哈爾濱工業大學

Harbin Institute of Technology

视听觉信号处理
实验报告

DPCM 编解码实验

课程名称：	视听觉信号处理
学院：	计算学部
专业：	计算机科学与技术
学号：	**1180300811**
姓名：	孙骁
实验地点：	格物 **207**
指导老师：	郑铁然
学期：	**2020** 秋季学期

2020 年 **12** 月 **23** 日

目录

一、 实验内容与步骤

1.1 DPCM 编解码算法

1.1.1 编码算法

1. 从 wav 文件中读入原始数据；
2. 与解码后的 $\bar{x}(n-1)$ 计算差值；
3. 对从 $d(n)$ 进行重新量化，要求 8bit，一个采样一个 BYTE；
   直接量化或量化因子法（推荐）；
4. 解码；
5. 对从 $d(n)$ 进行重新量化，要求 4bit，两个采样一个 BYTE；
   (a) 直接量化；
   (b) 设计一个因子 a；
   (c) 对数变换。
6. 解码计算 $\bar{x}(n-1)$
   (a) 量化因子法：$\bar{x}(n)=\bar{x}(n-1)+(c(n)-8)\times a$；
   (b) 对数变换法：$\bar{x}(n)=\bar{x}(n-1)+(-1)^{\text{sgn}(c(n)-8)}\times a^{c(n)\&7}$
7. 封装 $c(n)$；
8. 保存编码到文件（后缀用”.dpc”）。

1.1.2 解码算法

解码还原后文件，保存为 pcm 文件。

1.2 改进思路

请提出你的若干改进策略（可聚焦于 8bit 量化编码进行改进）。

1.3 计算信噪比

二、 实验环境

1. Anaconda 4.8.4
2. Python 3.7.4
3. PyCharm 2020.3 (Professional Edition)
4. Windows 10 2004

三、 实验原理

3.1 DPCM 编解码算法

由于相邻采样值之间的差值远小于采样值本身，可以对差值进行编码，而不是对采样值本身进行编码，这就是差分脉冲编码 DPCM。

产生差分信号的过程：直接存储前一次的采样值，用本次的采样值计算差值，经过量化得到数字语音编码。解码端做相反处理，恢复原信号。用 Z 变换考查各点信号的时域关系，有

$$C(z) = X(z)\left(1 - z^{-1}\right) + E(z) \tag{1}$$

和

$$\bar{X}(z) = \frac{C(z)}{1 - z^{-1}} = X(z) + \frac{E(z)}{1 - z^{-1}}. \tag{2}$$

式中，$E(z)$ 为量化器量化噪声 $e(n)$ 的 Z 变换。

由于量化器的量化噪声被累积叠加到了输出信号中，即每次的量化噪声信号都被保存，叠加到下次的输出当中，如果量化噪声始终是同一方向，则输出信号会越来越偏离原信号。因此，编码器应采用前一次解码后的采样值代替前一次的输入采样值，以生成差分信号。

则

$$\widetilde{X}(z) = \frac{C(z)\, z^{-1}}{1 - z^{-1}}. \tag{3}$$

编码结果为

$$C(z) = X(z) - \widetilde{X}(z) + E(z), \tag{4}$$

代入后有

$$C(z) = (X(z) + E(z))\left(1 - z^{-1}\right). \tag{5}$$

因此有

$$\bar{X}(z) = \frac{C(z)}{1 - z^{-1}} = X(z) + E(z), \tag{6}$$

由此可见，已经消除了量化噪声的累积。

DPCM 仅仅用到了两个采样值之间的相关性。然而，当前输入采样不仅与上一时刻的采样值相关，也与前面若干采样值相关，可以使用线性预测分析的方法实现一般形式的差分脉冲编码。根据线性预测分析的原理，可以用过去的一些采样值的线性组合来预测和推断当前的采样值，得到一组线性预测系数，且预测误差 $e(n)$ 的动态范围和平均能量均比信号 $x(n)$ 小得多，预测阶数越高，预测误差越小，相应的编码速率可以越低。

四、 实验步骤及相应结果

此部分内容与 word 模板相应章节对应，单击图片引用序号可以跳转查看相应图片结果。

4.1 8bits DPCM 编解码算法

当量化因子取 1 时，量化因子法即为直接量化法，故在算法介绍时不分开介绍，在计算信噪比时做区分。

4.1.1 简述算法内容

8bit 的编码算法，要求对 $d(n)$ 重新量化，一个采样一个 Byte。8 位能表示的无符号数的范围是 $(-128, 127)$，编码时采用取自然对数变换压缩编码，因此 8 位编码可表示的范围远大于信号的范围。

