哈爾濱工業大學

实验报告

实 验（二）

题　　目 DPCM 编解码实验

专　　业 人工智能

学　　号 1190201215

班　　级 1903602

学　　生 冯开来

指 导 教 师 郑铁然

实 验 地 点 格物 207

实 验 日 期 2021.12.11

计算机科学与技术学院

一、 8 比特 DPCM 编解码算法

1.1 简述算法的内容

1.1.1 直接量化法

假设原本得到的音频数据为 x(n),编码的数据为 c(n),预测的音频数据为 y(n)。差分编码最简单的方法就是直接存储前一次的采样值，然后用本次采样值去计算差值 d(n)，对差值进行量化得到数字语音编码。解码端则是做相反的处理恢复原信号。

直接量化又是这些量化方法里面最简单的。算法原理就是差值在-127 和 128 之间，不变换，而大于 128 的变换为 128，小于-127 变换为-127。最后加上 128，这样变换后的结果就是编码的数据 c(n)。

对 c(n)采取 8bit 量化，即一个采样点一个 Byte，这就对应了为什么上面要映射到-127 到 128。其实挺好做的，就是一个 c(n)的值就保存为一个 Byte 类型写入文件。

在让 c(n)加上之前预测的音频数据 y(n-1)得到的就是当前预测的音频数据 y(n)。

我对这个算法简单的理解就是首先这里的差值不是原音频每个采样点的差值，而是当前真实采样点和前一个预测采样点的差值。其次就是编码数据对差值进行了压缩。最后是通过编码数据预测下一个音频数据，不难发现，这其中是一个一边编码一边解码的过程。

```
# 8-bit直接量化
if method == 0 and bit == 8: # 8-bit直接量化
    # 对第一位进行编码
    if d[0] > 127:
        c[0] = 127
    elif d[0] < -128:
        c[0] = -128
    else:
        c[0] = d[0]
    # 对接下来的每一位差值进行编码
    for i in range(1, n):
        d[i] = data[i] - x[i-1]
        if d[i] > 127:
            c[i] = 127
        elif d[i] < -128:
            c[i] = -128
        else:
            c[i] = d[i]
        # 在量化的过程中嵌入了解码的过程
        x[i] = x[i-1] + c[i] # 解码过程
    print(cal_snr(wave_data, x)) # 计算信噪比
# 将预测的音频写入文件，即解码的文件
f = wave.open("DPCM\\2_" + str(bit) +"bit.pcm", "wb")
f.setparams((1, 2, 16000, len(x), 'NONE', 'HIT-1190201215'))
f.writeframes(x.astype(np.short))
f.close()
# 将cn写入文件，即编码文件，c[0]是16-bit，其余皆是一个采样点一个byte
g = open("DPCM\\2_" + str(bit) + "bit.dpc", "wb")
g.write(np.short(c[0]))
for i in range(1, len(c)):
    g.write(np.int8(c[i]))
g.close()
```

1.1.2 量化因子法（取量化因子为 250）

量化因子法在直接量化的基础上加了一点东西。记量化因子为 a，具体方法是当差值大于 127×a 时，数据转换为 127，差值小于-128×a 时，数据转换为-128，其余时候以 a 为一个步长，从-128×a 到 127×a 中寻找当(i-1)a<d(n)≤ia 的时候转换为 i。

对 c(n)的 8bit 量化和写入文件以及对下一个音频数据的预测和上面一样。

其实换个角度想，直接量化法就是特殊的量化因子法，将步长变成了 1，而量化因子法增加步长之后，是将更多连续且相关性更高的信号压缩到一起，这样对于减少语音信号中的信息冗余度更加有效。

```
# 8-bit量化因子
elif method == 1 and bit == 8:
    # 对第一位进行编码
    if d[0] > 127 * a:
        c[0] = 127
    elif d[0] < -128 * a:
        c[0] = -128
    else:
        for j in range(-128, 128):
            if (j-1) * a < d[0] <= j * a:
                c[0] = j
                break
    # 对剩下的差值进行编码
    for i in range(1, n):
        d[i] = data[i] - x[i - 1]
        if d[i] > 127 * a:
            c[i] = 127
        elif d[i] < -128 * a:
            c[i] = -128
        else:
            for j in range(-128, 128):
                if (j - 1) * a < d[i] <= j * a:
                    c[i] = j
                    break
        c[i] += 128 # 每个c[i]加上128也是编码过程，我的认为是为了'居中'
        x[i] = x[i - 1] + (c[i] - 128) * a # 解码过程
```

