected to be in each Gaussian component. The re-estimation formulas are consistent with our intuition. These MLE formulas calculate the weighted contribution of each data sample according to the mixture posterior probability γ_k^i.

In fact, VQ is an approximate version of EM algorithms. A traditional VQ with the Mahalanobis distance measure is equivalent to a mixture Gaussian VQ with the following conditions:

$$c_k = 1/K \tag{4.109}$$

$$\gamma_k^i = \begin{cases} 1, & \mathbf{y}_i \in C_k \\ 0, & \text{otherwise} \end{cases} \tag{4.110}$$

The difference between VQ and the EM algorithm is that VQ performs a hard assignment of the data sample $\mathbf{y}_i$ to clusters (cells) while the EM algorithm performs a soft assignment of the data sample $\mathbf{y}_i$ to clusters. As discussed in Chapter 8, this difference carries over to the case of the Viterbi algorithm vs. the Baum-Welch algorithm in hidden Markov models.

4.5. CLASSIFICATION AND REGRESSION TREES

Classification and regression trees (CART) [15, 82] have been used in a variety of pattern recognition applications. Binary decision trees, with splitting questions attached to each

node, provide an easy representation that interprets and predicts the structure of a set of data. The application of binary decision trees is much like playing the *number-guessing* game, where the examinee tries to deduce the chosen number by asking a series of binary number-comparing questions.

Consider a simple binary decision tree for height classification. Every person's data in the study may consist of several measurements, including race, gender, weight, age, occupation, and so on. The goal of the study is to develop a classification method to assign a person one of the following five height classes: *tall* (T), *medium-tall* (t), *medium* (M), *medium-short*(s) and *short* (S). Figure 4.14 shows an example of such a binary tree structure. With this binary decision tree, one can easily predict the height class for any new person (with all the measured data, but no height information) by traversing the binary trees. Traversing the binary tree is done through answering a series of yes/no questions in the traversed nodes with the measured data. When the answer is *no*, the right branch is traversed next; otherwise, the left branch will be traversed instead. When the path ends at a leaf node, you can use its attached label as the height class for the new person. If you have the average height for each leaf node (computed by averaging the heights from those people who fall in the same leaf node during training), you can actually use the average height in the leaf node to predict the height for the new person.

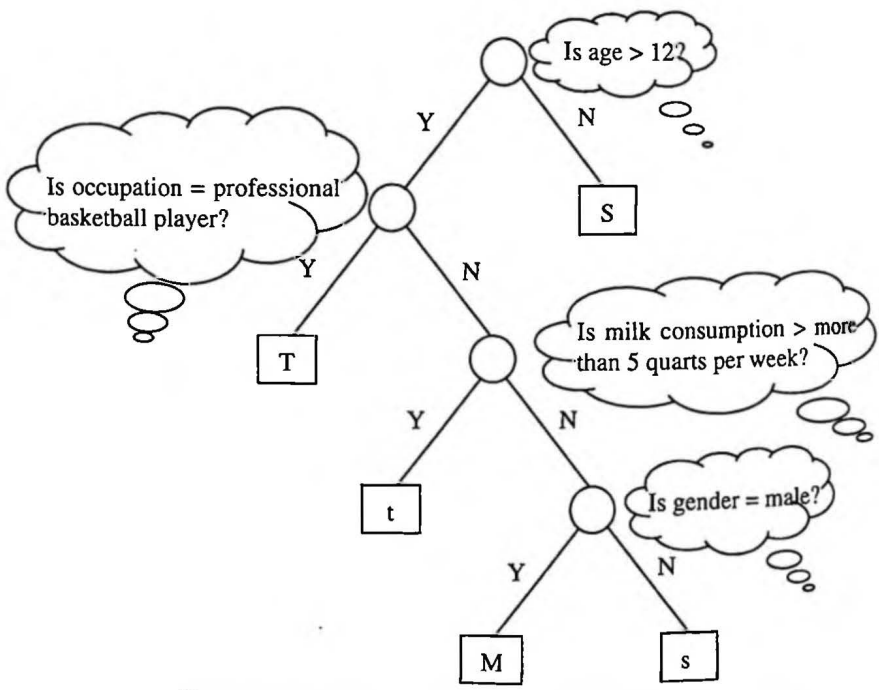


Figure 4.14 A binary tree structure for height classification.

This classification process is similar to a rule-based system where the classification is carried out by a sequence of decision rules. The choice and order of rules applied in a rule-based system is typically designed subjectively by hand through an introspective analysis based on the impressions and intuitions of a limited number of data samples. CART, on the other hand, provides an automatic and data-driven framework to construct the decision process based on objective criteria. Most statistical pattern recognition techniques are designed for data samples having a standard structure with homogeneous variables. CART is designed instead to handle data samples with high dimensionality, mixed data types, and nonstandard data structure. It has the following advantages over other pattern recognition techniques:

- CART can be applied to any data structure through appropriate formulation of the set of potential questions.
- The binary tree structure allows for compact storage, efficient classification, and easily understood interpretation of the predictive structure of the data.
- It often provides, without additional effort, not only classification and recognition, but also an estimate of the misclassification rate for each class.
- It not only handles missing data, but also is very robust to outliers and mislabeled data samples.

To construct a CART from the training samples with their classes (let's denote the set as $\mathcal{S}$), we first need to find a set of questions regarding the measured variables; e.g., "*Is age > 12*?", "*Is occupation = professional basketball player*?", "*Is gender = male*?" and so on. Once the question set is determined, CART uses a greedy algorithm to generate the decision trees. All training samples $\mathcal{S}$ are placed in the root of the initial tree. The *best question* is then chosen from the question set to split the root into two nodes. Of course, we need a measurement of how well each question splits the data samples to pick the best question. The algorithm recursively splits the most promising node with the best question until the right-sized tree is obtained. We describe next how to construct the question set, how to measure each split, how to grow the tree, and how to choose the right-sized tree.

4.5.1. Choice of Question Set

Assume that the training data has the following format:

$$\mathbf{x} = (x_1, x_2, \ldots x_d) \tag{4.111}$$

where each variable x_i is a discrete or continuous data type. We can construct a *standard set* of questions Q as follows:

1. Each question is about the value of only a single variable. Questions of this type are called *simple* or *singleton* questions.

2. If x_i is a discrete variable from the set $\{c_1, c_2, \ldots, c_K\}$, Q includes all questions of the following form:

$$\{\text{Is } x_i \in S?\} \quad \text{where } S \text{ is any subset of } \{c_1, c_2, \ldots, c_K\} \tag{4.112}$$

3. If x_i is a continuous variable, Q includes all questions of the following form:

$$\{\text{Is } x_i \le c?\} \text{ for } c \in (-\infty, \infty) \tag{4.113}$$

The question subset generated from discrete variables (in condition 2 above) is clearly a finite set $(2^{K-1}-1)$. On the other hand, the question subset generated from continuous variables (in condition 3 above) seems to be an infinite set based on the definition. Fortunately, since the training data samples are finite, there are only finite number of distinct splits for the training data. For a continuous variable x_i, the data points in $\mathcal{S}$ contain at most M distinct values $v_1, v_2, \ldots, v_M$. There are only at most M different splits generated by the set of questions in the form:

$$\{\text{Is } x_i \le c_n\} \quad n = 1, 2, \ldots, M \tag{4.114}$$

where $c_n = \frac{v_{n-1} + v_n}{2}$ and $v_0 = 0$. Therefore, questions related to a continuous variable also form a finite subset. The fact that Q is a finite set allows the enumerating of all possible questions in each node during tree growing.

The construction of a question set is similar to that of rules in a rule-based system. Instead of using the all-possible question set Q, some people use knowledge selectively to pick a subset of Q, which is sensitive to pattern classification. For example, the vowel subset and consonant subset are a natural choice for these sensitive questions for phoneme classification. However, the beauty of CART is the ability to use all possible questions related to the measured variables, because CART has a statistical data-driven framework to determine the decision process (as described in subsequent sections). Instead of setting some constraints on the questions (splits), most CART systems use all the possible questions for Q.

4.5.2. Splitting Criteria

A question in CART framework represents a split (partition) of data samples. All the leaf nodes (L in total) represent L disjoint subsets $A_1, A_2, \ldots, A_L$. Now we have the entire potential question set Q, the task is how to find the best question for a node split. The selection of the best question is equivalent to finding the best split for the data samples of the node.

Since each node t in the tree contains some training samples, we can compute the corresponding class probability density function $P(\omega | t)$. The classification process for the node can then be interpreted as a random process based on $P(\omega | t)$. Since our goal is classi-

fication, the objective of a decision tree is to reduce the uncertainty of the event being decided upon. We want the leaf nodes to be as pure as possible in terms of the class distribution. Let Y be the random variable of classification decision for data sample $\mathbf{X}$. We could define the weighted entropy for any node t as follows:

$$\bar{H}_t(Y) = H_t(Y)P(t) \tag{4.115}$$

$$H_t(Y) = -\sum_i P(\omega_i \mid t) \log P(\omega_i \mid t) \tag{4.116}$$

where $P(\omega_i \mid t)$ is the percentage of data samples for class i in node t; and $P(t)$ is the prior probability of visiting node t (equivalent to the ratio of number of data samples in node t and the total number of training data samples). With this weighted entropy definition, the splitting criterion is equivalent to finding the question which gives the greatest entropy reduction, where the entropy reduction for a question q to split a node t into nodes l and r can be defined as:

$$\Delta\bar{H}_t(q) = \bar{H}_t(Y) - (\bar{H}_l(Y) + \bar{H}_r(Y)) = \bar{H}_t(Y) - \bar{H}_t(Y \mid q) \tag{4.117}$$

The reduction in entropy is also the mutual information between Y and question q. The task becomes that of evaluating the entropy reduction $\Delta\bar{H}_q$ for each potential question (split), and picking the question with the greatest entropy reduction, that is,

$$q^* = \underset{q}{\operatorname{argmax}}(\Delta\bar{H}_t(q)) \tag{4.118}$$

If we define the entropy for a tree, T, as the sum of weighted entropies for all the terminal nodes, we have:

$$\bar{H}(T) = \sum_{t \text{ is terminal}} \bar{H}_t(Y) \tag{4.119}$$

It can be shown that the tree-growing (splitting) process repeatedly reduces the entropy of the tree. The resulting tree thus has a better classification power. For continuous pdf, likelihood gain is often used instead, since there is no straightforward entropy measurement [43]. Suppose one specific split divides the data into two groups, $\mathbf{X}_1$ and $\mathbf{X}_2$, which can then be used to train two Gaussian distributions $N_1(\mu_1, \Sigma_1)$ and $N_2(\mu_2, \Sigma_2)$. The log-likelihoods for generating these two data groups are:

$$L_1(\mathbf{X}_1 \mid N_1) = \log \prod_{\mathbf{x}_1} N(\mathbf{x}_1, \mu_1, \Sigma_1) = -\left(d \log 2\pi + \log|\Sigma_1| + d\right) a/2 \tag{4.120}$$

$$L_2(\mathbf{X}_2 \mid N_2) = \log \prod_{\mathbf{x}_2} N(\mathbf{x}_2, \mu_2, \Sigma_2) = -\left(d \log 2\pi + \log|\Sigma_2| + d\right) b/2 \tag{4.121}$$

where d is the dimensionality of the data; and a and b are the sample counts for the data groups $\mathbf{X}_1$ and $\mathbf{X}_2$ respectively. Now if the data $\mathbf{X}_1$ and $\mathbf{X}_2$ are merged into one group and modeled by one Gaussian $N(\mu,\Sigma)$, according to MLE, we have

$$\mu = \frac{a}{a+b}\mu_1 + \frac{b}{a+b}\mu_2 \tag{4.122}$$

$$\Sigma = \frac{a}{a+b}\left[\Sigma_1 + (\mu_1 - \mu)(\mu_1 - \mu)^t\right] + \frac{b}{a+b}\left[\Sigma_2 + (\mu_2 - \mu)(\mu_2 - \mu)^t\right] \tag{4.123}$$

Thus, the likelihood gain of splitting the data $\mathbf{X}$ into two groups $\mathbf{X}_1$ and $\mathbf{X}_2$ is:

$$\begin{aligned}\Delta\bar{L}_t(q) &= L_1(\mathbf{X}_1 \mid N) + L_2(\mathbf{X}_2 \mid N) - L_x(\mathbf{X} \mid N) \\ &= (a+b)\log|\Sigma| - a\log|\Sigma_1| - b\log|\Sigma_2|\end{aligned} \tag{4.124}$$

For regression purposes, the most popular splitting criterion is the mean squared error measure, which is consistent with the common *least squared* regression methods. For instance, suppose we need to investigate the real height as a regression function of the measured variables in the height study. Instead of finding height classification, we could simply use the average height in each node to predict the height for any data sample. Suppose Y is the actual height for training data $\mathbf{X}$, then overall regression (prediction) error for a node t can be defined as:

$$E(t) = \sum_{\mathbf{X}\in t} |Y - d(\mathbf{X})|^2 \tag{4.125}$$

where $d(\mathbf{X})$ is the regression (predictive) value of Y.

Now, instead of finding the question with greatest entropy reduction, we want to find the question with largest squared error reduction. That is, we want to pick the question q^* that maximizes:

$$\Delta E_t(q) = E(t) - (E(l) + E(r)) \tag{4.126}$$

where l and r are the leaves of node t. We define the expected square error $V(t)$ for a node t as the overall regression error divided by the total number of data samples in the node.

$$V(t) = E\left(\sum_{\mathbf{X}\in t} |Y - d(\mathbf{X})|^2\right) = \frac{1}{N(t)}\sum_{\mathbf{X}\in t} |Y - d(\mathbf{X})|^2 \tag{4.127}$$

Note that $V(t)$ is actually the variance estimate of the height, if $d(\mathbf{X})$ is made to be the average height of data samples in the node. With $V(t)$, we define the weighted squared error $\bar{V}(t)$ for a node t as follows.

$$\bar{V}(t) = V(t)P(t) = \left(\frac{1}{N(t)}\sum_{\mathbf{X}\in t} |Y - d(\mathbf{X})|^2\right)P(t) \tag{4.128}$$

Finally, the splitting criterion can be rewritten as:

$$\Delta \bar{V}_t(q) = \bar{V}(t) - (\bar{V}(l) + \bar{V}(r)) \tag{4.129}$$

Based on Eqs. (4.117) and (4.129), one can see the analogy between entropy and variance in the splitting criteria for CART. The use of entropy or variance as splitting criteria is under the assumption of uniform misclassification costs and uniform prior distributions. When nonuniform misclassification costs and prior distributions are used, some other splitting might be used for splitting criteria. Noteworthy ones are *Gini index of diversity* and *twoing rule*. Those interested in alternative splitting criteria can refer to [11, 15].

For a wide range of splitting criteria, the properties of the resulting CARTs are empirically insensitive to these choices. Instead, the criterion used to get the right-sized tree is much more important. We discuss this issue in Section 4.5.6.

4.5.3. Growing the Tree

Given the question set Q and splitting criteria $\Delta \bar{H}_t(q)$, the tree-growing algorithm starts from the initial root-only tree. At each node of tree, the algorithm searches through the variables one by one, from x_1 to x_N. For each variable, it uses the splitting criteria to find the best question (split). Then it can pick the best question out of the N best single-variable questions. The procedure can continue splitting each node until either of the following conditions is met for a node:

1. No more splits are possible; that is, all the data samples in the node belong to the same class;
2. The greatest entropy reduction of best question (split) falls below a pre-set threshold β, i.e.:

$$\max_{q \in Q} \Delta \bar{H}_t(q) < \beta \tag{4.130}$$

3. The number of data samples falling in the leaf node t is below some threshold α. This is to assure that there are enough training tokens for each leaf node if one needs to estimate some parameters associated with the node.