编码方法为，如果采样值为正数，则取自然对数的绝对值，转化为无符号 int 类型，保存为无符号 int 类型；如果采样值为负数，则取自然对数的绝对值，转化为无符号 int 类型，用 255 减去这个数，保存相减的数值。解码时，如果待解码数据范围为 $[0, 127)$，说明在编码前原数据是正数，直接做幂运算即可；如果待解码数据范围为 $[128, 255)$，说明在编码前原数据为负数，先用 255 减去待解码数据，再做幂运算。

编码的过程中，为了避免量化噪声叠加到输出信号中，采用边编码边解码的方法，即采用前一次解码后的采样值代替前一次的输入采样值生成差分信号。在编码过程中维护两个列表，code_data 保存编码后的数据，decode_data 保存解码的数据，以备下一位编码使用。

写入 dpc 文件时，按照二进制格式打开，直接将编码后的 8 bit 无符号数写入即可。从 dpc 文件读出解码时，以二进制的形式读取文件，每次读取一个字节，调用 struct.unpack 函数，按照 unsigned char 解码得到对应的十进制数，直接解码第一个数；对于其他数据，将解码后与前一个解压缩的数据相加即得到本位的解压数据。

重新写回 pcm 文件时，需要为 pcm 文件配置声道数，量化位数和采样频率，将解码得到的列表转化为二进制数据写入.pcm 文件中。

按照公式 (7) 计算信噪比。

$$\mathrm{SNR} = 10 \times \log_{10} \frac{\sum_{n=0}^{M} x^2(n)}{\sum_{n=0}^{M} \left(\bar{x}(n) - x(n)\right)^2} \tag{7}$$

