第5章 量化与编码

CHAPTER 5

5.1 本章目标

学习完本章，读者应该：

(1) 熟悉标量量化的原理，能够解释矢量量化器的工作流程；

(2) 熟悉最小冗余码字分配的原理，理解无损信源编码的重要算法；

(3) 能够解释包含 DCT 在内的几种图像压缩方法；

(4) 理解波形和参数语音编码的基本方法，能够解释每种方法的优缺点；

(5) 能够解释音频编码器的关键要求以及组成音频编码系统的构建模块。

5.2 内容介绍

量化是指分配一个数字值(通常是一个整数)来表示一个或多个模拟值的过程。因为每一个采样值只能用数量有限的比特表示，所以只存在数量有限的离散电平。因此，只能以最接近的近似值表示真正的模拟信号。谨慎地选择表示方法意味着可以用尽可能少的比特实现这一过程。

除了表示电平的数量之外，还要有足够的采样密度，在数字化音频中指的是每秒的采样数，在数字化图像中指的是空间密度。采样音频信号的总数据速率(单位为比特每秒，即 bps)为每秒的采样数乘以每个采样点的比特位数。采样图像的总数据速率为给定图像的采样数乘以每个采样点的比特位数。数字信号的最紧凑表示(即在可接受的表示方法中，尽可能使用少的比特位数)对于数字化音频和图像的传输和存储都是至关重要的。

5.3 预备知识

本节将简要回顾概率的一些概念，这些概念在通信信道的误差建模中很有用；以及差分方程的相关概念，这些概念被广泛用于信号编码中。

5.3.1 概率函数

本章考虑的是量化，即对每个采样点都分配一个二进制表示，因此需要对采样值的特性进行研究，这些特性不仅用于确定信号取值的可能范围，还描述了各个采样值出现的概率。

如果有一个连续变量，如电压，它可以在一个特定范围内取任意值。例如，电压可以是 1.23V 或 6.4567V。除非需要精确的测量，通常情况下都并不需要知道准确的值，只需要一个近似的值。通常希望得到采样值出现在某一范围内的可能性，如－2～－1V、－1～0V、0～1V 和 1～2V。而图像中的像素也可以采用离散值 0，1，2，…，255 表示，并用这种方法将其“归类”到一组范围中。如果测量的信号是在某段时间内，就可以计算信号在每个范围内出现的次数。这就变成了一个直方图（Histogram），如图 5.1 所示。

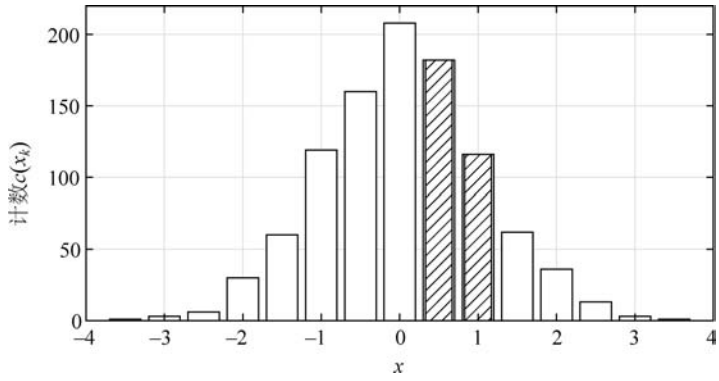


图 5.1　位于范围(x_k, x_{k+1})的信号值计数的直方图

注：阴影部分分别表示中心为 0.5 和 1.0 的分组范围内的计数值。

这种类型的表示有双重用途。首先，可以看到，在这种情况下，低于－4 或高于＋4 的计数都很少，信号最有可能的取值为－1～＋1。此外，如果想知道某些分组的计数比例，如图 5.1 中阴影部分，表示中心 x_k 为 0.5 和 1.0 的分组，就可以对每个分组分别进行计数。由于总的计数是所有分组的总和，因此，比例就是将所需范围内的计数除以总计数所得到的比值。

另外，如果有一个连续信号，并且不希望把它分成多个分组，就可以使用概率密度函数（Probability Density Function，PDF）来描述。图 5.2 给出了 PDF 的一个示例，可以看到它与直方图非常相似。一个关键的区别就是没有把数据分为离散的区间。重要的是，PDF 的高度不是计数或概率，而是密度。例如，无法准确找到 $x=1.234\,565$ 出现的可能性。然而，可以找到信号落在一定范围内的概率。图 5.1 中阴影部分的面积对应于信号的 x 值在特定范围内的概率。显然，对于不同的取值范围和不同的概率密度函数，结果都将发生变化。此外，PDF 曲线下的总面积必须为 1，因为信号必须始终落在可能的范围内（也就是说，信号 100％会取某个值）。这也等价于在离散直方图中各个分组计数的总和必须等于观察总数。

PDF 的数学形式在分析中很有用，其中高斯分布是迄今为止最常用的，它可以很好地近似多种常见类型的噪声。高斯分布的概率密度函数的数学表示为

$$f(x)=\frac{1}{\sigma\sqrt{2\pi}}\mathrm{e}^{-(x-\mu)^2/2\sigma^2} \tag{5.1}$$

其中，$f(x)$是针对 x 的概率密度，σ^2 是方差，μ 是平均值。如果只有从一组样本中得到的平均值可用，则称为样本均值（Sample Average），通常用 $\bar{x}$ 表示。而真实的平均值是总体均值（Population Mean），用 μ 表示。图 5.2 所示为均值为 1，方差为 1 的高斯分布概率密度曲线。

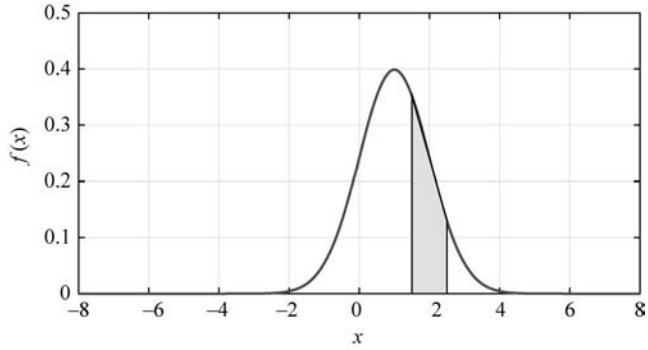


图 5.2　连续采样的概率密度曲线

注：从 $x=1.5$ 到 $x=2.5$ 的总概率为阴影区域的面积。

5.3.2　差分方程与 z 变换

通信系统的许多构建模块都是以数字化的方式实现的，差分方程就描述了它们逐步运行的方式。在采样时刻 n，输入 $x(n)$产生了输出 $y(n)$，差分方程根据当前输入、过去输入和过去输出的加权和确定每个输出。一个简单的例子如下。

$$y(n)=0x(n)+0.8x(n-1)+1.5y(n-1)-0.64y(n-2) \tag{5.2}$$

对于每一个新的输入样本 $x(n)$，都会计算得到一个输出 $y(n)$，$x(n-1)$表示 $x(n)$前一时刻的采样值。除了计算之外，过去的输入和过去的输出，即此例中的 $x(n-1)$、$y(n-1)$和 $y(n-2)$，还需要被存储起来。当然，还能扩展到任意项。此外，如果某项不存在，如此例中的 $x(n-2)$，意味着它的系数为 0。

差分方程的分析工具是 z 变换。这种方法使用运算符 z 的幂次方，即 z^{-D} 代表 D 个采样间隔的延迟。因此，$x(n-2)$项转换后就变成 $X(z)z^{-2}$。然后将示例中的差分方程进行如下转换，其目的是得到输出与输入的比值，即 $Y(z)/X(z)$。

$$\begin{aligned}
&y(n)=0x(n)+0.8x(n-1)+1.5y(n-1)-0.64y(n-2)\\
&Y(z)z^{0}=0X(z)z^{0}+0.8X(z)z^{-1}+1.5Y(z)z^{-1}-0.64Y(z)z^{-2}\\
&Y(z)z^{0}=X(z)(0z^{0}+0.8z^{-1})+Y(z)(1.5z^{-1}-0.64z^{-2})\\
&Y(z)(1z^{0}-1.5z^{-1}+0.64z^{-2})=X(z)(0z^{0}+0.8z^{-1})\\
&\frac{Y(z)}{X(z)}=\frac{0z^{0}+0.8z^{-1}}{1z^{0}-1.5z^{-1}+0.64z^{-2}}
\end{aligned} \tag{5.3}$$

因此，输入与输出的关系就是关于 z 的两个多项式的比值。z 变换表达式中的分子 $\boldsymbol{b}$ 和分母 $\boldsymbol{a}$ 的系数为

$$\boldsymbol{b}=[0 \quad 0.8]$$
$$\boldsymbol{a}=[1 \quad -1.5 \quad 0.64]$$

注意，$\boldsymbol{a}$ 中的第一个系数值为 1，因为输出为 $1\cdot y(n)$。此外，与差分方程相比，后续的系数也不同，在差分方程中，后续系数分别为+1.5 和−0.64。这是由于从差分方程[式(5.2)]到 z 域方程[式(5.3)]中的项进行了重新排列。

上述差分方程的 MATLAB 实现如下。输入信号采用脉冲(Impulse)函数，它是一个值为+1 的单峰，之后的数据都为零，这是最简单的输入类型。

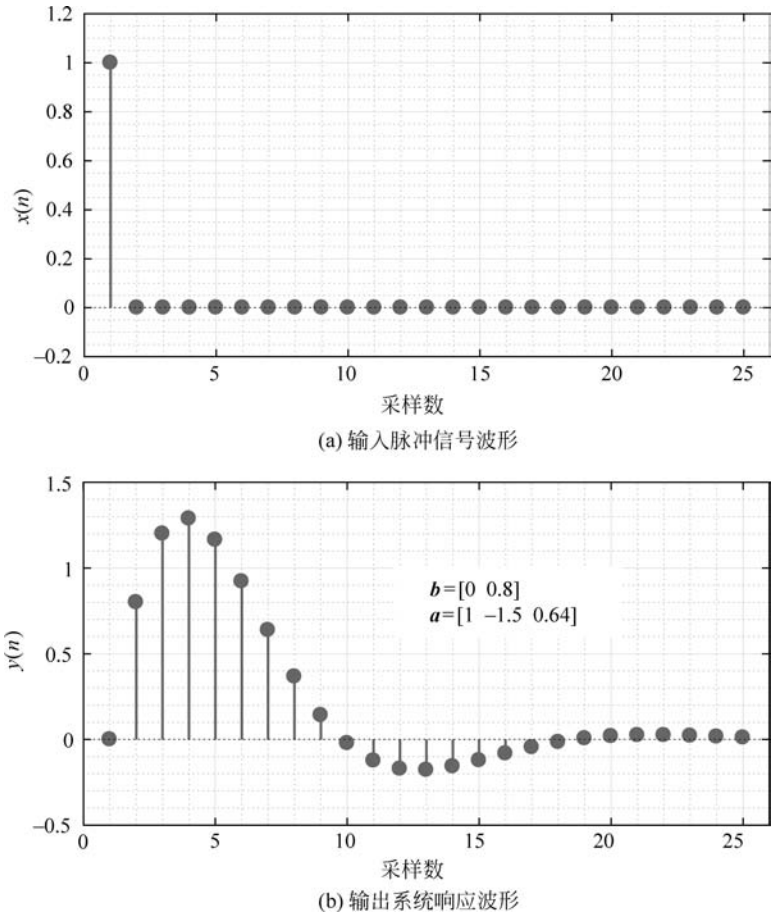
```
% 传递函数 Y(z)/X(z) 有分子 b 和分母 a
% 将它转化为差分方程时请注意:
% 总有 a(1) = 1 且 a(2:end) 是差分方程系数的负数
b = [0 0.8];
a = [1 -1.5 0.64];

% 脉冲输入
x = zeros(25, 1);
x(1) = 1;

% 计算 y 中的输出样本序列,对应 x 的每个输入
% b 和 a 中的系数定义了 z 变换的系数
% 等价地,它们也是差分方程的系数
% 但要参见上面的注释
y = filter(b, a, x);

stem(y);
```

差分方程的脉冲输入和输出如图 5.3 所示。这里有几点值得注意,第一个输出为 0,这是因为此例中 $x(n)$的系数为 0。实际上,这将使输出延迟一个采样点;更重要的是,由于输出取决于过去的输出(以及输入),系统响应会延续一段时间,这称为递归(Recursive)系统。



(a) 输入脉冲信号波形

(b) 输出系统响应波形

图 5.3　差分方程的脉冲输入和输出

一般情况的差分方程可写成如下形式。

$$1y(n)=[b_0x(n)+b_1x(n-1)+\cdots+b_Nx(n-N)]-$$
$$[a_1y(n-1)+a_2y(n-2)+\cdots+a_My(n-M)] \tag{5.4}$$

这种形式很有用，因为它反映了 MATLAB 使用 filter()函数实现差分方程的方法。注意 $y(n)$的系数为 1，正如之前已经明确说过的，这也导致 $\boldsymbol{a}$ 的第一个系数为 1。还要注意 $\boldsymbol{a}$ 中剩余系数的负号，与前面的数值示例相同。

将 z 变换推至一般情况，如下所示。

$$\frac{Y(z)}{X(z)}=\frac{b_0z^0+b_1z^{-1}+\cdots+b_Nz^{-N}}{1+a_1z^{-1}+a_2z^{-2}+\cdots+a_Mz^{-M}}$$

也就是说，可以将 filter()函数的系数矢量表示如下。

$$\boldsymbol{b}=[b_0 \quad b_1 \quad b_2 \quad \cdots]$$
$$\boldsymbol{a}=[1 \quad a_1 \quad a_2 \quad \cdots]$$

差分方程在电信系统中得到广泛应用，它的分析最好在 z 域进行。任何系统的零点(Zero)都是使分子等于零的 z 值；任何系统的极点(Pole)都是使分母等于零的 z 值。系统的稳定性及其振荡频率由极点决定。从根本上说，这是因为极点决定了递归(或反馈)项的系数。

5.4 数字信道容量

当想要传输数字信息时，自然会提出一个问题：在给定的时间里，能传输多少信息？或者说，数据传输是否存在最大速率？首先假设传输的是二进制信息，即比特，它是由 J. W. Tukey 提出的(Shannon，1948)。早期关于信道容量的研究是针对使用开关型或摩斯电码型信号的电报系统进行的，当时的想法是将几个电报流复用到一个长途载波上。

Hartley(1928)提出使用对数度量信息中的信息量。如果使用的是二进制系统，对数的底数自然选为 2。如果有 M 个电平(如电压)，那么可以用 $\log_2M$ 位编码。例如，如果要传输 0，1，2，…，15V 的电压，共有 16 个电平，很显然，编码需要 4 位(lb16＝4)。当然，测量单位不是必须为伏特，它可以是任意选定的量。

著名的奈奎斯特采样定理本质上是说，一个带宽为 B 的信号需要以大于 $2B$ 的速率采样才能实现完美重构(Nyquist，1924ab)。将这一点与 Hartley 关于信息量的观察相结合，得到如下的一个信道容量的公式，通常称为哈特利定律。

$$C=2B\text{lb}M \quad \text{bps} \tag{5.5}$$

其中，B 为信道带宽(Hz)，M 为用于每个信号成分的电平数，bps 表示传输速率，即比特每秒。

下面用一个例子来说明这一公式并展示其重要性。如果通过传输线发送语音信号，保持语音正确性的最小带宽约为 3000Hz。因此，该线路的带宽至少为 $B\approx3$kHz。在这个信道上，如果要发送两个电压电平编码的数据(每个信号周期传一个比特)，即 $M=2$，那么总信道容量就为 $C=2\times3000\times\text{lb}2=6$kbps。因为实际中带宽都是有限的，所以获得更高信息速率的唯一方法是增加电平数 M。

这个公式作为上限是很有用的，但忽略了噪声的存在。Shannon(1948)推导出了著名的最大容量公式，有时也称为香农-哈特利定律(Shannon-Hartley Law)。根据各种假设，将信道容量 C 预测为

$$C=B\text{lb}\left(1+\frac{S}{N}\right) \quad \text{bps} \tag{5.6}$$

与前面一样，B 为信道带宽(Hz)，S 为用于传输信息的信号功率(W)，N 为当信号从发射机传输到接收机时经过信道所附加的噪声功率，单位也是 W。S/N 为信号功率与噪声功率的比值，通常情况下，信噪比(SNR)用分贝(dB)表示，即

$$\mathrm{SNR}=10\lg\left(\frac{S}{N}\right)\quad \mathrm{dB} \tag{5.7}$$

为了说明这一点，还是考虑一个带宽 B 为 3kHz 的模拟电路，同时信噪比为 1000。因为 $S/N=1000$，所以 $\mathrm{SNR}=10\times\lg10^3=30\mathrm{dB}$。还需要计算 lb1000，注意到 $2^{10}=1024$，所以要求的对数大约为 10(准确计算为 $\mathrm{lb}1000=\lg1000/\lg2=3/0.301\approx9.967$)。所以可计算出 $C=3000\times\mathrm{lb}(1+1000)\approx3000\times10=30\mathrm{kbps}$。

有两个计算 C 的公式，下面讨论这两个公式之间的关系。

$$C=2B\,\mathrm{lb}M$$

$$C=B\,\mathrm{lb}\left(1+\frac{S}{N}\right)$$

将这两个公式联立，可以得到

$$\begin{aligned}M&=\sqrt{1+\frac{S}{N}}\\&\approx\sqrt{\frac{S}{N}}\end{aligned} \tag{5.8}$$

这种方法似乎是合理的，因为在相同的带宽中承载更多的信息就意味着在每个采样点都包含更多的信息，因此 M 也会更高。需要权衡的是，随着 M 的增加，相邻电平的差距变得更小，因此噪声对数据产生的影响更大。在本例中，使用上述方法意味着需要大约 32 个电平(即约为$\sqrt{S/N}$)。此外，由于信噪比的平方根产生均方根电压，可以粗略地认为误差与电压差成正比。

当然，所有这些只是理论上的讨论，它没有说明如何对信号进行编码使速率达到最大化，只是说明了最大速率可能是多少。为了了解香农-哈特利容量界限推导背后的原因，可以参考 Shannon 的原始论文。首先，假设信号和噪声都服从零均值高斯分布，它们的概率密度函数为

$$f(x)=\frac{1}{\sigma\sqrt{2\pi}}\mathrm{e}^{-x^2/2\sigma^2} \tag{5.9}$$

5.6.1 节将解释熵(Entropy)与信息量的概念。这里需要的结果是，在给定概率密度 f_X 的情况下，可以在 X 范围内取值的信号的熵 $H(X)$为

$$H(X)=\int_{-\infty}^{\infty}f_X(x)\,\mathrm{lb}\left[\frac{1}{f_X(x)}\right]\mathrm{d}x \tag{5.10}$$

用式(5.9)表示的零均值高斯分布代替概率密度函数，可得高斯信号的信息量为

$$H(X)=\frac{1}{2}\mathrm{lb}2\pi\mathrm{e}\sigma^2 \tag{5.11}$$

记 P 为信号功率，σ^2 为噪声功率。信号与噪声的熵可分别计算为

$$H_{\mathrm{s}}(X)=\frac{1}{2}\mathrm{lb}2\pi\mathrm{e}(P+\sigma^2) \tag{5.12}$$

$$H_{\mathrm{n}}(X)=\frac{1}{2}\mathrm{lb}2\pi\mathrm{e}\sigma^2 \tag{5.13}$$

这样写是为了明确地说明信道的影响，将信道建模为一个随机变量 X。Shannon 推断信号和噪声都含有信息量，而信道容量（单位为 b）则是总信息（即信号加噪声）的熵或信息量减去噪声的熵，即

$$\begin{aligned} H(X) &= H_s(X) - H_n(X) \\ &= \frac{1}{2}\mathrm{lb}(P+\sigma^2) - \frac{1}{2}\mathrm{lb}\sigma^2 \\ &= \frac{1}{2}\mathrm{lb}\left(\frac{P+\sigma^2}{\sigma^2}\right) \\ &= \frac{1}{2}\mathrm{lb}\left(1+\frac{P}{\sigma^2}\right) \\ &= \frac{1}{2}\mathrm{lb}(1+\mathrm{SNR}) \end{aligned} \tag{5.14}$$

而信道容量（单位为 bps）是这个熵值乘以每秒 $2B$ 个采样点，所以有

$$C = B\mathrm{lb}(1+\mathrm{SNR}) \tag{5.15}$$

这些都是后续研究的基础，还有两个问题需要解决。首先，需要确定一个最佳的方法，为输入信号（语音、音频、图像或其他要传输的数据）分配幅度步长，也称为量化电平。其次，还需要确定如何把可用的二进制数字（即比特）分配给这些量化电平，以便形成编码的符号流。解码本质上是相反的，将符号映射回二进制码，再从二进制码中得到近似的原始幅度。

5.5 量化

量化（Quantization）通常是指标量量化。也就是说，每次采集一个采样点，并确定一个二进制数值来代表该时刻的模拟输入电平（通常为电压）。模/数（A/D）转换中有两个重要的问题。首先是确定所需的电平数量。对于给定的信号类型，通常都可能采用尽量多的电平，但是这会产生更高的比特率。其次是确保涵盖信号的范围。例如，一般对话中的语音会有一个很大范围的取值，从很小到很大，对应着声音从轻柔到响亮。因此，会存在一种平衡，一方面，对于一个固定的电平数，表示更大的动态范围必然意味着它的精度会降低；另一方面，如果最大范围减小，用相同数量的电平表示这个较小的范围，那么更小的幅度将被更精确地量化。然而，由于范围缩小，一些极端情况下的信号幅度可能无法表示。这两种情况都会导致重构的模拟信号出现失真。因此，目标就是平衡这些相互矛盾的要求，即精度和动态范围的折中。

5.5.1 标量量化

图 5.4 给出了一个量化信号的简单说明。将连续波形（在本例中为正弦波）在离散的时刻进行采样。在每个时刻，分配一个电平来表示它。当可用于表示电平的比特有 N 个时，每个采样点的代表值都是 2^N 个电平中的一个。从图 5.4 中可以清楚地看出，在量化时存在采样值精度的损失。该示例显示了一个非常粗略的量化，只用了 8 个电平，这就意味着每个采样点用 3 位比特表示。例如，如果使用的是 8 位量化，那么将有 256 个电平；如果再将比特位数增加一倍，将有 65 536 个电平。电平数与比特位数呈指数关系，或者说，给每个采样点增加一位比特可以使电平数翻倍。

这一过程称为模/数（A/D）转换，是使用模数转换器（Analog to Digital Converter，ADC）执行的。相反，数/模（D/A）转换是将给定的二进制数值转换为相应的模拟电压电平。在 A/D 过程中，会损失一

些精度，即图 5.4 所示的平滑正弦曲线被阶梯曲线近似代替。

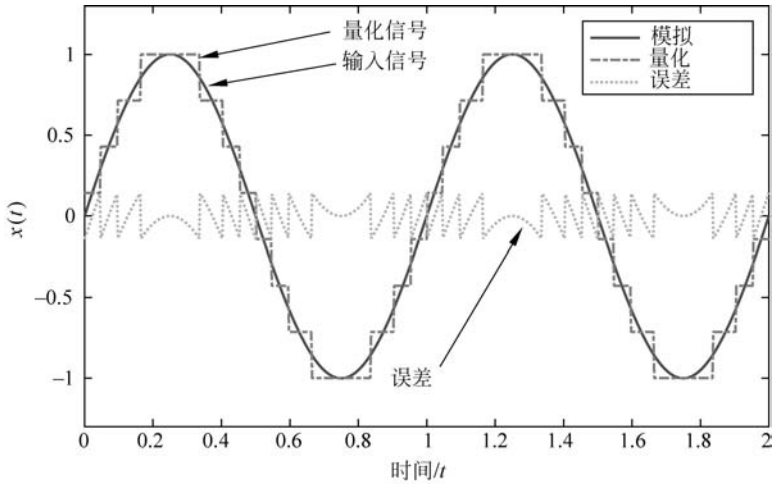


图 5.4　正弦波量化示例(使用 3b 中升量化器)

将量化的描述及其确切作用形式化是十分有必要的。图 5.5 显示了从横轴输入到纵轴输出的量化映射关系。输入幅度可以是 X 轴范围上的任意位置，并从 Y 轴上的相应电平中读取重构值。输入电平称为决策(Decision)电平，因为它决定了模拟电平将落入哪个区间。输出电平称为重构(Reconstruction)电平，因为它是在输出波形中重构出每个采样点的固定电平。