When a node cannot be further split, it is declared a terminal node. When all active (non-split) nodes are terminal, the tree-growing algorithm stops.

The algorithm is greedy because the question selected for any given node is the one that seems to be the best, without regard to subsequent splits and nodes. Thus, the algorithm constructs a tree that is locally optimal, but not necessarily globally optimal (but hopefully globally *good enough*). This tree-growing algorithm has been successfully applied in many applications [5, 39, 60]. A dynamic programming algorithm for determining global optimality is described in [78]; however, it is suitable only in restricted applications with relatively few variables.

4.5.4. Missing Values and Conflict Resolution

Sometimes, the available data sample $\mathbf{x} = (x_1, x_2, \ldots x_d)$ has some value x_j missing. This missing-value case can be handled by the use of *surrogate questions* (*splits*). The idea is intuitive. We define a similarity measurement between any two questions (splits) q and $\tilde{q}$ of a node t. If the best question of node t is the question q on the variable x_i, we can find the question $\tilde{q}$ that is most similar to q on a variable other than x_i. $\tilde{q}$ is our best surrogate question. Similarly, we find the second-best surrogate question, third-best and so on. The surrogate questions are considered as the backup questions in the case of missing x_i values in the data samples. The surrogate question is used in descending order to continue tree traversing for those data samples. The surrogate question gives CART unique ability to handle the case of missing data. The similarity measurement is basically a measurement reflecting the similarity of the class probability density function [15].

When choosing the best question for splitting a node, several questions on the same variable x_i may achieve the same entropy reduction and generate the same partition. As in rule-based problem solving systems, a *conflict resolution* procedure [99] is needed to decide which question to use. For example, discrete questions q_1 and q_2 have the following format:

$$q_1 : \{\text{Is } x_i \in S_1 ?\} \tag{4.131}$$

$$q_2 : \{\text{Is } x_i \in S_2 ?\} \tag{4.132}$$

Suppose S_1 is a subset of S_2, and one particular node contains only data samples whose x_i value contains only values in S_1, but no other. Now question q_1 or q_2 performs the same splitting pattern and therefore achieves exactly the same amount of entropy reduction. In this case, we call q_1 a sub-question of question q_2, because q_1 is a more specific version.

A *specificity ordering* conflict resolution strategy is used to favor the discrete question with fewer elements because it is more specific to the current node. In other words, if the elements of a question are a subset of the elements of another question with the same entropy reduction, the question with the subset of elements is preferred. Preferring more specific questions will prevent decision trees from over-generalizing. The specificity ordering conflict resolution can be implemented easily by presorting the set of discrete questions by the number of elements they contain in descending order, before applying them to decision trees. A similar specificity ordering conflict resolution can also be implemented for continuous-variable questions.

4.5.5. Complex Questions

One problem with allowing only simple questions is that the data may be over-fragmented, resulting in similar leaves in different locations of the tree. For example, when the best ques-

tion (rule) to split a node is actually a composite question of the form "*Is* $x_i \in S_1$?" or "*Is* $x_i \in S_2$?", a system allowing only simple questions will generate two separate questions to split the data into three clusters rather than two as shown in Figure 4.15. Also data for which the answer is *yes* are inevitably fragmented across two shaded nodes. This is inefficient and ineffective since these two very similar data clusters may now both contain insufficient training examples, which could potentially handicap future tree growing. Splitting data unnecessarily across different nodes leads to unnecessary computation, redundant clusters, reduced trainability, and less accurate entropy reduction.

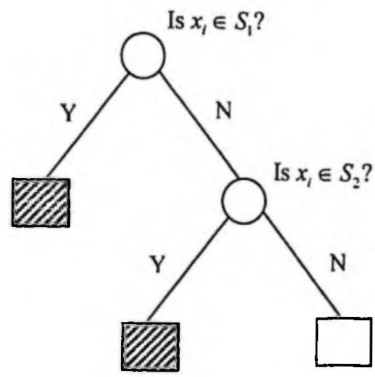


Figure 4.15 An over-split tree for the question "*Is* $x_i \in S_1$?" or "*Is* $x_i \in S_2$?"

We deal with this problem by using a composite-question construction [38, 40]. It involves conjunctive and disjunctive combinations of all questions (and their negations). A composite question is formed by first growing a tree with simple questions only and then clustering the leaves into two sets. Figure 4.16 shows the formation of one composite question. After merging, the structure is still a binary question. To construct the composite question, multiple OR operators are used to describe the composite condition leading to either one of the final clusters, and AND operators are used to describe the relation within a particular route. Finally, a Boolean reduction algorithm is used to simplify the Boolean expression of the composite question.

To speed up the process of constructing composite questions, we constrain the number of leaves or the depth of the binary tree through heuristics. The most frequently used heuristics is the limitation of the depth when searching a composite question. Since composite questions are essentially binary questions, we use the same greedy tree-growing algorithm to find the best composite question for each node and keep growing the tree until the stop criterion is met. The use of composite questions not only enables flexible clustering, but also improves entropy reduction. Growing the sub-tree a little deeper before constructing the composite question may achieve longer-range optimum, which is preferable to the local optimum achieved in the original greedy algorithm that used simple questions only.

The construction of composite questions can also be applied to continuous variables to obtained complex rectangular partitions. However, some other techniques are used to obtain

general partitions generated by hyperplanes not perpendicular to the coordinate axes. Questions typically have a linear combination of continuous variables in the following form [15]:

$$\{\text{Is } \sum_i a_i x_i \le c ?\} \tag{4.133}$$

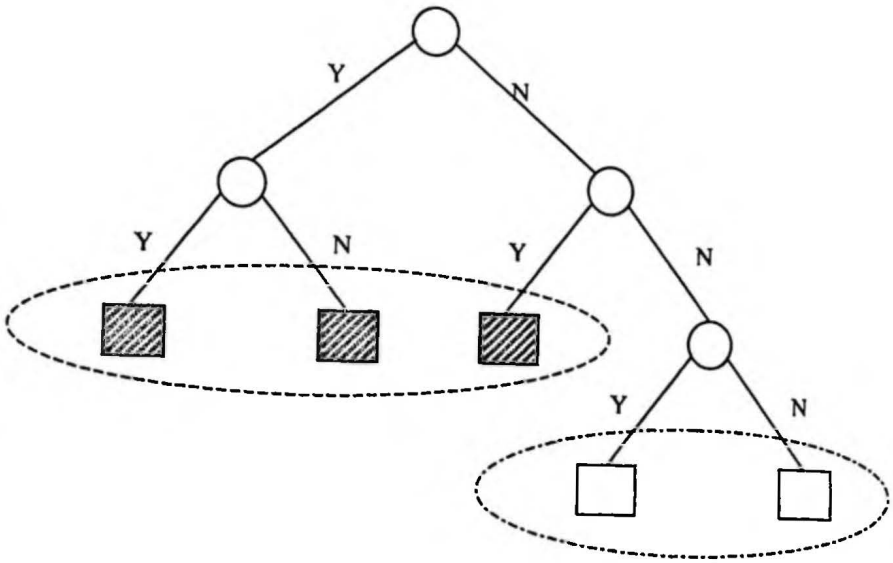


Figure 4.16 The formation of a composite question from simple questions.

4.5.6. The Right-Sized Tree

One of the most critical problems for CART is that the tree may be strictly tailored to the training data and has no generalization capability. When you split a leaf node in the tree to get entropy reduction until each leaf node contains data from one single class, that tree possesses a zero percent classification error on the training set. This is an over-optimistic estimate of the test-set misclassification rate. Independent test sample estimation or cross-validation is often used to prevent decision trees from over-modeling idiosyncrasies of the training data. To get a right-sized tree, you can minimize the misclassification rate for future independent test data.

Before we describe the solution for finding the right sized tree, let's define a couple of useful terms. Naturally we will use the plurality rule $\delta(t)$ to choose the class for a node t:

$$\delta(t) = \underset{i}{\text{argmax}}\, P(\omega_i \mid t) \tag{4.134}$$

Similar to the notation used in Bayes' decision theory, we can define the misclassification rate $R(t)$ for a node t as:

$$R(t) = r(t)P(t) \tag{4.135}$$

where $r(t) = 1 - \max_i P(\omega_i \mid t)$ and $P(t)$ is the frequency (probability) of the data falling in node t. The overall misclassification rate for the whole tree T is defined as:

$$R(T) = \sum_{t \in \tilde{T}} R(t) \tag{4.136}$$

where $\tilde{T}$ represents the set of terminal nodes. If a nonuniform misclassification cost $c(i \mid j)$, the cost of misclassifying class j data as class i data, is used, $r(t)$ is redefined as:

$$r(t) = \min_i \sum_j c(i \mid j)P(j \mid t) \tag{4.137}$$

As we mentioned, $R(T)$ can be made arbitrarily small (eventually reduced to zero) for the training data if we keep growing the tree. The key now is how we choose the tree that can minimize $R^*(T)$, which is denoted as the misclassification rate of independent test data. Almost no tree initially grown can perform well on independent test data. In fact, using more complicated stopping rules to limit the tree growing seldom works, and it is either stopped too soon at some terminal nodes, or continued too far in other parts of the tree. Instead of inventing some clever stopping criteria to stop the tree growing at the right size, we let the tree over-grow (based on rules in Section 4.5.3). We use a pruning strategy to gradually cut back the tree until the minimum $R^*(T)$ is achieved. In the next section we describe an algorithm to prune an over-grown tree, *minimum cost-complexity pruning*.

4.5.6.1. Minimum Cost-Complexity Pruning

To prune a tree, we need to find a subtree (or branch) that makes the least impact in terms of a cost measure, whether it is pruned or not. This candidate to be pruned is called the *weakest subtree*. To define such a weakest subtree, we first need to define the cost measure.

DEFINITION 1: For any sub-tree T of $T_{\max}$ ($T \prec T_{\max}$), let $|\tilde{T}|$ denote the number of terminal nodes in tree T.

DEFINITION 2: Let $\alpha \geq 0$ be a real number called the *complexity parameter*. The cost-complexity measure can be defined as:

$$R_\alpha(T) = R(T) + \alpha |\tilde{T}| \tag{4.138}$$

DEFINITION 3: For each α, define the minimal cost-complexity subtree $T(\alpha) \prec T_{\max}$ that minimizes $R_\alpha(T)$, that is,

$$T(\alpha) = \arg\min_{T \prec T_{\max}} R_\alpha(T) \tag{4.139}$$

Based on Definitions 2 and 3, if α is small, the penalty for having a large tree is small and $T(\alpha)$ will be large. In fact, $T(0) = T_{max}$ because T_{max} has a zero misclassification rate, so it will minimize $R_o(T)$. On the other hand, when α increases, $T(\alpha)$ becomes smaller and smaller. For a sufficient large α, $T(\alpha)$ may collapse into a tree with only the root. The increase of α produces a sequence of pruned trees and it is the basis of the pruning process. The pruning algorithm rests on two theorems. The first is given as follows.

THEOREM 1: For every value of α, there exists a unique minimal cost-complexity subtree $T(\alpha)$ as defined in Definition 3.[12]

To progressively prune the tree, we need to find the weakest subtree (node). The idea of a weakest subtree is the following: *if we collapse the weakest subtree into a single terminal node, the cost-complexity measure would increase least.* For any node t in the tree T, let $\{t\}$ denote the subtree containing only the node t, and T_t denote the branch starting at node t. Then we have

$$R_\alpha(T_t) = R(T_t) + \alpha |\tilde{T}_t| \tag{4.140}$$

$$R_\alpha(\{t\}) = R(t) + \alpha \tag{4.141}$$

When α is small, T_t has a smaller cost-complexity than the single-node tree $\{t\}$. However, when α increases to a point where the cost-complexity measures for T_t and $\{t\}$ are the same, it makes sense to collapse T_t into a single terminal node $\{t\}$. Therefore, we decide the critical value for α by solving the following inequality:

$$R_\alpha(T_t) \le R_\alpha(\{t\}) \tag{4.142}$$

We obtain:

$$\alpha \le \frac{R(t) - R(T_t)}{|\tilde{T}_t| - 1} \tag{4.143}$$

Based on Eq. (4.143), we define a measurement $\eta(t)$ for each node t in tree T:

$$\eta(t) = \begin{cases} \dfrac{R(t) - R(T_t)}{|\tilde{T}_t| - 1}, & t \notin \tilde{T} \\ +\infty, & t \in \tilde{T} \end{cases} \tag{4.144}$$

Based on measurement $\eta(t)$, we then define the weakest subtree T_{t_1} as the tree branch starting at the node t_1 such that

[12] You can find the proof to this in [15].

$$t_1 = \arg\min_{t \in T} \eta(t) \tag{4.145}$$

$$\alpha_1 = \eta(t_1) \tag{4.146}$$

As α increases, the node t_1 is the first node such that $R_\alpha(\{t\})$ becomes equal to $R_\alpha(T_t)$. At this point, it would make sense to prune subtree T_{t_1} (collapse T_{t_1} into a single-node subtree $\{t_1\}$), and α_1 is the value of α where the pruning occurs.

Now the tree T after pruning is referred to as T_1, i.e.,

$$T_1 = T - T_{t_1} \tag{4.147}$$

We then use the same process to find the weakest subtree T_{t_2} in T_1 and the new pruning point α_2. After pruning away T_{t_2} from T_1 to form the new pruned tree T_2, we repeat the same process to find the next weakest subtree and pruning point. If we continue the process, we get a sequence of decreasing pruned trees:

$$T \succ T_1 \succ T_2 \succ T_2 \cdots \succ \{r\} \tag{4.148}$$

where $\{r\}$ is the single-node tree containing the root of tree T with corresponding pruning points:

$$\alpha_0 < \alpha_1 < \alpha_2 < \alpha_3 < \cdots \tag{4.149}$$

where $\alpha_0 = 0$.

With the process above, the following theorem (which is basic for the minimum cost-complexity pruning) can be proved.

THEOREM 2 : Let T_0 be the original tree T.

$$\text{For } k \geq 0,\ \alpha_k \leq \alpha < \alpha_{k+1},\ T(\alpha) = T(\alpha_k) = T_k \tag{4.150}$$

4.5.6.2. Independent Test Sample Estimation

The minimum cost-complexity pruning algorithm can progressively prune the over-grown tree to form a decreasing sequence of subtrees $T \succ T_1 \succ T_2 \succ T_2 \cdots \succ \{r\}$, where $T_k = T(\alpha_k)$, $\alpha_0 = 0$ and $T_0 = T$. The task now is simply to choose one of those subtrees as the optimal-sized tree. Our goal is to find the optimal-sized tree that minimizes the misclassification for

independent test set $R^*(T)$. When the training set $\mathfrak{I}$ is abundant, we can afford to set aside an independent test set $\mathfrak{R}$ from the training set. Usually $\mathfrak{R}$ is selected as one third of the training set $\mathfrak{I}$. We use the remaining two thirds of the training set $\mathfrak{I}-\mathfrak{R}$ (still abundant) to train the initial tree T and apply the minimum cost-complexity pruning algorithm to attain the decreasing sequence of subtrees $T \succ T_1 \succ T_2 \succ T_2 \cdots \succ \{r\}$. Next, the test set $\mathfrak{R}$ is run through the sequence of subtrees to get corresponding estimates of test-set misclassification $R^*(T), R^*(T_1), R^*(T_2), \cdots, R^*(\{r\})$. The optimal-sized tree T_{k^*} is then picked as the one with minimum test-set misclassification measure, i.e.:

$$k^* = \arg\min_k R^*(T_k) \tag{4.151}$$

The *independent test sample estimation* approach has the drawback that it reduces the effective training sample size. This is why it is used only when there is abundant training data. Under most circumstances where training data is limited, *cross-validation* is often used.