4.1.2 解码信号的信噪比

8bit DPCM 编码算法使用直接量化法的信噪比为 15.225，采用量化因子为 23 的量化因子算法的信噪比为 13.199。

直接量化法解码后的波形如图 (1) 所示，量化因子算法解码后的波形如图 (2) 所示。

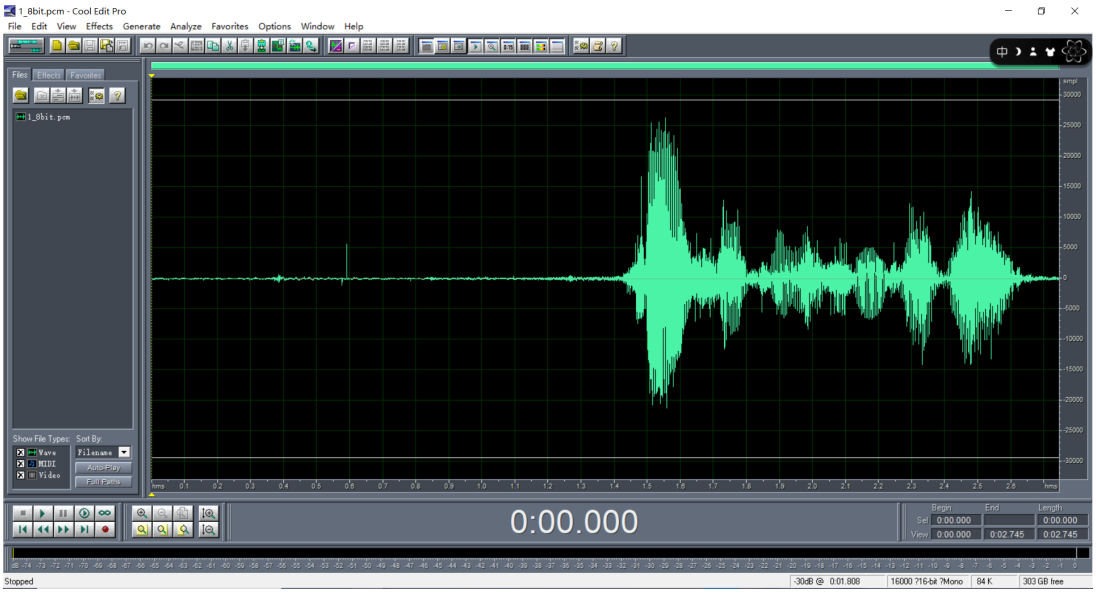

图 **1**　**8 bits DPCM** 编码直接量化法解码波形

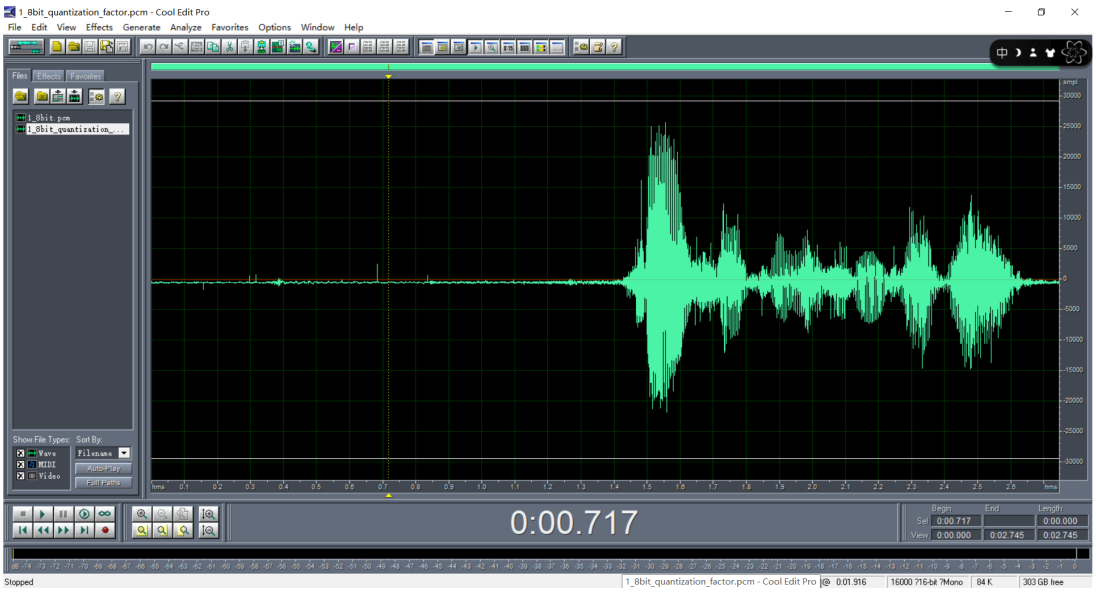

图 **2**　**8 bits DPCM** 编码量化因子法解码波形

4.2 4bits DPCM 编解码算法

4.2.1 你所采用的量化因子

量化因子为 23.

4.2.2 拷贝你的算法，加上适当的注释说明

```
def difference_pulse_coding_modulation_4(self):
    """
    4 bits 编码的主函数
    """
    print('DPCM 4bits')
    for key in self.wav_dic.keys():
        code_data = np.zeros_like(self.wav_dic[key][4]) # 编码的数据
        decode_data = np.zeros_like(self.wav_dic[key][4]) # 边编码边解码保存的临时数据
        length = len(self.wav_dic[key][4])
        code_data[0], decode_data[0] = self.coding_first_4(self.wav_dic[key][4][0]) #
            对第一位特殊处理，计算编码和解码数据
        for i in range(1, length):
            err = self.wav_dic[key][4][i] - decode_data[i - 1]
            code_data[i] = self.dpcm_4(err) # 计算编码数据
            decode_data[i] = decode_data[i - 1] - self.__quantization_factor * np.exp(
                15 - code_data[i]) if err < 0 else decode_data[i - 1] +
                    self.__quantization_factor * np.exp(
                code_data[i]) # 计算解码数据
        self.code_dic_4[key] = code_data # 将对应的文件存储到相应键值中
        # print(code_data)

def coding_first_4(self, data):
    """
    对第一位数据进行4 bits编码
    :param data: 第一位数据
    :return: 第一位数据编码结果与解码结果
    """
    if data < 0:
        code_data = 15 if data > -self.__quantization_factor else 15 - (
                np.uint8(int(abs(np.log(-data / self.__quantization_factor)) + 0.5) << 4)
                    >> 4) # 若采样值小于零
        code_data = max(8, code_data)
        return code_data, np.exp(15 - code_data) * -self.__quantization_factor
    else:
        code_data = 0 if data < self.__quantization_factor else np.uint8(
            int(abs(np.log(data / self.__quantization_factor)) + 0.5) << 4) >> 4 #
                若采样值大于零
        code_data = min(7, code_data)
        return code_data, np.exp(code_data) * self.__quantization_factor


def dpcm_4(self, data):
    """
    对其他数据进行4 bits编码
    :param data: 相应采样数据
    :return: 采样编码结果与解码结果
```

```
    """
    if data < 0:
        code_data = 15 if data > -self.__quantization_factor else 15 - (
                np.uint8(int(abs(np.log(-data / self.__quantization_factor)) + 0.5) << 4)
                    >> 4) # 若采样值小于零
        code_data = max(8, code_data)
        # print(code_data)
        return code_data
    else:
        code_data = 0 if data < self.__quantization_factor else np.uint8(
            int(abs(np.log(data / self.__quantization_factor)) + 0.5) << 4) >> 4 #
                若采样值大于零
        code_data = min(7, code_data)
        # print(code_data)
        return code_data
```