1.2 解码信号的信噪比

1.2.1 信噪比计算公式

$$SNR = 10 * \log_{10}\left\{\frac{\sum_{n=0}^{M}(x(n))^2}{\sum_{n=0}^{M}(\overline{x}(n)-x(n))^2}\right\}$$

信噪比计算代码实现：

其中计算每一项的平方的和我是通过一维矩阵的二范数实现的，为了防止溢出，我对于每一项除以了数组长度进行归一化处理。

```
def cal_snr(x, X):
    lengh = len(x)
    s1 = np.dot(x/lengh, x.T/lengh) # 除以lengh归一化处理，防止溢出
    s2 = np.dot((x-X)/lengh, (x-X).T/lengh)
    return 10 * np.log10(s1/s2)
```

1.2.2 直接量化法

信噪比如下：

-0.07993551532572893

可以看到，直接量化法的信噪比竟然是负数，可以理解为量化误差比原音频都高，说明效果还是很差的。

1.2.3 量化因子法

信噪比如下：

39.299333941766726

经过不断尝试，我发现当量化因子取 250 左右的时候，其信噪比最高，听起来的效果也是最好的。

二、4 比特 DPCM 编解码算法

2.1 简述算法内容，给出所采用的编码参数（如所采用的量化因子等）

2.1.1 直接量化法

这里采用 4bit 量化压缩，所以算法原理是映射到-8 和 7 之间，不变换，而大于 8 的变换为 8,小于-7 变换为-7。变换后的结果就是编码的数据再加上 8 就 c(n)。

对 c(n)采取 4bit 量化，即两个采样点一个 Byte，就是对于 c(0)来说仍然按照 16bit 保存，剩下的 c(n)，两个为一组，如 c(1)和 c(2)为一个 byte，计算方法将就是将前面一个 c(i)左移 4 位（×16）然后加上 c(i+1)，保存为一个 Byte 类型写入文件。

再让 c(n)-8 加上之前预测的音频数据 y(n-1)得到的就是当前预测的音频数据 y(n)。

```
# 4-bit直接量化
elif method == 0 and bit == 4:
    # 对一个差值进行量化
    if d[0] > 7:
        c[0] = 7
    elif d[0] < -8:
        c[0] = -8
    else:
        c[0] = d[0]
    for i in range(1, n): # 对剩下的差值进行量化
        d[i] = data[i] - x[i-1]
        if d[i] > 7:
            c[i] = 7
        elif d[i] < -8:
            c[i] = -8
        else:
            c[i] = d[i]
        c[i] += c[i] + 8
        x[i] = x[i-1] + (c[i] - 8) # 解码过程
    print(cal_snr(wave_data, x))
# 将预测的数据（解码数据）写入文件
f = wave.open("DPCM\\2_" + str(bit) + "bit.pcm", "wb")
f.setparams((1, 2, 16000, len(x), 'NONE', 'HIT-1190201215'))
f.writeframes(x.astype(np.short))
f.close()
#将c(n)写入文件
lengh = int(np.ceil((c.shape[0]+1) / 2)) # 计算两个采集点为一个byte的数组长度
c_dpc = np.zeros(lengh) # 初始化新数组，为即将写入文件的编码
g = open("DPCM\\2_" + str(bit) + "bit.dpc", "wb")
c_dpc[0] = c[0]
g.write(np.short(c_dpc[0])) # 首位仍然采用16bit写入文件
for i in range(1, len(c)-1, 2): # 步长为2，为放置溢出，终止条件为原c(n)-1
    n = int(np.ceil(i/2))
    c_dpc[n] = c[i]*16 + c[i+1]
    g.write(np.int8(n))
if len(c) % 2 == 0: # 当原数组是2的整数倍也就是除去首位后无法被2整除，则最后一位单独处理
    n = lengh -1
    c_dpc[n] = c[len(c)-1]*16
    g.write(np.int8(c_dpc[n]))
g.close()
```