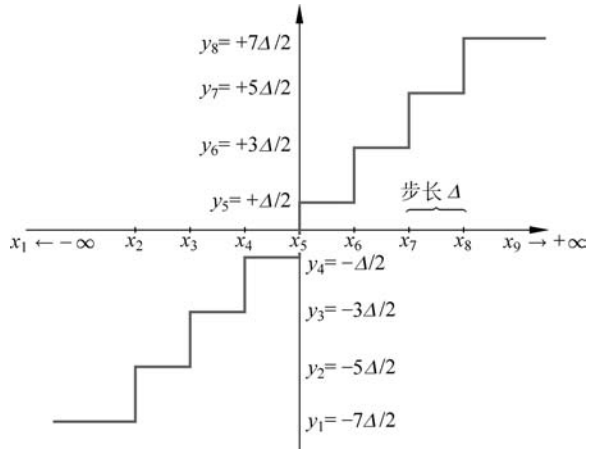


图 5.5　输入-输出量化映射关系(3b 中升量化特性)

注：x 是模拟输入，y_k 表示量化值。

图 5.5 的量化器有一个步长 Δ。0～Δ 的任意输入值 x_k 都将被映射到相应的重构值 y_k，在此例中为 $\Delta/2$。类似地，-3Δ～-2Δ 的输入都将被重构为 $-5\Delta/2 = -2.5\Delta$。这似乎完全合理，但是当输入值高于 3Δ 或低于 -3Δ 时会发生什么呢？它们会被划到各自的最大值与最小值 $\pm 3.5\Delta$。

值得注意的是，上例所示的量化类型可以被表征为零偏移、均匀步长、中升的量化器，但是这种量化特征并不是唯一的。如果较小的幅值要比较大的幅值更精确地表示，步长有可能不均匀。也有可能不是以零为中心，而是以其他某个正电平为中心。最后，中升也可以用“中平”来代替，其优点是零值被精

确量化为零。然而，这也意味着还有奇数个电平，因此可能会有一个电平未被使用（因为电平的数量为2^N），或者在横轴的一侧比另一侧有更多的电平。

现在分析量化误差，了解误差是如何产生的，因为这些误差会导致采样波形的失真。通过分析误差，对于给定的信号以及“典型”信号，就可以使用统计的方法计算有多少误差会出现。很明显，信号有大量时间落入的平均位置应该是量化最精确的区域，可以减少总的平均失真。

假设量化特性中有 L 个电平，并将第 k 个决策电平表示为 x_k，将第 k 个重构电平表示为 y_k。模拟输入 x 映射到重构 y_k 的误差即为$(x-y_k)$，再对此求平方，因为误差可以是正的，也可以是负的。

信号的概率密度 $f(x)$显示了 x 取何值是更有可能（或更不可能）的。因此，误差还要再与该值进行加权，并在 $x_k \sim x_{k+1}$ 求和，这是输出 y_k 对应的输入范围。

最后，对每一个量化电平（共 L 个）进行上述计算，最终得到的量化误差为（Jayant，et al.，1990）

$$e_q^2 = \sum_{k=1}^{L} \int_{x_k}^{x_{k+1}} (x - y_k)^2 f(x) \mathrm{d}x \tag{5.16}$$

其中，L 为量化器输出电平的数量，$f(x)$为输入变量的概率密度函数，x_k 为第 k 个决策电平，y_k 为相应的重构电平。

为了简单地解释这个方程的应用，对于 1b（两电平）中升量化器，重构信号按照以下方式进行：

（1）当 x 值为$-\infty \sim 0$ 时，y 输出值为$-\Delta/2$；

（2）当 x 值为 $0 \sim +\infty$时，y 输出值为$+\Delta/2$。

假设输入信号的概率密度 $f(x)$是不变的，也就是说，所有的取值都是等可能的。当然，这一点应与实际情况相符，或者至少假设这种分布能够充分地代表输入。输入 x 的概率密度函数不变意味着信号的两个取值是等可能的。假设 x 的平均值为零，最大幅度的取值为$\pm A/2$。那么概率密度函数可写为

$$f(x) = \begin{cases} 0, & x < -\dfrac{A}{2} \\ \dfrac{1}{A}, & -\dfrac{A}{2} \leqslant x \leqslant \dfrac{A}{2} \\ 0, & x > \dfrac{A}{2} \end{cases} \tag{5.17}$$

这表明 $f(x)$在$-A/2 \sim +A/2$ 的概率密度必须都等于 $1/A$。这是因为概率密度函数曲线下的面积必须等于 1（也就是说，信号必须在已知范围内的某个位置），并且这样形成的矩形的面积也为 1。

因此，对于假设的输入概率密度函数，需要计算出怎样的量化器参数可以最小化量化误差。首先，需要一个代价函数（Cost Function），即需要定义一个量等价于代价，并将其与设计参数联系起来。此例中，代价可以定义为平均量化误差，并且可以调整量化器的步长以最小化这一代价。所以这个问题可以表述为“有没有一个最佳的量化器步长可以最小化量化信号的平均误差”。

式(5.16)显示误差在平方后与概率密度进行加权，在上述假设的特殊情况下，量化误差为

$$e_q^2 = \sum_{k=1}^{2} \int_{x_k}^{x_{k+1}} (x - y_k)^2 \frac{1}{A} \mathrm{d}x \tag{5.18}$$

重构电平 y_k 为$\pm\Delta/2$，将其代入式(5.18)，并展开本例中的两个电平的和，平方误差就变为

$$\begin{aligned} e_q^2 &= \frac{1}{A}\int_{-\infty}^{0} \left(x - \frac{-\Delta}{2}\right)^2 \mathrm{d}x + \frac{1}{A}\int_{0}^{+\infty} \left(x - \frac{+\Delta}{2}\right)^2 \mathrm{d}x \\ &= \frac{1}{A}\int_{-\infty}^{0} \left(x^2 + x\Delta + \frac{\Delta^2}{4}\right) \mathrm{d}x + \frac{1}{A}\int_{0}^{+\infty} \left(x^2 - x\Delta + \frac{\Delta^2}{4}\right) \mathrm{d}x \end{aligned}$$

现在需要计算积分。由于分布是均匀的，积分的无限范围可以用最大的信号幅度来代替，计算如下。

$$e_q^2=\frac{1}{A}\left(\frac{x^3}{3}+\frac{x^2\Delta}{2}+\frac{x\Delta^2}{4}\right)\Big|_{x=-\frac{A}{2}}^{0}+\frac{1}{A}\left(\frac{x^3}{3}-\frac{x^2\Delta}{2}+\frac{x\Delta^2}{4}\right)\Big|_{0}^{x=+\frac{A}{2}}$$

$$=\frac{1}{A}\left[0-\left(\frac{-A^3}{24}+\frac{A^2\Delta}{8}+\frac{-A\Delta^2}{8}\right)\right]+\frac{1}{A}\left[\left(\frac{-A^3}{24}-\frac{A^2\Delta}{8}+\frac{A\Delta^2}{8}\right)-0\right]$$

化简可得

$$e_q^2=\frac{1}{A}\left(\frac{A^3}{12}-\frac{A^2\Delta}{4}+\frac{A\Delta^2}{4}\right)$$

$$e_q^2=\frac{A^2}{12}-\frac{A\Delta}{4}+\frac{\Delta^2}{4} \tag{5.19}$$

最后，得到平均平方误差的表达式，而使它最小化唯一可以改变的参数是步长 Δ。于是可以用微分确定这个最佳值，因为这个方程实际上是关于 Δ 的二次方程。因此，记最佳步长为 Δ^*，可以写出导数为

$$\frac{de_q^2}{d\Delta}=-\frac{A}{4}+\frac{\Delta^*}{2} \tag{5.20}$$

这类问题已经很熟悉了，将导数设为 0 可得

$$\frac{A}{4}=\frac{\Delta^*}{2}$$

$$\Delta^*=\frac{A}{2} \tag{5.21}$$

根据信号幅度参数 A，上述方程提供了最佳步长的表达式。最佳步长是幅度的一半的结果，似乎也与直观相符。还可以根据信号功率或方差重写公式。首先，需要一个用概率密度函数表示的方差。一般来说，对于任何范围为 $\pm A/2$ 的均匀概率密度函数，方差都可写为

$$\sigma^2\triangleq\int_{-\infty}^{+\infty}x^2f(x)\,dx$$

$$\sigma^2=\int_{-\frac{A}{2}}^{+\frac{A}{2}}x^2\left(\frac{1}{A}\right)dx \tag{5.22}$$

其结果为

$$\sigma^2=\frac{A^2}{12} \tag{5.23}$$

根据信号 x 的方差反过来表示其幅度

$$\sigma_x^2=\frac{A^2}{12}$$

$$A=2\sigma_x\sqrt{3} \tag{5.24}$$

将其代入最佳步长的方程，得到

$$\Delta^*=\frac{A}{2}$$

$$=\frac{2\sigma_x\sqrt{3}}{2}$$

$$=\sigma_x\sqrt{3} \tag{5.25}$$

这个结果显示，量化器步长应该是信号标准差的$\sqrt{3}$（≈1.73）倍，以便最小化误差。这样做将最小化量

化噪声，或者等效地，最大化信噪比。

因此，可以得出的一个普遍结论，对于给定的概率密度函数，最佳步长与标准差成正比。但是，当改变电平数量时，量化每个采样点所需的比特位数当然也会变化，那么结论又会怎样变化呢？

量化信噪比是信号功率除以噪声功率，用分贝表示为

$$\mathrm{SNR}=10\lg\left[\frac{\sum_{n}x^2(n)}{\sum_{n}e^2(n)}\right] \tag{5.26}$$

其中，误差为采样值 $x(n)$ 和量化值 $\hat{x}(n)$ 之间的差值，即

$$e(n)=x(n)-\hat{x}(n) \tag{5.27}$$

对于均匀的 Nb 量化器，步长 Δ 为峰值幅度与量化等级数量的比值，即

$$\Delta=\frac{2x_{\max}}{2^N} \tag{5.28}$$

如果量化噪声服从均匀分布，则方差为

$$\sigma^2\triangleq\int_{-\infty}^{+\infty}x^2f(x)\mathrm{d}x$$

$$\sigma_{\mathrm{e}}^2=\int_{-\Delta/2}^{+\Delta/2}x^2\left(\frac{1}{\Delta}\right)\mathrm{d}x$$

$$\sigma_{\mathrm{e}}^2=\frac{\Delta^2}{12} \tag{5.29}$$

将步长 Δ 代入其中，可得

$$\begin{aligned}\sigma_{\mathrm{e}}^2&=\frac{\Delta^2}{12}\\&=\frac{4x_{\max}^2}{12\times2^{2N}}\\&=\frac{x_{\max}^2}{3\times2^{2N}}\end{aligned} \tag{5.30}$$

可以把信噪比写为

$$\begin{aligned}\mathrm{SNR}&=\frac{\sigma_{\mathrm{x}}^2}{\sigma_{\mathrm{e}}^2}\\&=\frac{\sigma_{\mathrm{x}}^2}{\frac{x_{\max}^2}{3\cdot2^{2N}}}\\&=3\left(\frac{\sigma_{\mathrm{x}}^2}{x_{\max}^2}\right)\times2^{2N}\end{aligned} \tag{5.31}$$

通常的方法是用分贝(dB)来表示，即取对数后再乘以 10。使用标准对数规则，式(5.31)变为

$$\begin{aligned}\mathrm{SNR}&=10\lg3+10\lg2^{2N}+10\lg\left(\frac{\sigma_{\mathrm{x}}^2}{x_{\max}^2}\right)\\&\approx4.77+6.02N+20\lg\left(\frac{\sigma_{\mathrm{x}}}{x_{\max}}\right)\quad\mathrm{dB}\end{aligned}$$

假设最大值是 4 倍标准差，即 $x_{\max}=4\sigma_x$，就有

$$\mathrm{SNR}\approx 6N-7.3 \quad \mathrm{dB} \tag{5.32}$$

也就是说，可以说信噪比与比特数 N 成正比，即

$$\mathrm{SNR}\propto N \tag{5.33}$$

其中，比例常数为 6。一个重要的结论是，当量化器增加 1b，大约可以提高 6dB 信噪比。请注意，由于在推导过程中做了各种假设，这个结果实际上仅适用于有大量的量化等级，同时没有（或只有几个）采样点在 $\pm 4\sigma_x$ 外的情况。

5.5.2 压扩

上述关于量化的讨论中假设采样点是线性（Linearly）量化的。也就是说，每个步长都是相等的。如果步长不相等会发生什么？此外，为什么随着时间的推移步长却是固定的？现在将问题转向步长的选择，并研究动态地改变步长。虽然对于一个量化器的所有电平，具有相等的步长更加常见，但是压缩和扩展电平的技术已经广泛用于某些类型的音频，并且也在更高级的编码方案中得到应用。压扩的想法归功于 Eugene Peterson（尤金·彼得森），他观察到“微弱的声音比强烈的声音需要更微妙的处理”（Bennett，1984）。

如果保持一个固定的步长，就意味着量化时所有振幅对噪声的贡献是相等的。然而，由于量化只是一个近似值，可以放宽对精度的要求。对于高振幅，较大的步长可能就足够了。这种在较大幅度下牺牲一定精度的做法，还可以做到幅度越小，步长越小。根据信号的分布，如果幅度更小的信号更可能出现，那么平均误差实际上也可能会减小。此外，在音频中，从人耳听力的角度来说，较大的信号幅度会掩盖更多的噪声，使噪声变得更小。

这就产生了“压扩”的概念，意味着压缩和扩展范围。图 5.6 将此显示为从输入到输出的映射。在横轴上可以看到输入幅度，输出显示在纵轴上。对于低电平的输入幅度 x，输入的微小变化 δx 都会导致相当大的输出变化 δy。然而，对于高输入振幅，给定输入变化 δx 时，输出变化 δy 相对较小。这实现了目标，即对于较大的幅度，有较大的步长。

该过程采用模拟映射时，可用一组公式来描述。然而，以数字方式实现时，使用查找表（LUT）更方便。实际上，可以使用一组 13b（A 律）或 14b（μ 律）的电平量化信号源，并将其映射到一组 8b 等效电平上。这可以使每个采样点使用更少的比特，但是总体上保持大致相同的感知失真水平。

请注意，前面描述的过程是编码或压缩阶段，扩展阶段与此操作相反。这些操作可以由一个预先计算好的 LUT 来执行，这个 LUT 包含用于压缩和扩展的所有输入-输出对。

图 5.6 显示了前面讨论的两种压扩特性比较。线性响应表示为 45°的直线，此时输出等于输入。μ 律特性由映射方程定义为

$$c_\mu(x)=\mathrm{sign}(x)\frac{\ln(1+\mu\mid x\mid)}{\ln(1+\mu)},\quad 0\leqslant x\leqslant 1 \tag{5.34}$$

它具有“饱和”特性，即输入信号的较小值与较大值被区别对待。图 5.6 中还显示了 A 律的特性，它在概念上与 μ 律相似，但定义略有不同。

$$c_A(x)=\begin{cases}\mathrm{sign}(x)\dfrac{A\mid x\mid}{1+\ln A}, & 0\leqslant\mid x\mid\leqslant\dfrac{1}{A}\\[2ex] \mathrm{sign}(x)\dfrac{1+\ln(A\mid x\mid)}{1+\ln A}, & \dfrac{1}{A}\leqslant\mid x\mid\leqslant 1\end{cases} \tag{5.35}$$

注意，A 律将输入范围划分为不同的区域，较小的振幅为线性区域，较大的振幅为非线性区域。图 5.6

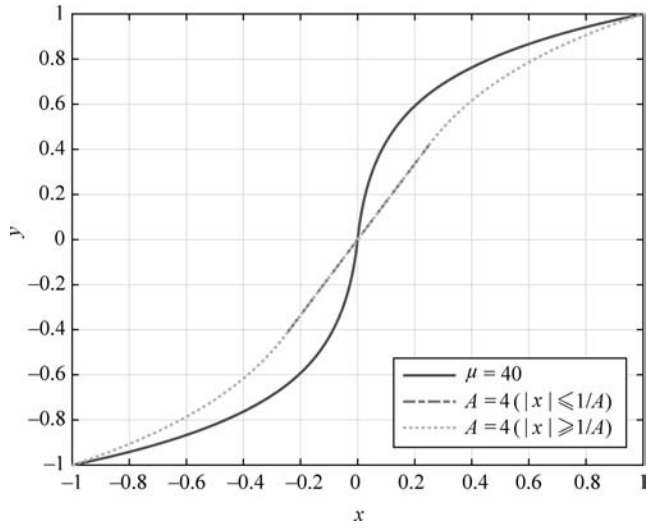


图 5.6 μ 律与 A 律压扩的典型比较

注：μ 值与 A 值的选择是为了强调 A 律的分段特性。

中选择的 μ 值和的 A 值是为了说明其特性的差异。将 μ 律压缩幅度信号转换回原始值的相反步骤为

$$x = \text{sign}(c_\mu(x)) \frac{1}{\mu}[(1+\mu)^{|c_\mu(x)|} - 1] \tag{5.36}$$

图 5.7 显示了 8b 线性、12b 线性和 8b 压扩信号的信噪比比较。对于非常小的输入幅度，使用 8b 的 μ 律压扩甚至能够比 12b 线性量化表现得更好(因为信噪比明显更高)。随着输入幅度的增加，线性量化器显示出的性能越来越好，信噪比越来越高，而压扩信号在幅度较高时达到了一个平稳状态。这是合理的，因为不可能做到在一个方面有所改进而在其他方面没有牺牲。然而，这种牺牲发生在较高的幅度，这在统计上是不太可能出现的。而且，当幅度较大时，噪声在感知上也不太明显。

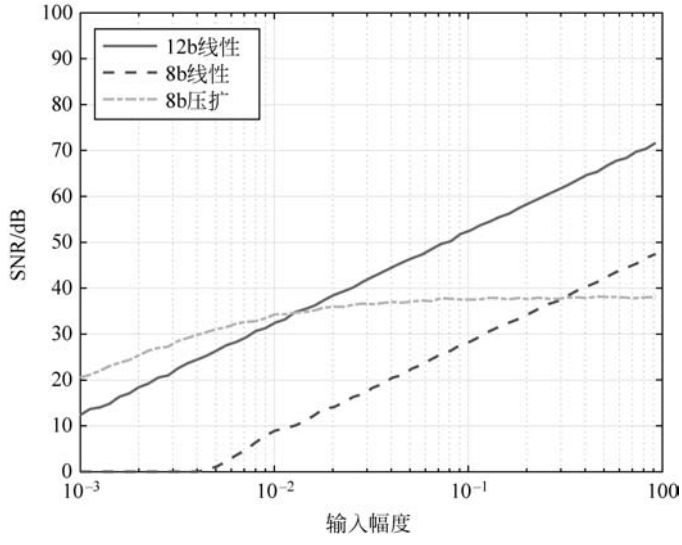


图 5.7 线性量化与压扩量化器的性能比较

注：压扩的折中是显而易见的，低电平时性能更好，高电平时性能较差。

5.5.3 非均匀步长量化

如5.5.2节所述，使用非均匀步长确实有一些优点。具体而言，幅度越大，步长越大；幅度越小，步长越小，这样对语音信号的效果很好。使用非线性输入输出特性进行压扩是实现这个效果的一种方法，用到了线性A/D和D/A转换。另一种方法就是将非线性步长构建到量化器本身，这可以使用Lloyd-Max算法来完成。这种方法假设概率密度函数是不均匀的，也就是说，某些信号幅度比起其他信号幅度更有可能出现。

Lloyd-Max算法假设输入信号的概率密度函数是已知的(Jayant, et al., 1990)，这就意味着必须基于典型数据做出假设。两种适合于近似真实数据的分布是拉普拉斯概率密度函数

$$f(x)=\frac{1}{\sigma\sqrt{2}}\mathrm{e}^{-\sqrt{2}|x-\bar{x}|/\sigma} \tag{5.37}$$

和高斯概率密度函数

$$f(x)=\frac{1}{\sigma\sqrt{2\pi}}\mathrm{e}^{-(x-\bar{x})^2/2\sigma^2} \tag{5.38}$$

为了描述Lloyd-Max算法，首先使用前面出现过的量化误差表达式

$$e_{\mathrm{q}}^2=\sum_{k=1}^{L}\int_{x_k}^{x_{k+1}}(x-y_k)^2 f(x)\mathrm{d}x \tag{5.39}$$

然而，现在决策(输入)电平 x_k 和重构(输出)电平 y_k 都可以变化。因此，为了解决最小失真，需要改变这些量，并找到以下方程的解。

$$\frac{\partial e_{\mathrm{q}}^2}{\partial x_k}=0,\quad k=2,3,\cdots,L \tag{5.40}$$

$$\frac{\partial e_{\mathrm{q}}^2}{\partial y_k}=0,\quad k=1,2,\cdots,L \tag{5.41}$$

除了少数简单的概率密度函数，上述方程很难求出解析解，此时需要采用迭代的数值方法求解。此外，从式(5.39)中注意到：

(1) 决策电平 x_k 在相邻重构电平的中间；

(2) 每个重构电平 y_k 是适当间隔内的概率密度函数的质心。

从数学上说，这就可以采用Lloyd-Max算法(Jayant, et al., 1990)，该算法按照以下方式迭代求解。

$$x_k^*=\frac{1}{2}(y_k^*+y_{k-1}^*),\quad k=2,3,\cdots,L$$

$$x_1^*=-\infty$$

$$x_{L+1}^*=+\infty$$

$$y_k^*=\frac{\int_{x_k^*}^{x_{k+1}^*}xf(x)\mathrm{d}x}{\int_{x_k^*}^{x_{k+1}^*}f(x)\mathrm{d}x},\quad k=1,2,\cdots,L \tag{5.42}$$

迭代上述方程组直至收敛。图 5.8 和图 5.9 分别说明了 $L=4$ 和 $L=8$ 个电平的过程。请注意，迭代不需要任何导数的显式计算，只需要评估概率密度函数和进行决策-重构电平的逐步调整。

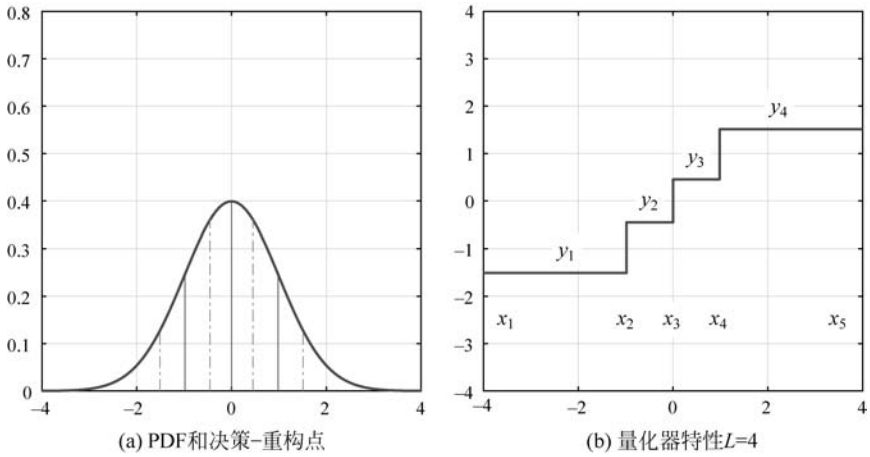


图 5.8 Lloyd-Max PDF 优化量化器($L=4$)

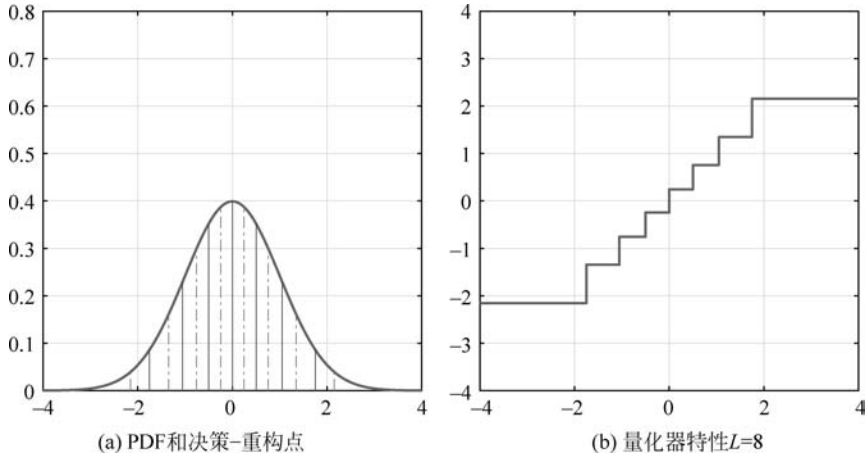


图 5.9 Lloyd-Max PDF 优化量化器($L=8$)

5.5.4 自适应标量量化

前面章节讨论的量化器在设计完成后就固定了。编码器工作时，总是使用相同的决策点和重构电平。然而，语音和音频信号不是不变的，它们会随着时间变化。这就引出了在信号编码时改变步长的思路。对于小幅度信号，步长可以固定，但是很小。对于大幅度信号，步长可以更大，但要再次固定。根据输入信号的相对大小决定使用小步长还是大步长。这可以基于采样点的瞬时值来计算，或根据一组采样点的能量(方差)来计算。