4.2.3 解码信号的信噪比

采用直接量化法的信噪比为 3.948，采用量化因子法的信噪比为 15.606。

直接量化法解码后的波形如图 (3) 所示，量化因子算法解码后的波形如图 (4) 所示。

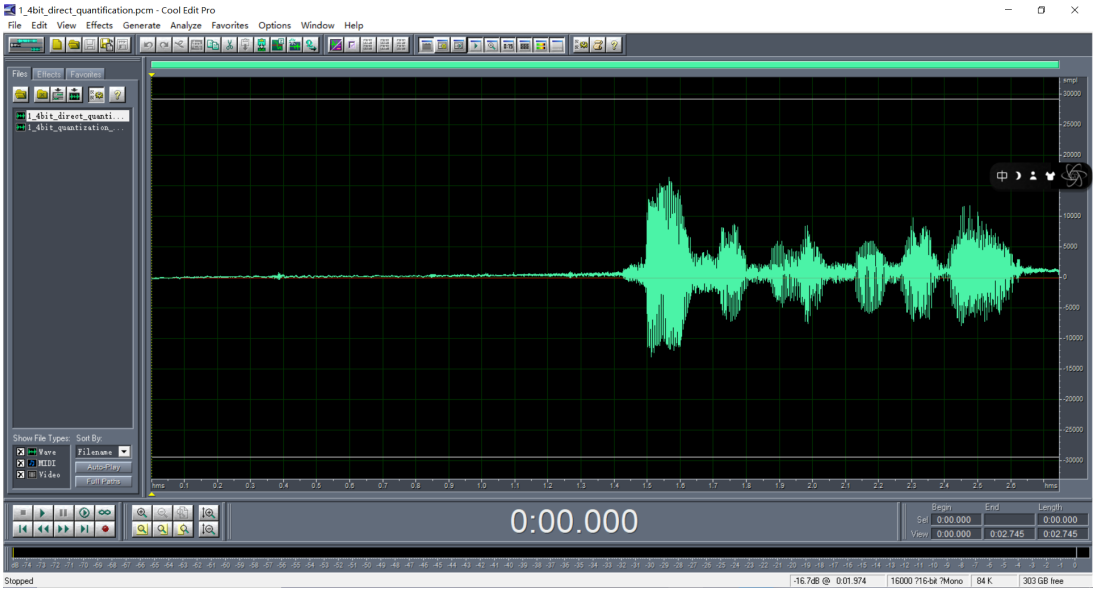

图 3　4 bits DPCM 编码直接量化法解码波形

4.3 改进策略

4.3.1 对采样值做对数变换

在章节4.1.1中，提到将原数据先取自然对数的绝对值再做编码，实际上最开始的实现是直接将原数据与编码后数据作差得到新的数据，但是在实际过程中发现舍入误差

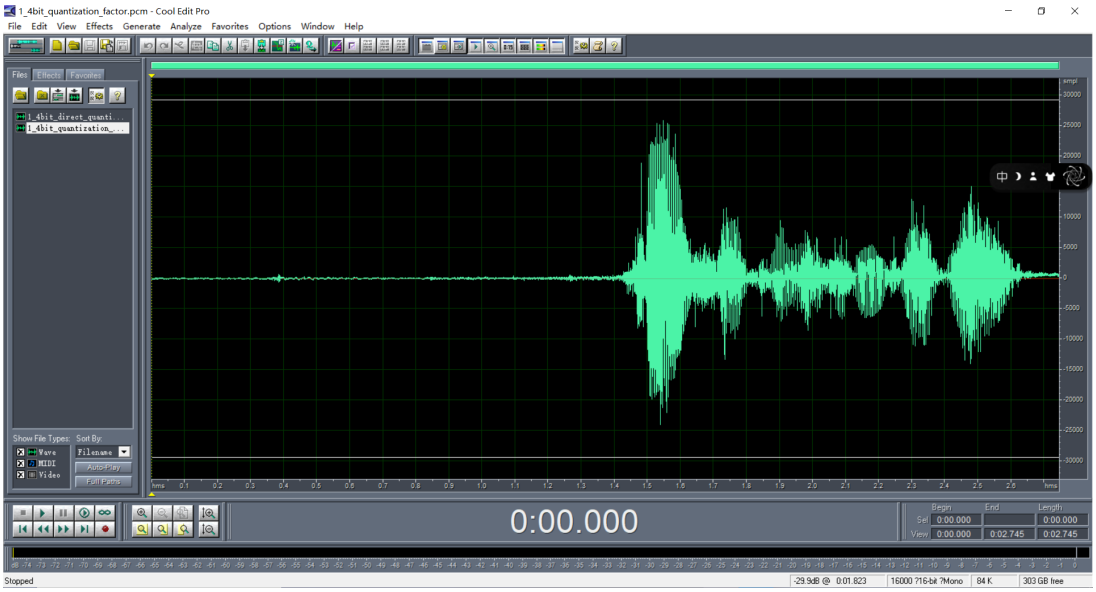

图 4　**4 bits DPCM** 编码量化因子法解码波形

过大，因此转为先取对数在计算差分，取对运算为严格的单调递增变换，所以不影响数据的变化趋势，好处是用较小的值表示了较大范围的数据。

修改后的编码和解码方式为

$$\begin{aligned} d(n) &= \log x(n) - \log \bar{x}(n) \\ \bar{x}(n) &= \bar{x}(n-1) + x(n) \times a \end{aligned} \tag{8}$$

其中，a 为量化因子，针对采样值为正数和负数的不同情况还要做不同处理。

采用量化因子编码，8 bit 编码修改前的信噪比为 5.432，修改后的信噪比为 13.199。

4.3.2 将小于量化因子的采样编码为固定值

在 8 bits 编码时，将绝对值小于量化因子的采样值直接编码为 0，这样可以使很多较小采样值的编码一致，在语音恢复后，也不会有较大的影响，同样也提高了信噪比。

4.3.3 可能的优化方向

如果有更多的时间，我会做更一般的 DCPM 算法，因为实验中实现 DPCM 仅仅用到了两个采样值之间的相关性。然而，当前输入采样不仅与上一时刻的采样值相关，也与前面若干采样值相关，可以使用线性预测分析的方法实现更一般形式的差分脉冲编码。根据线性预测分析的原理，可以用过去的一些采样值的线性组合来预测和推断当前的采样值，得到一组线性预测系数，且预测误差 $e(n)$ 的动态范围和平均能量均比信号 $x(n)$ 小得多，预测阶数越高，预测误差越小，相应的编码速率可以越低。

4.4 简述你对量化误差的理解

4.4.1 什么是量化误差

一般量化值都用二进制表示，如果用 m 个二进制数表示量化值，即量化字长，那么一般将幅度值划分为 2^m 个等分区间。从量化的过程中可以看出，信号在经过量化后，一定存在一个量化误差。其定义为

$$e(n) = \bar{x}(n) - x(n) \tag{9}$$

式中，$e(n)$ 为量化误差或者噪声，$\bar{x}(n)$ 为量化后的采样值，即量化器的输出，$x(n)$ 为未量化的采样值，即量化器的输入。

此外，在信号压缩存储的过程中，也会产生量化误差。

4.4.2 为什会编码器中会有一个解码器

在章节3.1的 DPCM 算法介绍中，已经对此部分做了解释。简言之，如果对编码信号做差分计算，会导致每次编码的量化误差都被保存下来，如果量化噪声始终是同一方向，则输出信号会越来越偏离原信号。因此，编码器应采用前一次解码后的采样值代替前一次的输入采样值，以生成差分信号。

五、 总结

5.1 请总结本次实验的收获

对 DPCM 编解码算法有了新的认识，对于不同的量化方法有了深刻的了解。

5.2 请给出对本次实验内容的建议

无。

参考文献

[1] 韩纪庆，张磊，郑铁然. 语音信号处理（第 3 版）. 清华大学出版社, 2019.

