哈爾濱工業大學

实验报告

实 验（二）

题　　目 DPCM 编解码实验

专　　业 视听觉信息处理

学　　号 1180300419

班　　级 1803106 班

学　　生 刘晓慧

指 导 教 师 郑铁然

实 验 地 点 格物楼 207

实 验 日 期 2020 年 12 月 23 日星期三

计算机科学与技术学院

一、 8 比特 DPCM 编解码算法

1.1 简述算法内容

算法思想：

DPCM 即差分 PCM 编码。因为语音信号存在冗余性，即语音信号相邻采样间表现出很强的相关性。因此，对相邻采样值的差值进行编码可以大大减少每个采样点的编码比特数。

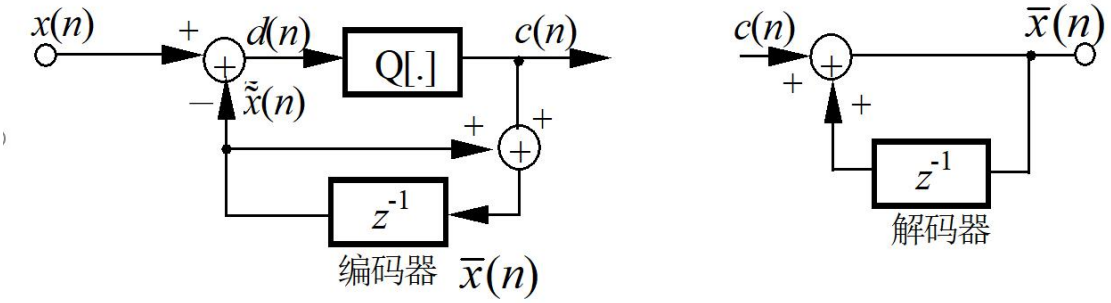
算法实现：

DPCM 包括编码器和解码器两个部分。因为编码器中包含一个解码器，因此下面首先介绍解码器的内容：

解码器的内容：编码器编码之后的数据为 $c(n)$,则解码器端利用$\widehat{x(n)} = x(\widehat{n-1}) + c(n)$，即可得到对原始信号的预测值。

编码器的内容：为了防止造成误差值的累积，因此在编码器端利用：$d(n) = x(n) - x(\widehat{n-1})$，$c(n) = d(n) + e(n)$和 $d(1) = \widehat{x(1)} = x(n)$来进行编码，即在编码器端包含了一个解码器，通过该方法防止量化误差的累积。

该算法需要注意的是量化方法的选取，较简单的方法是采取直接量化方法，该方法较简单，但是量化噪声会比较大（以 8 比特量化为例:可以表示的 dn 的范围是 -128 到 127，因此超出该范围的 dn 无法准确量化，带来很大的量化误差），因此解码之后的语音质量不好；相对比较复杂的量化方法可以采用量化因子法，该方法的思想是：找出误差分布的主要范围，将该范围均分成等长的小区间；之后，每个区间映射到一个 cn 值，解码的时候，只需要做相应的逆映射，则可以大致还原 dn。该方法的缺点是：即使 dn 值很小，也会带来量化误差；优点是大大增加了可以量化的 dn 的范围，实验时观察到通过合理选择量化因子，该方法编码的语音质量相对于直接量化有很大的提高。



实际 DPCM 结构图

1.2 解码信号的信噪比

信噪比也叫做 SNR。指一个电子设备或者电子系统中原始信号与噪声的能量的比值。原始信号指原始输入信号，噪声指原始信号与其经过电子设备或者电子系统处理之后得到的输出信号的差值。数学表达式为