4.5.6.3. Cross-Validation

CART can be pruned via *v-fold cross-validation*. It follows the same principle of cross validation described in Section 4.2.3. First it randomly divides the training set $\mathfrak{I}$ into v disjoint subsets $\mathfrak{I}_1, \mathfrak{I}_2, \cdots, \mathfrak{I}_v$, each containing roughly the same data samples. It then defines the i^{th} training set

$$\mathfrak{I}^i = \mathfrak{I} - \mathfrak{I}_i \quad i = 1, 2, \ldots, v \tag{4.152}$$

so that $\mathfrak{I}^i$ contains the fraction $(v-1)/v$ of the original training set. v is usually chosen to be large, like 10.

In v-fold cross-validation, v auxiliary trees are grown together with the main tree T grown on $\mathfrak{I}$. The i^{th} tree is grown on the i^{th} training set $\mathfrak{I}^i$. By applying minimum cost-complexity pruning, for any given value of the cost-complexity parameter α, we can obtain the corresponding minimum cost-complexity subtrees $T(\alpha)$ and $T^i(\alpha)$, $i = 1, 2, \ldots, v$. According to Theorem 2 in Section 4.5.6.1, those minimum cost-complexity subtrees will form $v+1$ sequences of subtrees:

$$T \succ T_1 \succ T_2 \succ T_2 \cdots \succ \{r\} \text{ and} \tag{4.153}$$

$$T^i \succ T_1^i \succ T_2^i \succ T_3^i \cdots \succ \{r^i\} \quad i = 1, 2, \ldots, v \tag{4.154}$$

ALGORITHM 4.5: *THE CART ALGORITHM*

Step 1: Question Set: Create a standard set of questions Q that consists of all possible questions about the measured variables.
Step 2: Splitting Criterion: Pick a splitting criterion that can evaluate all the possible questions in any node. Usually it is either entropy-like measurement for classification trees or mean square errors for regression trees.
Step 3: Initialization: Create a tree with one (root) node, consisting of all training samples.
Step 4: Split Candidates: Find the best composite question for each terminal node:
a. Generate a tree with several simple-question splits as described in Section 4.5.3.
b. Cluster leaf nodes into two classes according to the same splitting criterion.
c. Based on the clustering done in (b), construct a corresponding composite question.
Step 5: Split: Out of all the split candidates in Step 4, split the one with best criterion.
Step 6: Stop Criterion: If all the leaf nodes containing data samples from the same class or all the potential splits generating improvement fall below a pre-set threshold β, go to Step 7; otherwise go to Step 4.
Step 7: Use *independent test sample estimate* or *cross-validation estimate* to prune the original tree into the optimal size.

The basic assumption of cross-validation is that the procedure is *stable* if v is large. That is, $T(\alpha)$ should have the same classification accuracy as $T^i(\alpha)$. Although we cannot directly estimate the test-set misclassification for the main tree $R^*(T(\alpha))$, we could approximate it via the test-set misclassification measure $R^*(T^i(\alpha))$, since each data sample in $\mathcal{S}$ occurs in one and only one test set $\mathcal{S}_i$. The v-fold cross-validation estimate $R^{CV}(T(\alpha))$ can be computed as:

$$R^{CV}(T(\alpha)) = \frac{1}{v}\sum_{i=1}^{v} R^*(T^i(\alpha)) \tag{4.155}$$

Similar to Eq. (4.151), once $R^{CV}(T(\alpha))$ is computed, the optimal v-fold cross-validation tree $T_{k^{CV}}$ can be found through

$$k^{CV} = \arg\min_k R^{CV}(T_k) \tag{4.156}$$

Cross-validation is computationally expensive in comparison with independent test sample estimation, though it makes more effective use of all training data and reveals useful information regarding the stability of the tree structure. Since the auxiliary trees are grown on a smaller training set (a fraction $v-1/v$ of the original training data), they tend to have a higher misclassification rate. Therefore, the cross-validation estimates $R^{CV}(T)$ tend to be an over-estimation of the misclassification rate. The algorithm for generating a CART tree is illustrated in Algorithm 4.5.

4.6. HISTORICAL PERSPECTIVE AND FURTHER READING

Pattern recognition is a multidisciplinary field that comprises a broad body of loosely related knowledge and techniques. Historically, there are two major approaches to pattern recognition – the statistical and the syntactical approaches. Although this chapter is focused on the statistical approach, syntactical pattern recognition techniques, which aim to address the limitations of the statistical approach in handling contextual or structural information, can be complementary to statistical approaches for spoken language processing, such as parsing. Syntactic pattern recognition is based on the analogy that complex patterns can be decomposed recursively into simpler subpatterns, much as a sentence can be decomposed into words and letters. Fu [24] provides an excellent book on syntactic pattern recognition.

The foundation of statistical pattern recognition is Bayesian theory, which can be traced back to the 18th century [9, 54] and its invention by the British mathematician Thomas Bayes (1702-1761). Chow [20] was the first to use Bayesian decision theory for pattern recognition. Statistical pattern recognition has been used successfully in a wide range of applications, from optical/handwritten recognition [13, 96], to speech recognition [7, 86] and to medical/machinery diagnosis [1, 27]. The books by Duda et al. [22] and Fukunaga [25] are two classic treatments of statistical pattern recognition. Duda et al. have a second edition of the classic pattern recognition book [23] that includes many up-to-date topics.

MLE and MAP are two most frequently used estimation methods for pattern recognition because of their simplicity and efficiency. In Chapters 8 and 9, they play an essential role in model parameter estimation. Estimating the recognition performance and comparing different recognition systems are important subjects in pattern recognition. The importance of a large number of test samples was reported in [49]. McNemar's test is dated back to the 1940s [66]. The modification of the test for continuous speech recognition systems presented in this chapter is based on an interesting paper [30] that contains a general discussion on using hypothesis-testing methods for continuous speech recognition.

Gradient descent is fundamental for most discriminant estimation methods, including MMIE, MCE, and neural networks. The history of gradient descent can be traced back to Newton's method for root finding [72, 81]. Both the book by Duda et al. [23] and the paper by Juang et al. [48] provide a good description of gradient descent. MMIE was first proposed in [16, 71] for the speech recognition problem. According to these two works, MMIE is more robust than MLE to incorrect model assumptions. MCE was first formulated by Juang et al. [48] and successfully applied to small-vocabulary speech recognition [47].

The modern era of neural networks was brought to the scientific community by McCulloch and Pitts. In the pioneering paper [64], McCulloch and Pitts laid out the mathematical treatment of the behavior of networks of simple neurons. The most important result they showed is that a network would compute any computable function. John von Neumann was influenced by this paper to use switch-delay elements derived from the McCulloch-Pitts neuron in the construction of the EDVAC (Electronic Discrete Variable Automatic Computer) that was developed based on ENIAC (Electronic Numerical Integrator and Computer) [2, 35]. The ENIAC was the famous first general-purpose electronic computer built at the

Moore School of Electrical Engineering at the University of Pennsylvania from 1943 to 1946 [31]. The two-layer perceptron work [87] by Rosenblatt, was the first to provide rigorous proofs about perceptron convergence. A 1969 book by Minsky and Papert [68] reveals that there are fundamental limits to what single-layer perceptrons can compute. It was not until the 1980s that the discovery of multi-layer perceptrons (with hidden layers and nonlinear threshold functions) and back-propagation [88] reawakened interest in neural networks. The two-volume PDP book [90, 91], *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, edited by Rummelhart and McClelland, brought the back-propagation learning method to the attention of the widest audience. Since then, various applications of neural networks in diverse domains have been developed, including speech recognition [14, 58], speech production and perception [93, 94], optical and handwriting character recognition [55, 92], visual recognition [26], game playing [97], and natural language processing [63]. There are several good textbooks for neural networks. In particular, the book by Haykin [35] provides a very comprehensive coverage of all foundations of neural networks. Bishop [12] provides a thoughtful treatment of neural networks from the perspective of pattern recognition. Short, concise tutorial papers on neural networks can be found in [44, 57].

Vector quantization originated from speech coding [17, 32, 45, 61]. The *k*-means algorithm was introduced by Lloyd [59]. Over the years, there have been many variations of VQ, including fuzzy VQ [10], learning VQ (LVQ) [51], and supervised VQ [18, 42]. The first published investigation toward the EM-like algorithm for incomplete data learning can be attributed to Pearson [79]. The modern EM algorithm was formalized by Dempster, Laird, and Rubin [21]. McLachlan and Krishnan [65] provide a thorough overview and history of the EM algorithm. The convergence of the EM algorithm is an interesting research topic and Wu [100] has an extensive description of the rate of convergence. The EM algorithm is the basis for all unsupervised learning that includes hidden variables. The famous HMM training algorithm, as described in Chapter 8, is based on the EM algorithm.

CART uses a very intuitive and natural principle of sequential questions and answers, which can be traced back to 1960s [70]. The popularity of CART is attributed to the book by Breiman *et al.* [15]. Quinlan proposed some interesting variants of CART, like ID3 [82] and C4.5 [84]. CART has recently been one of the most popular techniques in machine learning. Mitchell includes a good overview chapter on the latest CART techniques in his machine-learning book [69]. In addition to the strategies of node splitting and pruning mentioned in this chapter, [62] used a very interesting approach for splitting and pruning criteria based on a statistical significance testing of the data's distributions. Moreover, [28] proposed an iterative expansion pruning algorithm which is believed to perform as well as cross-validation pruning and yet is computationally cheaper [52]. CART has been successfully used in a variety of spoken language applications such as letter-to-sound conversion [46, 60], allophone model clustering [8, 38, 39], language models [5], automatic rule generation [83], duration modeling of phonemes [74, 80], and supervised vector quantization [67].

REFERENCES

[1] Albert, A. and E.K. Harris, *Multivariate Interpretation of Clinical Laboratory Data*, 1987, New York, Marcel Dekker.

[2] Aspray, W. and A. Burks, "Papers of John von Neumann on Computing and Computer Theory" in *Charles Babbage Institute Reprint Series for the History of Computing* 1986, Cambridge, MA, MIT Press.

[3] Atal, B.S. and M.R. Schroeder, "Predictive Coding of Speech Signals and Subjective Error Criteria," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1979, **ASSP-27**(3), pp. 247-254.

[4] Bahl, L.R., *et al.*, "A New Algorithm for the Estimation of Hidden Markov Model Parameters," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1988, New York, NY, pp. 493-496.

[5] Bahl, L.R., *et al.*, "A Tree-Based Statistical Language Model for Natural Language Speech Recognition," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, 1989, **37**(7), pp. 1001-1008.

[6] Bahl, L.R., *et al.*, "Estimating Hidden Markov Model Parameters so as to Maximize Speech Recognition Accuracy," *IEEE Trans. on Speech and Audio Processing*, 1993, **1**(1), pp. 77-83.

[7] Bahl, L.R., F. Jelinek, and R.L. Mercer, "A Maximum Likelihood Approach to Continuous Speech Recognition," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 1983, **5**(2), pp. 179-190.

[8] Bahl, L.R., *et al.*, "Decision Trees for Phonological Rules in Continuous Speech" in *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing* 1991, Toronto, Canada, pp. 185-188.

[9] Bayes, T., "An Essay Towards Solving a Problem in the Doctrine of Chances," *Philosophical Tansactions of the Royal Society*, 1763, **53**, pp. 370-418.

[10] Bezdek, J., *Pattern Recognition with Fuzzy Objective Function Algorithms*, 1981, New York, NY, Plenum Press.

[11] Bhargava, T.N. and V.R.R. Uppuluri, "Sampling Distribution of Gini's Index of Diversity," *Applied Mathematics and Computation*, 1977, **3**, pp. 1-24.

[12] Bishop, C.M., *Neural Networks for Pattern Recognition*, 1995, Oxford, UK, Oxford University Press.

[13] Blesser, B., *et al.*, "A Theoretical Approach for Character Recognition Based on Phenomenological Attributes," *Int. Journal of Man-Machine Studies*, 1974, 6(6), pp. 701-714.

[14] Bourlard, H. and N. Morgan, *Connectionist Speech Recognition – A Hybrid Approach*, 1994, Boston, MA, Kluwer Academic Publishers.

[15] Breiman, L., *et al.*, *Classification and Regression Trees*, 1984, Pacific Grove, CA, Wadsworth.

[16] Brown, P.F., *The Acoustic-Modeling Problem in Automatic Speech Recognition*, PhD Thesis in Computer Science Department 1987, Carnegie Mellon University, Pittsburgh, PA.

[17] Buzo, A., *et al.*, "Speech Coding Based upon Vector Quantization," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1980, **28**(5), pp. 562-574.

[18] Cerf, P.L., W. Ma, and D.V. Compernolle, "Multilayer Perceptrons as Labelers for Hidden Markov Models," *IEEE Trans. on Speech and Audio Processing*, 1994, **2**(1), pp. 185-193.

[19] Chang, P.C. and B.H. Juang, "Discriminative Training of Dynamic Programming Based Speech Recognizers," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1992, San Fancisco.

[20] Chow, C.K., "An Optimum Character Recognition System Using Decision Functions," *IRE Trans.*, 1957, pp. 247-254.

[21] Dempster, A.P., N.M. Laird, and D.B. Rubin, "Maximum-Likelihood from Incomplete Data via the EM Algorithm," *Journal of Royal Statistical Society ser. B*, 1977, **39**, pp. 1-38.

[22] Duda, R.O. and P.E. Hart, *Pattern Classification and Scene Analysis*, 1973, New York, N.Y., John Wiley and Sons.

[23] Duda, R.O., D.G. Stork, and P.E. Hart, *Pattern Classification and Scene Analysis: Pattern Classification*, 2nd ed, 1999, John Wiley & Sons.

[24] Fu, K.S., *Syntactic Pattern Recognition and Applications*, 1982, Englewood Cliffs, NJ, Prentice Hall.

[25] Fukunaga, K., *Introduction to Statistical Pattern Recognition*, 2nd ed., 1990, Orlando, FL, Academic Press.

[26] Fukushima, K., S. Miykake, and I. Takayuki, "Neocognition: A Neural Network Model for a Mechanism of Visual Pattern Recognition," *IEEE Trans. on Systems, Man and Cybernetics*, 1983, **SMC-13**(5), pp. 826-834.

[27] Gastwirth, J.L., "The Statistical Precision of Medical Screening Procedures: Application to Polygraph and AIDS Antibodies Test Data (with Discussion)," *Statistics Science*, 1987, **2**, pp. 213-238.

[28] Gelfand, S., C. Ravishankar, and E. Delp, "An Iterative Growing and Pruning Algorithm for Classification Tree Design," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 1991, **13**(6), pp. 163-174.