附录 A　从 wav 文件读取数据——read_wav_from_file.py

```python
import os
import wave
import numpy as np


def read_wav_from_file():
    file_dir_path = '../data/'
    file_list = os.listdir(file_dir_path)
    wav_data_dic = {}
    for file in file_list:
        file_path = os.path.join(file_dir_path, file)
        print("open wav file " + file_path)
        file_name = int(file.split('.', 1)[0])
        # print(file_name)
        with wave.open(file_path, "rb") as f:
            params = f.getparams()
            n_channels, samp_width, frame_rate, n_frames = params[:4]
            wav_data = f.readframes(n_frames)
            wave_data_short = np.frombuffer(wav_data, dtype=np.short)
            wav_data_dic[file_name] = [n_channels, samp_width, frame_rate, n_frames,
                wave_data_short]
    # print(wav_data_dic.keys())
    # print(bin(wav_data_dic[1][4][0] & 0b1111))
    return wav_data_dic


# def main():
#     read_wav_from_file()
#
#
# if __name__ == '__main__':
#     main()
```

附录 B　计算信噪比——calculate_snr.py

```python
import numpy as np


def calculate_snr(code_data, decode_data):
    molecular = np.longdouble(0)
    denominator = np.longdouble(0)
    for i in range(len(code_data)):
```

```
        # print(f'before: {code_data[i]}, after: {decode_data[i]}')
        molecular += pow(code_data[i], 2)
        denominator += pow(code_data[i] - decode_data[i], 2)
    snr = 10 * np.log10(molecular / denominator)
    print(f'SNR: {snr}')
    return snr
```