2.1.2 量化因子法（取量化因子为 1260）

量化因子法在直接量化的基础上加了一点东西。记量化因子为 b，具体方法是当差值大于 7×b 时，数据转换为 7，差值小于-8×b 时，数据转换为-8，其余时候以 b 为一个步长，从-8×b 到 7×b 中寻找当(i-1)b<d(n)≤ib 的时候转换为 i。

对 c(n)的 8bit 量化和写入文件以及对下一个音频数据的预测和上面一样。

其实换个角度想，直接量化法就是特殊的量化因子法，将步长变成了 1，而量化因子法增加步长之后，是将更多连续且相关性更高的信号压缩到一起，这样对于减少语音信号中的信息冗余度更加有效。

```
# 4-bit量化因子
elif method == 1 and bit == 4:
    if d[0] > 7 * b:
        c[0] = 7
    elif d[0] < -8 * b:
        c[0] = -8
    else:
        for j in range(-8, 8):
            if (j-1) * b < d[0] <= j * b:
                c[0] = j
                break
    for i in range(1, n):
        d[i] = data[i] - x[i-1]
        if d[i] > 7 * b:
            c[i] = 7
        elif d[i] < -8 * b:
            c[i] = -8
        else:
            for j in range(-8, 8):
                if (j - 1) * b < d[i] <= j * b:
                    c[i] = j
                    break
        c[i] += 8
        x[i] = x[i-1] + (c[i] - 8) * b # 解码过程
    print(cal_snr(wave_data, x))
```

2.2 拷贝你的算法，加上适当的注释说明

（包含直接量化，量化因子）

```
# 4-bit 直接量化
elif method == 0 and bit == 4:
    # 对一个差值进行量化
    if d[0] > 7:
        c[0] = 7
    elif d[0] < -8:
        c[0] = -8
    else:
        c[0] = d[0]
    for i in range(1, n): # 对剩下的差值进行量化
        d[i] = data[i] - x[i-1]
        if d[i] > 7:
            c[i] = 7
        elif d[i] < -8:
            c[i] = -8
        else:
            c[i] = d[i]
        c[i] += c[i] + 8
```

```
        x[i] = x[i-1] + (c[i] - 8) # 解码过程
    print(cal_snr(wave_data, x))
    # 将预测的数据（解码数据）写入文件
    f = wave.open("DPCM\\2_" + str(bit) + "bit.pcm", "wb")
    f.setparams((1, 2, 16000, len(x), 'NONE', 'HIT-1190201215'))
    f.writeframes(x.astype(np.short))
    f.close()
    #将 c(n) 写入文件
    lengh = int(np.ceil((c.shape[0]+1) / 2)) # 计算两个采集点为一个 byte 的
数组长度
    c_dpc = np.zeros(lengh) # 初始化新数组，为即将写入文件的编码
    g = open("DPCM\\2_" + str(bit) + "bit.dpc", "wb")
    c_dpc[0] = c[0]
    g.write(np.short(c_dpc[0])) # 首位仍然采用 16bit 写入文件
    for i in range(1, len(c)-1, 2): # 步长为 2，为放置溢出，终止条件为原 c(n)-
1
        n = int(np.ceil(i/2))
        c_dpc[n] = c[i]*16 + c[i+1]
        g.write(np.int8(n))
    if len(c) % 2 == 0: # 当原数组是 2 的整数倍也就是除去首位后无法被 2 整除，则最
后一位单独处理
        n = lengh -1
        c_dpc[n] = c[len(c)-1]*16
        g.write(np.int8(c_dpc[n]))
    g.close()

    return x, c_dpc

# 4-bit 量化因子
elif method == 1 and bit == 4:
    if d[0] > 7 * b:
        c[0] = 7
    elif d[0] < -8 * b:
        c[0] = -8
    else:
        for j in range(-8, 8):
            if (j-1) * b < d[0] <= j * b:
                c[0] = j
                break
    for i in range(1, n):
        d[i] = data[i] - x[i-1]
        if d[i] > 7 * b:
            c[i] = 7
        elif d[i] < -8 * b:
            c[i] = -8
        else:
            for j in range(-8, 8):
                if (j - 1) * b < d[i] <= j * b:
                    c[i] = j
                    break
        c[i] += 8
        x[i] = x[i-1] + (c[i] - 8) * b # 解码过程
    print(cal_snr(wave_data, x))

    f = wave.open("DPCM\\2_" + str(bit) + "bit.pcm", "wb")
```

```
    f.setparams((1, 2, 16000, len(x), 'NONE', 'HIT-1190201215'))
    f.writeframes(x.astype(np.short))
    f.close()

    lengh = int(np.ceil((c.shape[0]+1) / 2))
    c_dpc = np.zeros(lengh)
    g = open("DPCM\\2_" + str(bit) + "bit.dpc", "wb")
    c_dpc[0] = c[0]
    g.write(np.short(c_dpc[0]))
    for i in range(1, len(c)-1, 2):
        n = int(np.ceil(i/2))
        c_dpc[n] = c[i]*16 + c[i+1]
        g.write(np.int8(n))
    if len(c) % 2 == 0:
        n = lengh -1
        c_dpc[n] = c[len(c)-1]*16
        g.write(np.int8(c_dpc[n]))
    g.close()

    return x, c_dpc
```