[29] Gersho, A., "On the Structure of Vector Quantization," *IEEE Trans. on Information Theory*, 1982, **IT-28**, pp. 256-261.

[30] Gillick, L. and S.J. Cox, "Some Statistical Issues in the Comparison of Speech Recognition Algorithms," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1989, Glasgow, Scotland, UK, IEEE, pp. 532-535.

[31] Goldstine, H., *The Computer from Pascal to von Neumann*, 2nd ed, 1993, Princeton, NJ, Princeton University Press.

[32] Gray, R.M., "Vector Quantization," *IEEE ASSP Magazine*, 1984, **1**(April), pp. 4-29.

[33] Gray, R.M. and E.D. Karnin, "Multiple Local Optima in Vector Quantizers," *IEEE Trans. on Information Theory*, 1982, **IT-28**, pp. 256-261.

[34] Hartigan, J.A., *Clustering Algorithm*, 1975, New York, NY, J. Wiley.

[35] Haykin, S., *Neural Networks: A Comprehensive Foundation*, 2nd ed, 1999, Upper Saddle River, NJ, Prentice-Hall.

[36] Hedelin, P., P. Knagenhjelm, and M. Skoglund, "Vector Quantization for Speech Transmission" in *Speech Coding and Synthesis*, W.B. Kleijn and K.K. Paliwal, eds., 1995, Amsterdam, pp. 311-396, Elsevier.

[37] Hinton, G.E., "Connectionist Learning Procedures," *Artificial Intelligence*, 1989, **40**, pp. 185--234.

[38] Hon, H.W., *Vocabulary-Independent Speech Recognition: The VOCIND System*, Ph.D Thesis in Department of Computer Science 1992, Carnegie Mellon University, Pittsburgh, PA.

[39] Hon, H.W. and K.F. Lee, "On Vocabulary-Independent Speech Modeling," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1990, Albuquerque, NM, pp. 725-728.

[40] Hon, H.W. and K.F. Lee, "Vocabulary-Independent Subword Modeling and Adaptation," *IEEE Workshop on Speech Recognition*, 1991, Harriman, NY, Arden House.

[41] Huang, X.D., Y. Ariki, and M.A. Jack, *Hidden Markov Models for Speech Recognition*, 1990, Edinburgh, U.K., Edinburgh University Press.

[42] Hunt, M.J., *et al.*, "An Investigation of PLP and IMELDA Acoustic Representations and of Their Potential for Combination" in *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing* 1991, Toronto, Canada, pp. 881-884.

[43] Hwang, M.Y. and X.D. Huang, "Dynamically Configurable Acoustic Modelings for Speech Recognition," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1998, Seattle, WA.

[44] Jain, A., J. Mao, and K.M. Mchiuddin, "Artificial Neural Networks: A Tutorial," *Computer*, 1996, **29**(3), pp. 31-44.

[45] Jayant, N.S. and P. Noll, *Digital Coding of Waveforms*, 1984, Englewood Cliffs, NJ, Prentice-Hall.

[46] Jiang, L., H.W. Hon, and X.D. Huang, "Improvements on a Trainable Letter-to-Sound Converter," *Eurospeech*, 1997, Rhodes, Greece.

[47] Juang, B.H., W. Chou, and C.H. Lee, "Statistical and Discriminative Methods for Speech Recognition" in *Automatic Speech and Speaker Recognition – Advanced Topics*, C.H. Lee, F.K. Soong, and K.K. Paliwal, eds., 1996, Boston, MA, pp. 109-132, Kluwer Academic Publishers.

[48] Juang, B.H. and S. Katagiri, "Discriminative Learning for Minimum Error Classification," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1992, **SP-40**(12), pp. 3043-3054.

[49] Kanal, L.N. and B. Chandrasekaran, "On Dimensionality and Sample Size in Statistical Pattern Classification," *Proc. of NEC*, 1968, **24**, pp. 2-7.

[50] Kanal, L.N. and N.C. Randall, "Recognition System Design by Statistical Analysis," *ACM Proc. of 19th National Conf.*, 1964, pp. D2.5-1-D2.5-10.

[51] Kohonen, T., *Learning Vector Quantization for Pattern Recognition*, 1986, Helsinki University of Technology, Finland.

[52] Kuhn, R. and R.D. Mori, "The Application of Semantic Classification Trees to Natural Language Understanding," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 1995(7), pp. 449-460.

[53] Lachenbruch, P.A. and M.R. Mickey, "Estimation of Error Rates in Discriminant Analysis," *Technometrics*, 1968, **10**, pp. 1-11.

[54] Laplace, P.S., *Theorie Analytique des Probabilities*, 1812, Paris, Courcier.

[55] Lee, D.S., S.N. Srihari, and R. Gaborski, "Bayesian and Neural Network Pattern Recognition: A Theoretical Connection and Empirical Results with Handwriting Characters" in *Artificial Neural Network and Statistical Pattern Recognition: Old and New Connections*, I.K. Sethi and A.K. Jain, eds., 1991, Amsterdam, North-Holland.

[56] Linde, Y., A. Buzo, and R.M. Gray, "An Algorithm for Vector Quantizer Design," *IEEE Trans. on Communication*, 1980, **COM-28**(1), pp. 84-95.

[57] Lippmann, R.P., "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, 1987, pp. 4-22.

[58] Lippmann, R.P., "Review of Neural Nets for Speech Recognition," *Neural Computation*, 1989, **1**, pp. 1-38.

[59] Lloyd, S.P., "Least Squares Quantization in PCM," *IEEE Trans. on Information Theory*, 1982, **IT-2**, pp. 129-137.

[60] Lucassen, J.M. and R.L. Mercer, "An Information-Theoretic Approach to the Automatic Determination of Phonemic Baseforms," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1984, San Diego, California, pp. 42.5.1-42.5.4.

[61] Makhoul, J., S. Roucos, and H. Gish, "Vector Quantization in Speech Coding," *Proc. of the IEEE*, 1985, **73**(11), pp. 1551-1588.

[62] Martin, J.K., "An Exact Probability Metric for Decision Tree Splitting and Stopping," *Articial Intelligence and Statistics*, 1995, **5**, pp. 379-385.

[63] McClelland, J., "The Programmable Blackboard Model" in *Parallel Distributed Processing – Explorations in the Microstructure of Cognition, Volume II: Psychological and Biological Models* 1986, Cambridge, MA, MIT Press.

[64] McCulloch, W.S. and W. Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, 1943.

[65] McLachlan, G. and T. Krishnan, *The EM Algorithm and Extensions*, 1996, New York, NY, Wiley Interscience.

[66] McNemar, E.L., "Note on the Sampling Error of the Difference Between Correlated Proportions or Percentages," *Psychometrika*, 1947, **12**, pp. 153-157.

[67] Meisel, W.S., *et al.*, "The SSI Large-Vocabulary Speaker-Independent Continuous Speech Recognition System," 1991, pp. 337-340.

[68] Minsky, M. and S. Papert, *Perceptrons*, 1969, Cambridge, MA, MIT Press.

[69] Mitchell, T., *Machine Learning*, McGraw-Hill Series in Computer Science, 1997, McGraw-Hill.

[70] Morgan, J.N. and J.A. Sonquist, "Problems in the Analysis of Survey Data and a Proposal," *Journal of American Statistics Association*, 1962, **58**, pp. 415-434.

[71] Nadas, A., "A Decision-Theoretic Formulation of a Training Problem in Speech Recognition and a Comparison of Training by Unconditional Versus Conditional Maximum Likelihood," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1983, **4**, pp. 814-817.

[72] Newton, I., *Philosophiae Naturalis Principlea Mathematics*, 1687, London, Royal Society Press.

[73] Neyman, J. and E.S. Pearson, "On the Problem of the Most Efficient Tests of Statistical Hypotheses," *Philosophical Trans. of Royal Society*, 1928, **231**, pp. 289-337.

[74] Ostendorf, M. and N. Velleux, "A Hierarchical Stochastic Model for Automatic Prediction of Prosodic Boundary Location," *Computational Linguistics*, 1995, **20**(1), pp. 27-54.

[75] Paliwal, K. and W.B. Kleijn, "Quantization of LPC Parameters" in *Speech Coding and Synthesis*, W.B. Kleijn and K.K. Paliwal, eds., 1995, Amsterdam, pp. 433-466, Elsevier.

[76] Paul, D.B., "An 800 bps Adaptive Vector Quantization in Speech Coding," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1983, pp. 73-76.

[77] Paul, D.B., "The Lincoln Tied-Mixture HMM Continuous Speech Recognizer" in *Morgan Kaufmann Publishers* 1990, San Mateo, CA, pp. 332-336.

[78] Payne, H.J. and W.S. Meisel, "An Algorithm for Constructing Optimal Binary Decision Trees," *IEEE Trans. on Computers*, 1977, **C-26**(September), pp. 905-916.

[79] Pearson, K., "Contributions to The Mathematical Theorem of Evolution," *Philosophical Trans. of Royal Society*, 1894, **158A**, pp. 71-110.

[80] Pitrelli, J. and V. Zue, "A Hierarchical Model for Phoneme Duration in American English" in *Proc. of Eurospeech*, 1989.

[81] Press, W.H., *et al.*, *Numerical Recipes in C*, 1988, Cambridge, UK, Cambridge University Press.

[82] Quinlan, J.R., "Introduction of Decision Trees" in *Machine Learning: An Artifical Intelligence Approach*, R. Michalski, J. Carbonell, and T. Mitchell, eds., 1986, Boston, M.A., pp. 1-86, Kluwer Academic Publishers.

[83] Quinlan, J.R., "Generating Production Rules from Decision Trees," *Int. Joint Conf. on Artificial Intelligence*, 1987, pp. 304-307.

[84] Quinlan, J.R., *C4.5: Programs for Machine Learning*, 1993, San Francisco, Morgan Kaufmann.

[85] Rabiner, L. and B.H. Juang, "Speech Recognition System Design and Implementation Issues" in *Fundamentals of Speech Recognition*, L. Rabiner and B.H. Juang, eds., 1993, Englewood Cliffs, NJ, Prentice Hall, pp. 242-340.

[86] Reddy, D.R., "Speech Recognition by Machine: A Review," *IEEE Proc.*, 1976, **64**(4), pp. 502-531.

[87] Rosenblatt, F., "The Perceptron – A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review*, 1958, **65**, pp. 386-408.

[88] Rummelhart, D.E., G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation" in *Parallel Distributed Processing*, D.E. Rumelhart and J.L. McClelland, eds., 1986, Cambridge, MA, MIT Press, pp. 318-362.

[89] Rummelhart, D.E., G.E. Hinton, and R.J. Williams, "Learning Representations by Back-Propagating Errors," *Nature*, 1986, **323**, pp. 533-536.

[90] Rummelhart, D.E. and J.L. McClelland, *Parallel Distributed Processing – Explorations in the Microstructure of Cognition, Volume I: Foundations*, 1986, Cambridge, MA, MIT Press.

[91] Rummelhart, D.E. and J.L. McClelland, *Parallel Distributed Processing – Explorations in the Microstructure of Cognition, Volume II: Psychological and Biological Models*, 1986, Cambridge, MA, MIT Press.

[92] Schalkoff, R.J., *Digital Image Processing and Computer Vision*, 1989, New York, NY, John Wiley & Sons.

[93] Sejnowski, T.J. and C.R. Rosenberg, "Parallel Networks that Learn to Pronounce English Text," *Complex Systems*, 1987, **1**, pp. 145-168.

[94] Sejnowski, T.J., *et al.*, "Combining Visual and Acoustic Speech Signals with a Neural Network Improve Intelligibility" in *Advances in Neural Information Processing Systems* 1990, San Mateo, CA, Morgan Kaufmann, pp. 232-239.

[95] Sparkes, J.J., "Pattern Recognition and a Model of the Brain," *Int. Journal of Man-Machine Studies*, 1969, **1**(3), pp. 263-278.

[96] Tappert, C., C.Y. Suen, and T. Wakahara, "The State of the Art in On-Line Handwriting Recognition," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 1990, **12**(8), pp. 787-808.

[97] Tesauro, G. and T.J. Sejnowski, "A Neural Network That Learns to Play Backgammon," *Neural Information Processing Systems, American Institute of Physics*, 1988, pp. 794-803.

[98] White, H., "Learning in Artificial Neural Networks: A Statistical Perspective," *Neural Computation*, 1989, **1**(4), pp. 425-464.

[99] Winston, P.H., *Artificial Intelligence*, 1984, Reading, MA, Addison-Wesley.

[100] Wu, C.F.J., "On the Convergence Properties of the EM Algorithm," *The Annals of Statistics*, 1983, **11**(1), pp. 95-103.

[101] Young, J.Z., *Programmes of the Brain*, 1975, Oxford, England, Oxford University Press.

PART II

SPEECH PROCESSING

CHAPTER 5

Digital Signal Processing

One of the most popular ways of characterizing speech is in terms of a *signal* or acoustic waveform. Shown in Figure 5.1 is a representation of the speech signal that ensures that the information content can be easily extracted by human listeners or computers. This is why digital signal processing plays a fundamental role for spoken language processing. We describe here the fundamentals of digital signal processing: digital signals and systems, frequency-domain transforms for both continuous and discrete frequencies, digital filters, the relationship between analog and digital signals, filterbanks, and stochastic processes. In this chapter we set the mathematical foundations of frequency analysis that allow us to develop specific techniques for speech signals in Chapter 6.

The main theme of this chapter is the development of frequency-domain methods computed through the Fourier transform. When we boost the bass knob in our amplifier we are increasing the gain at low frequencies, and when we boost the treble knob we are increasing the gain at high frequencies. Representation of speech signals in the frequency domain is especially useful because the frequency structure of a phoneme is generally unique.

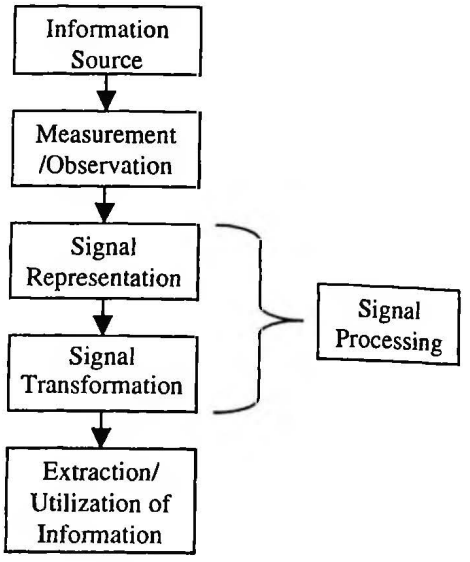


Figure 5.1 Signal processing is both a representation and a transformation that allows a useful information extraction from a source. The representation and transformation are based on a model of the signal, often parametric, that is convenient for subsequent processing.

5.1. DIGITAL SIGNALS AND SYSTEMS

To process speech signals, it is convenient to represent them mathematically as functions of a continuous variable t, which represents time. Let us define an *analog signal* $x_a(t)$ as a function varying continuously in time. If we sample the signal x with a sampling period T (i.e., $t = nT$), we can define a discrete-time signal as $x[n] = x_a(nT)$, also known as *digital signal.*[1] In this book we use parentheses to describe an analog signal and brackets for digital signals. Furthermore we can define the sampling frequency F_s as $F_s = 1/T$, the inverse of the sampling period T. For example, for a sampling rate $F_s = 8\,\text{kHz}$, its corresponding sampling period is 125 microseconds. In Section 5.5 it is shown that, under some circumstances, the analog signal $x_a(t)$ can be recovered exactly from the digital signal $x[n]$. Figure 5.2 shows an analog signal and its corresponding digital signal. In subsequent figures, for convenience, we will sometimes plot digital signals as continuous functions.

