语音信号处理实践作业报告

题目： 语音信号的的自适应差分脉冲编码调制(ADPCM)编解码实现

姓名：

班级：

学号：

2024年4月

目录

[1. 技术原理与背景 1](#_Toc169768153)

[1.1 技术原理 1](#_Toc169768154)

[1.2 背景 1](#_Toc169768155)

[2. 算法思想与步骤 2](#_Toc169768156)

[2.1 APDCM编码函数算法思想 2](#_Toc169768157)

[2.1.1差分编码： 2](#_Toc169768158)

[2.1.2自适应量化： 2](#_Toc169768159)

[2.1.3步长查找表： 2](#_Toc169768160)

[2.1.4符号位编码： 2](#_Toc169768161)

[2.1.5预测机制： 2](#_Toc169768162)

[2.2 APDCM编码函数设计步骤 2](#_Toc169768163)

[2.2.1 输入预处理： 3](#_Toc169768164)

[2.2.2 构建量化步长查找表： 3](#_Toc169768165)

[2.2.3 初始化编码状态： 3](#_Toc169768166)

[2.2.4 编码循环： 3](#_Toc169768167)

[2.2.5 输出编码结果： 3](#_Toc169768168)

[2.3 APDCM解码函数算法思想 3](#_Toc169768169)

[2.3.1 初始化参数与环境： 3](#_Toc169768170)

[2.3.2 逐位解码与预测更新： 3](#_Toc169768171)

[2.3.3自适应步长调整： 3](#_Toc169768172)

[2.3.4信号重构与输出： 4](#_Toc169768173)

[2.2 APDCM解码函数设计步骤 4](#_Toc169768174)

[2.4.1 初始化解码器状态： 4](#_Toc169768175)

[2.4.2 构建量化步长查找表： 4](#_Toc169768176)

[2.4.3 解码循环： 4](#_Toc169768177)

[2.4.4 输出信号归一化： 4](#_Toc169768178)

[3.编程实现 5](#_Toc169768179)

[3.1 初始化与环境设置 5](#_Toc169768180)

[3.2 定义ADPCM编码参数 5](#_Toc169768181)

[3.3 执行ADPCM编码 5](#_Toc169768182)

[3.4 执行ADPCM解码 5](#_Toc169768183)

[3.5 计算信号质量指标 5](#_Toc169768184)

[3.6 绘制图形结果 5](#_Toc169768185)

[4.调试结果 6](#_Toc169768186)

[4.1 编码前语音 6](#_Toc169768187)

[4.2 编码后语音 6](#_Toc169768188)

[5.源代码 7](#_Toc169768189)

[5.1 main 7](#_Toc169768190)

[5.2 APDCM编码函数 7](#_Toc169768191)

[5.3 APDCM解码函数 9](#_Toc169768192)

1. 技术原理与背景

脉冲编码调制（Pulse Code Modulation，PCM）是一种将模拟信号数字化的方法，它通过采样、量化和编码三个步骤将连续的模拟信号转换成数字信号。然而，PCM虽然能提供高质量的声音，但其数据率相对较高，例如，在电话系统中，典型的PCM编码使用8位量化，采样频率为8kHz，这导致数据速率为64kbps。

自适应差分脉冲编码调制（Adaptive Differential Pulse Code Modulation，ADPCM）则是一种改进的编码方式，旨在减少PCM所需的带宽，同时保持可接受的音质。ADPCM的关键在于它利用了相邻样本之间的相关性，并且动态调整量化步长来适应信号幅度的变化。

1.1 技术原理

采样：模拟信号首先被按照一定频率（如8kHz）采样，得到一系列的离散时间点上的信号幅度。

差分编码：ADPCM不是直接对信号样本进行编码，而是编码样本与前一个样本（或预测样本）之间的差异（差分）。这个差分值通常比原始样本值小得多，因此可以用更少的比特数来表示。

自适应量化：根据信号的变化速率，动态地调整量化步长。当信号变化缓慢时，使用较小的步长；当信号变化快时，使用较大的步长。这样可以在信号复杂度高时保持精度，而在信号简单时节省比特数。

编码：差分值经过量化后，用较少的比特数（如4位）编码，从而降低数据率。

编码流程

1. 输入：PCM信号（8位/样本）
2. 预测：根据过去的几个样本预测当前样本的值。
3. 差分：计算实际样本值与预测值的差。
4. 量化：根据自适应量化表量化这个差值。
5. 编码：用较少的比特（如4位）表示量化的差值。