上述方案需要注意一些细节。首先，一些信号，如语音，往往会从小振幅快速变为大振幅。因此，在调整步长时，也必须快速地改变。同样，当信号的平均电平降低时，步长必须快速降低。但此过程又不能进行得太快，因此必须注意能量估算的块大小。实际上，这种方案必须基于过去的采样点估计当前和

未来采样点的能量。

如果可以允许缓存少量的信号，就可以基于“未来”采样点估算方差，也就是解码器暂时还没有看到的那些采样点。这个延迟需要保持得很小，以便不被察觉。重要的是，在给定时间内所使用的步长必须以某种方式发送给解码器，以便它对于给定的二进制码字，能知道重构使用的正确幅度电平。步长通过边信道传输，也就是交织在编码比特流中。这个系统称为前向自适应量化（Adaptive Quantization Forwards，AQF）。

另一种选择是后向自适应量化（Adaptive Quantization Backwards，AQB），即编码器和解码器使用的步长都是基于过去量化采样点的估计。这是可行的，因为解码器知道它已经重构的采样点。这种方法不需要步长的显式传输，然而，由于它基于过去的采样点，步长的自适应调整可能需要更长的时间并且不太精确。

5.5.5　矢量量化

前面所有关于量化的讨论都假设一次只有一个采样点被量化。然而，如果不是一个，而是一个数据块中的几个采样点同时被量化了呢？这种技术称为矢量量化（Vector Quantization，VQ），而与此区别，传统的一次量化一个采样点则称为标量量化。图 5.10 显示了一个 VQ 码本的设计，它包含了用于匹配源采样点矢量的代表性码矢量。

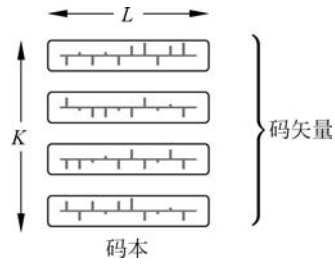


图 5.10　VQ 码本的设计

注：每个码矢量的维数为 L，共有 K 个码矢量。

矢量量化可以代表一维矢量的量化，通常都是这样的情况。然而，当量化块（如图像）时，这种方法可能称为“矩阵量化”更好。然而，矢量量化一词通常对于 $N\times1$ 和 $N\times M$ 的数据块都是适用的。

一次量化几个采样点的主要优点是可以利用采样点之间的相关性。首先，编码器和解码器都有一个相同的“典型”矢量的码本（Codebook），构造该码本的数据源在统计上类似于实际中要量化的数据。码本由大量的码矢量组成，每个码矢量的尺寸等于要编码的数据块（矢量）的大小。然后，编码器将得到的采样点缓冲为大小与矢量维数相等的块，并在码本中搜索最接近的匹配。匹配标准必须是具有某种数学相似性的形式，通常选择具有最小均方误差的码矢量。然后，最接近匹配的索引被发送到解码器。接着，解码器简单地使用这个索引查找相应的码矢量。最后从该码矢量中读取单个的采样点，以重建采样流。上述过程如图 5.11 所示。

这种方法有几个明显的问题，最明显的是在一开始如何生成码本，这个问题放在后面讨论。首先来考虑 VQ 的数据缩减的可能性。假设问题是对有 8 个采样点的数据块进行编码，每个采样点的精度为 8 位。当使用传统的标量量化时，需要的位数为

$$8\frac{\cancel{\text{sample}}}{\text{vector}}\times 8\frac{\text{b}}{\cancel{\text{sample}}}=64\frac{\text{b}}{\text{vector}} \tag{5.43}$$

假设在这个例子中，码本包含 1024 项。也就是说，1024 个维度为 8 的矢量。索引该码本所需的比特位数为 lb1024＝10。那么使用了 VQ 的平均比特率为

$$\frac{10\text{b/vector}}{8\text{sample/vector}}=1.25\frac{\text{b}}{\text{sample}} \tag{5.44}$$

换句话说，每个矢量的比特位数已经从 64（使用标量量化）降低到 10（使用 VQ）；同样，每个采样点的比

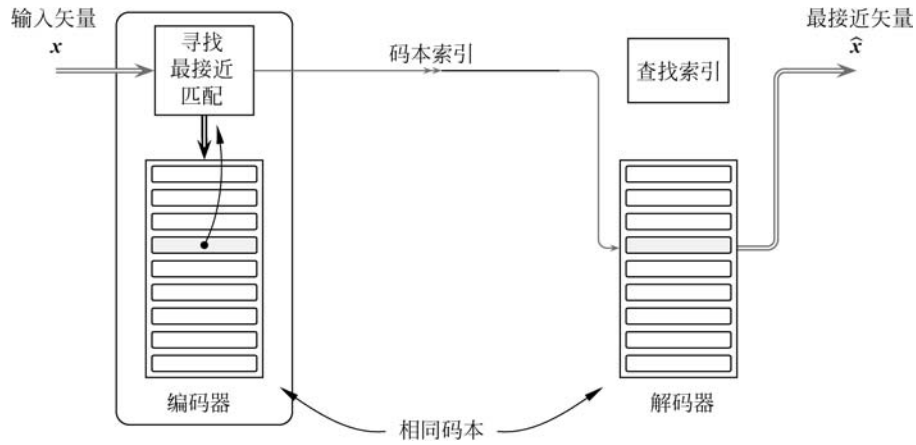


图 5.11 用矢量量化器的编码与解码

注：编码器必须搜索最佳匹配矢量，而解码器只须查找索引（通过信道传输）对应的矢量。

特位数从 8 降低到 1.25。

从这个简单的例子中，应该清楚地知道，增加码矢量的大小（不是码矢量的数量）会降低平均速率，这也是所希望的。然而，平均失真将会显著增加，因为码本必须用较少的码矢量来近似更大数量的样本矢量。每个码矢量必须有效地代表更大范围的源矢量。

另外，增加码矢量的数量（即码本中有更多项）意味着有更多的代表性码矢量可供选择，因此，平均失真会减小。然而，这也意味着要搜索更多的矢量，矢量索引只增加一位就会使码本的大小翻倍。

因此，计算复杂度可能成为一个重大障碍。考虑到编码器搜索码本时必然会执行操作。记源矢量为 $\boldsymbol{x}$，码本矢量为 $\boldsymbol{c}_k$，对于维度为 L 的矢量，其误差（或失真）计算为

$$d_k = \sum_{j=0}^{L-1} (x_j - c_{kj})^2 \tag{5.45}$$

在本例中，码本共有 1024 项，每项的维度均为 8。每个矢量的比较操作需要分别进行 8 次减法、平方和加法运算，来计算均方误差。如果这个代码是在顺序循环中实现的，那么索引矢量值还会有额外的开销，而迭代块也会有循环开销。因此，计算量的近似值约为每个矢量 25 次运算，乘以码本中的 1000 个矢量，共 25 000 次运算。每个待编码的矢量（块）都需要这样的操作，这导致计算量相当大。增加码本大小或码矢量维数都会增加这个值。此外，增加码本大小会导致复杂度的指数性增长，因为添加到矢量索引的每个额外比特都会使码本的大小翻倍。

上面概述的方法称为穷举搜索（Exhaustive Search），因为对于每个编码块，码本中每个可能的码矢量都要被检测。目前已经提出了各种方法来降低搜索的复杂度，通常采用的折中策略是增大存储空间。

使用穷举搜索的编码可以更加正式地定义如下。

（1）给定码本大小为 K，矢量维数为 L。

（2）对于每个待编码的源矢量 $\boldsymbol{x}$，将失真 $d_{\min}$ 设置为一个非常大的数字，对于每个码矢量（索引为 k），完成以下步骤：

a）对于码本 $\boldsymbol{C}$ 中的每项候选 $\boldsymbol{c}_k$，使用所选度量计算失真，通常是平方误差 $d_k = \| \boldsymbol{x} - \boldsymbol{c}_k \|^2$；

b）如果失真 d 小于目前为止的最小失真（$d_{\min}$），设置 $d_k \rightarrow d_{\min}$ 并保存索引 $k \rightarrow k^*$。目前最接近

的矢量 $\hat{\boldsymbol{x}}$ 保存为此码矢量：$\boldsymbol{c}_k \rightarrow \hat{\boldsymbol{x}}$；

c）对码本中的所有 K 个码矢量重复上述操作。

而解码器只须使用传输的索引 k^* 查找相应的码矢量。显然，编码器的复杂度远远大于解码器。这种方案称为不对称编码器（Asymmetric Coder）。

如果码本是待编码数据样本的合理表示，则该方法是令人满意的。但是包含着所有码矢量的码本又是如何确定的呢？与标量量化器所概述的解析方案不同，通常会使用典型的源矢量训练得到迭代解。假设这些训练矢量在实践中能够以足够的精度表示源数据，这种训练算法本质上是将典型的源矢量映射到若干聚类中。图 5.12 说明了映射过程。除了矢量维数和码本大小外，还需要选择足够的训练矢量，这些矢量代表了典型的源特征。

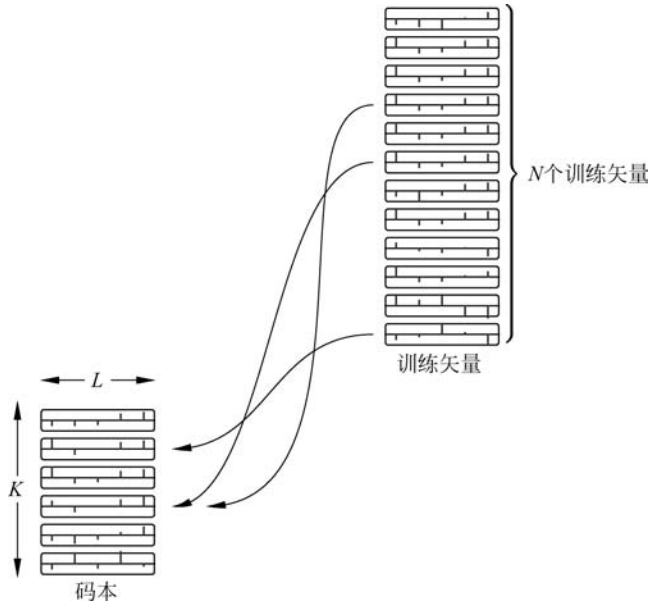


图 5.12　VQ 训练过程

注：多个的训练矢量可能映射到同一个给定的码本矢量。

训练算法有很多选择。k-Means 或 LBG 训练算法步骤基本如下。

（1）确定一个大小为 K，矢量维度为 L 的码本 $\boldsymbol{C}$。

（2）得到 N 个特征矢量 $\boldsymbol{x}_k$，$k=0,1,\cdots,N-1$。

（3）任意选择 K 个特征矢量来创建初始码本 $\boldsymbol{C}$。

（4）对于训练集中的每个矢量：

a）在当前码本中进行搜索以找到最接近的近似值；

b）将此矢量 $\tilde{\boldsymbol{x}}_k$ 保存在索引（码字）k 处，增加索引 k 的映射矢量的计数。

（5）对于码本中的每个码矢量：

a）利用保存的矢量集合和计数计算映射到码矢量的所有训练矢量的质心；

b）用这个质心替换码矢量。

（6）用新的码本对每个训练矢量进行编码，并计算平均失真。如果足够小，退出；否则，从步骤（4）

开始重复。

质心(或平均值)的计算步骤可以用图 5.13 的 Voronoi 图表示,该图描绘了一个二维的 VQ。每个输入矢量都由空间中的一个点来表示,每个矢量的两个分量分别是横纵坐标。用圆圈表示的每个码矢量,对于落在指定区域内的任意输入矢量都是最佳匹配。每个区域的边界随着训练过程在每次迭代中更新。使用 MATLAB 中的 voronoi()函数可以很容易地生成这样的 Voronoi 图。

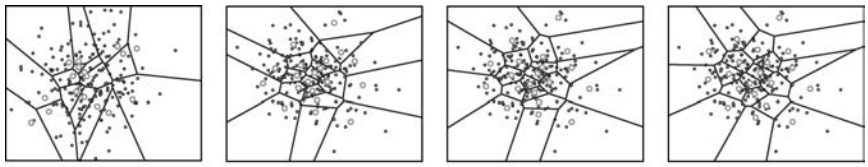


图 5.13 VQ 训练迭代与收敛(小点是训练数据,圆圈是质心)

5.6 信源编码

数字信道上的数据由最低层的比特流组成。然而,传输的内容可以包括文本、图像、声音或其他类型的信息。因此,呈现的内容与编码比特流之间的映射是很有必要的,这称为信源编码(Source Coding)。编码器执行这一映射,解码器执行相反的操作,尽管所涉及的步骤不一定是完全相反的。

不同类型的源文件有不同的要求。最基本的是对精确重构的要求,也就是说,接收机所解码的比特流是否与呈现给编码器的比特流完全相同。虽然看起来精确的逐位匹配是必要的,但并不总是如此。例如,如果要重构数据文件,那么精确的比特传输显然是必要的。但是有些信道会有误差,可能很难始终保证收到的比特是正确的。

此外,如果愿意牺牲比特精确性,就有可能对某些类型的源数据进行更多的压缩。图像、语音和视频是这一类型中最常见的。如果单个像素或音频采样点损坏了,可能对声音或图像的感知质量影响很小或没有影响。事实上,在不同的场景下,相对较高的误差水平是可以容忍的。如果一个特定的压缩算法能够显著减少数据量,并且尽可能少地影响感知,那么它可能是一个更好的选择。

精确比特的压缩算法称为无损(Lossless)压缩,而有损(Lossy)压缩算法可能会通过牺牲一些重构的精确性来实现更大的压缩。这两种技术在实践中都被广泛应用,并且经常在给定的电信应用中相互补充。

在不同的应用中,可能需要压缩文本或媒体(如音频或视频)。编码器发出的数据由符号(Symbol)组成,这些符号以某种转换的方式表示数据。可用的符号集通常称为字母表(Alphabet),它与含有字母的字母表类似,但具有更一般的含义。例如,如果一幅图像仅由 64 种颜色组成,那么字母表就由这 64 种颜色中的每一种所组成,并且需要 lb64=6b 来唯一指定每个符号。然而,一个符号还可以代表更多,如一个符号可以表示有一定顺序的几个像素。

5.6.1 无损编码

本节介绍无损编码,即可以准确无误地再现源数据流的编码方法。这种类型的编码也称为熵编码(Entropy Codes)。

1. 熵与码字

在对数据源进行编码时，有必要确定待编码值的可能范围。正如将要显示的，每个单独的值的概率是很重要的。从理论上说，信息源的信息速率(或熵)决定了对信息源发出的数据进行编码所需的最小比特位数。那么编码这些数据需要多少位呢？熵回答了这个问题。例如，如果只需要编码英文字母表中的大写字母，那么就需要 26 种二进制表达式。这意味着至少需要 5 位来覆盖 32 种可能性(因为 $2^5=32$，大于 26)。反过来，最低要求是 lb26 位。计算中隐含的假设是，这些表达式中的每一个都是等可能的。这种假设在实践中通常不会成立，这是一件好事。事实证明，符号出现的可能性不相等是一个有利条件。

熵的概念正是用在这里。对于从字母表 S 中提取的源符号 s_i，每个符号出现的概率记为 $\Pr(s_i)$，则熵的定义为

$$H=-\sum_{s_i\in S}\Pr(s_i)\mathrm{lb}\Pr(s_i)\quad \mathrm{b/symbol} \tag{5.46}$$

熵是理论上的下边界，它显示了给定符号概率的情况下，编码每个符号所需的最小比特位数。重要的是，可以看到，如果概率并不相等，还可能有机会减少熵，减少每个符号的平均比特位数。所以问题是：要如何将比特分配给每个符号，以便最有效地利用比特？

对于给定的信息源，概率可能是固定的，或者可能随时间稍有变化。如果是这种情况，如何将二进制码字分配给每个源符号，以便最小化输出比特率？将较短的码字分配给更可能出现的符号，将其他剩余的更长的码字分配给不太可能出现的符号，似乎是合乎逻辑的。数学上，希望最小化符号块上编码的平均码字长度。如果可以这样做，知道了每个源符号的码字长度，那么平均码字长度将是加权平均值，计算如下。

$$L_{\mathrm{av}}=\sum_{s_i\in S}\Pr(s_i)L(s_i)\quad \mathrm{b/symbol} \tag{5.47}$$

理论上这是很好的，但是如何解出最佳码字分配呢？举一个具体的例子，假设有 4 个符号：A、B、C 和 D。如果分配码字，令符号 A 为 0110，符号 B 为 101，符号 C 为 011，符号 D 为 10，那么就可以得到一个可变字长编码(Variable-Wordlength Code，VWLC)。根据前面的讨论，只有在 D 比 A 更有可能出现的情况下这才有意义，因为如果 D 更有可能出现并且有更短的码字长度，那么总体上才可能需要更少的比特。

根据式(5.47)，总的平均码字长度将更小。请注意，可能最后得到的是小数比特率，因为目标量是每个符号的平均比特位数。例如，给定符号 A、B、C 和 D 的比特长度分别为 4、3、3 和 2，假设每个符号的概率为 25%。根据式(5.47)，平均码字长度将是 $0.25\times4+0.25\times3+0.25\times3+0.25\times3+0.25\times2=3.0$。但是假设概率实际上是 50%、20%、20%和 10%，在这种情况下，计算结果如下。

```
% 表示每个符号的比特长度
s = [4  3  3  2];

% 每个符号的概率
pr = [0.5  0.2  0.2  0.1];

% 验算——概率的和应为 1
sum(pr)
ans =
    1.0000
```

```
%平均码字长度
sum(pr.*s)
ans =
    3.4000
```

平均值为 3.4b/symbol。这不是一个整数，即使每个单独的符号都被分配了一个整数的比特位数。

对于给定的码字分配，解码中有可能出现混淆。假设发送了二进制流 1010110。解码器可以将其解码为 101 再接 0110，这就等同于源符号为 BA。或者，它可以被解码为 10 101 10，这就是 DBD。为了避免这种歧义，可以使用一种特殊的比特表达式作为分隔符，即“逗号”。假设 110 表示这个分隔符，那么有分隔符编码的块 ABCD 就变成 A，B，C，D，编码结果将是 0110 110 101 110 011 110 10。这种方法的问题很明显，它增加了信息块中的平均比特位数。此外，仍然需要非常小心，使分隔符表达式不会出现在任何合法码字的开头，甚至不会在合法码字的中间，因为这会使解码器混淆。这两个问题都有一个巧妙的解决方法，叫作霍夫曼编码（Huffman，1952）。霍夫曼编码既解决了解码混淆问题，又为每个符号分配了一个整数的最小长度编码。

2. 霍夫曼编码

式(5.46)清楚地表明，信息源的熵取决于符号概率，它提供了一个最低的理论界限，但没有说明如何以最佳方式分配码字。因此，还需要某种方法为每个符号分配比特，从而形成码字。在理想情况下，L_{av} 等于熵 H。但这只是理论上的，在实际中平均码字长度接近熵，但从未达到熵。本质上，理论的下边界给了我们努力的方向。

霍夫曼编码以最佳方式为符号分配一个整数的比特位数来形成码字。除了减小平均码字长度之外，这些编码还是唯一可解的，即一个码字的结束和下一个码字的开始不会混淆。重要的是，不需要使用特殊的分隔符。

霍夫曼编码的工作原理如下。假设有一组已知概率的源符号，如图 5.14 所示。叶节点是已定义的符号值（在此例中为 A、B、C、D 和 E），并且已经根据前面符号的组合对内部节点进行了注释，如 DE 是 D 和 E 的组合。这不是算法要求的一部分，但是它使接下来的编码开发变得更加容易。

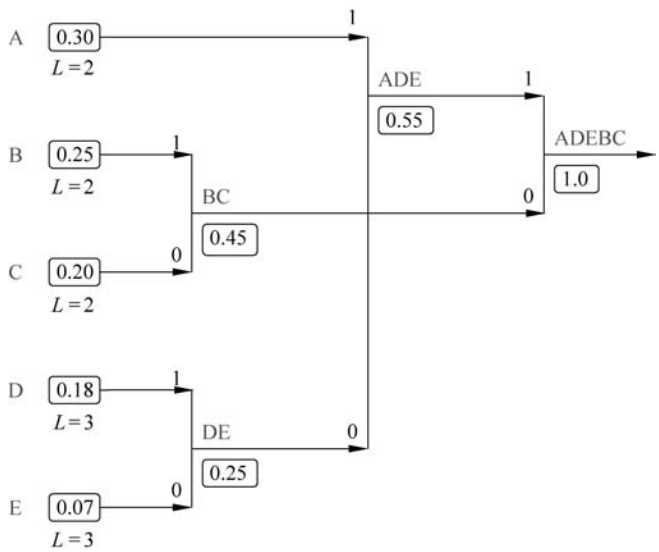


图 5.14　霍夫曼树的生成过程

注：按照惯例，组合两个节点时，为更高概率的节点分配比特 1。每一步的概率以图中所示的方式组合，最终得到的平均长度为 2.25b/symbol。

首先，将符号及其对应的概率画在左边的一列，这些称为叶节点。然后，继续组合符号，一次两个，以创建中间节点。例如，D 和 E 是组合的，组合概率为 0.18＋0.07＝0.25。约定用二进制 1 表示较高的概率，0 表示较低的概率（当然这是任意的，只要保持一致，任何约定都可以使用）。B 和 C 也是如此。然后，以相同的方式将这些中间节点与叶节点或其他中间节点重新组合，即将概率和为 0.25（标记为 DE）的节点和 0.30（A）的节点组合，得到 0.55（标记为 ADE），然后将此结果与 0.45（BC）组合。最后，到达右边的单个节点（根节点）结束，其概率明显应该等于 1.0，因为达到这个阶段已经将所有的源概率相加（尽管是间接地通过连续配对相加）。

图 5.15 显示了使用这种（侧向）树结构编码源符号的过程。假设希望编码符号 D。从叶节点开始，沿着路径到达最右边的根节点。在这个路径中，记录下经过的节点。所以在第一个中间节点，得到一个 1，因为在这个编码设计阶段，经过的是概率较高的上分支（0.18＞0.07）。沿着所示路径，接下来会在交叉点遇到 0（0.25＜0.30）。最后，在根节点，比特值为 1（0.55＞0.45）。

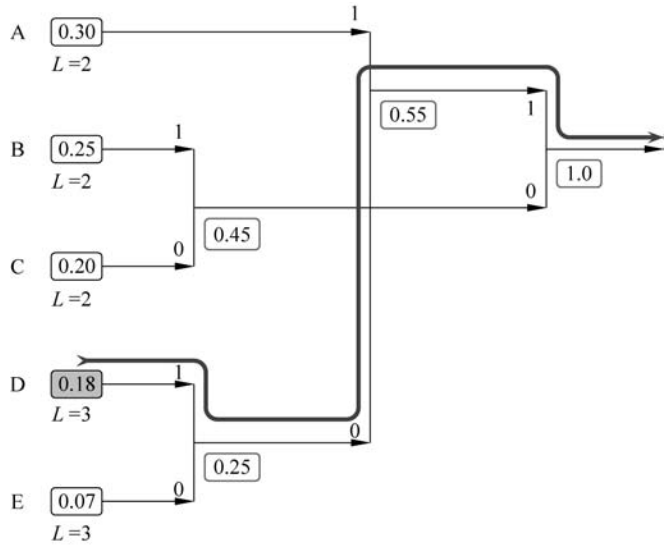


图 5.15　霍夫曼编码的编码过程

注：从对应待编码符号的叶节点开始，遵循节点的连接，直到到达根节点。每个节点处路径经过的分支决定编码比特值。

接收机的解码过程与编码相反。如图 5.16 所示，有一个相同的树，但是现在要从右向左遍历。这是有意义的，因为解码器在开始时并不知道下一个预期的符号，它只在比特流到达时获取到单个比特。从根节点开始，如果接收到 1，意味着必须取上分支。接下来，取图 5.16 中的下分支（0），然后根据比特值 1 取上分支。最后，到达叶节点，显示了符号 D。

因此，解码过程与编码过程相反。在编码中，从叶节点开始，根据遇到下一个节点的分支来分配 1/0，并递归重复，直到到达根节点。解码器从根节点开始，根据接收到的 1/0，来选择分支。显然，符号的比特流在发送到解码器之前必须反转。