附录 C　编码核心代码——code_wav_file.py

```
from src.write_dpc_to_file import *
from src.read_wav_from_file import *
from src.calculate_snr import *
import numpy as np
import struct


class Codewavfile(object):
    def __init__(self, wav_dic):
        self.wav_dic = wav_dic
        self.__quantization_factor = 23
        self.code_dic_4 = {}
        self.code_dic_8 = {}

    def difference_pulse_coding_modulation_4(self):
        print('DPCM 4bits')
        for key in self.wav_dic.keys():
            code_data = np.zeros_like(self.wav_dic[key][4])
            decode_data = np.zeros_like(self.wav_dic[key][4])
            length = len(self.wav_dic[key][4])
            code_data[0], decode_data[0] = self.coding_first_4(self.wav_dic[key][4][0])
            for i in range(1, length):
                err = self.wav_dic[key][4][i] - decode_data[i - 1]
                code_data[i] = self.dpcm_4(err)
                decode_data[i] = decode_data[i - 1] - self.__quantization_factor * np.exp(
                    15 - code_data[i]) if err < 0 else decode_data[i - 1] +
                        self.__quantization_factor * np.exp(
                    code_data[i])
            self.code_dic_4[key] = code_data
            # print(code_data)

    def coding_first_4(self, data):
        """
        对第一位进行4 bits编码
        :param data: 第一位数据
```

```
        :return: 第一位数据编码结果与解码结果
        """
        if data < 0:
            code_data = 15 if data > -self.__quantization_factor else 15 - (
                  np.uint8(int(abs(np.log(-data / self.__quantization_factor)) + 0.5) <<
                       4) >> 4)
            code_data = max(8, code_data)
            return code_data, np.exp(15 - code_data) * -self.__quantization_factor
        else:
            code_data = 0 if data < self.__quantization_factor else np.uint8(
               int(abs(np.log(data / self.__quantization_factor)) + 0.5) << 4) >> 4
            code_data = min(7, code_data)
            return code_data, np.exp(code_data) * self.__quantization_factor

    def dpcm_4(self, data):
        if data < 0:
            code_data = 15 if data > -self.__quantization_factor else 15 - (
                  np.uint8(int(abs(np.log(-data / self.__quantization_factor)) + 0.5) <<
                       4) >> 4)
            code_data = max(8, code_data)
            # print(code_data)
            return code_data
        else:
            code_data = 0 if data < self.__quantization_factor else np.uint8(
               int(abs(np.log(data / self.__quantization_factor)) + 0.5) << 4) >> 4
            code_data = min(7, code_data)
            # print(code_data)
            return code_data

    def difference_pulse_coding_modulation_8(self):
        print('DPCM 8bits')
        for key in self.wav_dic.keys():
            code_data = []
            decode_data = []
            length = len(self.wav_dic[key][4])
            a, b = self.coding_first_8(self.wav_dic[key][4][0])
            code_data.append(a)
            decode_data.append(b)
            # print(code_data[0], decode_data[0])
            for i in range(1, length):
                err = self.wav_dic[key][4][i] - decode_data[i - 1]
                err2 = self.wav_dic[key][4][i] - code_data[i - 1]
                if err < 0:
                    code_i = 255 - np.uint8(abs(np.log(-err)) + 0.5)
                    code_i = np.uint8(128) if code_i < 128 else np.uint8(code_i)
                    code_data.append(code_i)
                    decode_data.append(decode_data[i - 1] - np.exp(255 * 1.0 -
```

```
                    code_data[i]))
            elif err2 == 0:
                code_data.append(np.uint8(0))
                decode_data.append(decode_data[i - 1])
            else:
                code_i = np.uint8(0.5 + abs(np.log(1e-5 + self.wav_dic[key][4][i] -
                    1.0 * decode_data[i - 1])))
                code_i = np.uint8(127) if code_i >= 128 else np.uint8(code_i)
                code_data.append(code_i)
                decode_data.append(decode_data[i - 1] + np.exp(1.0 * code_data[i]))
        self.code_dic_8[key] = code_data
        # for i in range(len(code_data)): print(code_data[i])

    def coding_first_8(self, data):
        code_data = 255 - np.uint8(abs(np.log(-data / self.__quantization_factor)) +
            0.5) if data < 0 else np.uint8(
            abs(np.log(data / self.__quantization_factor)) + 0.5)
        code_data = np.uint8(128) if code_data < 128 else np.uint8(code_data)
        if data < 0:
            # print(code_data, -np.exp(255 - int(code_data)) *
                -self.__quantization_factor)
            return code_data, -np.exp(255 - int(code_data)) * -self.__quantization_factor
        else:
            # print(f'2code data {code_data}, decode {np.exp(code_data) *
                self.__quantization_factor}')