解码流程

1. 解码：将接收到的4位ADPCM代码解码回量化后的差分值。
2. 逆量化：使用相同的自适应量化表逆量化这个差分值。
3. 重构：将差分值加到预测值上，重构出原始信号的近似值。
4. 输出：重构的PCM信号（8位/样本）。

图示

描述已自动生成

**ADPCM解码器原理**

1.2 背景

历史发展：ADPCM最初在20世纪70年代被开发，目的是为数字通信系统提供一种有效的语音编码方法。

国际标准：国际电信联盟（ITU）制定了G.721和G.726等ADPCM标准，这些标准被广泛应用于数字电话网络和语音邮件系统。

应用领域：ADPCM被用于多种通信系统，包括固定电话网络、移动电话、VoIP（互联网电话）和卫星通信。

压缩效率：ADPCM通常能够将语音信号压缩到较低的比特率，例如32 kbps或40 kbps，同时保持可接受的语音质量。

计算复杂性：与其他语音编码技术相比，ADPCM具有较低的计算复杂性，这使得它适合于资源受限的系统。

改进与发展：随着技术的发展，ADPCM的一些局限性（如压缩效率和音质）已经被更先进的编码技术所克服，例如MP-MLQ（多脉冲最大似然量化）和现代的编码标准如Opus。

ADPCM作为一种早期的语音编码技术，在语音通信领域具有重要的历史意义。尽管现代编码技术在许多方面超越了ADPCM，但ADPCM的基本原理和概念仍然对理解语音编码和通信系统的发展具有指导意义。

ADPCM编码和解码可以通过软件算法实现，也可以通过专用硬件（如FPGA）实现，以满足实时处理的需求。在设计ADPCM编解码器时，重点在于优化预测算法和量化策略，以确保良好的音质和数据压缩比。

2. 算法思想与步骤

2.1 APDCM编码函数算法思想

APDCM（Adaptive Differential Pulse Code Modulation）算法是一种基于预测和自适应调整的高效音频编码技术，其核心目标是在保证重构信号质量的同时，尽可能减少所需的比特数，达到数据压缩的目的。以下是APDCM算法的几个关键思想：

### 2.1.1差分编码：

APDCM首先采用差分编码的思想，不是直接对原始信号采样值进行编码，而是对相邻采样点之间的差异（差值）进行编码。

这样做的好处是可以利用信号的连续性，尤其是在信号变化缓慢时，连续的差值往往较小，易于用较少的比特表示。

### 2.1.2自适应量化：

“自适应”是APDCM算法的关键特性。量化是将连续的差值转换为离散的、有限集合的过程，通常会导致量化误差。APDCM通过分析前几帧的量化误差，动态调整量化步长（即相邻两个可选量化值之间的差距）。

当信号变化剧烈时，增大量化步长，减少对细节的敏感度；当信号平缓时，则减小量化步长，提高精度。这使得编码更加灵活，能更好地匹配信号的实际动态范围。

### 2.1.3步长查找表：

为了实现自适应量化，算法维护一个步长查找表，根据当前量化误差的大小来选择下一次编码的量化步长。查找表的设计基于对数分布，确保了在不同信号动态范围内的高效编码。

### 2.1.4符号位编码：

通过对差值的符号进行编码（正或负），进一步压缩信息。在编码过程中，如果差值为负，则在编码比特流中加入一个额外的“符号位”，指示后续的量化值应理解为负数。这种编码方式结合了极性信息，使得单个量化值能够代表更宽的差值范围。

### 2.1.5预测机制：

在每次编码后，使用当前的量化输出作为下一个采样点预测的基础，不断更新预测值。这种机制利用了信号的时序相关性，使得连续的编码值能够反映信号的变化趋势，而非独立的绝对值。

综上所述，APDCM算法通过差分、自适应量化、步长调整、符号位编码和预测机制的综合运用，实现了对音频信号的高效压缩，同时保持了较好的重构质量。这种算法特别适合于语音通信和存储空间受限的应用场景。