The term *Digital Signal Processing* (DSP) refers to methods for manipulating the sequence of numbers $x[n]$ in a digital computer. The acronym DSP is also used to refer to a *Digital Signal Processor*, i.e., a microprocessor specialized to perform DSP operations.

[1] Actually the term digital signal is defined as a discrete-time signal whose values are represented by integers within a range, whereas a general discrete-time signal would be represented by real numbers. Since the term digital signal is much more commonly used, we will use that term, except when the distinction between them is necessary.

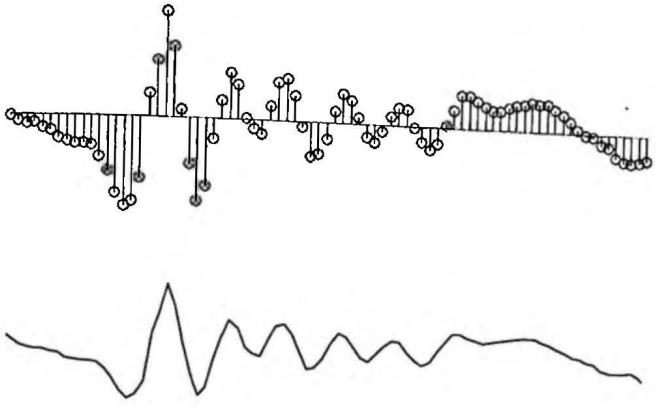


Figure 5.2 Analog signal and its corresponding digital signal.

We start with sinusoidal signals and show they are the fundamental signals for linear systems. We then introduce the concept of convolution and linear time-invariant systems. Other digital signals and nonlinear systems are also introduced.

5.1.1. Sinusoidal Signals

One of the most important signals is the sine wave or *sinusoid*

$$x_0[n] = A_0 \cos(\omega_0 n + \phi_0) \tag{5.1}$$

where A_0 is the sinusoid's amplitude, ω_0 the angular frequency, and ϕ_0 the phase. The angle in the trigonometric functions is expressed in radians, so that the angular frequency ω_0 is related to the normalized linear frequency f_0 by the relation $\omega_0 = 2\pi f_0$, and $0 \le f_0 \le 1$. This signal is *periodic*[2] with period $T_0 = 1/f_0$. In Figure 5.3 we can see an example of a sinusoid with frequency $f_0 = 0.04$, or a period of $T_0 = 25$ samples.

Sinusoids are important because speech signals can be decomposed as sums of sinusoids. When we boost the bass knob in our amplifier we are increasing the gain for sinusoids of low frequencies, and when we boost the treble knob we are increasing the gain for sinusoids of high frequencies.

[2] A signal $x[n]$ is periodic with period N if and only if $x[n]=x[n+N]$, which requires $\omega_0 = 2\pi / N$. This means that the digital signal in Eq. (5.1) is not periodic for all values of ω_0, even though its continuous signal counterpart $x(t) = A_0 \cos(\omega_0 t + \phi_0)$ is periodic for all values of ω_0 (see Section 5.5).

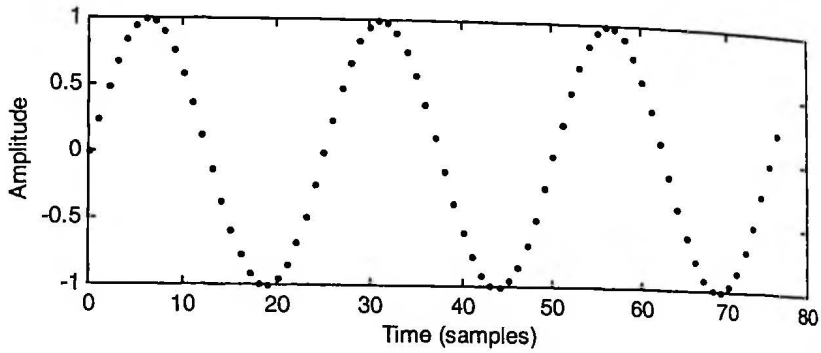


Figure 5.3 A digital sinusoid with a period of 25 samples.

What is the sum of two sinusoids $x_0[n]$ and $x_1[n]$ of the same frequency ω_0 but different amplitudes A_0 and A_1, and phases ϕ_0 and ϕ_1? The answer is another sinusoid of the same frequency but a different amplitude A and phase ϕ. While this can be computed through trigonometric identities, it is somewhat tedious and not very intuitive. For this reason we introduce another representation based on complex numbers, which proves to be very useful when we study digital filters.

A complex number x can be expressed as $z = x+jy$, where $j = \sqrt{-1}$, x is the real part and y is the imaginary part, with both x and y being real numbers. Using Euler's relation, given a real number ϕ, we have

$$e^{j\phi} = \cos\phi + j\sin\phi \tag{5.2}$$

so that the complex number z can also be expressed in polar form as $z = Ae^{j\phi}$, where A is the amplitude and ϕ is the phase. Both representations can be seen in Figure 5.4, where the real part is shown in the abscissa (x-axis) and the imaginary part in the ordinate (y-axis).

Using complex numbers, the sinusoid in Eq. (5.1) can be expressed as the real part of the corresponding complex exponential

$$x_0[n] = A_0\cos(\omega_0 n + \phi_0) = \mathrm{Re}\{A_0 e^{j(\omega_0 n+\phi_0)}\} \tag{5.3}$$

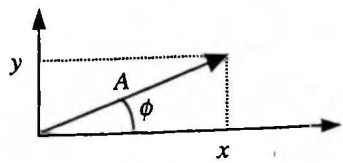


Figure 5.4 Complex number representation in Cartesian form $z = x + jy$ and polar form $z = Ae^{j\phi}$. Thus $x = A\cos\phi$ and $y = A\sin\phi$.

and thus the sum of two complex exponential signals equals

$$A_0 e^{j(\omega_0 n+\phi_0)} + A_1 e^{j(\omega_0 n+\phi_1)} = e^{j\omega_0 n}\left(A_0 e^{j\phi_0} + A_1 e^{j\phi_1}\right) = e^{j\omega_0 n} A e^{j\phi} = A e^{j(\omega_0 n+\phi)} \tag{5.4}$$

Taking the real part in both sides results in

$$A_0 \cos(\omega_0 n+\phi_0) + A_2 \cos(\omega_0 n+\phi_1) = A\cos(\omega_0 n+\phi) \tag{5.5}$$

or in other words, the sum of two sinusoids of the same frequency is another sinusoid of the same frequency.

To compute A and ϕ, dividing Eq. (5.4) by $e^{j\omega_0 n}$ leads to a relationship between the amplitude A and phase ϕ:

$$A_0 e^{j\phi_0} + A_1 e^{j\phi_1} = A e^{j\phi} \tag{5.6}$$

Equating real and imaginary parts in Eq. (5.6) and dividing them we obtain:

$$\tan\phi = \frac{A_0 \sin\phi_0 + A_1 \sin\phi_1}{A_0 \cos\phi_0 + A_1 \cos\phi_1} \tag{5.7}$$

and adding the squared of real and imaginary parts and using trigonometric identities[3]

$$A^2 = A_0^2 + A_1^2 + 2A_0 A_1 \cos(\phi_0 - \phi_1) \tag{5.8}$$

This complex representation of Figure 5.5 lets us analyze and visualize the amplitudes and phases of sinusoids of the same frequency as vectors. The sum of N sinusoids of the same frequency is another sinusoid of the same frequency that can be obtained by adding the real and imaginary parts of all complex vectors. In Section 5.2.1 we show that the output of a linear time-invariant system to a sinusoid is another sinusoid of the same frequency.

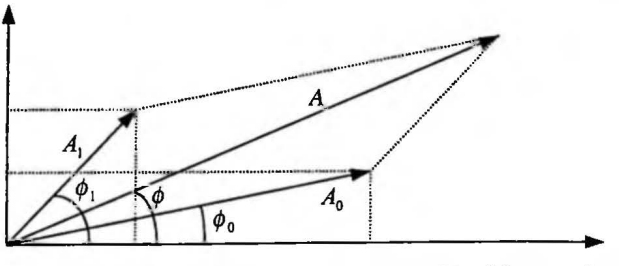


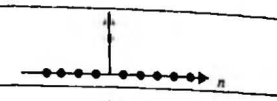
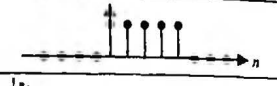
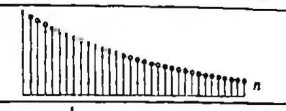
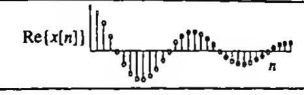
Figure 5.5 Geometric representation of the sum of two sinusoids of the same frequency. It follows the complex number representation in Cartesian form of Figure 5.4.

[3] $\sin^2\phi + \cos^2\phi = 1$ and $\cos(a-b) = \cos a \cos b + \sin a \sin b$.

5.1.2. Other Digital Signals

In the field of digital signal processing there are other signals that repeatedly arise and that are shown in Table 5.1.

Table 5.1 Some useful digital signals: the Kronecker delta, unit step, rectangular signal, real exponential ($a<1$) and real part of a complex exponential ($r<1$).

Kronecker delta, or unit impulse	$\delta[n]=\begin{cases}1 & n=0\\ 0 & otherwise\end{cases}$	
Unit step	$u[n]=\begin{cases}1 & n\geq 0\\ 0 & n<0\end{cases}$	
Rectangular signal	$\text{rect}_N[n]=\begin{cases}1 & 0\leq n<N\\ 0 & otherwise\end{cases}$	
Real exponential	$x[n]=a^n u[n]$	
Complex exponential	$x[n]=a^n u[n]=r^n e^{jn\omega_0}u[n]$ $=r^n(\cos n\omega_0+j\sin n\omega_0)u[n]$	

If $r=1$ and $\omega_0\neq 0$ we have a complex sinusoid as shown in Section 5.1.1. If $\omega_0=0$ we have a real exponential signal, and if $r<1$ and $\omega_0\neq 0$ we have an exponentially decaying oscillatory sequence, also known as a damped sinusoid.

5.1.3. Digital Systems

A digital system is a system that, given an input signal $x[n]$, generates an output signal $y[n]$:

$$y[n]=T\{x[n]\} \tag{5.9}$$

whose input/output relationship can be seen in Figure 5.6.

In general, a digital system T is defined to be linear *iff* (if and only if)

$$T\{a_1x_1[n]+a_2x_2[n]\}=a_1T\{x_1[n]\}+a_2T\{x_2[n]\} \tag{5.10}$$

for any values of a_1, a_2 and any signals $x_1[n]$ and $x_2[n]$.

Here, we study systems according to whether or not they are linear and/or time invariant.

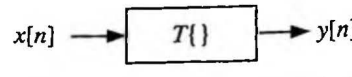


Figure 5.6 Block diagram of a digital system whose input is digital signal $x[n]$, and whose output is digital signal $y[n]$.

5.1.3.1. Linear Time-Invariant Systems

A system is *time-invariant* if given Eq. (5.9), then

$$y[n-n_0]=T\{x[n-n_0]\} \tag{5.11}$$

Linear digital systems of a special type, the so-called *linear time-invariant* (LTI),[4] are described by

$$y[n]=\sum_{k=-\infty}^{\infty} x[k]h[n-k]=x[n]*h[n] \tag{5.12}$$

where $*$ is defined as the *convolution* operator. It is left to the reader to show that the linear system in Eq. (5.12) indeed satisfies Eq. (5.11).

LTI systems are completely characterized by the signal $h[n]$, which is known as the system's *impulse response* because it is the output of the system when the input is an impulse $x[n]=\delta[n]$. Most of the systems described in this book are LTI systems.

Table 5.2 Properties of the convolution operator.

Commutative	$x[n]*h[n]=h[n]*x[n]$
Associative	$x[n]*(h_1[n]*h_2[n])=(x[n]*h_1[n])*h_2[n]=x[n]*h_1[n]*h_2[n]$
Distributive	$x[n]*(h_1[n]+h_2[n])=x[n]*h_1[n]+x[n]*h_2[n]$

The convolution operator is commutative, associative and distributive as shown in Table 5.2 and Figure 5.7.

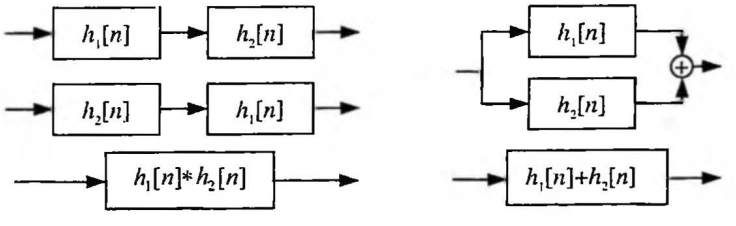


Figure 5.7 The block diagrams on the left, representing the commutative property, are equivalent. The block diagrams on the right, representing the distributive property, are also equivalent.

[4] Actually the term linear time-invariant (LTI) systems is typically reserved for continuous or analog systems, and linear shift-invariant system is used for discrete-time signals, but we will use LTI for discrete-time signals too since it is widely used in this context.

5.1.3.2. Linear Time-Varying Systems

An interesting type of digital systems is that whose output is a linear combination of the input signal at different times:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]g[n,n-k] \tag{5.13}$$

The digital system in Eq. (5.13) is linear, since it satisfies Eq. (5.10). The Linear Time-Invariant systems of Section 5.1.3.1 are a special case of Eq. (5.13) when $g[n,n-k] = h[n-k]$. The systems in Eq. (5.13) are called *linear time-varying* (LTV) systems, because the weighting coefficients can vary with time.

A useful example of such system is the so-called *amplitude modulator*

$$y[n] = x[n]\cos\omega_0 n \tag{5.14}$$

used in AM transmissions. As we show in Chapter 6, speech signals are the output of LTV systems. Since these systems are difficult to analyze, we often approximate them with linear time-invariant systems.

5.1.3.3. Nonlinear Systems

Many *nonlinear* systems do not satisfy Eq. (5.10). Table 5.3 includes a list of typical nonlinear systems used in speech processing. All these nonlinear systems are memoryless, because the output at time n depends only on the input at time n, except for the *median smoother* of order $(2N + 1)$ whose output depends also on the previous and the following N samples.

5.2. CONTINUOUS-FREQUENCY TRANSFORMS

A very useful transform for LTI systems is the Fourier transform, because it uses complex exponentials as its basis functions, and its generalization: the z-transform. In this section we cover both transforms, which are continuous functions of frequency, and their properties.

5.2.1. The Fourier Transform

It is instructive to see what the output of a LTI system with impulse response $h[n]$ is when the input is a complex exponential. Substituting $x[n] = e^{j\omega_0 n}$ in Eq. (5.12) and using the commutative property of the convolution we obtain

$$y[n] = \sum_{k=-\infty}^{\infty} h[k]e^{j\omega_0(n-k)} = e^{j\omega_0 n}\sum_{k=-\infty}^{\infty} h[k]e^{-j\omega_0 k} = e^{j\omega_0 n}H(e^{j\omega_0}) \tag{5.15}$$

Table 5.3 Examples of nonlinear systems for speech processing. All of them are memoryless except for the median smoother.