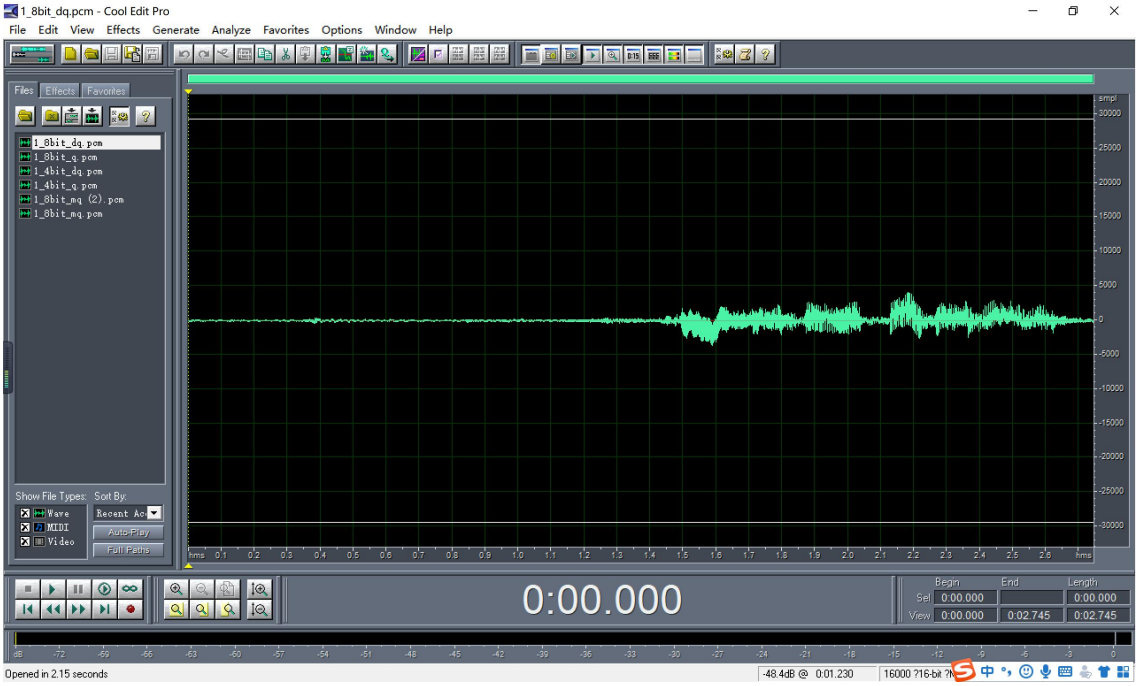
$SNR = 10 * \log\left\{\frac{\sum_{n=0}^{M}(s(n))^2}{\sum_{n=0}^{M}(s(n)-\hat{s}(n))^2}\right\}$，单位是分贝。信噪比可以反映输出信号的质量，信噪比越大，表示输出信号越接近原始输入信号。一般信噪比是 30db 以上，则表示输出信号质量比较好。

8 比特直接量化得到的解码信号的信噪比是: 0.16834839927412204 分贝

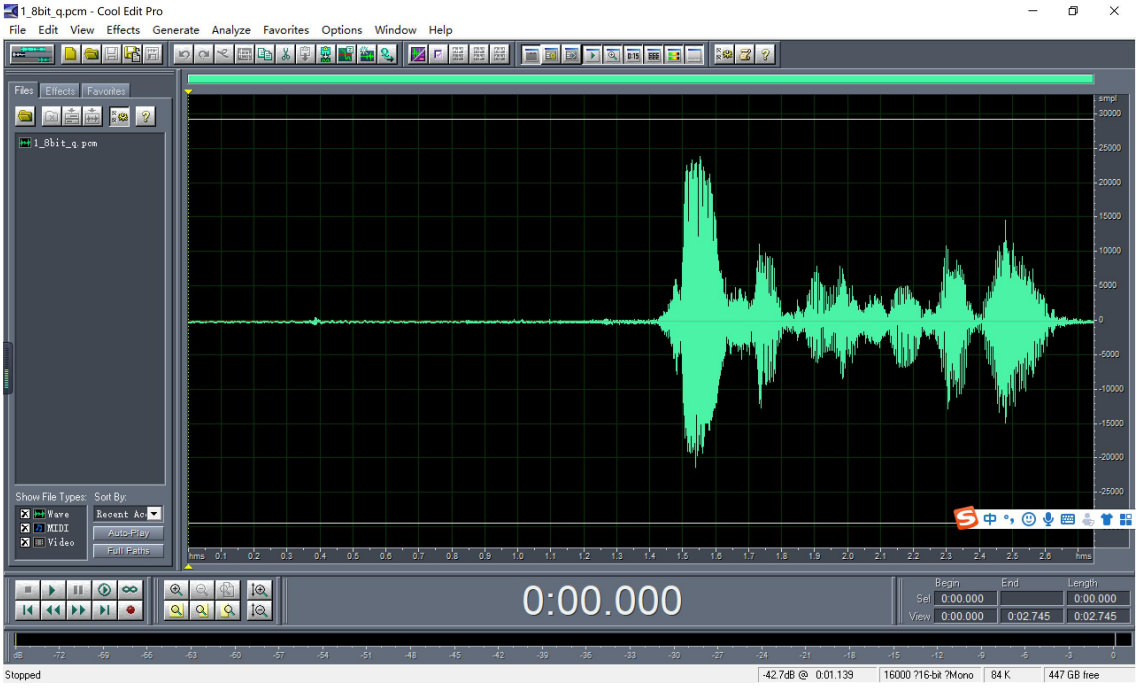
8 比特量化因子法得到的解码信号的信噪比是：25.086460049073406 分贝