在构造霍夫曼编码树时，用于选择分支的 1/0 分配是任意的。上例中用 1 表示较高的概率，称为上分支；用 0 表示较低的概率，称为下分支。只要编码器和解码器使用的是相同的约定，就没有问题。然而，在每个节点处组合符号的可能性却有很多。图 5.17 显示了一组不同的组合，得到了另一个霍夫曼树。仔细检查将会发现一个不同之处，就在于概率为 0.25 的组合节点 DE 被组合时。在图 5.14 的情况下，节点 B（概率为 0.25）和节点 C（概率为 0.20）组合在一起，而在图 5.17 中，已经组合的节点 DE（概

率为 0.25)与节点 C 再组合在一起。这种不同的树结构将导致不同的码字分配。事实上,平均码字长度仍然是一样的,但应该清楚的是,由于节点可以组合的方式多种多样,所以可能存在多种树结构。

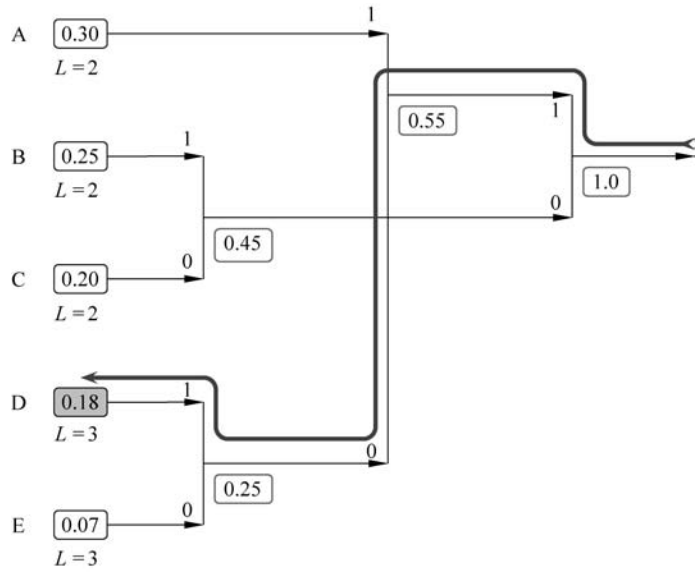


图 5.16 霍夫曼编码的解码过程

注:从根节点开始,接收到的每个连续比特决定了在每个节点取哪个分支,直到到达叶节点,这对应待解码的符号。

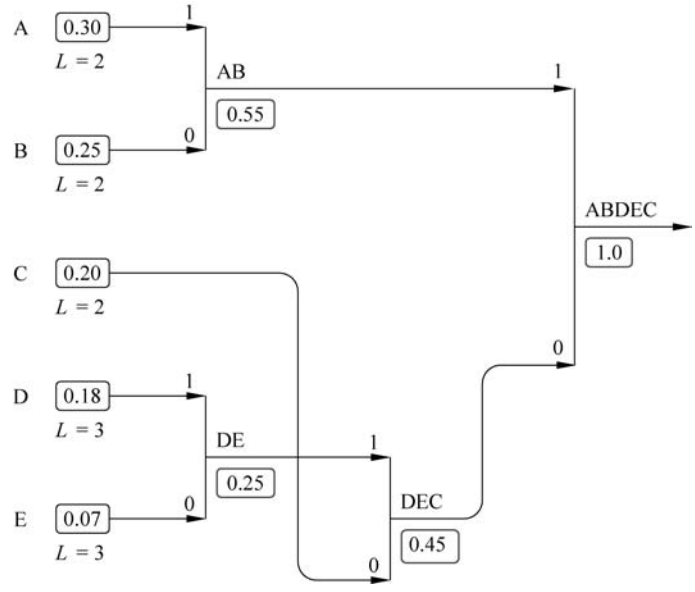


图 5.17 一个使用了替代分组的霍夫曼树生成过程

注:在每个阶段,两个最小概率被组合成一个新的内部节点。

然而,图 5.18 代表了一种不同的情况,导致平均码字长度更长。如何构造树来保证得到的结果有着最短平均码字长度呢?一个简单的规则是在每次迭代中只组合概率最小的那一对,这称为兄弟性质(Sibling Property)(Gallager,1978)。比较图 5.14 和图 5.18,可以应用式(5.47)计算平均码字长度,如

下所示。

```
%概率
p = [0.30  0.25  0.20  0.18  0.07];

%遵循和不遵循兄弟性质的码长度比较
nsib = [2  2  2  3  3];
nnosib = [1  2  3  4  4];

%平均码字长度——兄弟节点相组合
sum(p. * nsib)
ans =
    2.2500
%平均码字程度——非兄弟节点相组合
sum(p. * nnosib)
ans =
    2.4000
```

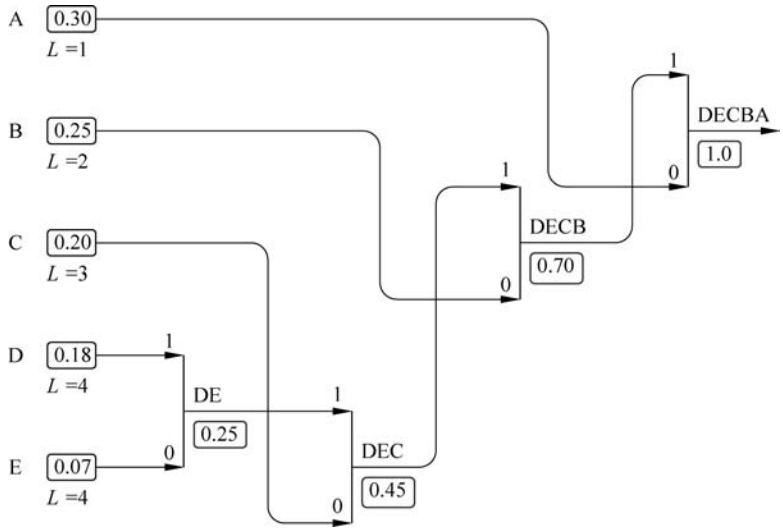


图 5.18　平均码字长度变长的霍夫曼树的生成过程

注：当每个阶段概率最小的节点(兄弟节点)没有按顺序连接时，平均码字长度为 2.40b/symbol。

因此，在遵循兄弟性质构建的树中，任意长消息的平均长度将接近 2.25b/symbol。然而，对于另一棵没有遵循兄弟性质构建的树，平均码字长度稍微长一点，为 2.40b/symbol。0.15b/symbol 的差异可能很小，但是在一个大的编码块上，这个小差异可能会变得显著。此外，在典型的编码场景中，源符号的数量将比这个简单示例中考虑的 5 个符号的字母表要大得多(数百个)。

请注意，仍然存在一些可能的模糊性，如用相同的概率构造的两种霍夫曼编码。在这种情况下，有两个概率均为 0.25 的节点，它们的组合顺序是任意的。自然，当遇到这种情况时，编码器和解码器必须遵循相同的决策逻辑。

除非是码字规模非常小的例子，霍夫曼编码的构造其实是相当冗长的。下面介绍如何在 MATLAB 中进行树的构造、编码和解码。代码使用 handle 操作符创建了一个指向节点的引用(或指

针),数据结构和 MATLAB 中的引用在 4.3.3 节中介绍过。每个节点都有一个与之相关联的概率和符号,内部节点根据其包含的组合节点分配到一个复合符号,以帮助理解。每个节点还分配了向上和向下的指针,而对于叶节点会是空指针。存储在每个节点中的比特值代表了从父节点到达该节点取的是上分支或下分支。

```
% 霍夫曼节点类,返回一个数据结构的句柄(指针)
classdef HuffmanNode < handle
    properties (SetAccess = private)
        Sym                                 % 该节点的符号
        Prob                                % 该符号的概率
    end

    properties
        IsRoot                              % 根节点
        IsLeaf                              % 叶节点或分支节点
        hUp                                 % 上指针对应分支 1
        hDown                               % 下指针对应分支 0
        hParent                             % 父节点
        BitValue                            % 1 代表向上,概率更大,0 代表向下,概率更小
    end

    methods
      function hNode = HuffmanNode(IsRoot, IsLeaf, Sym, Prob, hUp, hDown)

          hNode.IsRoot = IsRoot;
          hNode.IsLeaf = IsLeaf;
          hNode.Sym = Sym;
          hNode.Prob = Prob;

          if( nargin == 4 )
              hUp = HuffmanNode.empty;      % 未给定向上或向下的指针
              hDown = HuffmanNode.empty;    % 所以初始化为空
          end
          hNode.hUp = hUp;
          hNode.hDown = hDown;
          hNode.hParent = HuffmanNode.empty;
          hNode.BitValue = '-';
      end
    end
end
```

霍夫曼树本身由节点组成,根节点是搜索开始的起点。节点可以是叶节点(有符号和概率)或两个分支连接处的节点。

```
classdef HuffmanTree < handle
    properties (SetAccess = private)
        hRootNode                   % 根节点本身
        hAllNodes                   % 所有节点列表
    end
```

```
    methods
        % 构造函数
        function hTree = HuffmanTree(Syms, Probs)
            hRootNode = hTree.CreateTree(Syms, Probs);
        end

        %此处添加树方法
    end %结束方法
end
```

从根节点开始向下搜索树(用于解码),包括依次读取每个新比特,并取适当的分支去到向上或向下的节点,搜索在叶节点处终止。

```
function [DecSym] = Descend(hTree, BitStr)
    BitsGiven = length(BitStr);
    hCurrNode = hTree.hRootNode;

    BitsUsed = 0;
    while( ~hCurrNode.IsLeaf )
        BitsUsed = BitsUsed + 1;
        if( BitsUsed > length(BitStr) )
            % 解码符号
            DecSym = [];

            fprintf(1, 'Descend(): not enough bits\n');
            return;
        end

        CurrBit = BitStr(BitsUsed);
        fprintf(1, 'Currbit %c\n', CurrBit);
        if( CurrBit == '1' )
            % 向上
            hCurrNode = hCurrNode.hUp;
        else
            hCurrNode = hCurrNode.hDown;
        end
        disp(hCurrNode);
    end
    fprintf(1, 'At Leaf, symbol "%s" bits used = %d (given %d)\n', ...
                hCurrNode.Sym, BitsUsed, BitsGiven);

    if( BitsUsed == BitsGiven )
        DecSym = hCurrNode.Sym;
    else
        DecSym = [];
        fprintf(1, 'Error: bits used %d, bits given %d\n', ...
                    BitsUsed, BitsGiven);
    end
end
```

对于给定的符号，从叶节点开始向上搜索树（用于编码）。连续获取每个父指针，并将每个阶段的比特累积得到比特串。当向下搜索树时，比特串必须反转。

```
function [BitStr] = Ascend(hTree, Sym)
    % 在叶节点列表中找到开始节点
    NumNodes = length(hTree.hAllNodes);
    fprintf(1, 'Searching %d leaf nodes\n', NumNodes);
    StartLeaf = 0;
    for n = 1:NumNodes
        if( hTree.hAllNodes(n).IsLeaf )
            if( hTree.hAllNodes(n).Sym == Sym )
                StartLeaf = n;
                break;
            end
        end
    end

    if( StartLeaf == 0 )
        fprintf(1, 'Ascend() error: cannot find symbol in nodes\n');
        BitStr = [];                    %信号错误
        return;
    end

    % 对于给定的符号，从叶节点开始
    hCurrNode = hTree.hAllNodes(StartLeaf);
    BitStr = [];                        %从叶节点到根节点累积比特串

    while( ~isempty(hCurrNode) )
        disp(hCurrNode);

        if( ~hCurrNode.IsRoot )
            %沿着路线累积比特
            BitStr = [BitStr hCurrNode.BitValue];
        end

        hCurrNode = hCurrNode.hParent;
    end
end
```

实际上，创建霍夫曼树是最复杂的一步。首先，将给定的符号分配给叶节点。然后，确定所有候选配对的节点，并按照概率递增的顺序进行排序。请注意，搜索中的节点可能是叶节点或内部节点，一旦某个节点已被组合，它将被标记为不再可用。注释中提示了，简单地按照每对出现的顺序进行选择的位置，不能保证正确的兄弟顺序。根据兄弟顺序组合需要根据每个节点的概率进行排序。

一旦确定了下一对要组合的节点，一个新的父节点就被创建了，设置每个子节点的向上指针，以及父节点指向每个子节点的向上/向下指针。重复上述过程直到所有节点都被组合，此时到达根节点。

```
function hRootNode = CreateTree(hTree, Syms, Probs)
    % 创建叶节点
    Paired = []; % zeros(N, 1);
    N = length(Syms);
    for n = 1:N
        IsRoot = false;
        IsLeaf = true;
        hNewNode = HuffmanNode(IsRoot, IsLeaf, Syms(n), Probs(n));
        hTree.hAllNodes = [hTree.hAllNodes hNewNode];
        Paired = [Paired false];
    end
    % 成对组合节点,有 N-1 个配对节点
    for n = 1:N-1

        PairList = [];
        for k = 1:length(hTree.hAllNodes)
            if( ~Paired(k) )
                % 候选配对
                PairList = [PairList k];
            end
        end
        % 只须选择前两个,不保证是兄弟顺序
        % 因此也不保证产生最短的可能码字
        iup = PairList(1);
        idown = PairList(2);
        % 一个更好的方法是按照概率递增的顺序排序
        % 然后选择最小的两个进行组合
        ProbVals = [];
        for p = 1:length(PairList)
            i = PairList(p);
            ProbVals = [ProbVals hTree.hAllNodes(i).Prob];
        end

        %以升序对所有节点概率进行排序,返回排序的索引
        [SortedVals, SortIdx] = sort(ProbVals);

        %从配对列表中选择两个最小的概率,即排序索引列表中的前两个
        iup = PairList(SortIdx(2));
        idown = PairList(SortIdx(1));

        % 标记为已组合
        Paired(iup) = true;
        Paired(idown) = true;

        fprintf(1, 'selected up : %d, sym:" %s" prob: %f\n', ...
                  iup, hTree.hAllNodes(iup).Sym, hTree.hAllNodes(iup).Prob);
        fprintf(1, 'selected down: %d, sym:" %s" prob: %f\n', ...
                  idown, hTree.hAllNodes(idown).Sym, hTree.hAllNodes(idown).Prob);

        % 创建具有概率和的父节点
```

```
            ProbSum = hTree.hAllNodes(iup).Prob + hTree.hAllNodes(idown).Prob;

            % 通过组合子节点的名称,创建一个虚拟节点名称
            NodeSym = [hTree.hAllNodes(iup).Sym hTree.hAllNodes(idown).Sym];
            IsRoot = false;
            IsLeaf = false;
            hNewNode = HuffmanNode(IsRoot, IsLeaf, NodeSym, ProbSum, ...
                                        hTree.hAllNodes(iup), ...
                                        hTree.hAllNodes(idown));

            % 子节点指向父节点
            hTree.hAllNodes(iup).hParent = hNewNode;
            hTree.hAllNodes(iup).BitValue = '1';

            % 父节点的向上/向下指针指向子节点
            hTree.hAllNodes(idown).hParent = hNewNode;
            hTree.hAllNodes(idown).BitValue = '0';

            fprintf(1, 'created new node with prob %f\n', ProbSum);
            fprintf(1, 'Parent is: \n');
            disp(hNewNode);

            % 在所有节点列表中保存新的父节点,且尚未组合
            hTree.hAllNodes = [hTree.hAllNodes hNewNode];
            Paired = [Paired false];

            if( n == N-1 )
                % 这仅发生在最后一对的组合中,按定义即为根节点
                hNewNode.IsRoot = true;
                hRootNode = hNewNode;
                fprintf(1, 'Root node saved\n');
                hRootNode.disp();
            end
        end
        % 在树结构本身保存根节点
        hTree.hRootNode = hRootNode;
    end
```

为了测试霍夫曼树代码,可以进行一个测试。首先定义符号及其对应的概率,然后创建树,代码如下。

```
Syms = ['A' 'B' 'C' 'D' 'E'];
Probs = [0.3 0.25 0.20 0.18 0.07];

hTree = HuffmanTree(Syms, Probs);
```

编码一个字母是从叶节点到根节点进行的,使用 Tree.Ascend()函数;解码一个比特串则由 Tree.Descend()函数执行,如下所示。

```
Sym = 'A';
BitStr = hTree.Ascend(Sym);
fprintf(1, 'Symbol "%s" encoded as bit string "%s"\n', Sym, BitStr);

% 反转顺序
TransBitStr = fliplr(BitStr);

BitStr = '101';
DecSym = hTree.Descend(BitStr);
fprintf(1, 'Bit string "%s" decodes to symbol "%s"\n', BitStr, DecSym);
```

3. 调整概率表

编码器和解码器只要存储相同的霍夫曼树，前面建立霍夫曼树的方法就能良好工作。然而，存储该树的前提是符号概率是固定的并且是预先已知的。在特殊情况下，可能并非如此。此外，当数据流被编码时，概率可能会随时间变化。例如，文档的一部分可能先是包含文本，接着是图像。

概率表显然是高效编码的关键。如果有一组准确的统计数据，就能进行高效的编码。前面描述的霍夫曼方法使用固定的静态表，它在设计时就进行了设置。但是如果随着编码的进行，概率发生了改变呢？毕竟，源符号一直在输入，编码器就可以维护一个频率表。解码器也可以维护这样的一个表，因为它接收和解码了与数据源相同的符号集。

霍夫曼树可以根据每个新的压缩需求来构建。这可能适用于压缩数据文件，即通过一次数据来计算频率表，接下来的数据使用该表进行编码。该表本身需要预先添加到数据文件中(或传输)，以便解码器可以创建相同的霍夫曼编码表。然而，这对于电信系统来说并不理想，因为源数据在开始时可能不是都可用。因此，正如 Gallager(1978)指出的，需要一个自适应霍夫曼编码器。它在编码过程中保持了源符号的自适应计数，这对于符号概率事先未知且会随着编码的进行而改变的电信系统是很理想的。整个树不需要重建，只需要在一些节点的相对概率变化足够大的情况下，交换这些节点来进行调整。

当然，解码器必须始终与编码器保持同步，否则可能会出现不正确的解码。当编码器收到每个符号时，对其进行编码并更新表。解码器收到比特流，先解码输出，再更新表。请注意，此处的顺序至关重要，编码器必须在对符号进行编码之后才能更新自己的表，否则解码器的霍夫曼树可能与编码器的不同，导致传输失去同步。实际上，编码树的更新必须滞后一个符号的时间，以便匹配解码器。请注意，没有必要为每个符号完全重建霍夫曼树。根据霍夫曼树的工作原理，很显然，仅当一个符号概率增加到超过其兄弟符号概率时，才需要进行更新。

5.6.2 基于块的无损编码器

先前类型的编码器在编码单个符号时工作良好。但是如果相邻符号之间存在依赖关系呢？举一个简单的例子，假设要压缩屏幕上的一张颜色图像。如果一次取一行图像并尝试压缩它，可能会发现有相同颜色像素的长串。这就产生了一个想法：可以将相同颜色的块(或串)编码为符号对(像素颜色，串长度)，而不是为一个特定颜色编码一种比特模式。这样，就可以充分利用多个符号所产生的冗余。

请注意，颜色实际上可能不是单个字节，它可能是代表红色、绿色和蓝色分量的三元组。因此，正在编码的“符号”实际上可能是一组颜色值。正在编码的串长度值可以是一个字节或更多(或者可能更

少)。显然,需要限制来源于这个字段的大小,如果串长度是 4b 字段,那么串值将被限制为一次 16 个像素。还可能存在效率的权衡,对于与之前颜色不同的单个像素,仍然需要编码一个像素的长度。因此,在最坏的情况下,串长度输出实际上可能会导致数据量变大。

当不考虑单个字母,而是考虑字母组时,这种方法可以更加通用,并可扩展到文本。在解释时,将其视为"单词"可能更简便。在压缩文本时,有些单词比其他单词出现得更加频繁。

压缩数据时,解码器显然需要与编码器保持同步。这似乎是显而易见的,但仔细设计算法来确保始终做到这一点是很有必要的。此外,传输中的错误会导致灾难性的连锁反应。例如,如果由于一位比特的误差,将串长度 45 错误接收为 5,那么图像就会失真。

下面将介绍几种基于块的无损编码器。在下文中,用字符串(String)代表一个符号块。它可能是一个英语单词,但不一定必须如此。事实上,对于压缩算法,单词(Word)的概念完全是任意的,也可以用短语(Phrase)替代。编码器发送特定的值,如上面例子中的(串,长度)对,可以称为标记(Token),它通常用于表示单个条目或一组值。这些需要结合上下文来理解。

1. 滑动窗口无损编码器

一种众所周知的编码符号串或符号块的方法常被称为 Lempel-Ziv(简称为 LZ)算法。事实上,它不是一种算法,而是具有多种变化的一系列算法。

此处以文本编码为例。在某个时间点,解码器获得了最近解码的文本,而编码器可以看到尚未编码的文本块。给定过去的一个数据块,未来的数据可能包含相同或非常相似的符号串。利用这种性质的最简单的方法是游程编码器,但是它对于英语文本来说效果不是特别好。

可以通过以下方式利用数据流中的近因效应。编码器和解码器都需要维护一个最近解码的文本块,如图 5.19 所示。编码器查看要编码的下一个文本块。幸运的话,下一个块的某个短语在过去出现过。因此,为了将下一个符号块传送给解码器,首先需要搜索先前编码的文本以寻找相同的模式。然后发送该模式的索引以及该模式的长度。理想情况下,模式越长越好,因为可以同时编码更多的符号。该算法的执行是通过找到匹配项,然后搜索更长的匹配项,以此类推。直到某个时刻,模式匹配停止。解码器需要知道最后停止匹配的符号。

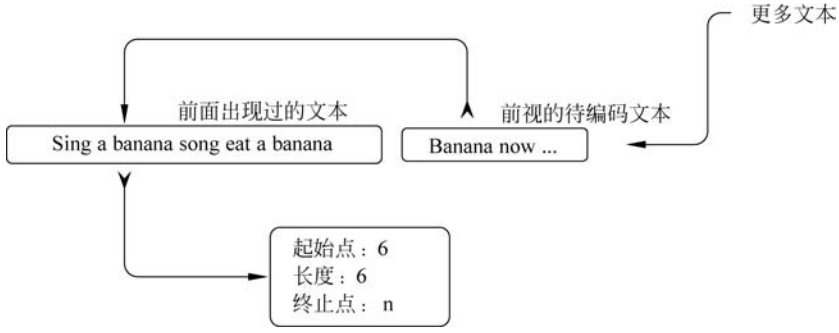


图 5.19 Lempel-Ziv 窗式压缩

注：为了编码"Banana now",需要使用索引 6,即先前编码窗口中的起始偏移量(清晰起见,忽略空格),长度正好也是 6。下一个字节是"n"(同样忽略空格)。

因此,有必要传输三元组(索引,长度,符号),其中最后的符号是遇到的不匹配符号的编码。解码器的工作是在前面出现的文本中,从给定的索引开始,复制相应长度的字节,然后添加不匹配的符号。此时,编码器和解码器可以沿着先前编码文本的窗口滑动,并重复该过程。

作为设计参数，显然有必要确定先前出现文本的缓冲区的长度，因为这决定了索引参数中的比特位数。还必须决定编码器前视块缓冲器中的符号数，因为这将决定长度参数所需的比特位数。一般来说，这些是不一样的，因为需要一个很大的先前数据窗口来最大化找到匹配模式的机会。例如，如果先前编码块为 4kB，前视块为 16B，那么索引将需要 12b，长度需要 4b。发送的每个编码符号将需要 12＋4＋8＝24b。如果平均模式长度为 3，并且模式将在编码缓冲区中一直出现，那么与原始字节流的直接传输相比，编码将会处于平衡。然而，如果找到的平均模式更长，将导致输出数据减少。

这一系列的编码器（尽管有许多变体）通常称为 LZ77 算法（Ziv，et al.，1977）。还可以添加后处理阶段，如利用霍夫曼方法编码（索引，长度，符号）标记或 3 个分量之一（如仅编码符号）。

除了调整缓冲区的最佳长度，计算问题也可能成为挑战。读入的每个新符号都需要在编码器的缓冲区中进行搜索，这可能需要时间。各种数据结构，如基于树的划分，可以用来加速编码搜索。相比之下，解码器仅有一个简单的索引和复制要求。这种类型的方案称为不对称（Asymmetric），因为编码器的复杂度要比解码器大得多。

2. 基于字典的无损编码器

一种使用模式表的相关方法是 LZ78 系列算法（Ziv，et al.，1978）。在这种情况下，使用的是表格，也称为字典（Dictionary），而不是滑动窗口，如图 5.20 所示。重要的是要理解“字典”并不是指英语单词，通常是指这样的表格。在最一般的情况下，字典中的项只是一个字节模式。