Nonlinear System	Equation
Median Smoother of order $(2N+1)$	$y[n] = \text{median}\{x[n-N],\cdots,x[n],\cdots,x[n+N]\}$
Full-Wave Rectifier	$y[n] = \lvert x[n] \rvert$
Half-Wave Rectifier	$y[n] = \begin{cases} x[n] & x[n] \geq 0 \\ 0 & x[n] < 0 \end{cases}$
Frequency Modulator	$y[n] = A\cos(\omega_0 + \Delta\omega x[n])n$
Hard-Limiter	$y[n] = \begin{cases} A & x[n] \geq A \\ x[n] & \lvert x[n] \rvert < A \\ -A & x[n] \leq -A \end{cases}$
Uniform Quantizer (L-bit) with $2N = 2^L$ intervals of width Δ	$y[n] = \begin{cases} (N-1/2)\Delta & x[n] \geq (N-1)\Delta \\ (m+1/2)\Delta & m\Delta \leq x[n] < (m+1)\Delta \quad 0 \leq m < N-1 \\ (-m+1/2)\Delta & -m\Delta \leq x[n] < -(m-1)\Delta \quad 0 < m < N-1 \\ (-N+1/2)\Delta & x[n] < -(N-1)\Delta \end{cases}$

which is another complex exponential of the same frequency and amplitude multiplied by the complex quantity $H(e^{j\omega_0})$ given by

$$H(e^{j\omega}) = \sum_{n=-\infty}^{\infty} h[n]e^{-j\omega n} \tag{5.16}$$

Since the output of a LTI system to a complex exponential is another complex exponential, it is said that complex exponentials are *eigensignals* of LTI systems, with the complex quantity $H(e^{j\omega_0})$ being their *eigenvalue*.

The quantity $H(e^{j\omega})$ is defined as the *discrete-time Fourier transform* of $h[n]$. It is clear from Eq. (5.16) that $H(e^{j\omega})$ is a periodic function of ω with period 2π, and therefore we need to keep only one period to fully describe it, typically $-\pi < \omega < \pi$ (Figure 5.8).

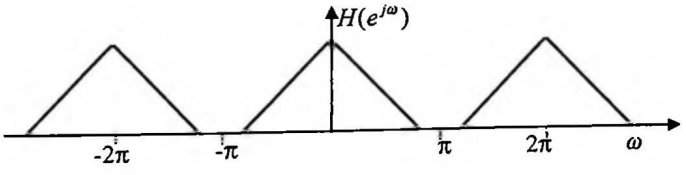


Figure 5.8 $H(e^{j\omega})$ is a periodic function of ω.

$H(e^{j\omega})$ is a complex function of ω which can be expressed in terms of the real and imaginary parts:

$$H(e^{j\omega}) = H_r(e^{j\omega}) + jH_i(e^{j\omega}) \tag{5.17}$$

or in terms of the magnitude and phase as

$$H(e^{j\omega}) = \left|H(e^{j\omega})\right| e^{j\arg\{H(e^{j\omega})\}} \tag{5.18}$$

Thus if the input to the LTI system is a sinusoid as in Eq. (5.1), the output will be

$$y_0[n] = A_0 \left|H(e^{j\omega_0})\right| \cos\left(\omega_0 n + \phi_0 + \arg\{H(e^{j\omega_0})\}\right) \tag{5.19}$$

according to Eq. (5.15). Therefore if $\left|H(e^{j\omega_0})\right| > 1$, the LTI system will amplify that frequency, and likewise it will attenuate, or *filter* it, if $\left|H(e^{j\omega_0})\right| < 1$. That is one reason why these systems are also called filters. The Fourier transform $H(e^{j\omega})$ of a filter $h[n]$ is called the system's *frequency response* or *transfer function.*

The angular frequency ω is related to the normalized linear frequency f by the simple relation $\omega = 2\pi f$. We show in Section 5.5 that linear frequency f_l and normalized frequency f are related by $f_l = fF_s$, where F_s is the sampling frequency.

The inverse *discrete-time Fourier transform* is defined as

$$h[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} H(e^{j\omega}) e^{j\omega n} d\omega \tag{5.20}$$

The Fourier transform is invertible, and Eq. (5.16) and (5.20) are transform pairs:

$$\begin{aligned} h[n] &= \frac{1}{2\pi}\int_{-\pi}^{\pi} H(e^{j\omega}) e^{j\omega n} d\omega = \frac{1}{2\pi}\int_{-\pi}^{\pi}\left(\sum_{m=-\infty}^{\infty} h[m] e^{-j\omega m}\right) e^{j\omega n} d\omega \\ &= \sum_{m=-\infty}^{\infty} h[m] \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{j\omega(n-m)} d\omega = \sum_{m=-\infty}^{\infty} h[m]\delta[n-m] = h[n] \end{aligned} \tag{5.21}$$

since

$$\frac{1}{2\pi}\int_{-\pi}^{\pi} e^{j\omega(n-m)} d\omega = \delta[n-m] \tag{5.22}$$

A sufficient condition for the existence of the Fourier transform is

$$\sum_{n=-\infty}^{\infty} \left|h[n]\right| < \infty \tag{5.23}$$

Although we have computed the Fourier transform of the impulse response of a filter $h[n]$, Eq. (5.16) and (5.20) can be applied to any signal $x[n]$.

5.2.2. Z-Transform

The *z-transform* is a generalization of the Fourier transform. The z-transform of a digital signal $h[n]$ is defined as

$$H(z) = \sum_{n=-\infty}^{\infty} h[n] z^{-n} \tag{5.24}$$

where z is a complex variable. Indeed, the Fourier transform of $h[n]$ equals its z-transform evaluated at $z = e^{j\omega}$. While the Fourier and z-transforms are often used interchangeably, we normally use the Fourier transform to plot the filter's frequency response, and the z-transform to analyze more general filter characteristics, given its polynomial functional form. We can also use the z-transform for unstable filters, which do not have Fourier transforms.

Since Eq. (5.24) is an infinite sum, it is not guaranteed to exist. A sufficient condition for convergence is:

$$\sum_{n=-\infty}^{\infty} |h[n]| |z|^{-n} < \infty \tag{5.25}$$

which is true only for a *region of convergence* (ROC) in the complex z-plane $R_1 < |z| < R_2$ as indicated in Figure 5.9.

For a signal $h[n]$ to have a Fourier transform, its z-transform $H(z)$ has to include the unit circle, $|z| = 1$, in its convergence region. Therefore, a sufficient condition for the existence of the Fourier transform is given in Eq. (5.23) by applying Eq. (5.25) to the unit circle.

An LTI system is defined to be *causal* if its impulse response is a causal signal, i.e. $h[n] = 0$ for $n < 0$. Similarly, a LTI system is *anti-causal* if $h[n] = 0$ for $n > 0$. While all physical systems are causal, noncausal systems are still useful since causal systems could be decomposed into causal and anti-causal systems.

A system is defined to be *stable* if for every bounded input it produces a bounded output. A necessary and sufficient condition for an LTI system to be stable is

$$\sum_{n=-\infty}^{\infty} |h[n]| < \infty \tag{5.26}$$

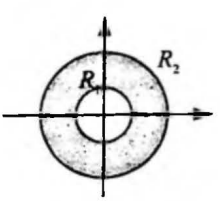


Figure 5.9 Region of convergence of the z-transform in the complex plane.

which means, according to Eq. (5.23), that $h[n]$ has a Fourier transform, and therefore that its z-transform includes the unit circle in its region of convergence.

Just as in the case of Fourier transforms, we can use the z-transform for any signal, not just for a filter's impulse response.

The *inverse z-transform* is defined as

$$h[n] = \frac{1}{2\pi j}\oint H(z)z^{n-1}dz \tag{5.27}$$

where the integral is performed along a closed contour that is within the *region of convergence*. Eqs. (5.24) and (5.27) plus knowledge of the region of convergence form a transform pair: *i.e.* one can be exactly determined if the other is known. If the integral is performed along the unit circle (i.e., doing the substitution $z = e^{j\omega}$) we obtain Eq. (5.20), the inverse Fourier transform.

5.2.3. Z-Transforms of Elementary Functions

In this section we compute the z-transforms of the signals defined in Table 5.1. The z-transforms of such signals are summarized in Table 5.4. In particular we compute the z-transforms of left-sided and right-sided complex exponentials, which are essential to compute the inverse z-transform of rational polynomials. As we see in Chapter 6, speech signals are often modeled as having z-transforms that are rational polynomials.

Table 5.4 Z-transforms of some useful signals together with their region of convergence.

Signal	Z-Transform	Region of Convergence
$h_1[n] = \delta[n-N]$	$H_1(z) = z^{-N}$	$z \neq 0$
$h_2[n] = u[n]-u[n-N]$	$H_2(z) = \frac{1-z^{-N}}{1-z^{-1}}$	$z \neq 0$
$h_3[n] = a^n u[n]$	$H_3(z) = \frac{1}{1-az^{-1}}$	$\lvert a\rvert < \lvert z\rvert$
$h_4[n] = -a^n u[-n-1]$	$H_4(z) = \frac{1}{1-az^{-1}}$	$\lvert z\rvert < \lvert a\rvert$

5.2.3.1. Right-Sided Complex Exponentials

A right-sided complex exponential sequence

$$h_3[n] = a^n u[n] \tag{5.28}$$

has a z-transform given by

$$H_3(z) = \sum_{n=0}^{\infty} a^n z^{-n} = \lim_{N\to\infty} \frac{1-(az^{-1})^N}{1-az^{-1}} = \frac{1}{1-az^{-1}} \quad \text{for} \quad |a| < |z| \tag{5.29}$$

by using the sum of the terms of a geometric sequence and making $N \to \infty$. This region of convergence ($|a| < |z|$) is typical of causal signals (those that are zero for $n < 0$).

When a z-transform is expressed as the ratio of two polynomials, the roots of the numerator are called *zeros*, and the roots of the denominator are called *poles*. Zeros are the values of z for which the z-transform equals 0, and poles are the values of z for which the z-transform equals infinity.

$H_3(z)$ has a pole at $z = a$, because its value goes to infinity at $z = a$. According to Eq. (5.26), $h_3[n]$ is a stable signal if and only if $|a| < 1$, or in other words, if its pole is inside the unit circle. In general, a causal and stable system has all its poles inside the unit circle. As a corollary, a system which has poles outside the unit circle is either noncausal or unstable or both. This is a very important fact, which we exploit throughout the book.

5.2.3.2. Left-Sided Complex Exponentials

A left-sided complex exponential sequence

$$h_4[n] = -a^n u[-n-1] \tag{5.30}$$

has a z-transform given by

$$H_4(z) = -\sum_{n=-\infty}^{-1} a^n z^{-n} = -\sum_{n=1}^{\infty} a^{-n} z^n = 1 - \sum_{n=0}^{\infty} a^{-n} z^n \qquad \text{for } |z| < |a| \tag{5.31}$$

$$= 1 - \frac{1}{1-a^{-1}z} = \frac{-a^{-1}z}{1-a^{-1}z} = \frac{1}{1-az^{-1}}$$

This region of convergence ($|z| < |a|$) is typical of noncausal signals (those that are nonzero for $n < 0$). Observe that $H_3(z)$ and $H_4(z)$ are functionally identical and only differ in the region of convergence. In general, the region of convergence of a signal that is nonzero for $-\infty < n < \infty$ is $R_1 < |z| < R_2$.

5.2.3.3. Inverse Z-Transform of Rational Functions

Integrals in the complex plane such as Eq. (5.27) are not easy to do, but fortunately they are not necessary for the special case of $H(z)$ being a rational polynomial transform. In this case, partial fraction expansion can be used to decompose the signal into a linear combination of signals like $h_1[n]$, $h_3[n]$ and $h_4[n]$ as in Table 5.4.

For example,

$$H_5(z)=\frac{2+8z^{-1}}{2-5z^{-1}-3z^{-2}} \tag{5.32}$$

has as roots of its denominator $z=3,-1/2$. Therefore it can be decomposed as

$$H_5(z)=\frac{A}{1-3z^{-1}}+\frac{B}{1+(1/2)z^{-1}}=\frac{(2A+2B)+(A-6B)z^{-1}}{2-5z^{-1}-3z^{-2}} \tag{5.33}$$

so that A and B are the solution of the following set of linear equations:

$$\begin{aligned} 2A+2B&=2\\ A-6B&=8 \end{aligned} \tag{5.34}$$

whose solution is $A=2$ and $B=-1$, and thus Eq. (5.33) is expressed as

$$H_5(z)=2\left(\frac{1}{1-3z^{-1}}\right)-\left(\frac{1}{1+(1/2)z^{-1}}\right) \tag{5.35}$$

However, we cannot compute the inverse z-transform unless we know the region of convergence. If, for example, we are told that the region of convergence includes the unit circle (necessary for the system to be stable), then the inverse transform of

$$H_4(z)=\frac{1}{1-3z^{-1}} \tag{5.36}$$

must have a region of convergence of $|z|<3$ according to Table 5.4, and thus be a left-sided complex exponential:

$$h_4[n]=-3^n u[-n-1] \tag{5.37}$$

and the transform of

$$H_3(z)=\frac{1}{1+(1/2)z^{-1}} \tag{5.38}$$

must have a region of convergence of $1/2<|z|$ according to Table 5.4, and thus be a right-sided complex exponential:

$$h_3[n]=(-1/2)^n u[n] \tag{5.39}$$

so that

$$h_5[n]=-2\cdot 3^n u[-n-1]-(-1/2)^n u[n] \tag{5.40}$$

While we only showed an example here, the method used generalizes to rational transfer functions with more poles and zeros.

5.2.4. Properties of the Z- and Fourier Transforms

In this section we include a number of properties that are used throughout the book and that can be derived from the definition of Fourier and z-transforms (see Table 5.5). Of special interest are the convolution property and Parseval's theorem, which are described below.

5.2.4.1. The Convolution Property

The z-transform of $y[n]$, convolution of $x[n]$ and $h[n]$, can be expressed as a function of their z-transforms:

$$\begin{aligned} Y(z) &= \sum_{n=-\infty}^{\infty} y[n]z^{-n} = \sum_{n=-\infty}^{\infty}\left(\sum_{k=-\infty}^{\infty} x[k]h[n-k]\right)z^{-n} \\ &= \sum_{k=-\infty}^{\infty} x[k]\left(\sum_{n=-\infty}^{\infty} h[n-k]z^{-n}\right) = \sum_{k=-\infty}^{\infty} x[k]\left(\sum_{n=-\infty}^{\infty} h[n]z^{-(n+k)}\right) \\ &= \sum_{k=-\infty}^{\infty} x[k]z^{-k}H(z) = X(z)H(z) \end{aligned} \tag{5.41}$$

which is the fundamental property of LTI systems: "The z-transform of the convolution of two signals is the product of their z-transforms." This is also known as the convolution property. The ROC of $Y(z)$ is now the intersection of the ROCs of $X(z)$ and $H(z)$ and cannot be empty for $Y(z)$ to exist.