            return code_data, np.exp(code_data) * self.__quantization_factor

    def dpcm_8(self, data):
        if data < 0:
            code_data = 255 - np.uint8(abs(np.log(-data)) + 0.5)
            code_data = np.uint8(128) if code_data < 128 else np.uint8(code_data)
            return code_data
        elif data == 0:
            code_data = np.uint8(data)
            return code_data

    def decode_4_bits(self):
        for key in self.code_dic_4.keys():
            file_name = f'../result/dpc/{key}_4bit.dpc'
            decode_four = []
            with open(file_name, 'rb') as f:
                num = -1
                while True:
                    data = f.read(1)
                    if not data: break
                    data = struct.unpack('B', data)
                    data = data[0]
```

```
                # print(data)
                if len(decode_four) == 0:
                    high_4 = data >> 4
                    low_4 = data & 15
                    # print(high_4, low_4)
                    read = self.__quantization_factor * np.exp(
                        high_4) if high_4 < 8 else -self.__quantization_factor *
                            np.exp(15 - int(high_4))
                    decode_four.append(read)
                    read = decode_four[0] + self.__quantization_factor * np.exp(low_4)
                         if low_4 < 8 else \
                        decode_four[0] - self.__quantization_factor * np.exp(15 -
                            int(low_4))
                    decode_four.append(read)
                    num += 2
                    # print(decode_four[0])
                else:
                    high_4 = data >> 4
                    low_4 = data & 15
                    # print(high_4, low_4)
                    # print(decode_four[num])
                    read = decode_four[num] + self.__quantization_factor *
                         np.exp(high_4) if high_4 < 8 else \
                        decode_four[num] - self.__quantization_factor * np.exp(15 -
                            int(high_4))
                    decode_four.append(read)
                    # print(read)
                    num += 1
                    # print(decode_four[num])
                    read = decode_four[num] + self.__quantization_factor *
                         np.exp(low_4) if low_4 < 8 else \
                        decode_four[num] - self.__quantization_factor * np.exp(15 -
                            int(low_4))
                    decode_four.append(read)
                    # print(read)
                    num += 1
            calculate_snr(self.wav_dic[key][4], decode_four)
            # print(decode_four)
            file_write_name = f'../result/pcm/{key}_4bit.pcm'
            with wave.open(file_write_name, 'wb') as f:
                f.setnchannels(1)
                f.setsampwidth(2)
                f.setframerate(16000)
                f.writeframes(np.array(decode_four).astype(np.short).tostring())

    def decode_8_bits(self):
        for key in self.code_dic_8.keys():
```

```
            file_name = f'../result/dpc/{key}_8bit.dpc'
            decode_eight = []
            with open(file_name, 'rb') as f:
                num = -1
                while True:
                    data = f.read(1)
                    if not data: break
                    data = struct.unpack("B", data)
                    data = data[0]
                    # print(data)
                    if len(decode_eight) == 0:
                        read = np.exp(data) if data < 128 else -np.exp(255 - int(data))
                        # print(read)
                        decode_eight.append(read)
                    else:
                        data_code = int(data) & int(np.uint8(128))
                        read = np.exp(data) if data_code == 0 else -np.exp(255 - int(data))
                        read += decode_eight[num]
                        # print(read)
                        decode_eight.append(read)
                    num += 1
            calculate_snr(self.wav_dic[key][4], decode_eight)
            file_write_name = f'../result/pcm/{key}_8bit.pcm'
            with wave.open(file_write_name, 'wb') as f:
                f.setnchannels(1)
                f.setsampwidth(2)
                f.setframerate(16000)
                f.writeframes(np.array(decode_eight).astype(np.short).tostring())


def main():

    wav_dic = read_wav_from_file()
    code_wav_file = Codewavfile(wav_dic)
    code_wav_file.difference_pulse_coding_modulation_4()
    code_wav_file.difference_pulse_coding_modulation_8()
    code_4_dic = code_wav_file.code_dic_4
    code_8_dic = code_wav_file.code_dic_8
    write_dpc_to_file(code_4_dic)
    write_8_dpc_to_file(code_8_dic)
    code_wav_file.decode_4_bits()
    code_wav_file.decode_8_bits()


if __name__ == '__main__':
    main()
```