虽然这两种量化方法得到的信噪比在数值上有很大差异，并且在人耳感觉和波形图上，可以明显地感觉到量化因子法得到的解码信号和原始信号更加接近。

8 比特直接量化，解码之后的波形图：



8 比特量化因子法，解码之后的波形图



二、4 比特 DPCM 编解码算法

2.1 你所采用的量化因子

量化因子： a = 370

选取方法：首先绘制出 dn 的密度分布，发现大部分数值都分布在[-2000, 2000]范围内。并且 4bit 量化可以选取的数值仅有{-8,-7,...,7}一共 16 个数字，因此 4000 这个范围可以被均分成 15 个区间，一个区间的大小大约是 266，因此在该值附近，结合 SNR 和人耳感觉到的声音质量，适当的改变 a，最终决定选取 a=370。

2.2 拷贝你的算法，加上适当的注释说明

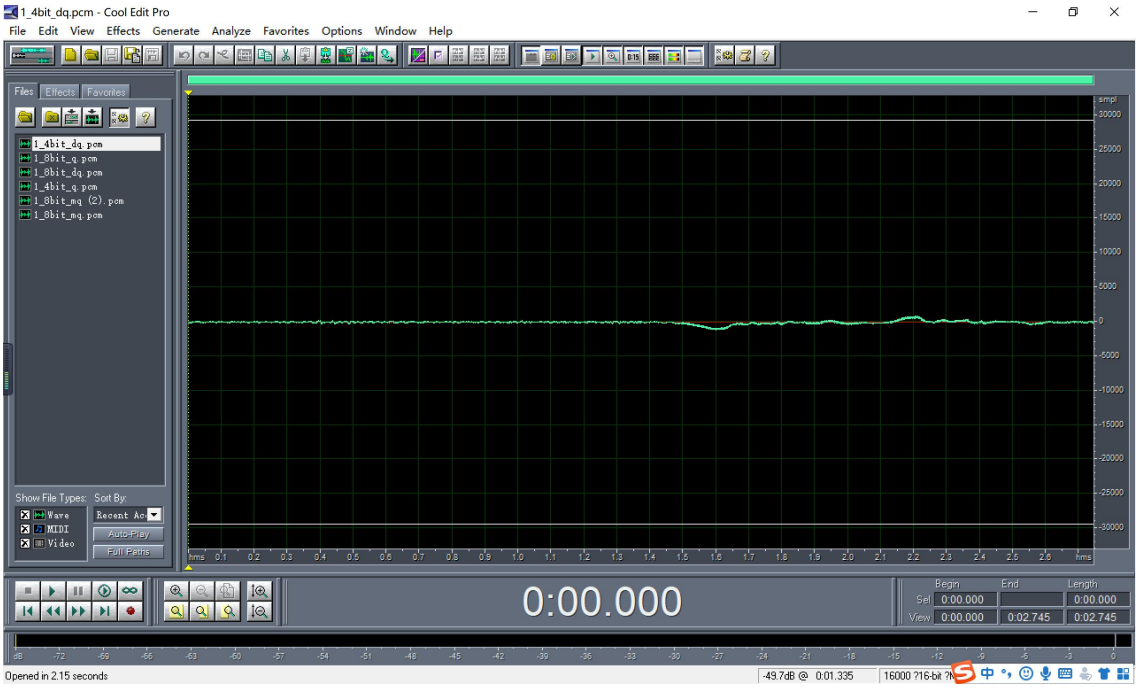
```
import wave
import numpy as np
import struct

class DPCM_4_Q:
    """
    '''
    该函数读取 filename 表示的 wave 文件，并且返回所有采样点组成的数组
    '''
    def read_data(self, filename):
        f = wave.open(filename, "rb")
        params = f.getparams()  # 获得文件的格式信息
        nchannels, sampwidth, framerate, nframes = params[:4]  # 依次获得各参数
        # 输出一些辅助参数
        print('该文件的声道数是:', nchannels)
        print('该文件中样本点占用的字节数是:', sampwidth)
        print('该文件的采样频率是:', framerate)
        print('该文件的采样个数是:', nframes)
        # 获得采样点的数据
        str_data = f.readframes(nframes)
        f.close()
        # 将波形数据转换成数组
        # 需要根据声道数和量化单位，将读取的二进制数据转换为一个可以计算的数组
        wave_data = np.frombuffer(str_data, dtype='i2')
        return wave_data
    '''
    wave_data:表示要进行编码的波形数据
    bits_digit:表示压缩编码之后，用多少比特
    a:量化参数取值
    该函数范围编码后的数据和 dn
    '''
    def encoder(self, wave_data, bits_digit, a):
        ans_data = []  # 用来存储编码后的数据
```

```
        dnn = []  # 辅助我们进行 a 的选取
        flag = False  # 用来判断是不是第一个待编码数据
        pre_data = 0  # 译码器需要用到的数据
        for data in wave_data:
            if not flag:
                ans_data.append(data)
                pre_data = data
                dnn.append(data)
                flag = True
            else:
                dn = data - pre_data
                dnn.append(dn)
                cn = self.quantitative(dn, bits_digit, a)
                pre_data = pre_data + cn * a - a // 2  # 更新该值
                ans_data.append(cn)  # 存储的是量化之后的数据
        return ans_data, dnn
    '''
    将 encoded_data 表示的数据写入 filename 表示的文件中
    '''
    def write_encoded_data(self, filename, encoded_data):
        f = open(filename, 'wb')
        f.write(struct.pack('h', encoded_data[0]))  # 转换为字节流
        for i in range(1, len(encoded_data) // 2 + 1):
            f.write(struct.pack('B', ((encoded_data[2 * i - 1] + 8) << 4) | (encoded_data[2 * i] + 8)))
        f.close()
    '''
    从 filename 表示的文件中读取编码之后的数据，返回差分数组
    '''
    def read_encoded_data(self, filename):
        ans_data = []
        with open(filename, 'rb') as f:
            ans, = struct.unpack('h', f.read(2))
            ans_data.append(ans)
            while True:
                tmp_data = f.read(1)
                if not tmp_data: break
                ans, = struct.unpack('B', tmp_data)
                ans_data.append((ans // 16) - 8)
                ans_data.append((ans % 16) - 8)
        return ans_data
    '''
    量化函数
    i：表示要量化的数字
    bits_digit:表示量化之后的位数
    a:表示量化因子
    '''
    def quantitative(self, i, bits_digit, a):
        high_level = np.power(2, bits_digit - 1) - 1
        low_level = -high_level - 1
        if i >= high_level * a:
            return high_level
        elif i <= low_level * a:
            return low_level
        floor = np.int(np.floor(np.abs(i) / a))
        ceil = np.int(np.ceil(np.abs(i) / a))
        if i >= 0: return ceil
```