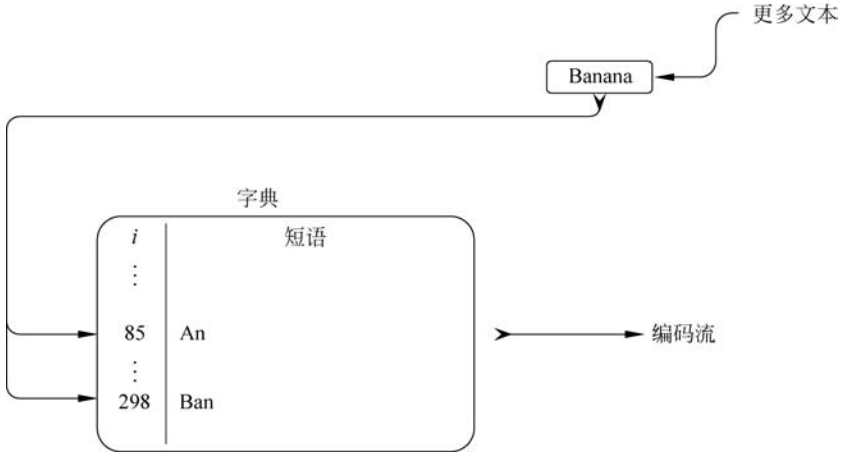


图 5.20　Lempel-Ziv 字典式压缩

注：字典中最长的匹配是“ban”，接着是“an”。接下来编码器和解码器可以将短语“bana”添加到它们各自的字典中。随着输入流中每一次出现“banana”，会逐渐建立“banan”，然后是“banana”，从而使未来的“banana”编码更加高效。

字典编码通过在编码器和解码器上维护字典来实现。编码主要包括在字典中搜索最长的匹配短语，并将该索引传输到解码器。因此，可以省去 LZ77 中的长度字段，因为长度隐含在每一项中。在编码器中搜索最长的匹配短语时，匹配必须在某个点断开。因为解码器需要建立一个相同的字典，所以有必要发送破坏匹配的第一个字符的字节编码，这类似于 LZ77。然后编码器和解码器基于刚刚编码的短语建立另一个短语及新的符号，这将成为后续编码的前缀。不匹配的字符成为附加在现有前缀上的后缀，因此字典是逐步建立的。

一个潜在的改进就是不匹配字符的编码。Lempel-Ziv-Welch（LZW）变体巧妙地解决了这个问题（Welch，1984）。它从一个字典开始，这个字典的第一项包含了所有待编码的标准字符。因此，如果只

对文本进行编码，字典将包括“A”“B”“C”…“Z”。这样，任意之前未出现的字符都可以通过字典索引进行编码。所有需要传输的是一系列索引，随着字典的建立，这些索引将覆盖单个符号和任意其他字符串。

例如，如果 XABC 和 XYZ 都在字典中，那么输入 XABCXYZ，将得到 XABC 的编码输出和 XYZ 的编码输出，以及有了新短语 XABCX 的字典更新。以这种方式更新字典似乎效率不高，但是在压缩了一定量的数据之后，字典将会被常用短语填充，并且编码变得越来越高效。

LZ78 的一个问题是字典最终会被填满。这可能需要一些时间，但总会在某个时候发生。已经有几种策略来应对这种可能性。最简单的方法是删除整个字典（除了 LZW 中单个符号的项），然后重新开始，但扔掉实际上可能在近期出现的短语会有些可惜。可以使用计数值标识每一项的使用频率，称为最少使用频率（Least Frequently Used，LFU）方法。但是，一些经常使用的模式（有很高的计数值）可能出现在一段时间之前。例如，文章前面章节中使用的单词可能会在后面章节中使用，但对于某些单词，其相对频率最终可能会降至零。因此，最近最少使用（Least Recently Used，LRU）方法可能会更好。考虑一个对电话号码列表进行编码的例子，由于索引是按字母顺序排序的，所以名字（对压缩算法来说只是短语）会表现出非常强的局部效应。然而，街道名称将具有非常弱的局部效应。

一个微妙的问题与重复的短语有关，以下例子很好地解释了这一点。假设 XABC 在编码器和解码器的字典中，而输入字符串是 XABCXABCX。编码器将输出 XABC 的编码，将 XABCX 添加到它的字典中，然后从第二个 X 开始尝试寻找匹配，它将在字典中找到 XABCX 并输出该编码。但在那时，解码器字典中还没有 XABCX，它仍在等待跟在 XABC 之后的符号的编码。必须处理这种异常，以确保正确解码。

目前有大量的无损数据压缩算法存在，并有许多的数据传输和存储的应用。一个例子是 LZO（Oberhumer，n. d.），用于火星探测器上的压缩。另一个例子是 BWT（Burrows，et al.，1994），用于一些公共领域的数据压缩程序。在特定情况下算法的选择取决于所需的数据压缩量、复杂度（对应于压缩所需的时间）和可用的内存。

5.6.3 差分脉冲编码调制

前面的章节讨论了无损编码，传输的总是原始数据，这样接收机就可以重新创建数据的精确副本。然而，对于真实世界采样的数据，如语音和图像，没有必要确保精确重建。不精确的重建方法就称为有损，即某些信息丢失了，这种方法会导致比特率的极大降低。

第一种也是最简单的方法是差分脉冲编码调制（Differential Pulse Code Modulation，DPCM）。PCM 是指脉冲编码调制，即对信号进行采样并以二进制形式传输采样值。接收机可以将二进制加权码转换回模拟电压，这是原始电压的近似值，不管它是像素强度、声音样本还是其他模拟信息源。PCM 所需的电平数（对应于比特位数）取决于数据类型和重建的预期分辨率或预期质量，但通常都需要 8～16b。由于每个采样点都有这么多的比特需要传输，它有效地充当了每秒比特数的乘数。也就是说，每秒比特数等于每秒采样点数乘以每个采样点的比特位数。

然而，连续采样点之间通常都有显著的相似性，导致了一定程度的冗余。这意味着采样点的值具有一定的可预测性，即一个或多个先前的采样点可用于预测下一个采样点的值。在最简单的情况下，n 时刻采样值是 $n+1$ 时刻采样值的良好预测。

但是这如何有助于降低比特率呢？基本前提是，如果预测良好，预测误差（实际采样值减去预测值）将是一个很小的值（理想情况下为零），这是 DPCM 的差分部分。如果预测误差很小，就可以使用更少

的比特来传输预测误差，而不是实际采样值本身。然后，解码器本质上执行相反的操作，它形成自己的预测，解出误差，重建原始值。

1. 逐点预测

首先，从大体上来看，假设预测总体上是很好的，如图 5.21 所示。数学模型是将真实信号 $x(n)$ 当成预测信号 $\hat{x}(n)$ 加上一个误差 $e(n)$，即

$$x(n)=\hat{x}(n)+e(n) \tag{5.48}$$

在每个采样时刻 n，误差 $e(n)$ 可以是正的或负的。源信号 $x(n)$ 是图 5.21 中预测函数的输入，从输入的真实值中减去对应的预测值，以形成误差信号，即

$$e(n)=x(n)-\hat{x}(n) \tag{5.49}$$

式(5.48)和式(5.49)是等价的。接收机通过执行相反的操作解码该值，如图 5.22 所示。解码器的预测是基于输出端的采样值(即用于重建的采样值)。误差信号接收自信道，并被加到预测值中。在数学上，接收机将预测值加到误差值中，可表示如下。

$$x(n)=\hat{x}(n)+e(n) \tag{5.50}$$

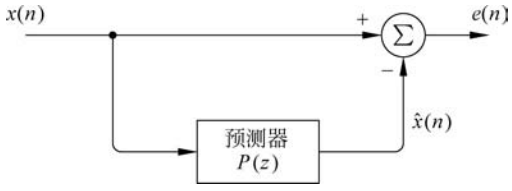


图 5.21　一种简化的差分编码器(不包括量化)

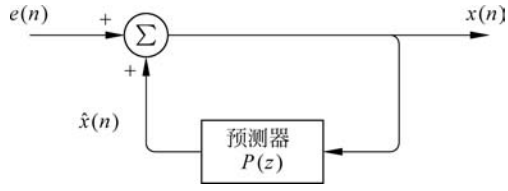


图 5.22　差分解码器

注：基于在解码器处形成的预测，加上通过信道接收的差值(预测误差)，从而形成输出。

然而，这种方法有一个小小的问题，它忽略了量化的影响。回想一下，量化将表示精度从理论上的无限个值降低到由有限比特位数决定的有限个值。因此，参考图 5.22，解码器预测的基础是重建值 $\hat{x}(n)$，而不是真值 $x(n)$。毕竟，接收机不知道编码器看到的真实信号。类似地，解码器中已知的误差是 $\hat{e}(n)$，而不是 $e(n)$。因此，必须要根据这些不精确的采样值来修改预测。

如果现在将计算的预测值表示为 $\tilde{x}(n)$，解码器的估计值表示为 $\hat{x}(n)$，那么基于量化误差 $\hat{e}(n)$，可得

$$\hat{x}(n)=\tilde{x}(n)+\hat{e}(n) \tag{5.51}$$

可以用一个简单的例子说明这一点。假设有采样值 20，33，42 和 35，如图 5.23 所示。每一步显示的误差值为＋13，＋9，－7。现在假设必须将误差量化为 4 的倍数，也就是说，只允许 0，±4 和±8 为误差值。从 20 开始，量化的预测误差为＋12，＋8，－8。如图 5.23 所示，得到的序列 20，32，40，32 与原始采样值不是精确匹配的。当然，由于量化，永远都不能精确匹配这些值。然而，还是希望二者尽可能接近，不希望在采样步骤中得到的值偏离了真实值。

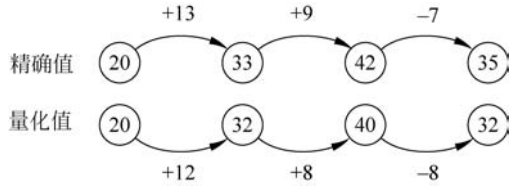


图 5.23　有量化和无量化的预测序列比较

对于少量的采样点，这似乎不是一个大问题，但是对于大量采样点，误差可能会累积。由于解码器只知道量化值，并且只能基于它自己计算的过去的输出值进行预测，因此预测应该仅基于这个前提。考

虑到这个事实，可以修改编码器，如图 5.24 所示。与图 5.21 仔细比较后可以发现，在环路中已经嵌入了量化操作，尽管也执行了相同的预测和误差计算，但它们的执行仅基于量化误差。

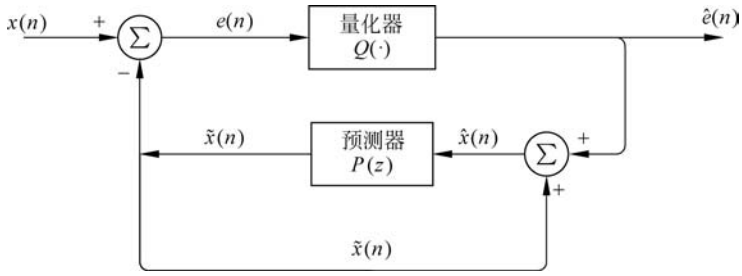


图 5.24　在预测环路中使用量化的 DPCM 编码器

注：最好的预测是基于解码器所知道的，而不是编码器所能看到的。

一个合理的问题是如何以最好的方式形成预测。当然，只能根据过去的采样值来进行。一个想法是形成以前采样点的平均值。可以通过引入加权线性和(Weighted Linear Sum)进一步推广它。忽略量化，对于一个 P 阶预测器，形成的加权和的预测为

$$\begin{aligned}\hat{x}(n) &= a_1 x(n-1) + a_2 x(n-2) + \cdots + a_P x(n-P) \\ &= \sum_{k=1}^{P} a_k x(n-k)\end{aligned} \tag{5.52}$$

其中，a_k 为第 k 阶的预测系数。这意味着采样值 $x(n)$ 通过一个或多个先前的采样值进行预测。最简单的情况，如前面的数值示例中所使用的，总是只使用前面一个采样值。在式(5.52)中，这将对应阶数 $P=1$ 的预测，加权系数 $a_1=1$。事实证明，对于图像数据，这是一个不错的预测。由此产生了几个问题：从什么意义上说，称得上是“最佳”或“最优”预测？在这种情况下，该如何计算最佳系数值 a_k？最佳预测阶数 P 又是多少？

对于前两个问题，预测器优化标准和预测器系数值是密切相关的，最好一起解决。预测器阶数不太容易确定，对于预测器阶数的增加，通常会有更好的估计结果，但这经常会伴随着综合效益的降低。高阶预测器有时只会产生稍微好一点的预测，这取决于待编码数据的性质。

为了解决预测器系数的求解问题，首先需要说明“最佳”的标准，这里是指一定长度内采样值有着最小的平均误差。误差可以是正的，也可以是负的。使用预测值和实际值之间的平方误差可以很好地工作，也便于使用数学推导来求解。再次考虑一个简单的一阶预测器，说明确定预测器系数 a_1 的最佳值的方法(即阶数 $P=1$)。预测值的形成可表示为

$$\hat{x}(n) = a_1 x(n-1) \tag{5.53}$$

此时预测误差为

$$\begin{aligned}e(n) &= x(n) - \hat{x}(n) \\ &= x(n) - a_1 x(n-1)\end{aligned} \tag{5.54}$$

为了消除正负误差的影响，可以取瞬时平方误差为

$$e^2(n) = [x(n) - a_1 x(n-1)]^2 \tag{5.55}$$

这给出了采样时刻的误差。但是信号会随着时间而改变，所以可以在一组 N 个采样点上对误差进行平均。平均平方误差表示为

$$\overline{e^2} = \frac{1}{N}\sum_n e^2(n)$$

$$=\frac{1}{N}\sum_n[x(n)-a_1x(n-1)]^2$$

$$=\frac{1}{N}\sum_n[x^2(n)-2x(n)a_1x(n-1)+a_1^2x^2(n-1)] \tag{5.56}$$

式(5.56)看起来很复杂,但是使用的是已知的采样值 $x(n)$,$x(n-1)$,…,而希望确定的是 a_1 的值。想要得到最小的平均误差,并且已经得到了关于 x 的多项式,所以对 a_1 求导数,可得

$$\frac{\mathrm{d}\overline{e^2}}{\mathrm{d}a_1}=\frac{1}{N}\sum_n[0-2x(n)x(n-1)+2a_1x^2(n-1)] \tag{5.57}$$

然后,与所有的最小化问题一样,将导数设为零,即

$$\frac{\mathrm{d}\overline{e^2}}{\mathrm{d}a_1}=0 \tag{5.58}$$

式(5.58)给出的预测值 a_1 实际上就是最佳预测值,将其记为 a_1^*,有

$$\frac{1}{N}\sum_n x(n)x(n-1)=a_1^*\ \frac{1}{N}\sum_n x^2(n-1) \tag{5.59}$$

解出 a_1^* 可得

$$a_1^*=\frac{1/N\sum_n x(n)x(n-1)}{1/N\sum_n x^2(n-1)} \tag{5.60}$$

理论上,需要一直对信号进行采样,才能在数学上实现这一点。然而,在实践中,可以不用这样更新预测器,而是使用有限大小的 N 个采样点的样本块。这是有意义的,因为信号实际上是随时间变化的,图像的扫描像素或采样的语音数据都是如此。统计特征在一小段时间内是近似不变的,但不会永远不变。因此,对大量采样值的统计特性描述可以使用自相关(Autocorrelations),定义如下。

$$R(0)=\frac{1}{N}\sum_n x^2(n) \tag{5.61}$$

$$R(1)=\frac{1}{N}\sum_n x(n)x(n-1) \tag{5.62}$$

那么解可以写为

$$a_1^*=\frac{R(1)}{R(0)} \tag{5.63}$$

对于典型的图像数据,相似性意味着 $R(1)$ 只比 $R(0)$ 小一点,并且发现系数为 0.8～0.95 的预测器可以得到误差很小的良好预测。

为了做出更好的预测,可以将加权平均扩展到更前面的采样点。对于一个二阶预测器,有

$$\hat{x}(n)=a_1x(n-1)+a_2x(n-2)$$

$$e(n)=x(n)-\hat{x}(n)$$

$$=x(n)-[a_1x(n-1)+a_2x(n-2)]$$

$$e^2(n)=\{x(n)-[a_1x(n-1)+a_2x(n-2)]\}^2 \tag{5.64}$$

同样,对于许多的采样值,平均平方误差为

$$\overline{e^2}=\frac{1}{N}\sum_n e^2(n)$$

$$=\frac{1}{N}\sum_n\{x(n)-[a_1x(n-1)+a_2x(n-2)]\}^2 \tag{5.65}$$

然而，这一次的优化问题涉及两个变量 a_1 和 a_2。可以使用偏导数对每一项分别进行优化，即

$$\frac{\partial\overline{e^2}}{\partial a_1}=\frac{1}{N}\sum_n\{2[x(n)-(a_1x(n-1)+a_2x(n-2))][-x(n-1)]\} \tag{5.66}$$

再将其设为零，如下。

$$\frac{\partial\overline{e^2}}{\partial a_1}=0$$

同样将最佳预测器用 * 进行标记，即 $a_1\to a_1^*$，$a_2\to a_2^*$，可得

$$\frac{1}{N}\sum_n\{2[x(n)-(a_1^*x(n-1)+a_2^*x(n-2))][-x(n-1)]\}=0$$

移项可得

$$\frac{1}{N}\sum_n x(n)x(n-1)=a_1^*\ \frac{1}{N}\sum_n x(n-1)x(n-1)+a_2^*\ \frac{1}{N}\sum_n x(n-1)x(n-2)$$

用自相关 $R(\cdot)$ 的定义表示为

$$R(1)=a_1^*R(0)+a_2^*R(1) \tag{5.67}$$

所以，现在有一个方程，但有两个未知数。然而，可以用同样的方法找到关于 a_2^* 的偏导数，最后可得

$$R(2)=a_1^*R(1)+a_2^*R(0) \tag{5.68}$$

现在有两个方程和两个未知数，写成矩阵的形式如下。

$$\begin{bmatrix}R(1)\\R(2)\end{bmatrix}=\begin{bmatrix}R(0)&R(1)\\R(1)&R(0)\end{bmatrix}\begin{pmatrix}a_1^*\\a_2^*\end{pmatrix} \tag{5.69}$$

或者简写为

$$\boldsymbol{r}=\boldsymbol{R}\boldsymbol{a}^* \tag{5.70}$$

其中，$\boldsymbol{r}$ 和 $\boldsymbol{R}$ 是从可用数据样本 $x(n)$ 中计算所得。然后，最佳预测系数矢量 $\boldsymbol{a}^*$ 可以通过求解矩阵方程(5.70)来确定。每个采样点的误差为

$$e(n)=x(n)-\overbrace{\sum_{k=1}^{P}a_kx(n-k)}^{\hat{x}(n)} \tag{5.71}$$

图 5.25 显示了这种方法的一个应用示例。从一个已知的测试例子开始，其数据由公式 $x(n)=1.71x(n-1)-0.81x(n-2)$ 生成。对于该系统的随机输入，使用 MATLAB 代码实现的最小均方预测方法得出的系数为 $a_1=1.62$，$a_2=-0.74$。当然，系数不会精确地等于预设值，因此，预测误差并不为零。然而，如图 5.25 所示，预测与输入有很好的一致性，因此误差通常都相当小。

下面的 MATLAB 代码展示了如何使用这种方法估计系统的参数。它从一个已知的系统开始，使用该系统的随机输入确定输出。目标是仅使用输出估计系统参数，也就是说，除了假定阶数已知(用来决定系数数量)之外，不需要对系统本身有任何的了解。

系统的输入是随机噪声，预测器系数的计算依据上文所述的自相关和矩阵方程求解。为了使用 filter()函数预测采样值，有必要将预测系数表示为扩充矢量的形式 am=[1；-a]，这是因为 filter()需要的数据形式为

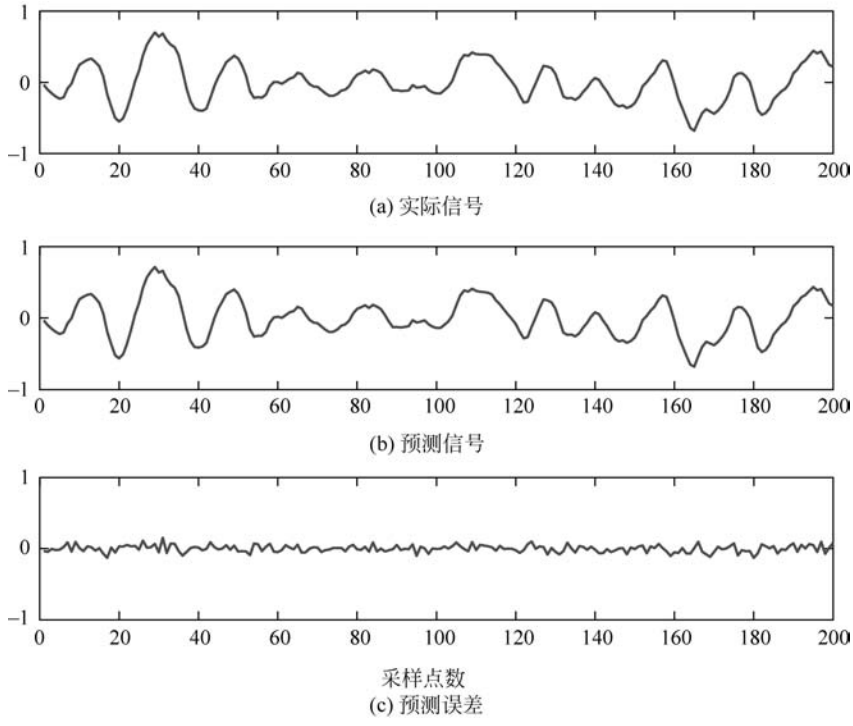


图 5.25　线性预测中的实际信号、预测信号以及预测误差曲线

注：计算较大样本块上的自相关一般来说会给出更好的预测结果。

$$a_1 y(n) = -a_2 y(n-1) - a_3 y(n-2) - \cdots + b_1 x(n) \tag{5.72}$$

输入为 x，输出为 y，而此处的问题可以表示为

$$x(n) = a_1 x(n-1) + a_2 x(n-2) + \cdots + e(n) \tag{5.73}$$

输入为误差 $e(n)$，输出 $x(n)$基于先前的采样值 $x(n-1)$，$x(n-2)$，…。预测值 $\hat{x}(n)$被表示为 xhat。

```
N = 200;

% 半径为 r 且角度为 omega 的极点
r = 0.9;
omega = pi/10;
p = r * exp(j * omega);

a = poly([p conj(p)]);
roots(a)

% 系统输入
e = 0.05 * randn(N, 1);

% 对输入的响应
x = filter(1, a, e);

% 计算自相关
R0 = sum(x . * x)/N;
R1 = sum(x(1:N-1) . * x(2:N))/(N);
```

```
R2 = sum(x(1:N-2) .* x(3:N))/(N);

% 自相关矩阵/矢量
R = [R0 R1 ; R1 R0 ];
r = [R1 ; R2];

% 最优预测解
a = inv(R) * r;

% 滤波器的最优预测参数
am = [1 ; -a];

% 估算的输出
xhat = filter(1, am, e);
```

2. 自适应预测

上面描述的方法在每个数据块编码时更新预测器。为了在接收机上重建数据,有必要同时知道误差样本和预测器参数。所以,预测器参数需要单独发送到解码器,或者解码器必须使用前一个块(解码器已有)计算预测器参数,并将其应用到当前块。前者的缺点是需要额外的比特编码预测器参数,而后者的缺点是处理的是过时信息。

为每个采样点实时更新预测器参数,而不是在缓冲完含有 N 个采样点的块之后,结果会怎么样?这称为自适应预测(Adaptive Prediction),与前面讨论的分块预测(Blockwise Prediction)是不同的。这里仍将预测器定义如下。

$$\begin{aligned} e(n) &= x(n) - \hat{x}(n) \\ &= x(n) - \sum_{k=1}^{P} h_k x(n-k) \end{aligned} \tag{5.74}$$

这与分块预测器基本相同,但是使用了 h_k 作为系数以避免混淆。如果允许预测器随着每个采样点变化,并将系数记为矢量 $\boldsymbol{h}$,可得

$$e(n) = x(n) - \boldsymbol{h}^{\mathrm{T}}(n)\boldsymbol{x}(n-1) \tag{5.75}$$

预测系数矢量随时间变化,记为

$$\boldsymbol{h}(n) = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_P \end{bmatrix} \tag{5.76}$$

采样值形成的矢量从最后一个开始,记为

$$\boldsymbol{x}(n-1) = \begin{bmatrix} x(n-1) \\ x(n-2) \\ \vdots \\ x(n-P) \end{bmatrix} \tag{5.77}$$