2.3 解码信号的信噪比

2.3.1 直接量化法

信噪比如下：

-0.07133474295031117

可以看到，4bit 量化的直接量化法的信噪比是负数，可以理解为量化误差比原音频都高，说明效果还是很差的。

2.3.2 量化因子法（量化因子取 1260）

信噪比如下：

16.98437053577291

经过不断尝试，我发现当量化因子取 1260 左右的时候，其信噪比最高，听起来的效果也是最好的。

三、改进策略

3.1 你提出了什么样的改进策略，效果如何

编码的核心其实就是映射到一个更小的范围，这样在传输信号的过程中造成的损失会更小，其实从某个角度来说，直接量化和量化因子的方法可以理解为双阈值筛选，通过两个阈值将数据过大和过小信号晒去，映射成中间的小范围。

所以对于映射到更小的范围，很容易想到的方法就是取对数，然后对这个结果，利用量化因子法映射到-7 到 8 之间。取对数的底数其实也要搜索尝试一番才能找到。首先假设底数为 e，量化因子为 q，对 d(n)的绝对值求对数，如果在-7×q 到 8×q 之外，直接令 c(n)为-7 或 8，而如果这个对数在-7 到 8 之间，搜索以 q 为步长下，满足(i-1)q＜ln(d(n))≤iq，将 ln(dn)映射到成 i。最后编码的时候都加上 8，以 4bit 编码写入文件。

而在解码的过程中首先就是还原，公式就是 pow(e, (c[n]-8)×q),但是此时的 d(n)都是正的，我们要根据一开始的 d(n)来判断是否为负数。

```
# 4-bit 自己优化的方法，结合了对数变换和量化因子法
elif method == 2 and bit == 4:
    c[0] = np.log(abs(d[0]))
    if c[0] < -7*q:
        c[0] = -7
    elif c[0] > 8*q:
        c[0] = 8
    else:
        for j in range(-7, 7):
            if (j-1)*q < c[0] <= j*q:
                c[0] = j
                break
    for i in range(1, n):
        d[i] = data[i] - x[i-1]
        if d[i] == 0:
            d[i] = 1
        c[i] = np.log(abs(d[i])) # 先对dn取对数，然后对对数进行量化因子法
        if c[i] < -7 * q:
            c[i] = -7
        elif c[i] > 8 * q:
            c[i] = 8
        else:
            for j in range(-7, 7):
                if (j - 1) * q < c[i] <= j * q:
                    c[i] = j
                    break
        c[i] += 8
        # 在解码的过程中要注意dn的正负
        if d[i] > 0:
            x[i] = x[i-1] + pow(np.e, (c[i]-8) * q)
        else:
            x[i] = x[i-1] - pow(np.e, (c[i]-8) * q)
```