Likewise, we can obtain a similar expression for the Fourier transforms:

$$Y(e^{j\omega}) = X(e^{j\omega})H(e^{j\omega}) \tag{5.42}$$

A dual version of the convolution property can be proven for the product of digital signals:

$$x[n]y[n] \leftrightarrow \frac{1}{2\pi} X(e^{j\omega}) * Y(e^{j\omega}) \tag{5.43}$$

whose transform is the continuous convolution of the transforms with a scale factor. The convolution of functions of continuous variables is defined as

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau \tag{5.44}$$

Note how this differs from the discrete convolution of Eq. (5.12).

5.2.4.2. Power Spectrum and Parseval's Theorem

Let's define the *autocorrelation* of signal $x[n]$ as

$$R_{xx}[n] = \sum_{m=-\infty}^{\infty} x[m+n]x^*[m] = \sum_{l=-\infty}^{\infty} x[l]x^*[-(n-l)] = x[n] * x^*[-n] \tag{5.45}$$

where the superscript asterisk (*) means complex conjugate[5] and should not be confused with the convolution operator.

Using the fundamental property of LTI systems in Eq. (5.42) and the symmetry properties in Table 5.5, we can express its Fourier transform $S_{xx}(\omega)$ as

$$S_{xx}(\omega) = X(\omega)X^*(\omega) = |X(\omega)|^2 \tag{5.46}$$

which is the *power spectrum*. The Fourier transform of the autocorrelation is the power spectrum:

$$R_{xx}[n] \leftrightarrow S_{xx}(\omega) \tag{5.47}$$

or alternatively

$$R_{xx}[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} S_{xx}(\omega)e^{j\omega n}d\omega \tag{5.48}$$

If we set $n = 0$ in Eq. (5.48) and use Eq. (5.45) and (5.46), we obtain

$$\sum_{n=-\infty}^{\infty} |x[n]|^2 = \frac{1}{2\pi}\int_{-\pi}^{\pi} |X(\omega)|^2 \, d\omega \tag{5.49}$$

which is called *Parseval's theorem* and says that we can compute the signal's energy in the time domain or in the frequency domain.

5.3. DISCRETE-FREQUENCY TRANSFORMS

Here we describe transforms, including the DFT, DCT and FFT, that take our discrete-time signal into a discrete frequency representation. Discrete-frequency transforms are the natural transform for periodic signals, though we show in Section 5.7 and Chapter 6 how they are also useful for aperiodic signals such as speech.

[5] If $z = x + jy = Ae^{j\phi}$, its complex conjugate is defined as $z^* = x - jy = Ae^{-j\phi}$.

Table 5.5 Properties of the Fourier and z-transforms.

Property	Signal	Fourier Transform	z-Transform
Linearity	$ax_1[n]+bx_2[n]$	$aX_1(e^{j\omega})+bX_2(e^{j\omega})$	$aX_1(z)+bX_2(z)$
Symmetry	$x[-n]$	$X(e^{-j\omega})$	$X(z^{-1})$
	$x^*[n]$	$X^*(e^{-j\omega})$	$X^*(z^*)$
	$x^*[-n]$	$X^*(e^{j\omega})$	$X^*(1/z^*)$
	$x[n]$ real	$X(e^{j\omega})$ is *Hermitian* $X(e^{-j\omega})=X^*(e^{j\omega})$ $\lvert X(e^{j\omega})\rvert$ is even[6] $\mathrm{Re}\{X(e^{j\omega})\}$ is even $\arg\{X(e^{j\omega})\}$ is odd[7] $\mathrm{Im}\{X(e^{j\omega})\}$ is odd	$X(z^*)=X^*(z)$
	$\mathrm{Even}\{x[n]\}$	$\mathrm{Re}\{X(e^{j\omega})\}$	
	$\mathrm{Odd}\{x[n]\}$	$j\,\mathrm{Im}\{X(e^{j\omega})\}$	
Time-shifting	$x[n-n_0]$	$X(e^{j\omega})e^{-j\omega n_0}$	$X(z)z^{-n_0}$
Modulation	$x[n]e^{j\omega_0 n}$	$X(e^{j(\omega-\omega_0)})$	$X(e^{-j\omega_0}z)$
	$x[n]z_0^n$		$X(z/z_0)$
Convolution	$x[n]*h[n]$	$X(e^{j\omega})H(e^{j\omega})$	$X(z)H(z)$
	$x[n]y[n]$	$\frac{1}{2\pi}X(e^{j\omega})*Y(e^{j\omega})$	
Parseval's Theorem	$R_{xx}[n]=\sum_{m=-\infty}^{\infty}x[m+n]x^*[m]$	$S_{xx}(\omega)=\lvert X(\omega)\rvert^2$	$X(z)X^*(1/z^*)$

A discrete transform of a signal $x[n]$ is another signal defined as

$$X[k]=\mathrm{T}\{x[n]\} \tag{5.50}$$

Linear transforms are special transforms that decompose the input signal $x[n]$ into a linear combination of other signals:

$$x[n]=\sum_{k=-\infty}^{\infty}X[k]\varphi_k[n] \tag{5.51}$$

[6] A function $f(x)$ is called even if and only if $f(x)=f(-x)$.

[7] A function $f(x)$ is called odd if and only if $f(x)=-f(-x)$.

where $\varphi_k[n]$ is a set of *orthonormal* functions

$$<\varphi_k[n],\varphi_l[n]>=\delta[k-l] \tag{5.52}$$

with the inner product defined as

$$<\varphi_k[n],\varphi_l[n]>=\sum_{n=-\infty}^{\infty}\varphi_k[n]\varphi_l^*[n] \tag{5.53}$$

With this definition, the coefficients $X[k]$ are the projection of $x[n]$ onto $\varphi_k[n]$:

$$X[k]=<x[n],\varphi_k[n]> \tag{5.54}$$

as illustrated in Figure 5.10.

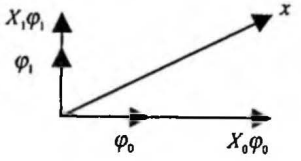


Figure 5.10 Orthonormal expansion of a signal $x[n]$ in a two-dimensional space.

5.3.1. The Discrete Fourier Transform (DFT)

If a $x_N[n]$ signal is periodic with period N then

$$x_N[n]=x_N[n+N] \tag{5.55}$$

and the signal is uniquely represented by N consecutive samples. Unfortunately, since Eq. (5.23) is not met, we cannot guarantee the existence of its Fourier transform. The *Discrete Fourier Transform* (DFT) of a periodic signal $x_N[n]$ is defined as

$$X_N[k]=\sum_{n=0}^{N-1}x_N[n]e^{-j2\pi nk/N} \qquad 0\le k<N \tag{5.56}$$

$$x_N[n]=\frac{1}{N}\sum_{k=0}^{N-1}X_N[k]e^{j2\pi nk/N} \qquad 0\le n<N \tag{5.57}$$

which are transform pairs. Equation (5.57) is also referred as a *Fourier series* expansion.

In Figure 5.11 we see the approximation of a periodic square signal with period $N=100$ as a sum of 19 *harmonic* sinusoids, i.e., we used only the first 19 $X_N[k]$ coefficients in Eq. (5.57).

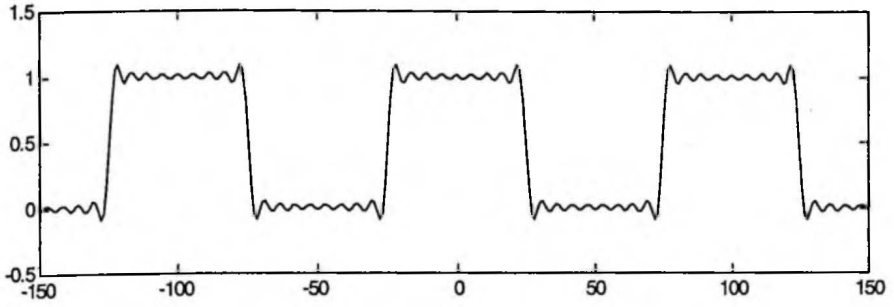


Figure 5.11 Decomposition of a periodic square signal with period 100 samples as a sum of 19 harmonic sinusoids with frequencies $\omega_k = 2\pi k/100$.

$$\tilde{x}_N[n] = \frac{1}{N}\sum_{k=-18}^{18} X_N[k]e^{j2\pi nk/N} = \frac{X_N[0]}{N} + \frac{2}{N}\sum_{k=1}^{18} X_N[k]\cos(2\pi nk/N) \tag{5.58}$$

Had we used 100 harmonic sinusoids, the periodic signal would have been reproduced *exactly*. Nonetheless, retaining a smaller number of sinusoids can provide a decent approximation for a periodic signal.

5.3.2. Fourier Transforms of Periodic Signals

Using the DFT, we now discuss how to compute the Fourier transforms of a complex exponential, an impulse train, and a general periodic signal, since they are signals often used in DSP. We also present a relationship between the continuous-frequency Fourier transform and the discrete Fourier transform.

5.3.2.1. The Complex Exponential

One of the simplest periodic functions is the complex exponential $x[n] = e^{j\omega_0 n}$. Since it has infinite energy, we cannot compute its Fourier transform in its strict sense. Since such signals are so useful, we devise an alternate formulation.

First, let us define the function

$$d_\Delta(\omega) = \begin{cases} 1/\Delta & 0 \le \omega < \Delta \\ 0 & otherwise \end{cases} \tag{5.59}$$

which has the following property

$$\int_{-\infty}^{\infty} d_\Delta(\omega)\,d\omega = 1 \tag{5.60}$$

for all values of $\Delta > 0$.

It is useful to define the continuous delta function $\delta(\omega)$, also known as the *Dirac* delta, as

$$\delta(\omega) = \lim_{\Delta \to 0} d_\Delta(\omega) \tag{5.61}$$

which is a *singular function* and can be seen in Figure 5.12. The Dirac delta is a function of a continuous variable and should not be confused with the Kronecker delta, which is a function of a discrete variable.

Using Eqs. (5.59) and (5.61) we can then see that

$$\int_{-\infty}^{\infty} X(\omega)\delta(\omega)d\omega = \lim_{\Delta \to 0} \int_{-\infty}^{\infty} X(\omega) d_\Delta(\omega) d\omega = X(0) \tag{5.62}$$

and similarly

$$\int_{-\infty}^{\infty} X(\omega)\delta(\omega - \omega_0)d\omega = X(\omega_0) \tag{5.63}$$

so that

$$X(\omega)\delta(\omega - \omega_0) = X(\omega_0)\delta(\omega - \omega_0) \tag{5.64}$$

because the integrals on both sides are identical.

Using Eq. (5.63), we see that the convolution of $X(\omega)$ and $\delta(\omega - \omega_0)$ is

$$X(\omega) * \delta(\omega - \omega_0) = \int_{-\infty}^{\infty} X(u)\delta(\omega - \omega_0 - u)du = X(\omega - \omega_0) \tag{5.65}$$

For the case of a complex exponential, inserting $X(\omega) = e^{j\omega n}$ into Eq. (5.63) results in

$$\int_{-\infty}^{\infty} \delta(\omega - \omega_0) e^{j\omega n} d\omega = e^{j\omega_0 n} \tag{5.66}$$

By comparing Eq. (5.66) with (5.20) we can then obtain

$$e^{j\omega_0 n} \leftrightarrow 2\pi\delta(\omega - \omega_0) \tag{5.67}$$

so that the Fourier transform of a complex exponential is an impulse concentrated at frequency ω_0.

Figure 5.12 Representation of the $\delta(\omega)$ function and its approximation $d_\Delta(\omega)$.

5.3.2.2. The Impulse Train

Since the impulse train

$$p_N[n] = \sum_{k=-\infty}^{\infty} \delta[n-kN] \tag{5.68}$$

is periodic with period N, it can be expanded in Fourier series according to (5.56) as

$$P_N[k] = 1 \tag{5.69}$$

so that using the inverse Fourier series Eq. (5.57), $p_N[n]$ can alternatively be expressed as

$$p_N[n] = \frac{1}{N}\sum_{k=0}^{N-1} e^{j2\pi kn/N} \tag{5.70}$$

which is an alternate expression to Eq. (5.68) as a sum of complex exponentials. Taking the Fourier transform of Eq. (5.70) and using Eq. (5.67) we obtain

$$P_N(e^{j\omega}) = \frac{2\pi}{N}\sum_{k=0}^{N-1} \delta(\omega - 2\pi k/N) \tag{5.71}$$

which is another impulse train in the frequency domain (See Figure 5.13). The impulse train in the time domain is given in terms of the Kronecker delta, and the impulse train in the frequency domain is given in terms of the Dirac delta.

Figure 5.13 An impulse train signal and its Fourier transform, which is also an impulse train.

5.3.2.3. General Periodic Signals

We now compute the Fourier transform of a general periodic signal using the results of Section 5.3.2.2 and show that, in addition to being periodic, the transform is also discrete. Given a periodic signal $x_N[n]$ with period N, we define another signal $x[n]$:

$$x[n] = \begin{cases} x_N[n] & 0 \le n < N \\ 0 & otherwise \end{cases} \tag{5.72}$$

so that

$$x_N[n] = \sum_{k=-\infty}^{\infty} x[n-kN] = x[n] * \sum_{k=-\infty}^{\infty} \delta[n-kN] = x[n] * p_N[n] \tag{5.73}$$

which is the convolution of $x[n]$ with an impulse train $p_N[n]$ as in Eq. (5.68). Since $x[n]$ is of finite length, it has a Fourier transform $X(e^{j\omega})$. Using the convolution property $X_N(e^{j\omega}) = X(e^{j\omega})P_N(e^{j\omega})$, where $P_N(e^{j\omega})$ is the Fourier transform of $p_N[n]$ as given by Eq. (5.71), we obtain another impulse train:

$$X_N(e^{j\omega}) = \frac{2\pi}{N}\sum_{k=-\infty}^{\infty} X(e^{j2\pi k/N})\delta(\omega - 2\pi k/N) \tag{5.74}$$

Therefore the Fourier transform $X_N(e^{j\omega})$ of a periodic signal $x_N[n]$ can be expressed in terms of samples $\omega_k = 2\pi k/N$, spaced $2\pi/N$ apart, of the Fourier transform $X(e^{j\omega})$ of $x[n]$, one period of the signal $x_N[n]$. The relationships between $x[n]$, $x_N[n]$, $X(e^{j\omega})$ and $X_N(e^{j\omega})$ are shown in Figure 5.14.

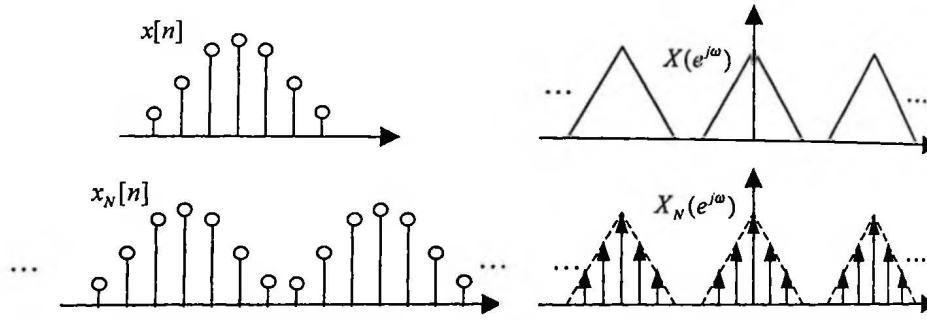


Figure 5.14 Relationships between finite and periodic signals and their Fourier transforms. On one hand, $x[n]$ is a length N discrete signal whose transform $X(e^{j\omega})$ is continuous and periodic with period 2π. On the other hand, $x_N[n]$ is a periodic signal with period N whose transform $X_N(e^{j\omega})$ is discrete and periodic.