同样,这是一个最小化问题。然而,这一次不是对一个块上的 N 个采样点求平均值,而是取每个采样值并更新预测器。预测器在 h_1 方向上梯度的估计值是计算偏导数,即

$$\begin{aligned}\hat{\nabla}_{h_1}e^2(n)&=\frac{\partial}{\partial h_1}[x(n)-\boldsymbol{h}^{\mathrm{T}}(n)\boldsymbol{x}(n-1)]^2\\&=\frac{\partial}{\partial h_1}\{x(n)-[h_1x(n-1)+h_2x(n-2)+\cdots]\}^2\\&=2\{x(n)-[h_1x(n-1)+h_2x(n-2)+\cdots]\}\times\\&\quad\frac{\partial}{\partial h_1}\{x(n)-[h_1x(n-1)+h_2x(n-2)+\cdots]\}\qquad(5.78)\\&=2e(n)[-x(n-1)]\\&=-2e(n)x(n-1)\qquad(5.79)\end{aligned}$$

最后一行很容易计算。类似的推导给出了 h_2 方向上梯度的估计值为

$$\hat{\nabla}_{h_2}e^2(n)=-2e(n)x(n-2)\qquad(5.80)$$

现在因为知道了梯度，就知道了误差的走向。对于每个新的采样值，目标是用一个与 $e^2(n)$ 的负梯度成比例的量更新预测器系数 $\boldsymbol{h}$，从而找到最小误差。

这是一个关键点，图 5.26 中的曲线代表了每设置一个可能的 h_1 参数时得到的平方误差，最小误差出现在曲线的最低点处，对应着 $h_1=h_1^*$。假设步骤 n 的更新在点 $h_1(n)$ 处执行。曲线在该点的梯度或斜率是正的，但是 h_1 的值必须要减小，以便更接近最小误差处的最佳点 h_1^*。类似地，在 h_1^* 的左边，曲线的梯度是负的，但是最佳值是较高的 h 值(向右移动)。这就是为什么要在负梯度方向上执行更新。

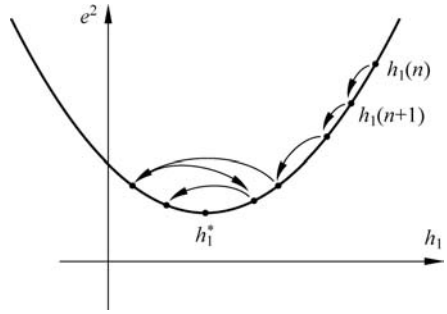


图 5.26　自适应线性预测(说明了预测器系数是如何收敛的)

从图 5.26 所示的步骤中可以看出，步长也很关键。随着梯度算法的迭代，有更好的参数估计出现，对于点 $h_1(n)$，$h_1(n+1)$，…，平方误差也会减小。步长越大，误差减小得越快。然而，有可能越过图 5.26 中所示的最小点，导致算法“搜寻”在最小误差处 h_1^* 的一侧。在这种情况下，显然步长越小越好，但这也意味着初始的收敛会更慢。

前面的讨论是以一个系数 h_1 为框架的，但是实际上有几个预测器系数 $h_1,h_2,h_3,\cdots$，同样的关于收敛和步长的论述都分别适用于它们。对于所有预测器系数，每个采样点处预测器系数的增量调整如下所示。

$$\boldsymbol{h}(n+1)=\boldsymbol{h}(n)-\mu\hat{\nabla}e^2(n)\qquad(5.81)$$

其中，μ 为自适应速率参数。使用前面计算的偏导数，式(5.81)写为

$$\boldsymbol{h}(n+1)=\boldsymbol{h}(n)+2\mu e(n)\boldsymbol{x}(n-1)\qquad(5.82)$$

展开可得

$$\begin{bmatrix}h_1(n+1)\\h_2(n+1)\\\vdots\\h_P(n+1)\end{bmatrix}=\begin{bmatrix}h_1(n)\\h_2(n)\\\vdots\\h_P(n)\end{bmatrix}+2\mu e(n)\begin{bmatrix}x(n-1)\\x(n-2)\\\vdots\\x(n-P)\end{bmatrix}\qquad(5.83)$$

这个更新步骤发生在每个采样点上，而不是像分块预测那样发生在每个样本块上。图 5.27 用一个数值示例说明了收敛过程。这里是一个含有两个系数 h_1 和 h_2 的二阶预测器。很明显，两个系数的最

终值都趋于收敛，但是也存在一些与预测器系数估计相关联的噪声。

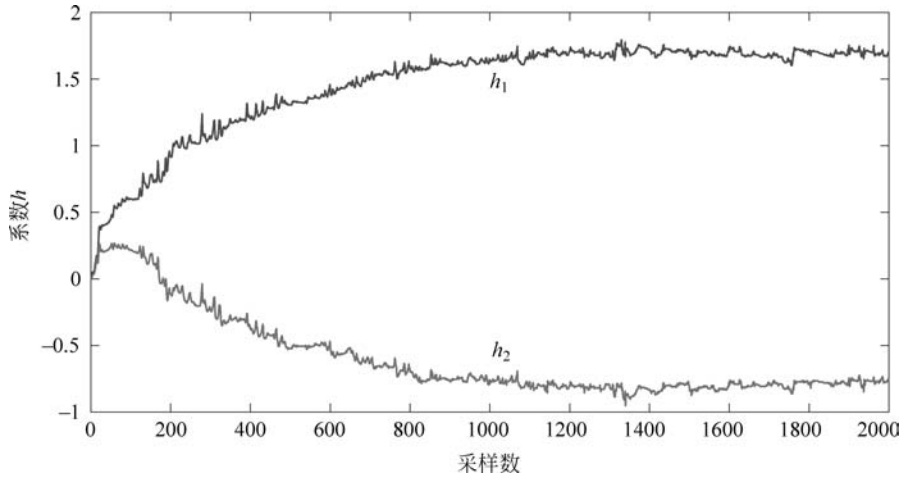


图 5.27　自适应线性预测系数的收敛曲线($\mu=0.001$)

注：图中显示给定步长参数 μ 时，预测系数 $\boldsymbol{h}=[h_1 \quad h_2]^{\mathrm{T}}$ 的收敛性。

5.7　图像编码

数字图像占用了大量的存储空间，因此传输可能需要很长时间。考虑尺寸为 1000×1000 像素(宽×高)的图像，它的分辨率不算特别高。当三原色(红，绿，蓝)中的每一种都用 1B 表示，则需要 3×1000×1000B[①]。如果两个尺寸都加倍，则需要 2×2=4 倍的空间。这个问题在视频上更加尖锐。如果这样的静止图像序列用作视频源，每秒重放 50 帧，那么每秒大概需要这个数字的 50 倍。一小时的视频序列需要 $3\times50\times60\times60\times1000^2\approx540$GB。这是相当大的存储空间。实时传输这样的视频将需要 1200Mbps 的传输速率。

因此，实际中几乎总是需要某种形式的数据压缩。在存储时，为了降低成本，需要压缩数据。在带宽受限信道中，压缩对于实现视频的实时传输是至关重要的。

幸运的是，视觉信息中有大量的冗余。利用这一事实，在许多情况下可以减少所需的数据量。这是因为图像、视频内容的感知依赖于显示设备和人类视觉系统(Human Visual System，HVS)。当然会有一个折中：通过丢弃一些信息来有效地减少信息量，但这将降低图像质量。然后，压缩算法的选择可以根据质量损失是否可感知，或者质量下降是否允许来决定。因此，对于电影内容，可能希望质量下降不可感知；而对于视频电话会议，一些质量的损失是可接受的。在某些情况下，如医学成像，质量下降是不可接受的，在这种情况下，必须回到前面讨论的无损算法，只是它可实现的压缩必然要低得多。

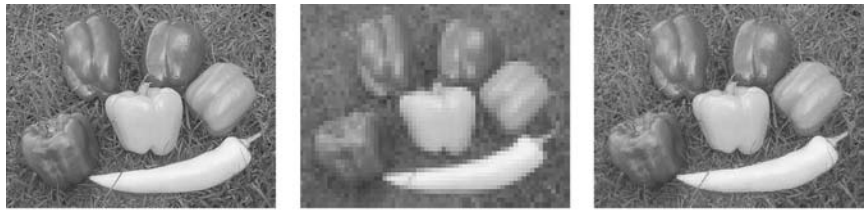
许多压缩算法在降低比特率方面非常有效，同时保持了可接受的质量，然而，它们的计算复杂度以及执行压缩所需的处理器速度可能非常重要。一般来说，计算速度越快，成本更高。此外，更快的速度必然意味着更高的功耗，这是移动设备一个关键的考虑因素。最后，复杂度也会影响内存需求，这也不是无关紧要的。通常，图像被分解成 8×8 像素的较小块(也称为子块)。原因有两个。首先，图像中像

① 在通常的用法中，megabyte(MB)是指 1024^2，但是有些人认为 megabyte 应该是 1000^2，应该用 mebibyte(MiB)指代 1024^2。

素的相似性通常是小范围延伸，因此选择有可能相似的小范围的像素是有意义的。其次，这种适度的块大小允许设计者制定计算单元来缓冲这些块并独立地处理它们。

5.7.1 块截断算法

减少数据量的最简单方法是对邻域内的像素进行平均，只传输平均电平。相邻像素之间存在大量的相关性，因此存在可以利用的大量冗余。对于一个边长为 N 个像素的正方形块，所得的 $M=N^2$ 个像素可以用一个单一值(即平均值)来表示。然而，这会导致重建图像中出现块效应(Blockiness)。这可以通过比较图 5.28(a)和图 5.28(b)看出来。



(a) 原始图像　(b) 平均值压缩图像(16×16)　(c) BTC图像(16×16)

图 5.28　原始图像和压缩图像的比较示例

注：图(a)是原始图像，图(b)是仅使用块平均值的图像，图(c)是 16×16 子块的 BTC 编码图像。请注意，仅使用块平均值的图像中存在明显的块效应，尽管这种情况下每个像素的平均比特位数很小。通过更好的算法和传输更多的参数，可以获得更好的图像质量。

块截断编码(Block Truncation Coding，BTC)算法是仅使用块平均值的思想的简单改进。BTC 最初是由 Delp 和 Mitchell(1979)提出的，它在重建块时不仅保留了平均值，还保留了平均值的方差[①]。本质上，平均值包含感知上重要的平均亮度电平，而方差表示平均值周围的平均亮度变化。目的是确定能够在原始和重建子块中实现相同均值和方差的最小参数集。虽然 BTC 的压缩率不够高，但考虑其原理是很有益处的，有助于理解性能更好的 DCT 算法(详见 5.7.2 节)。

在 BTC 中，重建块仅用两个值表示像素：如果当前源像素小于平均值 $\bar{x}$，则为 a；如果源像素大于平均值，则为 b。因此，每个像素需要 1b 来选择是 a 或 b，还需要 a 和 b 本身的值。必须证明，有可能计算出满足均值和方差特性的 a 和 b 值。

图像被分解成像素块，每个像素块用矩阵 $\boldsymbol{X}$ 表示，每边有 N 个像素，因此总共有 $M=N^2$ 个像素。将这些像素值视为一个简单的列表，平均值 $\bar{x}$、均方值 $\overline{x^2}$ 和方差 σ^2 计算如下。

$$\bar{x}=\frac{1}{M}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}x(i,j) \tag{5.84}$$

$$\overline{x^2}=\frac{1}{M}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}x^2(i,j) \tag{5.85}$$

$$\sigma^2=\frac{1}{M}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}[x(i,j)-\bar{x}]^2 \tag{5.86}$$

记总像素数为 M，大于平均值的像素数为 Q，可以写出原始块和重建块的平均值和均方值。为了保持重建块中的平均值和均方值，可以使用以下方程匹配平均值和均方值。

① 严格来说，这里指的是样本方差，在 MATLAB 中会使用 var(data,1)。

$$M\bar{x} = (M-Q)a + Qb \tag{5.87}$$

$$M\overline{x^2} = (M-Q)a^2 + Qb^2 \tag{5.88}$$

等式左侧代表原始图像，右侧由重建图像计算得出。因此，实际上有两个等式，其中含有两个未知参数 a 和 b。所有其他值(M,Q,$\bar{x}$,$\overline{x^2}$)都可以从源块中计算出来。方程的解有点复杂，因为含有非线性项，如 a^2 和 b^2。求解方程组式(5.87)和式(5.88)，得到较小的重建像素值为

$$a = \bar{x} - \sigma\sqrt{\frac{Q}{M-Q}} \tag{5.89}$$

较大的重建像素值为

$$b = \bar{x} + \sigma\sqrt{\frac{M-Q}{Q}} \tag{5.90}$$

以下示例说明了计算过程。像素值取自真实的灰度图像。使用尺寸为 4×4 的小块来说明，尽管实际中可以使用更大的尺寸获得更低的比特率。原始像素(8 位整数)块为

$$\boldsymbol{X} = \begin{bmatrix} 62 & 37 & 36 & 46 \\ 74 & 49 & 47 & 53 \\ 90 & 71 & 53 & 56 \\ 101 & 81 & 58 & 59 \end{bmatrix} \tag{5.91}$$

其中，$M=16$ 个像素的平均值为 $\bar{x}=60.81$，方差为 $\sigma^2=315.15$，均方值为 $\overline{x^2}=4013.31$。大于平均值的像素数 Q 为 6，因此位掩码为

$$\boldsymbol{B} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \tag{5.92}$$

其中，1 代表平均值以上的值，0 代表平均值以下的值。

根据这些数值，可以应用式(5.89)和式(5.90)计算出 $a=47.06$，$b=83.73$。然后，重建平均值和均方值的新子块为

$$\hat{\boldsymbol{X}} = \begin{bmatrix} 83.73 & 47.06 & 47.06 & 47.06 \\ 83.73 & 47.06 & 47.06 & 47.06 \\ 83.73 & 83.73 & 47.06 & 47.06 \\ 83.73 & 83.73 & 47.06 & 47.06 \end{bmatrix} \tag{5.93}$$

该子块相关的参数为 $\bar{x}=60.81$，$\sigma^2=315.15$，$\overline{x^2}=4013.31$。正如所期望的，这些值与原始块完全相同。近似后，像素值变为

$$\hat{\boldsymbol{Y}} = \begin{bmatrix} 84 & 47 & 47 & 47 \\ 84 & 47 & 47 & 47 \\ 84 & 84 & 47 & 47 \\ 84 & 84 & 47 & 47 \end{bmatrix} \tag{5.94}$$

重建块的平均值为 $\bar{x}=60.88$，方差为 $\sigma^2=320.86$，均方值为 $\overline{x^2}=4026.63$。由于最后一步中使用了像素近似，这些值接近但不等于理论值。

对于16×16的较大块尺寸，使用BTC的结果如图5.28所示。对于每个16×16的子块，仅使用块平均值时只需要一个参数，因此实现了相当大的压缩（实际上比例为256∶1）。然而，该方法在重建图像时，子块非常明显而且看起来不舒服。使用BTC算法时，编码整个子块，也只需要两个参数再加上位掩码。因此，位掩码中每个像素只有一位比特，再加上传输 a 和 b 所需的比特。结果表明，重建图像明显优于仅使用平均值的子块重建，并且在许多方面都与原始图像无法区分，它的比特率大约为1.06bpp（bpp即b/pixel，比特每像素）。

5.7.2 离散余弦变换

5.7.1节中介绍的BTC方法似乎运行良好，比特率低，质量好。然而，它需要相对较大的子块来实现低速率，导致图像整体的感知质量有更大的损失。此外，每个子块也都有一些保真度损失。

一种更先进的算法——离散余弦变换（Discrete Cosine Transform，DCT）在图像和视频压缩中都有非常广泛的应用。DCT的定义由Ahmed等（1974）给出，随后发现它在用于压缩图像时效果良好。DCT是当今使用的许多低速率算法的基础，如用于HDTV的MPEG和用于静止图像的JPEG。DCT的基本思想是尽可能多地去除图像块中的统计冗余，从这个角度来看，它与前面讨论的BTC方法没有实质性的不同。然而，所采取的方法却大不相同。此外，DCT能够实现分数比特率（小于1bpp），而BTC不能，因为后者需要传输位掩码矩阵。

要注意的是，DCT本身不会产生任何压缩，但是DCT的结果能够用更少的参数表示图像子块。

这里从结果开始分析，而不从数学理论开始。首先在图像中形成子块，一般都是8×8像素，再分别处理这些子块。在下文，假设像素值只表示亮度，即图像是一幅灰度图。通过颜色分量的分开处理，该方法可以扩展到彩色图像，这将在后面讨论。然而，现在要注意的是，颜色信息中有更多的冗余，因为HVS对亮度变化比对颜色精确度更为敏感。

第一步是进行8×8像素块的转换，这会得到相同数量的系数。如果考虑一幅或多幅图像上的大量子块并生成直方图，会得到如图5.29所示的结果。图5.29仅显示了9个系数，是整个8×8系数集的左上部分。索引为(0,0)的左上角系数通常称为DC系数，因为它实际上与整个块的平均亮度成比例。其他系数，如索引(0,1)，(1,0)，(1,1)等，明显呈尖峰分布。正如图5.29所示，这种类型的分布有利于更好地编码，因为有些值比其他值更有可能出现。出现可能性大的值可以用较短的码字编码（如霍夫曼编码），或者用相邻块之间的差分编码（如DPCM）来更有效地编码。尖峰分布还能允许更有效的量化，对于量化器中不太可能出现的值，可以采用更大的步长；而对于更可能出现的值，可以采用更小的步长。

这意味着，实际上，不需要为8×8子块中的每个像素都使用典型的8b，而可以为变换后的系数分配更少的比特。更有利的是，一些系数几乎确定为零或非常接近零，这意味着甚至根本不需要对其进行编码。

将DCT表示过程描绘成一组基图像（Basis Images）将有助于理解。如图5.30所示，基图像包含了8×8个较小子块，每个子块由8×8个像素组成。将每一个子块都称为一个“区块”（Tile），并想象8×8图像子块都由区块组合而成。事实上，真正用的是区块的加权组合。也就是说，平均值由左上角或(0,0)区块来表示，它可以编码为一个系数乘以区块上的各个像素值（在这种情况下，所有像素值都相等）。接下来，考虑区块(0,1)，它是一个竖条纹，左边的亮度较高，右边的亮度较低（较暗）。特定的图像子块可能碰巧具有类似的底层图案，因此可以用该区块乘以某个特定量（加权系数值）来近似真实图像子块。

类似地，横条纹区块，即索引(1,0)，(2,0)及以下的，也是可加权的。中间区块也是如此，其中存在各种棋盘图案组合。总的来说，图像子块由这些区块的组合来重建，其中每个区块的加权值可以不同。

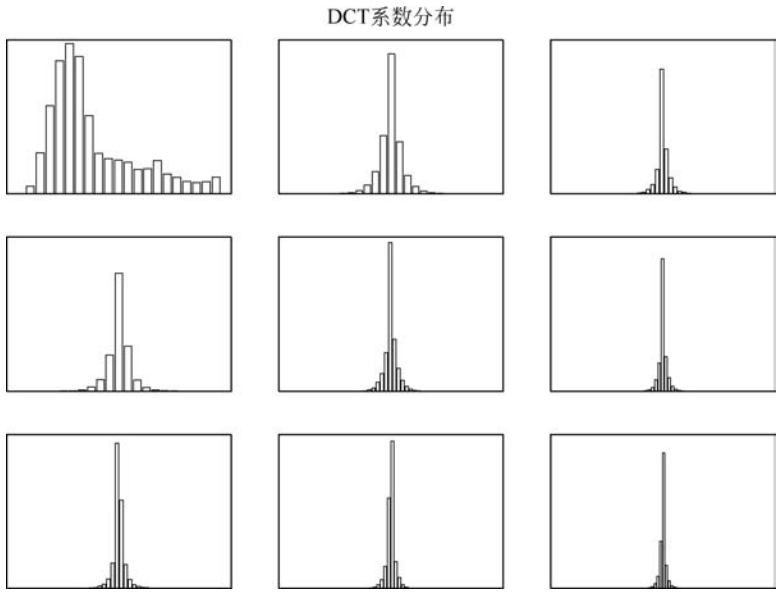


图 5.29　子块左上角的 DCT 系数直方图

注：对于 8×8 子块，将有相等数量的系数直方图，这里只显示左上角的 3×3 的系数直方图。

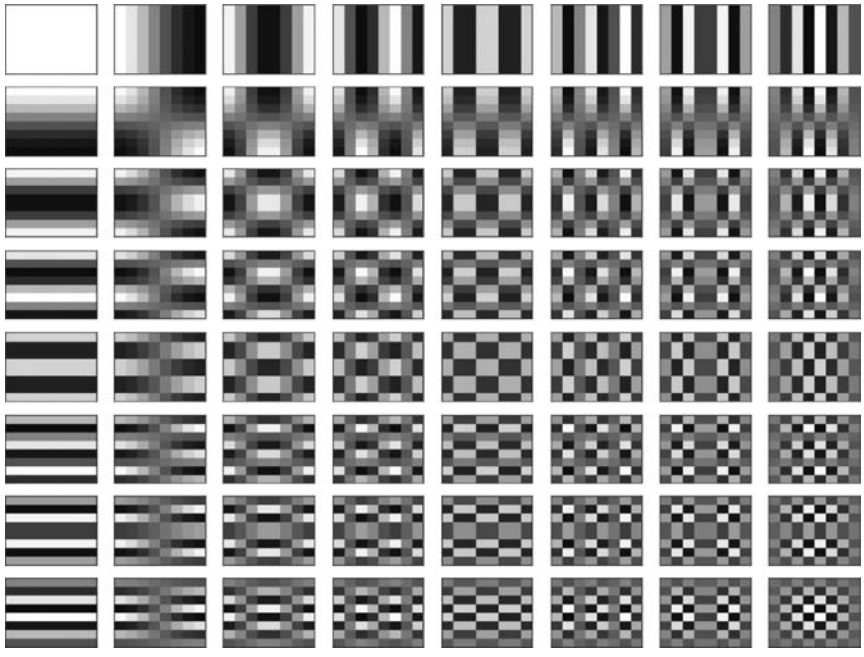


图 5.30　8×8 变换的 DCT 基图像

注：每个基图像是 8×8 的像素块，总共有 8×8 个基图像。

这就是 HVS 和显示设备分辨率发挥重要作用的地方。左上侧的区块表示了更“粗糙”的图像细节，一般是必要的。而沿对角线方向右下角的区块就可能不太需要了。如果准备牺牲其中一些更细节

的区块，那么就没有必要对相应的系数进行编码（编码值可设为零）。

因此，需要一种生成区块图案的方法，以及计算每个区块的权重以重构子块的方法。显然，图案是有规律的，事实证明，如果希望将最大能量用尽可能少的系数表示，余弦函数是一个非常好的选择。

这种变换对于图像是二维的，不过从一维开始讲解更容易。给定长度为 N 个像素的输入块（矢量），变换矢量 $\boldsymbol{y}$ 可用如下公式确定。

$$\boldsymbol{y}(k)=\sqrt{\frac{2}{N}}c_k\sum_{n=0}^{N-1}\boldsymbol{x}(n)\cos\left[\frac{(2n+1)k\pi}{2N}\right] \tag{5.95}$$

对于索引 $k=0,1,\cdots,N-1$，系数为

$$c_k=\begin{cases}\dfrac{1}{\sqrt{2}}, & k=0\\ 1, & k\neq 0\end{cases} \tag{5.96}$$

如果在编码器处执行变换，则有必要在解码器处对其进行反变换，以便获得图像像素。这是通过逆DCT（IDCT）完成的，如下所示。

$$\boldsymbol{x}(n)=\sqrt{\frac{2}{N}}\sum_{n=0}^{N-1}c_k\boldsymbol{y}(k)\cos\left[\frac{(2n+1)k\pi}{2N}\right] \tag{5.97}$$

其中，$n=0,1,\cdots,N-1$。变换和逆变换看起来非常相似，但是请注意缩放值 c_k 相对于求和的位置。在DCT中，它在求和符号之外，因为求和是在 n 上计算的；而在IDCT中，它必须在求和符号之内，因为求和是在 k 上计算的。当然，DCT是完全可逆的。也就是说，对矢量的DCT再进行IDCT运算，能够得到原始矢量。因此，理论上，给定所有系数时，原始图像可以被准确无误地还原。然而，为了实现更大程度的压缩，还是会放弃一些系数（通过将它们设置为零）或降低其他系数的量化精度。

前面讨论的区块实际上是基图像，即重建图像的基本块。在一维情况下，这些基矢量（Basis Vectors）$\boldsymbol{a}_k$ 的计算如下。

$$\boldsymbol{a}_k=\sqrt{\frac{2}{N}}c_k\cos\left[\frac{(2n+1)k\pi}{2N}\right] \tag{5.98}$$