信噪比如下：

4.9356239020974035

在这一过程中，有一个很尴尬的现象就是当 q 取 1 的时候信噪比最高，效果最好，但是 q=1 就是直接量化，所以整个过程我就是去了个 e 的对数操作，而且此

时的信噪比为 4 点几，效果没有那么理想。没有什么亮眼的地方，所以我打算对底数 e 进行优化。

肯定存在一个最优的底数使得取对数后的操作效果最好，经过搜索和尝试，我将底数确定为 3，发现一下子信噪比变成了 7 点几，效果有显著提升。

```
信噪比如下：
7.674161564922017
```

但是我相信肯定存在一个 p，q 使得信噪比比这个还要高，只是我还没找到。

。

四、 简述你对量化误差的理解

4.1 什么是量化误差？

在对差值 d(n)进行量化的时候，这里的差值可能过大，为便于存储，一般设置一个量化因子，可以理解位步长，在遍历的时候以步长为单位，该范围内的都映射到同一个值，比如，如果量化因子是 250 的话，差值 250 就可以映射到同一个数，用了更少的比特存储，但是会因此出现量化误差。

还有一种情况是所有的 d(n)都映射在同一个范围内，比如-127 到 128，-7 到 8，而如果差值过大或者过小，超出了这个范围，就是转换成边界，因此就会导致量化误差。

而这些差值（量化误差）可能超出了正确量化的范围，导致量化值精度不够，从而在解码过程中预测计算出的值与真实值不同。

4.2 为什会编码器中会有一个解码器

如果有一个数据出现了量化误差，那么后面的数据在还原的过程中就会在量化误差的基础上继续进行还原，这样的话，会让之前出现的误差加下去，继续影响后面所有的数据还原。

所以需要一边压缩，一边解压。利用当前真实的数据减去上一个解码预测的数据求得 d(n)，这样造成的量化误差只会影响当前节点，而不会影响后续采样点的预测。

五、 总结

5.1 请总结本次实验的收获

1. 了解了 DPCM 编解码原理
2. 学会了几种编码解码的方法
3. 学会了如何通过内嵌解码器减少量化误差的影响

5.2 请给出对本次实验内容的建议

1. 一开始没有理解 4-bit 量化和 8-bit 量化的区别，一直以为是在编码前对初始音频数据进行量化，后来知道是对编码后的数据进行 8/4bit 的量化。就在这上面浪费了挺多时间，但觉得纯粹是自己傻。
2. 没有什么建议，感觉很好！