2.2 APDCM编码函数设计步骤

### 2.2.1 输入预处理：

输入音频信号x乘以128进行尺度变换，以便量化处理。获取信号长度len。

### 2.2.2 构建量化步长查找表：

初始化量化步长调整索引数组index为[-1 4]，表示量化步长可能的增减情况。设置量化步长范围：起始值startval=1，结束值endval=127，确保量化步长在合理范围内。计算base，用于确定量化步长增长的指数关系。计算量化步长查找表table2的大小numSteps，并根据对数关系生成量化步长序列，存储于table2。

### 2.2.3 初始化编码状态：

初始化量化误差累积序列ss，设置初始量化步长为table2(1)。初始化过去样本预测值序列z和编码输出序列code为全零向量。初始化符号位标志neg为0。

### 2.2.4 编码循环：

遍历输入信号x的每一个样本（从第二个样本开始，索引为2）。计算当前样本与前一预测值的差值d(n)。根据差值正负更新符号位标志neg，并相应调整差值d(n)的正负。比较差值d(n)与当前量化步长ss(n-1)，若差值大于等于量化步长，则在编码输出code(n)上加1。根据符号位计算量化输出的实际量化值temp，并根据此计算量化误差temp2。更新预测值z(n)，并确保其在[-127, 127]范围内。根据量化输出调整量化步长索引currentIndex，并更新量化步长ss(n)，确保索引在有效范围内。

### 2.2.5 输出编码结果：

循环结束后，code即为经过ADPCM编码的比特流。

此adpcm\_encoder函数实现了ADPCM编码过程，通过自适应调整量化步长以适应输入信号的动态范围，提高了编码效率，同时保持了较好的重构质量。

2.3 APDCM解码函数算法思想

APDCM解码函数是自适应差分脉冲编码调制(Adaptive Differential Pulse Code Modulation)编码过程的逆操作，旨在从经过APDCM编码的比特流中恢复原始信号。解码过程遵循编码时的逆逻辑，主要包含以下几个关键步骤和思想：

### 2.3.1 初始化参数与环境：

首先，根据编码时相同的参数设置（如量化步长查找表的构建参数），初始化解码所需的各种变量和参数，包括步长查找表table2、当前索引currentIndex、量化步长范围边界startval和endval、以及用于计算步长的基数base等。这些初始化保证了编码和解码环境的一致性。

### 2.3.2 逐位解码与预测更新：

解码过程从第二位开始（因为第一位已知，用于初始化），每一步都基于前一位的预测值和当前的编码比特进行操作。首先，根据符号位判断当前编码值的正负，然后根据编码值和当前量化步长计算量化误差的绝对值，累加到前一时刻的预测值上，得到当前的预测值y(n)。这一过程复原了差分编码时的信息。

### 2.3.3自适应步长调整：

类似于编码过程，解码时也需要根据解码出的量化值调整量化步长，以适应信号动态范围的变化。通过解码得到的临时变量temp确定在步长查找表中的索引移动方向和距离，更新currentIndex，并据此获取新的量化步长ss2(n)。这确保了解码过程中的自适应性，能够根据信号变化动态调整量化精度。

### 2.3.4信号重构与输出：

随着每一步的迭代，解码得到的预测值序列逐步逼近原始信号的采样值。为了保持数值一致性，预测值y被限制在[-127, 127]的范围内。最后，将解码后的信号序列除以128，逆向缩放回原始信号的幅度范围，完成信号的完全恢复。

综上所述，APDCM解码算法通过逆向执行编码过程中的差分计算、自适应量化步长调整和预测更新，结合对符号位的正确解读，有效地从编码比特流中重构出原始信号，体现了自适应、逆向工程和信号预测的核心思想。

2.2 APDCM解码函数设计步骤

### 2.4.1 初始化解码器状态：

获取编码比特流code的长度len，初始化输出信号y为全零向量，长度与code相同。初始化量化步长序列ss2，其中第一个量化步长ss2(1)设为1。设置当前量化步长索引currentIndex初值为1，量化步长调整索引数组index为[-1 4]，表示根据编码比特决定量化步长是减小还是增加。定义量化步长的起始值startval=1，结束值endval=127，用于构建量化步长查找表。计算基础步长base，用于生成量化步长的对数分布。

### 2.4.2 构建量化步长查找表：

计算量化步长查找表table2的大小numSteps，确保覆盖从startval到endval的步长范围。重新计算base，使其在量化步长间隔中均匀分布。生成量化步长序列，通过const\*base.^n计算并四舍五入得到实际量化步长值，存储于table2中。