每个矢量 $\boldsymbol{a}_k$（索引 k 的范围为 $0,1,\cdots,N-1$）含有分量 $n=0,1,\cdots,N-1$。也可以将DCT（和IDCT）写成矩阵的乘法。毕竟，在一维的例子中，可以取一个 $N\times 1$ 的列矢量，并形成一个维数同样是 $N\times 1$ 的输出列矢量。根据矩阵论的理论，这可以通过矩阵乘以矢量来实现，即需要一个 $N\times N$ 的矩阵与矢量相乘，得到最终的输出矢量。也就是说，DCT可以视为如下的矩阵乘法。

$$\boldsymbol{y}=\boldsymbol{A}\boldsymbol{x} \tag{5.99}$$

例如，4×4 的DCT核矩阵为

$$\boldsymbol{A}=\begin{bmatrix}a & a & a & a\\ b & c & -c & -b\\ a & -a & -a & a\\ c & -b & b & -c\end{bmatrix} \tag{5.100}$$

其中

$$a=\frac{1}{2}$$

$$b=\frac{1}{\sqrt{2}}\cos\frac{\pi}{8}$$

$$c=\frac{1}{\sqrt{2}}\cos\frac{3\pi}{8}$$

为了简化矩阵-矢量方法，只考虑一个二像素的输入（它将相应地具有两个值的输出）。使用 DCT 公式，可以用矩阵形式重写计算过程，于是 2×2 的 DCT 转换矩阵可以写为

$$\boldsymbol{A}_{(2\times2)}=\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \tag{5.101}$$

如果仔细检查方程，就会发现基矢量构成了变换矩阵的行。类似地，基矢量也构成了逆变换矩阵的列。上面的 2×2 结果可以直观地解释。第一个输出系数是通过将两个输入像素乘以 1 而形成的，因此它是两者的平均值；第二个输出系数是通过将两个像素分别乘以＋1 和－1 而形成的，因此是差分运算。

因此，可以将正向变换 $\boldsymbol{y}=\boldsymbol{A}\boldsymbol{x}$ 写成行矢量表示，即

$$\begin{bmatrix}|\\ \boldsymbol{y}\\ |\end{bmatrix}=\begin{bmatrix}— & \boldsymbol{a}_0 & —\\ — & \boldsymbol{a}_1 & —\\ & \vdots & \end{bmatrix}\begin{bmatrix}|\\ \boldsymbol{x}\\ |\end{bmatrix} \tag{5.102}$$

这样就将基矢量作为变换矩阵的行矢量。$\boldsymbol{y}$ 中的每个输出系数都是矢量的点积，形式如下。

$$y_0=\begin{bmatrix}|\\ \boldsymbol{a}_0\\ |\end{bmatrix}\cdot\begin{bmatrix}|\\ \boldsymbol{x}\\ |\end{bmatrix} \tag{5.103}$$

另外，$y_1=\boldsymbol{a}_1\cdot\boldsymbol{x}$，以此类推，实际上是将输入矢量和基矢量相乘再相加，即加权（Weighting）。

根据类似的思想，逆变换也可表示为

$$\boldsymbol{x}=\boldsymbol{B}\boldsymbol{y} \tag{5.104}$$

同样，可以用矢量来改写矩阵，但这次是列矢量，可表示如下。

$$\begin{bmatrix}|\\ \boldsymbol{x}\\ |\end{bmatrix}=\begin{bmatrix}| & | & |\\ \boldsymbol{b}_0 & \boldsymbol{b}_1 & \cdots\\ | & | & |\end{bmatrix}\begin{bmatrix}y_0\\ y_1\\ \vdots\end{bmatrix} \tag{5.105}$$

其中，基矢量是列矢量。输出的写法可能会有所不同，即

$$\begin{bmatrix}|\\ \boldsymbol{x}\\ |\end{bmatrix}=y_0\begin{bmatrix}|\\ \boldsymbol{b}_0\\ |\end{bmatrix}+y_1\begin{bmatrix}|\\ \boldsymbol{b}_1\\ |\end{bmatrix}+\cdots \tag{5.106}$$

因此，可以看出，输出 $\boldsymbol{x}$ 是标量与矢量乘积的加权和。这也解释了正变换是确定基矢量系数的过程，而逆变换是将这些基矢量以正确比例系数相加的过程。

一维的情况也可以推广到二维。现在，需要的不是基矢量，而是基本块（或矩阵），这些就是前面讨论的区块。完整的二维 DCT 如下所示。

$$\begin{aligned}\boldsymbol{Y}(k,l)=&\frac{2}{N}c_k c_l\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}\boldsymbol{X}(m,n)\cdot\\ &\cos\left[\frac{(2m+1)k\pi}{2N}\right]\cos\left[\frac{(2n+1)l\pi}{2N}\right]\end{aligned} \tag{5.107}$$

其中，$k,l=0,1,\cdots,N-1$，相应的二维 IDCT 为

$$\boldsymbol{X}(m,n)=\frac{2}{N}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1}c_k c_l \boldsymbol{Y}(k,l)\cdot \cos\left[\frac{(2m+1)k\pi}{2N}\right]\cos\left[\frac{(2n+1)l\pi}{2N}\right] \tag{5.108}$$

其中，$m,n=0,1,\cdots,N-1$。

应该注意的是，运算所需的计算量相当大。二维 DCT 方程的直接实现需要对输入块中的所有 $N\times N$ 个像素求和，以获得一个输出系数。而每个子块有 $N\times N$ 个输出系数。当然，形成图像还需要大量的子块。因此，产生几种快速 DCT 算法就不足为奇了，其原理类似于快速傅里叶变换(FFT)。事实上，许多快速 DCT 算法都采用了 FFT(Narashima，et al.，1978)。

5.7.3　四叉树分解

图像的固定块有一些优点：它能够使比特率恒定，并且简化了编码图像所需的处理。固定比特率对于许多类型的信道都很重要，但是这并不意味着块编码就不能是可变比特率。虽然，在图像上叠加固定块可能有点不自然。如果将更多的比特分配给图像中“活跃”(Active)或细节丰富的区域，而将更少的比特分配给不太活跃的区域，将会更好。“活跃”的定义很简单，就是平均值的方差。由此引出了可变块大小编码，其中一种就是四叉树分解(Quadtree Decomposition)。

四叉树分解可以从图 5.31 中最左边的块开始，该块被细分为 4 个大小相等的块。如果这些块的活跃性度量基本相同，就不需要继续细分了。另外，如果这些块有一定程度的差异，还可以再继续细分。分割或不分割的行为可以用一位比特来表达(0 代表分割，1 代表不分割)。

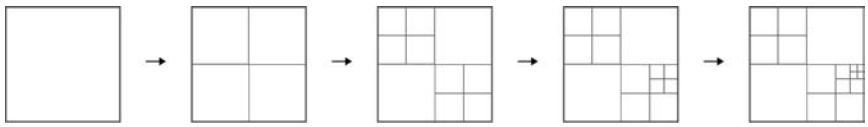


图 5.31　四叉树分解

注：从左到右的递归分解显示了一些子块是如何进一步细分的，而另一些子块则没有细分。

重要的是，该过程可以在得到的 4 个子块上进行重复。因此，在图中从左向右进行细分时，左上角和右下角的块开始被细分，但是右上角和左下角没有。这意味着未被细分的块可以用常数值(灰度或亮度，需要的话还有颜色)表示。继续向右，可以看到其中一个子块被进一步细分，以此类推。这样，该算法为图像中更活跃的区域选择出越来越小的块。这种算法适于递归实现，在每个阶段，输入块被分成 4 个，或者保持不变。如果一个块被分成 4 个较小的块，那么在这些较小的块上可以重复相同的过程。

图 5.32 显示了一个用这种方式分解的图像示例，其中块边界是可见的(实际上，块边界只是假设的，而不会真地编码在图像之中)。选择分解块的阈值会影响块的最终数量，阈值越大意味着细分的块会越少。

5.7.4　颜色表示

迄今为止，本节考虑的图像仅包含亮度。考虑颜色会引入一系列新的问题，大量的研究和标准化工作已经投入颜色的有效表示中。本质上，由主动(发光)显示器表示的颜色由 3 个主要成分组成：红色、绿色和蓝色(RGB)。就人们的感知而言，大部分“有价值的”信息包含在亮度中，而亮度实质上是 R、G、B 的和。在某种意义上，颜色信息可以叠加给仅含亮度的图像以添加颜色。

采样和显示设备可以被认为在 RGB 域中工作。颜色由 R、G 和 B 信号的相对权重组成。例如，红

色加绿色会产生黄色，但红色加50%绿色会产生橙色。赋予颜色的实际权重是相当主观的，当涉及颜色感知时，我们的视觉系统有些非线性。也就是说，一种颜色的特定刺激不会和另一种颜色表现得一样明显，有多种进化理论可以解释这一点。此外，因为图像捕捉和显示设备必须基于特定设备（特别是半导体）的物理性质，所以真实颜色的表示再次受限。

(a) 原始图像　　(b) 四叉树分解图像

图 5.32　灰度图像的四叉树分解示例

注：图中的块边界是可见的，以显示可变的块大小以及它们是如何对应于图像的局部活跃性。

由于HVS对亮度比对颜色更加敏感，可以减少编码颜色所需的数据量。传统上，在模拟电视系统中是使用所谓的色差信号来完成的，并且它一直持续到数字颜色采样的出现。亮度被表示为信号Y，还需要其他信号作为补充来传递颜色。因为开始有3个颜色信号（R，G，B），所以除了Y应该还有另外两个信号。这些就是色差，通常使用的是蓝色差（Cb）和红色差（Cr）信号。红色差Cr与（R－Y）成比例，蓝色差Cb与（B－Y）成比例。

最常用的权重是ITU-R BT601定义的（通常是使用CCIR601的早期定义）。以矩阵形式，可以将其写为（Acharya，et al.，2005；ITU-R，n.d.）

$$\begin{bmatrix} Y \\ Cb \\ Cg \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & 0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{5.109}$$

可以用全分辨率表示亮度，但是颜色的分辨率会降低。对于每4个像素，可用所谓的4∶4∶4来表示，其中包括每个像素的Y、Cr和Cb，如图5.33(a)所示。4∶2∶2表示降低了水平方向的颜色分辨率，如图5.33(b)所示。图5.33(c)还显示了另一种表示方法，即所谓的4∶2∶0，它将水平和垂直两个方向的颜色（或者更准确地说，色差）进行抽样。

(a) 全分辨率　(b) 水平抽样　(c) 水平和垂直抽样

图 5.33　4×4像素块的色度抽样示例

注：每个像素从RGB开始，然后转换为亮度Y加上色差CrCb。颜色可以如图(b)和图(c)所示进行抽样，对图像本身几乎没有可察觉的影响。

5.8 语音和音频编码

编码语音(或音频)最显而易见的方法是简单地对波形进行采样,然后(串行地)发送合成比特流。例如,以 8kHz 采样和 16b 量化的电话质量的语音能以 128kbps 的比特流速率进行编码。在一定带宽下,这使接收机能够重建出与信号源一样的波形。

然而,以这种方式进行直接采样所产生的比特率很大,并且在多路复用传输(互联网)和带宽受限信道(无线)的许多情况下都希望降低速率,以容纳更多的用户。比特率的降低通常都涉及了质量上的权衡,可达到的速率也取决于压缩算法的复杂度。

与前面讨论的自适应量化和压扩等简单的波形近似编码方法不同,这里采用参量编码器。顾名思义,它不是对波形本身进行编码,而是对表示波形的一些参数进行编码。其中的关键概念是摒弃了时域重构的精确性,而使用频域重构来衡量有效性。此外,经常使用的是感知标准,这意味着在设计中使用的是对波形声音的感知,而不是波形的确切形状。

5.8.1 线性预测语音编码

最大的一类低速率参数语音编码器采用的是综合分析法(Analysis-by-Synthesis Approach,ABS)。ABS 的关键概念是对语音进行采样,并分成大约 10～20 ms 的块(或帧),然后求出在频域中最能代表该帧的参数。接下来,这些参数被编码和发送,解码器重构近似(根据某些标准)原始波形频域特征的波形。原始语音和重构语音逐个采样点之间的精确对应关系将不再存在。

以这种方式,编码器的压缩阶段将大量采样点转换成合适的一组参数(即参数表达)。这样,解压缩阶段重建语音(或音频)通常只需较低的计算复杂度。

为了详细说明压缩阶段,先考虑预测问题,该问题已在 5.6.3 节中介绍了,当时是通过对采样值的加权线性和来预测编码器的新采样值。这种方法对于浊音语音(由肺和声道产生的语音片段,典型的元音如“ay”或“ee”)很有效,但对于清音语音(以 abrupt 结尾和辅音的发音,如“k”和“ss”)则不太有效。片段(或帧)中每个语音采样点的预测值可以写成如下线性加权和的形式。

$$\begin{aligned}\hat{x}(n) &= a_1 x(n-1) + a_2 x(n-2) + \cdots + a_P x(n-P) \\ &= \sum_{k=1}^{P} a_k x(n-k)\end{aligned} \tag{5.110}$$

而真正的采样值是预测值加上误差项,即

$$x(n) = \hat{x}(n) + e(n) \tag{5.111}$$

其中,$x(n)$为 n 时刻的语音信号采样值;$\hat{x}(n)$为信号的预测值;$e(n)$为预测的估计误差。

预测值 $\hat{x}(n)$可以被假设为确定性的成分,也就是说,它由模型参数表示。误差 $e(n)$在理想情况下当然为零,但实际上是一个很小且随机的值。将预测方程移项可得

$$e(n) = x(n) - \overbrace{\sum_{k=1}^{P} a_k \hat{x}(n-k)}^{\hat{x}(n)} \tag{5.112}$$

那么,问题是如何确定 P 个参数 a_k($k=1,2,\cdots,P$)。对于语音,取 $P=10$ 通常就足够了。因此,下面的讨论使用 10 阶预测器。首先,假设预测器参数已知,需要找到一个方法实现线性预测。直接传

输每个采样值的预测误差 $e(n)$，将使比特率小于原始比特率（假设误差很小），但这种方式本质上仍然是一种波形近似的编码器。为了将它转换为参数编码器，要认识到预测器本质上只是一个滤波操作（见5.3.2节），所需要的只是滤波器参数。然而，滤波器还需要某种输入。进一步研究发现，对于人类语音，输入可以是简单脉冲序列（用于浊音）或随机波形（用于清音）。这种方法应用于 LPC10 编解码器，其简化形式如图 5.34 所示。

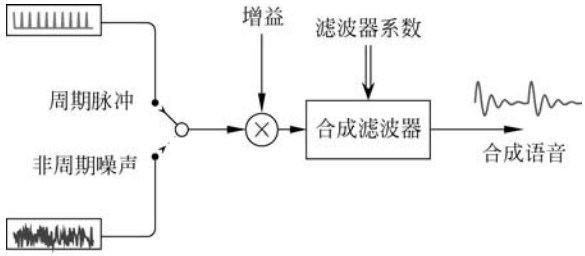


图 5.34　具有周期脉冲或随机噪声输入的线性预测编码器

为了导出 LPC 参数 a_k，需要用到自相关函数，将其定义如下。

$$R_{xx}(k)=\sum_{n=0}^{N-1}x(n)x(n-k) \tag{5.113}$$

其中，k 为相对偏移量。请注意，对于给定的数据记录，自相关是对称的，即

$$R_{xx}(k)=R_{xx}(-k) \tag{5.114}$$

P 阶线性预测估计值可写为

$$\hat{x}(n)=\sum_{k=1}^{P}a_k x(n-k) \tag{5.115}$$

平均误差为

$$\begin{aligned}\bar{e}&=E\{[x(n)-\hat{x}(n)]^2\}\\&=\sum_{n=1}^{N-1}[x(n)-\hat{x}(n)]^2\\&=\sum_{n=1}^{N-1}\left[x(n)-\sum_{k=1}^{P}a_k x(n-k)\right]^2\end{aligned} \tag{5.116}$$

为了找到最小均方误差，将关于预测器参数的导数设为零，即

$$\frac{\partial\bar{e}}{\partial a_m}=0 \tag{5.117}$$

然后，对导数应用链式法，可得

$$\sum_{n=0}^{N-1}2\left[x(n)-\sum_{k=1}^{P}a_k x(n-k)\right]\left\{\frac{\partial}{\partial a_m}\left[x(n)-\sum_{k=1}^{P}a_k x(n-k)\right]\right\}=0 \tag{5.118}$$

接下来需要计算导数项 $\partial/\partial a_m$。对于 $k\neq m$ 的所有 a_k，该项都将等于零。当 $k=m$ 时，它将简化为 $-x(n-m)$。数学上，可以将其表示为

$$\frac{\partial}{\partial a_m}\left[x(n)-\sum_{k=1}^{P}a_k x(n-k)\right]=\begin{cases}-x(n-m), & k=m\\0, & \text{其他}\end{cases},\quad m=1,2,\cdots,P \tag{5.119}$$

因此，需要最小化的表达式可以写为

$$\sum_{n=0}^{N-1}\left[x(n)-\sum_{k=1}^{P}a_k x(n-k)\right][-x(n-m)]=0$$

$$\begin{aligned}\sum_{n=0}^{N-1}x(n)x(n-m)&=\sum_{n=0}^{N-1}\left[\sum_{k=1}^{P}a_k x(n-k)x(n-m)\right]\\&=\sum_{k=1}^{P}a_k\left[\sum_{n=0}^{N-1}x(n-k)x(n-m)\right]\end{aligned}\tag{5.120}$$

左侧表达式可以认为是滞后 m 的自相关，对右侧表达式进行扩展，可得

$$\begin{aligned}R_{xx}(m)&=\sum_{k=1}^{P}a_k\left[\sum_{n=0}^{N-1}x(n-k)x(n-m)\right]\\&=a_1\sum_{n=0}^{N-1}x(n-1)x(n-m)+a_2\sum_{n=0}^{N-1}x(n-2)x(n-m)+\\&\quad\cdots+a_P\sum_{n=0}^{N-1}x(n-P)x(n-m)\end{aligned}\tag{5.121}$$

为了方便起见，去掉 xx 下标，当 $m=1$ 时，有

$$R(1)=a_1R(0)+a_2R(-1)+\cdots+a_PR(-(P-1))\tag{5.122}$$

当 $m=2$ 时，有

$$R(2)=a_1R(1)+a_2R(0)+\cdots+a_PR(-(P-2))\tag{5.123}$$

一直计算到 $m=P$，可得

$$R(P)=a_1R(P)+a_2R(P-1)+\cdots+a_PR(0)\tag{5.124}$$

因为在预测方程中有大量的项(对于语音通常取 $P=10$)，所以将所有这些方程写成矩阵形式更方便，即

$$\begin{bmatrix}R(0)&R(-1)&\cdots&R(-(P-1))\\R(1)&R(0)&\cdots&R(-(P-2))\\\vdots&\vdots&\ddots&\vdots\\R(P-1)&R(P-2)&\cdots&R(0)\end{bmatrix}\begin{bmatrix}a_1\\a_2\\\vdots\\a_P\end{bmatrix}=\begin{bmatrix}R(1)\\R(2)\\\vdots\\R(P)\end{bmatrix}\tag{5.125}$$

且由于自相关是对称的，即 $R(k)=R(-k)$，可得

$$\begin{bmatrix}R(0)&R(1)&\cdots&R(P-1)\\R(1)&R(0)&\cdots&R(P-2)\\\vdots&\vdots&\ddots&\vdots\\R(P-1)&R(P-2)&\cdots&R(0)\end{bmatrix}\begin{bmatrix}a_1\\a_2\\\vdots\\a_P\end{bmatrix}=\begin{bmatrix}R(1)\\R(2)\\\vdots\\R(P)\end{bmatrix}\tag{5.126}$$

式(5.126)可以更简洁地写成如下矩阵方程，即

$$\boldsymbol{Ra}=\boldsymbol{r}\tag{5.127}$$

其中

$$\boldsymbol{R}=\begin{bmatrix}R(0)&R(1)&R(2)&\cdots&R(P-1)\\R(1)&R(0)&R(1)&\cdots&R(P-2)\\\vdots&\vdots&\vdots&\ddots&\vdots\\R(P-1)&R(P-2)&R(P-3)&\cdots&R(0)\end{bmatrix}\tag{5.128}$$

是自相关值的矩阵，并且

$$\boldsymbol{r}=\begin{bmatrix}R(1)\\R(2)\\\vdots\\R(P)\end{bmatrix} \tag{5.129}$$

是自相关值的矢量。可以求解出期望的线性预测(Linear Prediction,LP)参数为

$$\boldsymbol{a}=\begin{bmatrix}a_1\\a_2\\\vdots\\a_P\end{bmatrix} \tag{5.130}$$

这提供了一种求解最优预测器参数 a_p 的方法,但是这个方法似乎需要相当大的计算复杂度(一个 $P\times P$ 的矩阵求逆)。然而,自相关矩阵 $\boldsymbol{R}$ 的对称性质有利于高效的求解,通常采用 Levinson-Durbin 递归算法来简化计算。

到目前为止,前面的理论给出了信号的良好预测(即误差很小)。最显而易见的方法是量化并发送误差信号,如本节开头所述,采用逐点恢复精确波形的方法,构成了一个波形编码器。但是,这种方法中每个采样点需要一位或多位比特,还可以有更好的方法,即使用参量的方法,放弃精确的时域重构要求,只发送预测器参数,然后这些参数就构成了解码器中的滤波器参数,而输出近似于语音的短时频谱。

那么,问题是用什么作为滤波器的输入呢?首先,一系列脉冲在短时间内可近似语音信号的音调,这给出了语音频谱的粗略表示,当然可以对其改进。对于具有强周期性的浊音语音来说,这种类型的激励已经足够了。当周期性不太明显时(在清音语音期间),可以采用白噪声激励。这样就得到了图 5.34 中给出的 LP 编码器,这种编码器能以低至 2kbps 的速率产生可接受的语音,这种语音虽然可以被人所理解,但听起来不够自然。对于某些应用,这是可以接受的(这种编码器最初是为军事应用开发的)。然而,对于商业语音通信服务来说,在一定程度上听起来比较自然是必须的,这就引出了下面的综合分析法。

5.8.2 综合分析法

为了提高 LP 编码器的感知质量和自然度,可以采用若干改进措施。最直观的方法是改变声源处的激励类型,不采用简单的二元浊音/清音的激励分类。

使用反馈回路,可以在编码器中产生和分析合成语音,而不是只能在解码器中产生合成语音。编码器根据某种定义的标准调整激励参数,以产生最佳语音,这种方法是非常有效的。

一种可能的激励情况是使用多脉冲,从而产生多脉冲激励(Multi-Pulse Excitation,MPE)编码器,它需要在激励框架内放置脉冲。因此,除了脉冲的位置,还必须确定脉冲的幅度大小。另一种更简单的变体是规则脉冲激励(Regular Pulse Excitation,RPE),其中脉冲间隔不是可变的,而是固定的。

最后,另一种编码方法为码激励线性预测(Code Excited Linear Prediction,CELP),如图 5.35 所示。在这种方法中,不是使用已知的脉冲型激励,而是将随机激励矢量存储在码本中。在编码器处测试每个激励矢量以合成语音,并选择性能最佳的一个。因此,选择的是样本矢量而不是脉冲幅度(以及可能的位置)。由于编码器和解码器具有相同的预存码本,因此编码并传输的是最佳匹配码矢量的索引。这种方法能够以非常低的速率得到高质量的语音。然而,这是以相当大的额外计算复杂度为代价的,因为每个可能的样本都必须在编码器处进行处理和测试。例如,对于 10b 的码本,如果要独立评估每一帧,就需要测试 $2^{10}\approx 1000$ 帧中的每一帧。

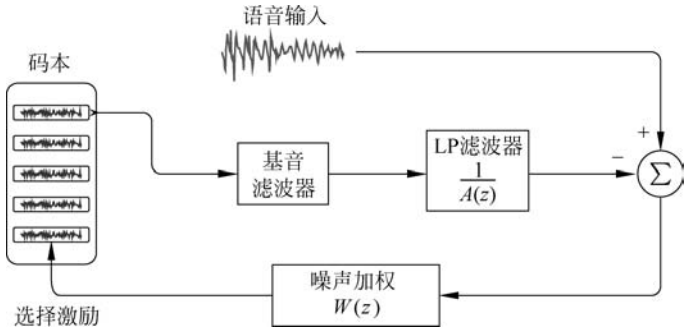


图 5.35　码激励线性预测编码器的原理

注：根据合成语音和原始语音之间的匹配来选择激励。

5.8.3　频谱响应和噪声加权

前面介绍的 LPC 方法通常在模拟产生低频信号方面优于高频。为了弥补这一点，可以使用一个称为预加重的过程，需要如下形式的高通滤波器。

$$H_{\text{pre}}(z)=1-\lambda z^{-1} \tag{5.131}$$

通常取 $\lambda=0.95$，在 LPC 处理阶段之前使用。同样，可以使用去加重滤波器来反转该过程，即该过程的倒数。

$$H_{\text{de}}(z)=\frac{1}{1-\eta z^{-1}} \tag{5.132}$$

如果取 $\eta=\lambda$，那么预加重和去加重过程将精确匹配。然而，在实际中，设置 $\eta=0.75$（略小于 λ）时效果更好，使解压缩后的语音更加清晰。