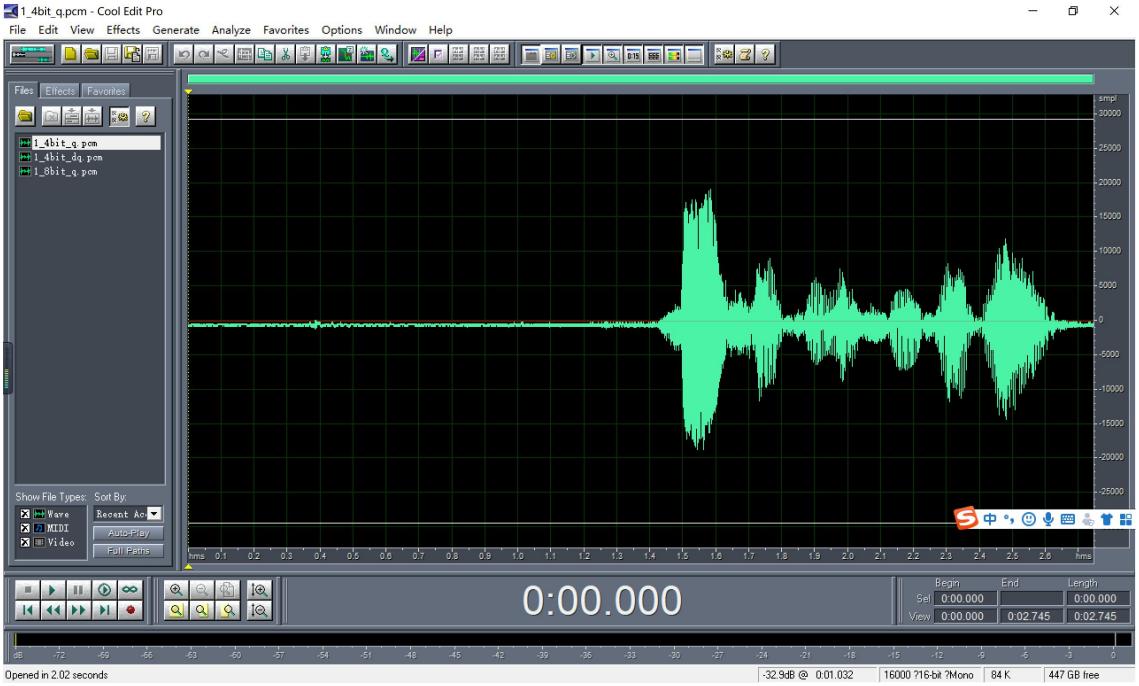
```
            else: return -floor
        '''
        encoded_data：表示待解码数据组成的数组,
        返回解码之后的数据
        '''
        def decoder(self, bits_digit, encoded_data, a):
            length = len(encoded_data)
            ans_data = np.zeros(length)
            pre_data = 0
            for i in range(length):
                ans_data[i] = pre_data + encoded_data[i] * a - a // 2
                pre_data = ans_data[i]
            ans_data = ans_data.astype(np.int16)
            return ans_data
        '''
        将解码后的数据 decoded_data 写进 filename 表示的文件中
        '''
        def write_decoded_data(self, filename, decoded_data):
            f = open(filename, 'wb')
            for i in range(len(decoded_data)):
                f.write(decoded_data[i])
            f.close()

my_dpcm = DPCM_4_Q()

# 编码步骤
a = 370
wave_data = my_dpcm.read_data('../data/1.wav')
encoded_data, dnn = my_dpcm.encoder(wave_data, 4, a)
import matplotlib.pyplot as plt
ans = []
# 绘制 dnn 的密度分布函数，如果不限制其范围是[-2000,2000]则绘制出的图像比较分散，不易观察
# 2000 是通过不断尝试选取的一个比较合适的范围
for i in range(len(dnn)):
    if np.abs(dnn[i]) <= 2000:
        ans.append(dnn[i])
plt.hist(ans, bins=50, color='steelblue')
plt.show()
my_dpcm.write_encoded_data('../results/encoded_data_4bits_q.dpc', encoded_data)
# 解码步骤
encoded_data = my_dpcm.read_encoded_data('../results/encoded_data_4bits_q.dpc')
decoded_data = my_dpcm.decoder(4, encoded_data, a)
my_dpcm.write_decoded_data('../results/decoded_data_4bits_q.pcm', decoded_data)
from SNR import *
snr = SNR(wave_data, decoded_data)
print('4 比特量化因子法得到的信噪比是: ', snr, '分贝')
```