### 2.4.3 解码循环：

遍历编码比特流code的每一个元素，从第二个元素开始（索引为2）。判断当前编码比特是否包含符号位信息，若包含则分离出实际的量化值temp。计算量化误差temp2，即量化值乘以前一时刻的量化步长ss2(n-1)，考虑符号位影响。更新当前输出样本值y(n)，将其设置为前一时刻的样本值加上量化误差，并限制在[-127, 127]范围内，以避免溢出。根据当前量化值调整量化步长索引currentIndex，确保其在有效范围内（1到numSteps）。更新当前量化步长ss2(n)，从查找表table2中根据索引currentIndex获取。

### 2.4.4 输出信号归一化：

解码完成后，将解码信号y除以128进行归一化，恢复原始信号的大致幅度范围。

通过以上步骤，adpcm\_decoder函数实现了从经过ADPCM编码的比特流code中，利用给定的符号位信息sign\_bit，恢复出原始信号y的过程。该过程依赖于对量化误差的累积和自适应调整量化步长，以适应信号动态范围的变化。

3.编程实现

这段MATLAB代码实现了一个完整的ADPCM（自适应差分脉冲编码调制）编解码实验流程，用于处理音频文件"C6\_3\_y.wav"。下面是详细的步骤解析：

### 3.1 初始化与环境设置

* 使用clear all, clc, close all命令清除工作区变量、清空命令窗口、关闭所有图形窗口，为实验准备干净的环境。
* 读取音频文件：[x,fs,numbits]=wavread('C6\_3\_y.wav')，其中x是音频信号的采样值数组，fs是采样频率，numbits是每个采样点的比特数（虽然在这个实验中未直接使用numbits）。

### 3.2 定义ADPCM编码参数

* 设置sign\_bit=2，表示使用两位来表示符号位，这是实验设定的特殊需求，通常ADPCM中符号位为1位。

### 3.3 执行ADPCM编码

* 调用自定义函数ss=adpcm\_encoder(x,sign\_bit)对音频信号x进行ADPCM编码，得到编码后的比特流ss。编码过程包括差分计算、量化、自适应步长调整等步骤。

### 3.4 执行ADPCM解码

* 对编码后的比特流ss进行解码，调用yy=adpcm\_decoder(ss,sign\_bit)'，这里的转置操作'是为了使解码后的信号与原始信号形状一致，便于后续的比较和显示。

### 3.5 计算信号质量指标

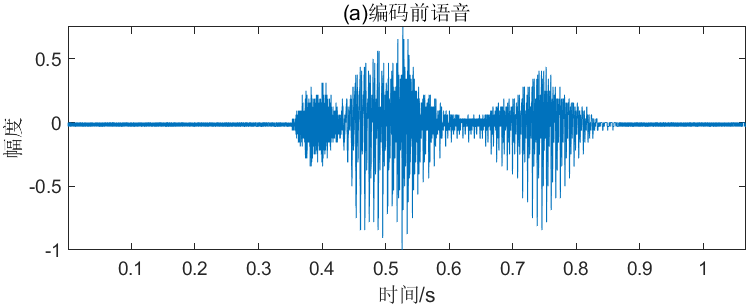
* 计算均方误差（MSE）：nq=sum((x-yy).\*(x-yy))/length(x)，衡量原始信号x与解码信号yy之间的差异。
* 计算解码信号的平均功率：sq=mean(yy.^2)。
* 计算信噪比（SNR）：snr=(sq/nq)，反映信号重构的质量。
* 计算SNR的分贝值：snrq=10\*log10(mean(snr))，便于直观理解。

### 3.6 绘制图形结果

* 使用subplot命令创建一个2行1列的图像布局，分别绘制原始信号和解码后的信号。
  + 第一个子图显示编码前的语音信号，使用plot(t,x/max(abs(x)))，其中t=(1:length(x))/fs计算时间轴，axis tight保持坐标轴紧凑，标题和轴标签说明了图示内容。
  + 第二个子图显示解码后的语音信号，同理使用plot(t,yy/max(abs(yy)))，展示了重构信号的波形。
* 最后，可以通过disp(['SNR(dB):', num2str(snrq)])的命令打印出信噪比的分贝值。