对这个想法进行进一步的扩展就是噪声整形（Noise Shaping）滤波器，使用的前提是在某个频段中感知的噪声能量小于有效信号的能量。也就是说，语音中更响亮的部分有效地掩盖了噪声。由此可推出，当某个频段具有较低的语音能量时，在该频段中会感知到更多的噪声。因此，在 LPC 滤波器的基础上进行噪声加权是很有意义的。如果 LPC 滤波器为 $A(z)$，那么遵循这些原则的噪声加权滤波器定义如下。

$$W(z)=\frac{A(z)}{A(z/\gamma)},\quad 0\leqslant\gamma\leqslant 1 \tag{5.133}$$

将 $A(z)$滤波器展开，噪声整形滤波器可写为

$$W(z)=\frac{1-\sum_{k=1}^{P}a_k z^{-k}}{1-\sum_{k=1}^{P}a_k\gamma^k z^{-k}},\quad 0\leqslant\gamma\leqslant 1 \tag{5.134}$$

系数 γ 通常选择为 0.9～0.95 的数量级，这样给出的频率响应略低于原始值。因为噪声整形滤波器与 LPC 滤波器是级联使用的，所以系统的整体响应变为 $1/A(z/\gamma)$。这实际上意味着什么，又是如何影响频率响应的？下面对其进行研究。因为实际上等价于发生了如下替换。

$$z\rightarrow\left(\frac{z}{\gamma}\right) \tag{5.135}$$

指数为 k 的每个含 z 项变为

$$\left(\frac{z}{\gamma}\right)^{k}=\gamma^{-k}z^{k} \tag{5.136}$$

因此，极点变为

$$\left(\frac{z}{\gamma}+p\right)=\frac{1}{\gamma}(z+\gamma p) \tag{5.137}$$

这意味着极点幅度减小了一个小的因子 γ。需要注意的是，受到影响的是幅度，而不是极点角度。因为极点角度决定了频率，所以总频率峰值位置不变。

为了说明这一点，图 5.36(a)显示了一个 LPC 函数的极点，该函数是使用前面所述的算法针对特定语音帧导出的，图 5.36(b)还显示了该滤波器的频率响应。

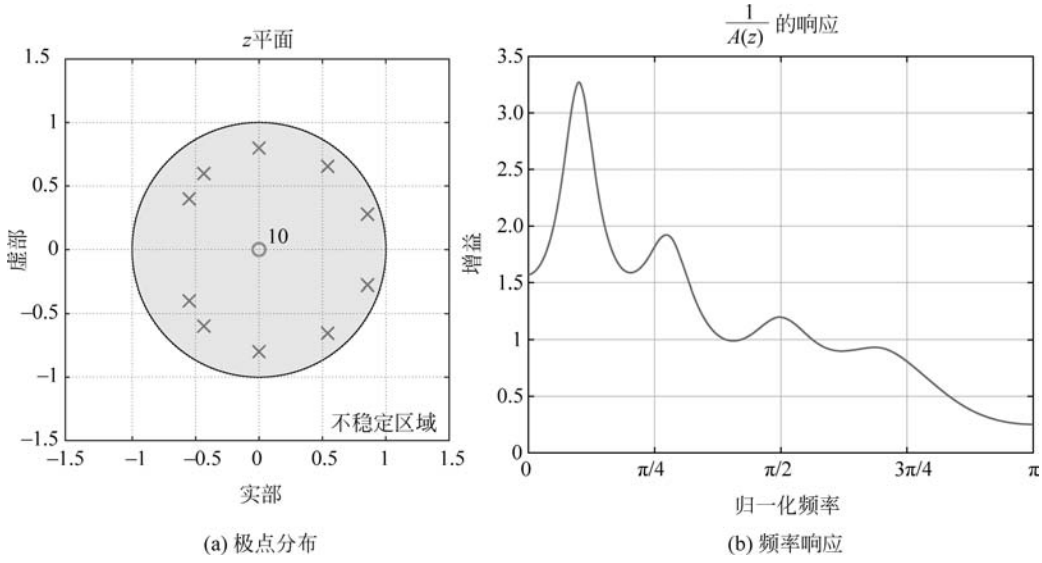


图 5.36　线性预测器的极点及相应的频率响应

注：由共振模拟声道以产生“合成”语音。

现在假设应用了上述噪声加权原理，取 $\gamma=0.85$。如图 5.37 所示，极点已经发生了改变。很明显，极点角度是不变的，但径向距离减小了。相应的频率响应在相同的位置出现峰值，然而，由于极点向内移动，峰值的大小有所减小，并且分散开来。与未改变的 LPC 相比，这会导致语音信号的声音略微减弱。

考虑到噪声加权滤波器 $W(z)$在图 5.35 的反馈环路中的位置，LPC 频率响应峰值周围的区域增益变小，这意味着更多的量化噪声被分配给频谱能量更强(情况更好)的区域，最终在 LPC 频率响应不太强的区域中会出现更少的量化噪声。因此，噪声整形是指量化噪声根据待编码信号的频谱进行整形。

5.8.4　音频编码

前面的章节讨论了语音编码。在这种应用中，主要目的是降低压缩语音的比特率。重构语音的质量通常是次要考虑因素，通常只需要足够“自然”。毕竟，采样速率通常低至 8kHz(带宽小于 4kHz)。

然而，对于音乐，音频编码是一个不同的问题，这里的目标是尽可能保证质量。这是通过几种机制来实现的，包括对音频频谱中感知不到的部分不进行编码。因此，它是一个有损编码器。

用于语音编码的线性预测方法通常不适用于音乐，因为它会产生非常差的音频质量(尤其是针对语音信号优化后)。因此，音频编码采用了不同的技术。音频编码(与语音编码相反)通常用于离线(非实

时)模式,这意味着信号不必立即压缩,可以放宽语音编码非常严格的实时性限制。实际上,缓冲期越长,能得到越大的压缩。此外,压缩的计算要求并不关键,因此允许算法更加复杂。

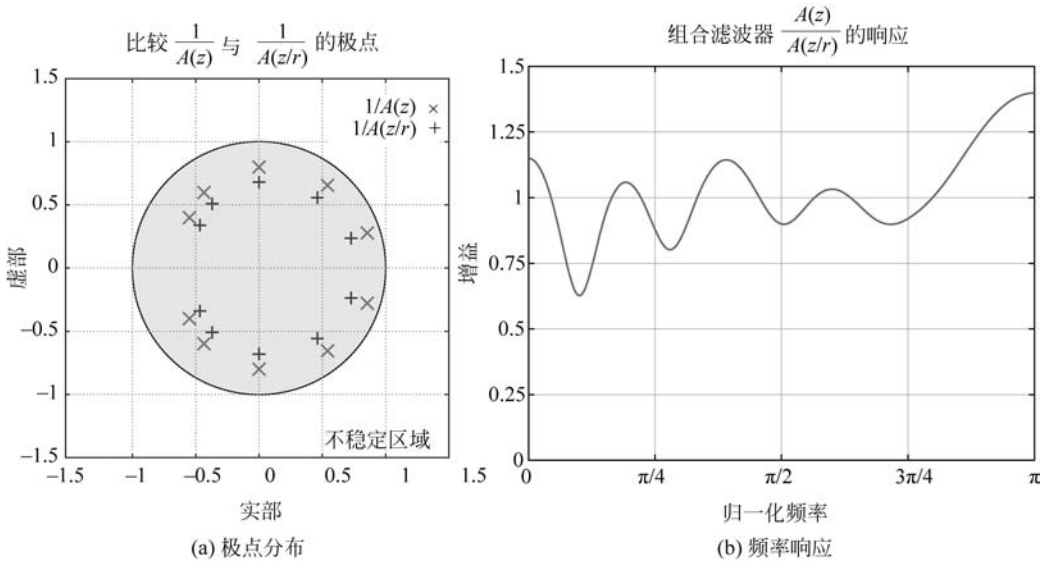


图 5.37　加入了噪声整形后的线性预测器的极点对比

注:图(a)展示了线性预测器的极点(×)和语音帧相应的噪声加权极点(+),对于指定值 $\gamma=0.85$,极点向内移动并使图 5.36 所示的频率响应变平。得到的噪声加权滤波器 $W(z)$ 的频率响应如图(b)所示。

音频编码器经过多年的研究发展起来,并且还在继续发展。这是一个复杂的领域,本节只介绍基本原理的概要。Brandenburg(1999)给出了一个更完整的概要,并引用了大量参考文献。

感知编码旨在降低分辨率,同时不会对感知的音频质量产生很大影响,从而节省必须编码和传输的参数。当然,典型的声音再现电子设备(放大器和扬声器)的保真度以及听觉环境本身都会以某种方式对音频产生影响。

图 5.38 给出了通用音频编码器的框图,如 MP3 系列编码器。在语音编码器中,有脉冲状激励就足够了。因为时域波形本身不被编码,而是用频域表示,这为编码器提供了便利。在音频编码器中,将源信号划分成频带,并根据它们的感知相关性分别进行编码,这是由滤波器组完成的。然后在每个频带上进行变换(修正 DCT),以便减少样本之间的相关性,这与图像编码器使用 DCT 的方式非常相似(除了现在是在一维上)。重叠帧用来减少可能被听出来的块效应。然后,量化阶段将变换后的滤波器组系数用适当的二进制表示。量化阶段用到了感知模型,而感知模型则利用了原始信号的傅里叶分析。最后,进行熵效率比特分配以产生编码比特流。很明显,编码器通常比解码器更复杂,因为它必须进行滤波、量化和感知分析。

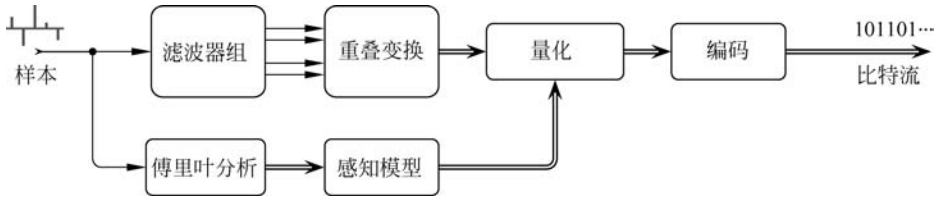


图 5.38　通用音频编码器框图

注:音频编码器包括了子带编码(滤波器组)、重叠变换、感知加权和熵编码。

在对人类听觉和声音感知进行大量研究后，滤波器组得到了优化。滤波器频带的数量及其带宽都很重要，并且每个频带的带宽通常都不相同。

通常所采用的 DCT 属于重叠变换类，被称为修正 DCT（Modified DCT，MDCT）（Princen，et al.，1987；Malvar，1990）。MDCT 不同于常见的变换，它的输出数量与输入并不相同。MDCT 有 $2N$ 个输入和 N 个输出，如下所示。

$$X(k)=\sum_{n=0}^{2N-1}x(n)\cos\left[\left(\frac{\pi}{N}\right)\left(n+\frac{1}{2}+\frac{N}{2}\right)\left(k+\frac{1}{2}\right)\right] \tag{5.138}$$

也可以按照完整的双倍长度输入块大小来改写，取 $M=2N$，即

$$X(k)=\sum_{n=0}^{M-1}x(n)\cos\left[\left(\frac{\pi}{2M}\right)(2n+1+N)(2k+1)\right] \tag{5.139}$$

逆 MDCT 有 N 个输入和 $2N$ 个输出，即

$$x(n)=\frac{1}{N}\sum_{k=0}^{N-1}X(k)\cos\left[\left(\frac{\pi}{N}\right)\left(n+\frac{1}{2}+\frac{N}{2}\right)\left(k+\frac{1}{2}\right)\right] \tag{5.140}$$

请注意，除了缩放常数之外，正向和逆向变换具有相同的形式。

重叠块的使用可以减少块边界处可能听得出的块效应。连续块的重叠和相加精确地重构了输出序列。重叠过程如图 5.39 所示，提供了精确重构原理的一个具体示例，是一个块大小 $M=4$ 的简化情况。每个块都得到长度 $N=2$ 的输出块，刚好是输入大小的一半。下面代码中显示的输入参数得到如图 5.39 所示的输出。当然，实际中的块大小要大得多。由于逆变换的输出数量大于输入，因此会出现混叠，并且如果直接提取逆变换的输出，不会得到原始序列。如图 5.39 所示，还需要将连续块相加，其中第一个和最后的子块没有重叠，因此没有被精确地重构。

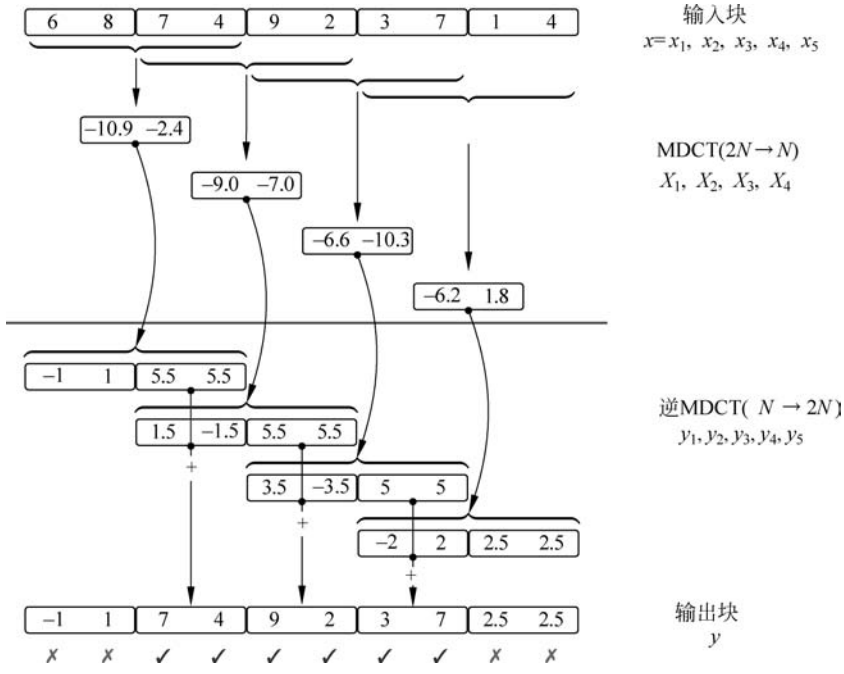


图 5.39　修正 DCT 的重叠块

注：图中所示的块大小为 $M=4$ 和 $N=2$，以说明连续块的重叠和相加可以得到精确的重构。

```
% mdct()和 imdct()函数对于任意长度都是通用的
% 下面的示例显示的是具有两点输出的 4 点分块
% 输入矢量
x = [6 8 7 4 9 2 3 7 1 4];
disp(x);

% 将输入矢量划分为 4 个样本的重叠块
x1 = x(1:4);
x2 = x(3:6);
x3 = x(5:8);
x4 = x(7:10);

disp(x1);
disp(x2);
disp(x3);
disp(x4);

% 对每个块进行 MDCT,由 4 到 2
X1 = mdct(x1);
X2 = mdct(x2);
X3 = mdct(x3);
X4 = mdct(x4);

% 对每个变换后的块进行 IMDCT,由 2 到 4
y1 = imdct(X1);
y2 = imdct(X2);
y3 = imdct(X3);
y4 = imdct(X4);

disp(y1');
disp(y2');
disp(y3');
disp(y4');

% 组合重叠输出块
y = zeros(1, length(x));

y(1:2) = y1(1:2)';                % 结果不正确
y(3:4) = y1(3:4)' + y2(1:2)';
y(5:6) = y2(3:4)' + y3(1:2)';
y(7:8) = y3(3:4)' + y4(1:2)';
y(9:10) = y4(3:4);                % 结果也不正确
```

以下代码显示了对于 $M=2N$ 个输入和 N 个输出的 MDCT 函数。

```
function [X] = mdct(x)
    x = x(:);
    M = length(x);
    N = round(M/2);

    X = zeros(N, 1);
    for k = 0:N-1
```

```
        s = 0;
        for n = 0:2*N-1 % M-1
            % 以下是等价的
            % s = s + x(n+1)*cos( (pi/N)*(n + 1/2 + N/2)*(k + 1/2) );
            s = s + x(n+1)*cos( (pi/(2*M))*(2*n + 1 + N)*(2*k + 1) );
        end
        X(k+1) = s;
    end
end
```

补充对于 $M=2N$ 个输出和 N 个输入的逆 MDCT 函数如下。

```
function [y] = imdct(X)
    X = X(:);
    N = length(X);

    y = zeros(2*N,1);
    for n = 0:2*N-1 % M-1
        s = 0;
        for k = 0:N-1
            s = s + X(k+1)*cos( (pi/N)*(n + 1/2 + N/2)*(k + 1/2) );
        end
        y(n+1) = s/N;
    end
end
```

感知编码非常依赖于感知研究(Schroeder, et al., 1979),不仅利用了人耳在不同频带上的非线性灵敏度,还利用了感知音调的能力。掩蔽现象指的是一个音调可能被相邻音调所掩盖或遮蔽的事实。如果是这种情况,就不需要对其进行编码。Painter 和 Spanias(1999,2000)对于音频和语音编码的感知方面进行了大量研究。

5.9 本章小结

下面是本章的要点:

- 标量量化,包括均匀和非均匀步长特性;
- 矢量量化,包括训练和搜索方法;
- 使用块变换的图像编码;
- 使用 ABS 的语音编码和使用变换的音频编码。

习题 5

5.1 香农限给出指定 SNR 的信道容量,试推导出香农限。其中需要高斯信号熵,首先,从集合 X 中取出的连续变量 x 的熵,定义如下。

$$H(X)=\int_{-\infty}^{\infty} f_X(x)\mathrm{lb}\left[\frac{1}{f_X(x)}\right]\mathrm{d}x \tag{5.141}$$

高斯分布的信号熵(信息量)可以表示为

$$H(X)=\frac{1}{2}\mathrm{lb}2\pi\mathrm{e}\sigma^2 \tag{5.142}$$

还会用到以下的标准积分和代换。

$$\int_{-\infty}^{\infty}x^2\mathrm{e}^{-ax^2}\mathrm{d}x=\frac{1}{2}\sqrt{\frac{\pi}{a^3}} \tag{5.143}$$

其中,$a=1/(2\sigma^2)$,且有

$$\int_{-\infty}^{\infty}\mathrm{e}^{-ax^2}\mathrm{d}x=\sqrt{\frac{\pi}{a}} \tag{5.144}$$

同样,$a=1/(2\sigma^2)$。

5.2　关于无损压缩,请回答如下问题。

(1) 什么是滑动窗口压缩? 从编码器传输到解码器的是什么? 这种方法的缺点是什么?

(2) 什么是基于字典的压缩? 每一阶段传输的是什么? 这种方法的缺点是什么?

5.3　研究熵理论与无损压缩之间的关系,要求如下。

(1) 编写并测试 MATLAB 代码,计算数据文件的熵(单位为 b/symbol)。使用未压缩格式的灰度图像文件进行测试,如 BMP 或无损 JPEG。

(2) 使用标准无损压缩(如 zip、bzip、gzip 或类似的压缩方式)来压缩上一问中使用的数据文件。得到的文件大小和压缩率是多少? 根据上一问中的程序,压缩文件的熵是多少?

5.4　霍夫曼编码是一种无损编码。

(1) 为概率分别为 0.2、0.4、0.25、0.15 的符号 A、B、C、D 构造霍夫曼编码,使用不遵循兄弟性质的树,计算熵和预期的平均比特率。

(2) 使用遵循兄弟性质的树重复上述操作,计算预期的平均比特率。与上一问中没有使用兄弟性质构造的码树所获得的结果进行比较。

5.5　为概率分别为 0.30、0.25、0.20、0.18、0.07 的符号 A、B、C、D、E 创建一个类似于图 5.14 所示的霍夫曼码树。使用提供的生成、编码和解码霍夫曼码 MATLAB 代码完成以下操作。

(1) 创建每个码字并确定集合的平均码字长度。

(2) 对每个码字进行编码和解码,并检查解码是否正确。

5.6　证明二维 DCT 是可分离的。也就是说,矩阵的 DCT 可以通过对行进行一维变换,然后对所得矩阵的列进行一维变换来完成。

5.7　对于维数 $N=4$ 的 DCT,使用以下正向二维 DCT 代码实现逆二维 DCT。验证先使用 DCT 再使用 IDCT 之后得到的输出与原始输入矢量 $\boldsymbol{X}$ 相等,但存在计算精度误差。

```
N = 4;                          % 块大小
X = rand(N, N);                 % 输入块
Y = zeros(N, N);                % 输出块

 % 每个输出系数
for k = 0:N-1
    for el = 0:N-1

        % 计算一个 DCT 系数
        s = 0;
```

```
            for n = 0:N-1
                for m = 0:N-1
                    s = s + X(m+1, n+1)*...
                              cos( (2*m+1)*k*pi/(2*N) ) * ...
                              cos( (2*n+1)*el*pi/(2*N) );
                end
            end

            if( k == 0 )
                ck = 1/sqrt(2);
            else
                ck = 1;
            end

            if( el == 0 )
                cl = 1/sqrt(2);
            else
                cl = 1;
            end

            Y(k+1, el+1) = 2/N*ck*cl*s;
        end
    end
```

5.8 数据序列的方差会影响它的可预测性。

(1) 利用 MATLAB 计算典型灰度图像的相关系数。

(2) 图像源$\{x(n)\}$具有零均值、单位方差,且相关系数 $\rho=0.95$。证明差分信号 $d(n)=x(n)-x(n-1)$的方差明显低于原始信号。

(3) 比值 σ_x^2/σ_d^2 可以用来衡量一组数据的预测性能的好坏。对于简单的一阶差分预测器 $\hat{x}(n)=a_1x(n-1)$,试确定其比值 σ_x^2/σ_d^2。

5.9 可以通过使用水平和垂直预测增强图像预测性能。

(1) 二维图像编码系统的实现方式如下,每个像素由它前一个像素和它前一行中紧接着它的像素的平均值来预测,即

$$\begin{array}{cc} \cdot & \cdot \quad x_c \quad n-1\text{ 行} \\ \cdot & x_b \quad x_a \quad n\text{ 行} \end{array}$$

预测方程为

$$\hat{x}_a=\frac{x_b+x_c}{2}$$

假设所有样本的平均值为零,垂直和水平方向的相关系数相同($\rho_{ac}=\rho_{ab}=\rho$),对角相关系数近似为 $\rho_{bc}=\rho^2$。求信号方差与预测方差的比值 σ_x^2/σ_e^2。

(2) 对于 $\rho=0.95$,求 σ_x^2/σ_e^2,并与前一题中的一维预测器进行比较。

5.10 拉普拉斯分布通常被用作图像像素分布的统计模型。由于像素的动态范围对于分配量化范围和步长都至关重要,因此知道像素超出范围被截断的可能性是很有意义的。给定零均值拉普拉斯分布如下。

$$f(x)=\frac{1}{\sigma\sqrt{2}}\mathrm{e}^{-\sqrt{2}|x|/\sigma}$$

求$|x|>\sigma$的概率。

5.11　一个均匀分布的随机变量使用四电平均匀中升量化器进行量化，试概述该量化器采用的决策—重构规则。通过最小化误差方差，证明最佳步长为$(\sigma\sqrt{3})/2$。对于不同的随机变量分布，最佳步长会改变吗？

5.12　用于视频会议的视频编码器以每秒10帧的速度对大小为512×512像素的图像进行编码。使用8×8的子块直接进行矢量量化，速率为0.5bpp。

(1) 确定每个图像的块数，以及编解码器(编码器—解码器)每秒必须编码的块数。

(2) 如果每个块被编码为一个平均值，用8b表示，剩余的比特用于矢量成形，确定VQ码本的大小。

(3) 确定每秒矢量比较的次数。

(4) 确定每秒算术运算次数，假设每次运算花费1ns，确定搜索时间。

5.13　生成随机高斯波形的100个样本。

(1) 将此样本输入滤波器传递函数

$$\hat{y}(n)=b_0x(n)+a_1y(n-1)+a_2y(n-2)$$

其中，$a_1=0.9$，$a_2=-0.8$，$b_0=1$。画出$y(n)$，$n=0,1,2,\cdots,N-1$。

(2) 用MATLAB估计二阶自相关矩阵$\boldsymbol{R}$和自相关矢量$\boldsymbol{r}$。

(3) 计算滤波器传递函数参数$\hat{a}_k$，并比较估计值$\hat{a}_k$与真实值a_k的接近程度。