2.3 解码信号的信噪比

4 比特量化因子法得到的信噪比是： 10.66459892122966 分贝
该信噪比优于 4 比特直接量化的-0.017762009289321784 分贝
4 比特直接量化，解码之后的波形图：



4 比特量化因子法，解码之后的波形图：



三、改进策略

3.1 你提出了什么样的改进策略，效果如何

改进策略：

改进主要集中在量化器上，具体方式是：

在 8 比特中，选取最高位表示是精确存储还是近似存储，如果待量化的数值属于[-32,31],则可以精确存储，此时最高位置是 0，否则需要进行近似存储，最高位是 1；

次高位表示量化的数值是正数还是负数，如果是正数则次高位是 0，如果是负数，则次高位是 1

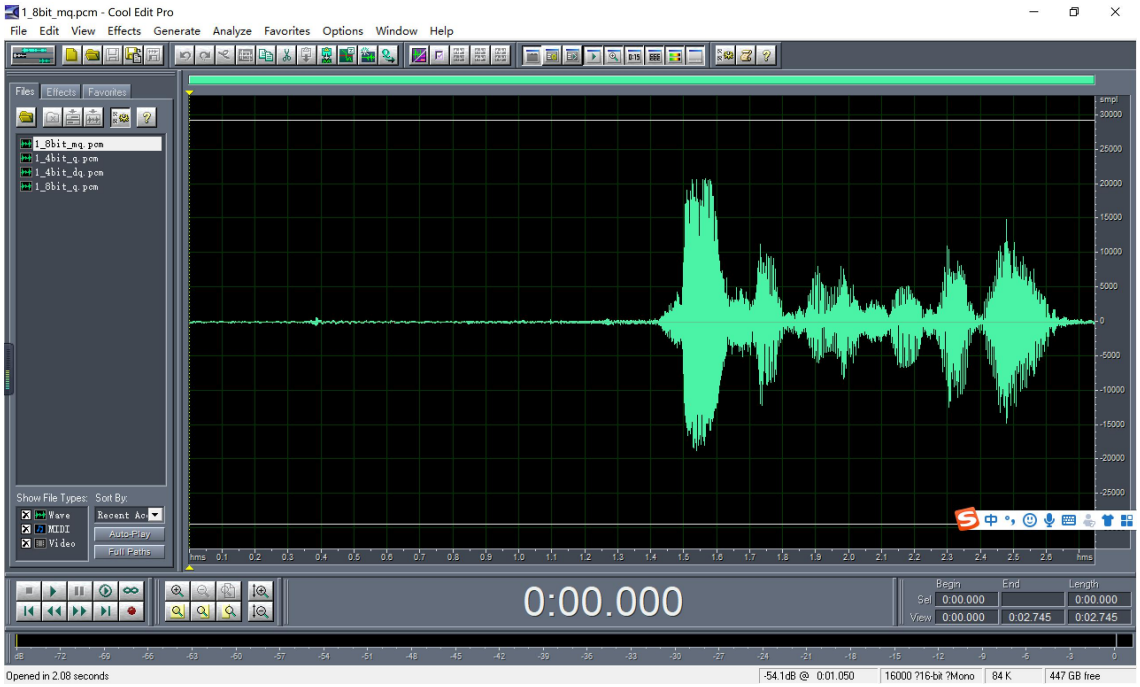
剩余 6 比特存储的信息根据最高位不同而有所差别。如果最高位是 0，即表示精确存储，则剩余 6 比特存储的是待量化数值的绝对值，如果最高位是 1，即表示近似存储，则剩余 6 比特存储的是待量化数值 i 的 round（sqrt（abs（i））））