5.3.3. The Fast Fourier Transform (FFT)

There is a family of fast algorithms to compute the DFT, which are called Fast Fourier Transforms (FFT). Direct computation of the DFT from Eq. (5.56) requires N^2 operations, assuming that the trigonometric functions have been pre-computed. The FFT algorithm only requires on the order of $N\log_2 N$ operations, so it is widely used for speech processing.

5.3.3.1. Radix-2 FFT

Let's express the discrete Fourier transform of $x[n]$

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi nk/N} = \sum_{n=0}^{N-1} x[n]W_N^{nk} \qquad 0 \le k < N \tag{5.75}$$

where we have defined for convenience

$$W_N = e^{-j2\pi/N} \tag{5.76}$$

Equation (5.75) requires N^2 complex multiplies and adds. Now, let's suppose N is even, and let $f[n] = x[2n]$ represent the even-indexed samples of $x[n]$, and $g[n] = x[2n+1]$ the odd-indexed samples. We can express Eq. (5.75) as

$$X[k] = \sum_{n=0}^{N/2-1} f[n]W_{N/2}^{nk} + W_N^k \sum_{n=0}^{N/2-1} g[n]W_{N/2}^{nk} = F[k] + W_N^k G[k] \tag{5.77}$$

where $F[k]$ and $G[k]$ are the $N/2$ point DFTs of $f[n]$ and $g[n]$, respectively. Since both $F[k]$ and $G[k]$ are defined for $0 \le k < N/2$, we need to also evaluate them for $N/2 \le k < N$, which is straightforward, since

$$F[k + N/2] = F[k] \tag{5.78}$$

$$G[k + N/2] = G[k] \tag{5.79}$$

If $N/2$ is also even, then both $f[n]$ and $g[n]$ can be decomposed into sequences of even and odd indexed samples and therefore its DFT can be computed using the same process. Furthermore, if N is an integer power of 2, this process can be iterated and it can be shown that the number of multiplies and adds is $N \log_2 N$, which is a significant saving from N^2. This is the *decimation-in-time* algorithm and can be seen in Figure 5.15. A dual algorithm called *decimation-in-frequency* can be derived by decomposing the signal into its first $N/2$ and its last $N/2$ samples.

5.3.3.2. Other FFT Algorithms

Although the radix-2 FFT is the best known algorithm, there are other variants that are faster and are more often used in practice. Among those are the radix-4, radix-8, split-radix and prime-factor algorithm.

The same process used in the derivation of the radix-2 decimation-in-time algorithm applies if we decompose the sequences into four sequences: $f_1[n] = x[4n]$, $f_2[n] = x[4n+1]$, $f_3[n] = x[4n+2]$, and $f_4[n] = x[4n+3]$. This is the radix-4 algorithm, which can be applied when N is a power of 4, and is generally faster than an equivalent radix-2 algorithm.

Similarly there are radix-8 and radix-16 algorithms for N being powers of 8 and 16 respectively, which use fewer multiplies and adds. But because of possible additional control logic, it is not obvious that they will be faster, and every algorithm needs to be optimized for a given processor.

There are values of N, such as $N = 128$, for which we cannot use radix-4, radix-8 nor radix-16, so we have to use the less efficient radix-2. A combination of radix-2 and radix-4, called *split-radix* [5], has been shown to have fewer multiplies than both radix-2 and radix-4, and can be applied to N being a power of 2.

Finally, another possible decomposition is $N = p_1 p_2 \cdots p_L$ with p_l being prime numbers. This leads to the *prime-factor algorithm* [2]. While this family of algorithms offers a similar number of operations as the algorithms above, it offers more flexibility in the choice of N.

5.3.3.3. FFT Subroutines

Typically, FFT subroutines are computed *in-place* to save memory and have the form `fft (float *xr, float *xi, int n)` where `xr` and `xi` are the real and imaginary parts respectively of the input sequence, before calling the subroutine, and the real and imaginary parts of the output transform, after returning from it. C code that implements a decimation-in-time radix-2 FFT of Figure 5.15 is shown in Figure 5.16.

The first part of the subroutine in Figure 5.16 is doing the so-called *butterflies*, which use the trigonometric factors, also called *twiddle factors*. Normally, those twiddle factors are pre-computed and stored in a table. The second part of the subroutine deals with the fact that the output samples are not linearly ordered (see Figure 5.15); in fact, the indexing has the bits reversed, which is why we need to do *bit reversal*, also called *descrambling*.

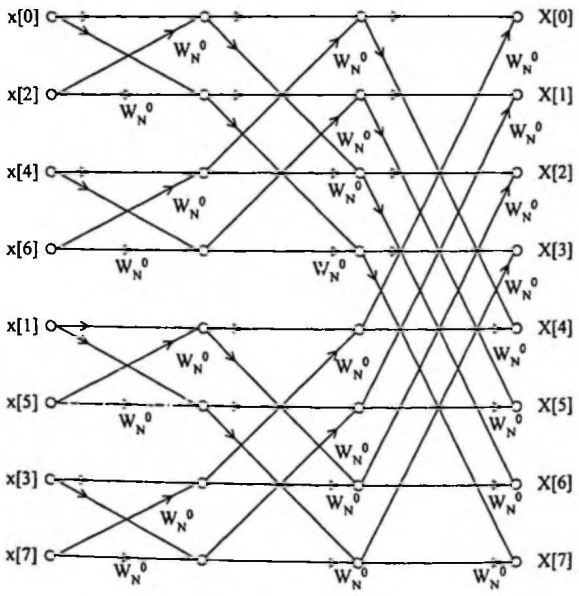


Figure 5.15 Decimation in time radix-2 algorithm for an 8-point FFT.

```
void fft2 (float *x, float *y, int n, int m)
{
     int n1, n2, i, j, k, l;
     float  xt, yt, c, s;
     double e, a;

     /* Loop through all m stages */
     n2 = n;
     for (k = 0; k < m; k++) {
         n1 = n2;
         n2 = n2 / 2;
         e = PI2 / n1;
         for (j = 0; j < n2; j++) {
             /* Compute Twiddle factors */
             a = j * e;
             c = (float) cos (a);
             s = (float) sin (a);

             /* Do the butterflies */
             for (i = j; i < n; i += n1) {
                 l = i + n2;
                 xt = x[i] - x[l];
                 x[i] = x[i] + x[l];
                 yt = y[i] - y[l];
                 y[i] = y[i] + y[l];
                 x[l] = c * xt + s * yt;
                 y[l] = c * yt - s * xt;
             }
         }
     }

     /* Bit reversal: descrambling */
     j = 0;
     for (i = 0; i < n - 1; i++) {
         if (i < j) {
             xt = x[j];
             x[j] = x[i];
             x[i] = xt;
             xt = y[j];
             y[j] = y[i];
             y[i] = xt;
         }
         k = n / 2;
         while (k <= j) {
             j -= k;
             k /= 2;
         }
         j += k;
     }
}
```

Figure 5.16 C source for a decimation-in-time radix-2 FFT. Before calling the subroutine, x and y contain the real and imaginary parts of the input signal respectively. After returning from the subroutine, x and y contain the real and imaginary parts of the Fourier transform of the input signal. n is the length of the FFT and is related to m by $n = 2^m$.

To compute the inverse FFT an additional routine is not necessary; it can be computed with the subroutine above. To see that, we expand the DFT in Eq. (5.56) into its real and imaginary parts:

$$X_R[k]+jX_I[k]=\sum_{n=0}^{N-1}\left(x_R[n]+jx_I[n]\right)e^{-j2\pi nk/N} \tag{5.80}$$

take complex conjugate and multiply by j to obtain

$$X_I[k]+jX_R[k]=\sum_{n=0}^{N-1}\left(x_I[n]+jx_R[n]\right)e^{j2\pi nk/N} \tag{5.81}$$

which has the same functional form as the expanded inverse DFT of Eq. (5.57)

$$x_R[k]+jx_I[k]=\frac{1}{N}\sum_{n=0}^{N-1}\left(X_R[n]+jX_I[n]\right)e^{j2\pi nk/N} \tag{5.82}$$

so that the inverse FFT can be computed by calling `fft (xi, xr, n)` other than the $(1/N)$ factor.

Often the input signal $x[n]$ is real, so that we know from the symmetry properties of Table 5.5 that its Fourier transform is Hermitian. This symmetry can be used to compute the length-N FFT more efficiently with a length $(N/2)$ FFT. One way of doing so is to define $f[n]=x[2n]$ to represent the even-indexed samples of $x[n]$, and $g[n]=x[2n+1]$ the odd-indexed samples. We can then define a length $(N/2)$ complex signal $h[n]$ as

$$h[n]=f[n]+jg[n]=x[2n]+jx[2n+1] \tag{5.83}$$

whose DFT is

$$H[k]=F[k]+jG[k]=H_R[k]+jH_I[k] \tag{5.84}$$

Since $f[n]$ and $g[n]$ are real, their transforms are Hermitian and thus

$$H^*[-k]=F^*[-k]-jG^*[-k]=F[k]-jG[k] \tag{5.85}$$

Using Eqs. (5.84) and (5.85), we can obtain $F[k]$ and $G[k]$ as a function of $H_R[k]$ and $H_I[k]$:

$$F[k]=\frac{H[k]+H^*[-k]}{2}=\left(\frac{H_R[k]+H_R[-k]}{2}\right)+j\left(\frac{H_I[k]-H_I[-k]}{2}\right) \tag{5.86}$$

$$G[k]=\frac{H[k]-H^*[-k]}{2j}=\left(\frac{H_I[k]+H_I[-k]}{2}\right)-j\left(\frac{H_R[k]-H_R[-k]}{2}\right) \tag{5.87}$$

As shown in Eq. (5.77), $X[k]$ can be obtained as a function of $F[k]$ and $G[k]$

$$X[k] = F[k] + G[k]W_N^{-k} \tag{5.88}$$

so that the DFT of the real sequence $x[n]$ is obtained through Eqs. (5.83), (5.86), (5.87) and (5.88). The computational complexity is a length ($N/2$) complex FFT plus N real multiplies and $3N$ real adds.

5.3.4. Circular Convolution

The convolution of two periodic signals is not defined according to Eq. (5.12). Given two periodic signals $x_1[n]$ and $x_2[n]$ with period N, we define their *circular convolution* as

$$y[n] = x_1[n] \otimes x_2[n] = \sum_{m=0}^{N-1} x_1[m]x_2[n-m] = \sum_{m=<N>} x_1[m]x_2[n-m] \tag{5.89}$$

where $m =< N >$ in Eq. (5.89) means that the sum lasts only one period. In fact, the sum could be over any N consecutive samples, not just the first N. Moreover, $y[n]$ is also periodic with period N. Furthermore, it is left to the reader to show that

$$Y[k] = X_1[k]X_2[k] \tag{5.90}$$

i.e., the DFT of $y[n]$ is the product of the DFTs of $x_1[n]$ and $x_2[n]$.

An important application of the above result is the computation of a regular convolution using a circular convolution. Let $x_1[n]$ and $x_2[n]$ be two signals such that $x_1[n] = 0$ outside $0 \le n < N_1$, and $x_2[n] = 0$ outside $0 \le n < N_2$. We know that their regular convolution $y[n] = x_1[n] * x_2[n]$ is zero outside $0 \le N_1 + N_2 - 1$. If we choose an integer N such that $N \ge N_1 + N_2 - 1$, we can define two periodic signals $\tilde{x}_1[n]$ and $\tilde{x}_2[n]$ with period N such that

$$\tilde{x}_1[n] = \begin{cases} x_1[n] & 0 \le n < N_1 \\ 0 & N_1 \le n < N \end{cases} \tag{5.91}$$

$$\tilde{x}_2[n] = \begin{cases} x_2[n] & 0 \le n < N_2 \\ 0 & N_2 \le n < N \end{cases} \tag{5.92}$$

where $x_1[n]$ and $x_2[n]$ have been *zero padded*. It can be shown that the circular convolution $\tilde{y}[n] = \tilde{x}_1[n] \otimes \tilde{x}_2[n]$ is identical to $y[n]$ for $0 \le n < N$, which means that $y[n]$ can be obtained as the inverse DFT of $\tilde{Y}[k] = \tilde{X}_1[k]\tilde{X}_2[k]$. This method of computing the regular convolution of two signals is more efficient than the direct calculation when N is large. While the crossover point will depend on the particular implementations of the FFT and convolution, as well as the processor, in practice this has been found beneficial for $N \ge 1024$.

5.3.5. The Discrete Cosine Transform (DCT)

The Discrete Cosine Transform (DCT) is widely used for speech processing. It has several definitions. The DCT-II $C[k]$ of a real signal $x[n]$ is defined by:

$$C[k]=\sum_{n=0}^{N-1}x[n]\cos\left(\pi k(n+1/2)/N\right) \quad \text{for } 0\le k<N \tag{5.93}$$

with its inverse given by

$$x[n]=\frac{1}{N}\left\{C[0]+2\sum_{k=1}^{N-1}C[k]\cos\left(\pi k(n+1/2)/N\right)\right\} \quad \text{for } 0\le n<N \tag{5.94}$$

The DCT-II can be derived from the DFT by assuming $x[n]$ is a real periodic sequence with period $2N$ and with an even symmetry $x[n]=x[2N-1-n]$. It is left to the reader to show, that $X[k]$ and $C[k]$ are related by

$$X[k]=2e^{j\pi k/2N}C[k] \quad \text{for } 0\le k<N \tag{5.95}$$

$$X[2N-k]=2e^{-j\pi k/2N}C[k] \quad \text{for } 0\le k<N \tag{5.96}$$

It is left to the reader to prove Eq. (5.94) is indeed the inverse transform using Eqs. (5.57), (5.95), and (5.96). Other versions of the DCT-II have been defined that differ on the normalization constants but are otherwise the same.

There are eight different ways to extend an N-point sequence and make it both periodic and even, such that can be uniquely recovered. The DCT-II is just one of the ways, with three others being shown in Figure 5.17.

The DCT-II is the most often used discrete cosine transform because of its *energy compaction*, which results in its coefficients being more concentrated at low indices than the DFT. This property allows us to approximate the signal with fewer coefficients [10].

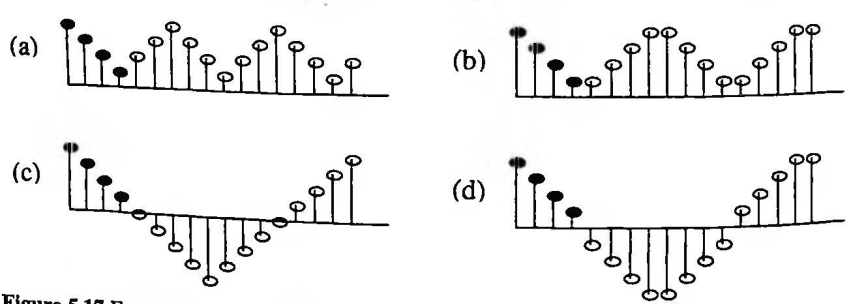


Figure 5.17 Four ways to extend a four-point sequence $x[n]$ to make it both periodic and have even symmetry. The figures in (a), (b), (c) and (d) correspond to the DCT-I, DCT-II, DCT-III and DCT-IV respectively.