4.调试结果

4.1 编码前语音



4.2 编码后语音

图表, 折线图, 直方图

描述已自动生成

5.源代码

5.1 main

clear all;

clc;

% 使用audioread读取音频文件

[x, fs] = audioread('C6\_3\_y.wav');

% 使用audioinfo读取音频文件信息

info = audioinfo('C6\_3\_y.wav');

numbits = info.BitsPerSample;

sign\_bit=2; % 两位ADPCM算法

ss=adpcm\_encoder(x,sign\_bit);

yy=adpcm\_decoder(ss,sign\_bit)';

% 计算均方误差

nq=sum((x-yy).^2)/length(x);

% 计算信号能量

sq=mean(yy.^2);

% 计算信噪比

snr=(sq/nq);

% 时间轴

t=(1:length(x))/fs;

% 绘制图形

subplot(211)

plot(t,x/max(abs(x)))

axis tight

title('(a)编码前语音')

xlabel('时间/s')

ylabel('幅度')

subplot(212)

plot(t,yy/max(abs(yy)))

axis tight

title('(b)解码后语音')

xlabel('时间/s')

ylabel('幅度')

% 计算并显示量化信噪比

snrq=10\*log10(mean(snr));

disp(['Quantization SNR: ', num2str(snrq), ' dB']);

5.2 APDCM编码函数

% APDCM编码函数

function code=adpcm\_encoder(x,sign\_bit)

x = x\*128;

len = length(x);

% 生成步长查找表

index = [-1 4];

currentIndex = 2;

startval = 1; % 大于 1

endval = 127; % 大于startval且小于128- 1

base = exp( log(2)/8 ); % 间隔在(1,2)间

% 近似步长

const = startval/base;

numSteps = round( log(endval/const) / log(base) );

n = 1:numSteps;

base = exp( log(endval/startval) / (numSteps-1) );

const = startval/base;

table2 = round( const\*base.^n );

ss = zeros(1,len);

ss(1) = table2(1);

z = zeros(1,len);

code = zeros(1,len);

neg = 0;

for n = 2:len

d(n) = x(n) - z(n-1);

if (d(n) < 0)

neg = 1;

code(n) = code(n) + sign\_bit;

d(n) = -d(n);

else

neg = 0;

end

if (d(n) >= ss(n-1))

code(n) = code(n) + 1;

end

%计算量化距离

if (neg)

temp = code(n) - sign\_bit;

else

temp = code(n);

end

temp2 = (temp+.5)\*ss(n-1);

if (neg)

temp2 = -temp2;

end

z(n) = z(n-1) + temp2;

if (z(n) > 127)

z(n) = 127;

elseif (z(n) < -127)

z(n) = -127;

end

% 计算新的步长

temp = temp + 1;

currentIndex = currentIndex + index(temp);

if (currentIndex < 1)

currentIndex = 1;

elseif (currentIndex > numSteps)

currentIndex = numSteps;

end

ss(n) = table2(currentIndex);

end

5.3 APDCM解码函数

% APDCM解码函数

function y=adpcm\_decoder(code,sign\_bit)

len=length(code);

y = zeros(1,len);

ss2 = zeros(1,len);

ss2(1) = 1;

currentIndex =1;

index = [-1 4];

startval = 1;

endval = 127;

base = exp( log(2)/8 );

% 近似步长

const = startval/base;

numSteps = round( log(endval/const) / log(base) );

n = 1:numSteps;

base = exp( log(endval/startval) / (numSteps-1) );

const = startval/base;

table2 = round( const\*base.^n );

for n = 2:len

% 计算量化距离

neg = code(n) >= sign\_bit;

if (neg)

temp = code(n) - sign\_bit;

else

temp = code(n);

end

temp2 = (temp+.5)\*ss2(n-1);

if (neg)

temp2 = -temp2;

end

y(n) = y(n-1) + temp2;

if (y(n) > 127)

y(n) = 127;

elseif (y(n) < -127)

y(n) = -127;

end

% 计算新的步长

temp = temp + 1;

currentIndex = currentIndex + index(temp);

if (currentIndex < 1)

currentIndex = 1;

elseif (currentIndex > numSteps)

currentIndex = numSteps;

end

ss2(n) = table2(currentIndex);

end

y = y/128;