解码器解码时，依次读取这些位的信息，按照其含义进行解码即可。

改进之后的效果：

8 比特量化因子法得到的信噪比是: 16.396514876974596 分贝

解码之后的波形如图所示，语音效果较好。



四、 简述你对量化误差的理解

4.1 什么是量化误差？

百度百科中对量化误差的定义所示:量化误差是指由于对模拟信号进行量化而产生的误差。

在本次实验中，量化误差是指由于用来存储 dn 的空间（1 字节或者 4 比特）小于精确存储 dn 所需要的空间，因此导致量化之后的值 cn 和原始值 dn 之间的误

差。

4.2 为什会编码器中会有一个解码器

为了防止解码信号中产生误差的累积。

分析过程如下：如果在编码器端没有解码器，则编码器端只能通过 $dn = x(n) - x(n-1)$ ， $c(n) = d(n) + e(n)$ 来计算误差值；而解码器端仍旧通过 $\hat{x}(n) = \hat{x}(n-1) + c(n)$ 进行解码，则：

$$\hat{x}(n) = \hat{x}(n-1) + c(n)$$
$$= \hat{x}(n-1) + d(n) + e(n)$$
$$= \hat{x}(n-1) + x(n) - x(n-1) + e(n)$$

则有 $\hat{x}(n) - x(n) = \hat{x}(n-1) - x(n-1) + e(n)$

即 $\hat{x}(n) = x(n) + \sum_{m=1}^{n} e(m)$

由上面的分析可以明显的看到误差具有累积效应，想要消除该误差，可以采用的方法有：将 $x(n-1)$ 传递到解码器或者在编码器中包含一个解码器，利用 $\mathrm{dn} = x(n) - \hat{x}(n-1)$ 计算误差。方法一需要很大的网络带宽，和我们压缩编码的初衷相违背，因此采用第二种方法，该方法可以有效避免解码器端误差的累积效应，并且不会增加网络传输的负担。

五、 总结

5.1 请总结本次实验的收获

1、对 DPCM 的理解更加深刻，掌握更加牢固。
2、熟练了如何在 Python 中进行位操作。
3、学会了如何在理论上估算量化因子，以及如何借助信噪比来定量地衡量量化因子选取的质量。

5.2 请给出对本次实验内容的建议

实验设计很好，做了之后收获很大。

六、附录

此打包文件中：
code 文件夹下存储代码源码：
DPCM_4_DQ:表示使用 4 比特直接量化法进行 DPCM 编解码
DPCM_4_Q:表示使用 4 比特量化因子法进行 DPCM 编解码
DPCM_8_DQ:表示使用 8 比特直接量化法进行 DPCM 编解码
DPCM_8_Q:表示使用 8 比特量化因子法进行 DPCM 编解码
MY_DPCM:表示使用 8 比特自己设计的量化器进行 DPCM 编解码
data 文件夹下存储本次实验使用的语料：
1.wav:表示本次实验使用的语料文件
results 文件夹下存储本次实验的结果数据：
1_8bit_q.dpc:表示使用 8 比特量化因子法，编码后的数据
1_8bit_q.pcm:表示使用 8 比特量化因子法，解码后的数据
1_8bit_dq.dpc:表示使用 8 比特直接量化法，编码后的数据
1_8bit_dq.pcm:表示使用 8 比特直接量化法，解码后的数据
1_4bit_q.dpc:表示使用 4 比特量化因子法，编码后的数据

l_4bit_q.pcm:表示使用 4 比特量化因子法，解码后的数据

l_4bit_dq.dpc:表示使用 4 比特直接量化法，编码后的数据

l_4bit_dq.pcm:表示使用 4 比特直接量化，解码后的数据

l_8bit_mq.pcm:表示使用 8 比特自己设计的量化器，解码后的数据

l_8bit_mq.dpc:表示使用 8 比特自己设计的量化器，编码后的数据