附录 D　解码过程读取编码后的文件——read_dpc_from_file.py

```python
import os
import struct


def read_dpc_from_file():
    file_dir_path = '../result/dpc/'
    file_list = os.listdir(file_dir_path)
    wav_data_dic = {}
    for file in file_list:
        file_path = os.path.join(file_dir_path, file)
        print("open dpc file " + file_path)
        file_name = int(file.split('.', 1)[0])
        # print(file_name)
        dpc_data = []
        with open(file_path, "rb") as f:
            data = struct.unpack('h', f.read(2))
            dpc_data.append(data[0])
            while True:
                data_byte = f.read(1)
                if not data_byte:
                    break
                else:
                    data = struct.unpack('B', data_byte)
                    dpc_data.append(data[0])

    return wav_data_dic


def main():
    read_dpc_from_file()


if __name__ == '__main__':
    main()
```

附录 E　将编码后的结果写入文件——write_dpc_to_file.py

```python
import numpy as np

file_dir_path = '../result/dpc/'
suffix = '.dpc'

```

```
6
7  def write_dpc_to_file(code_dic):
8      for key in code_dic.keys():
9          print(f'write dpc code with 4 bits.')
10         file_name = '1_4bit' + suffix
11         with open(file_dir_path + file_name, 'wb') as f:
12             length = len(code_dic[key])
13             for i in range(0, length - 1, 2):
14                 data = (code_dic[key][i] << 4) + code_dic[key][i + 1]
15                 # print(np.uint8(data))
16                 f.write(np.uint8(data))
17             if length % 2 == 1:
18                 data = code_dic[key][length - 1] << 4
19                 f.write(np.uint8(data))
20
21
22 def write_8_dpc_to_file(code_dic):
23
24     for key in code_dic.keys():
25         print(f'write dpc code with 8 bits.')
26         file_name = '1_8bit' + suffix
27         with open(file_dir_path + file_name, 'wb') as f:
28             length = len(code_dic[key])
29             for i in range(length):
30                 f.write(code_dic[key][i])
```