六、 附录（代码）

```
import os
import math
import wave
import numpy as np
import matplotlib.pylab as plt

# 读取文件
def readWaveData(file_path):
    f = wave.open(file_path, "rb")
    params = f.getparams()
    # 声道数，量化位数，采样频率，采样点数
    nchannels, sampwidth, framerate, nframes = params[:4]
    # 得到每个采样点的值
    str_data = f.readframes(nframes)
    f.close()
    # 转成 short 类型
    wave_data = np.frombuffer(str_data, dtype=np.short)
    # 通过采样点数和取样频率计算每个取样的时间
    time = np.arange(0, nframes) / framerate
    return wave_data, time, nchannels, sampwidth, framerate



def DPCM(data, method, bit):
    a = 110 # 量化因子 8bit
    b = 1260 # 量化因子 4bit
    q = 1 # 自己优化方法中的量化因子
    p = 3 # 自己优化方法中取对数的底数
    n = len(data)
    x = np.zeros(n)
    c = np.zeros(n)
    d = np.zeros(n)
    x[0] = data[0]
    d[0] = data[0]

    # method: 0-直接量化法，1-量化因子法，2-自己优化的方法，结合了对数变换和量化
因子法
    # bit: 4-4bit 量化，8-8bit 量化

    # 8-bit 直接量化
    if method == 0 and bit == 8: # 8-bit 直接量化
        # 对第一位进行编码
        if d[0] > 127:
            c[0] = 255
        elif d[0] < -128:
            c[0] = 0
        else:
            c[0] = d[0]
        # 对接下来的每一位差值进行编码
        for i in range(1, n):
```

```
            d[i] = data[i] - x[i-1]
            if d[i] > 127:
                c[i] = 127
            elif d[i] < -128:
                c[i] = -128
            else:
                c[i] = d[i]
            c[i] += 128
            # 在量化的过程中嵌入了解码的过程
            x[i] = x[i-1] + c[i]-128 # 解码过程
        print(cal_snr(wave_data, x)) # 计算信噪比

        # 将预测的音频写入文件，即解码的文件
        f = wave.open("DPCM\\2_" + str(bit) +"bit.pcm", "wb")
        f.setparams((1, 2, 16000, len(x), 'NONE', 'HIT-1190201215'))
        f.writeframes(x.astype(np.short))
        f.close()
        # 将 cn 写入文件，即编码文件，c[0]是 16-bit，其余皆是一个采样点一个 byte
        g = open("DPCM\\2_" + str(bit) + "bit.dpc", "wb")
        g.write(np.short(c[0]))
        for i in range(1, len(c)):
            g.write(np.int8(c[i]))
        g.close()

        return x, c

    # 8-bit 量化因子
    elif method == 1 and bit == 8:
        # 对第一位进行编码
        if d[0] > 127 * a:
            c[0] = 127
        elif d[0] < -128 * a:
            c[0] = -128
        else:
            for j in range(-128, 128):
                if (j-1) * a < d[0] <= j * a:
                    c[0] = j
                    break
        # 对剩下的差值进行编码
        for i in range(1, n):
            d[i] = data[i] - x[i - 1]
            if d[i] > 127 * a:
                c[i] = 127
            elif d[i] < -128 * a:
                c[i] = -128
            else:
                for j in range(-128, 128):
                    if (j - 1) * a < d[i] <= j * a:
                        c[i] = j
                        break
            c[i] += 128 # 每个 c[i]加上 128 也是编码过程，我的认为是为了'居中'
            x[i] = x[i - 1] + (c[i] - 128) * a # 解码过程

        print(cal_snr(wave_data, x))
```

```
        f = wave.open("DPCM\\2_" + str(bit) +"bit.pcm", "wb")
        f.setparams((1, 2, 16000, len(x), 'NONE', 'HIT-1190201215'))
        f.writeframes(x.astype(np.short))
        f.close()

        c_dpc = np.zeros((c.shape[0]))
        g = open("DPCM\\2_" + str(bit) + "bit.dpc", "wb")
        g.write(np.short(c[0]))
        for i in range(1, len(c)):
            g.write(np.int8(c[i]))
        g.close()

        return x, c

    # 4-bit 直接量化
    elif method == 0 and bit == 4:
        # 对一个差值进行量化
        if d[0] > 7:
            c[0] = 7
        elif d[0] < -8:
            c[0] = -8
        else:
            c[0] = d[0]
        for i in range(1, n): # 对剩下的差值进行量化
            d[i] = data[i] - x[i-1]
            if d[i] > 7:
                c[i] = 7
            elif d[i] < -8:
                c[i] = -8
            else:
                c[i] = d[i]
            c[i] += c[i] + 8
            x[i] = x[i-1] + (c[i] - 8) # 解码过程
        print(cal_snr(wave_data, x))
        # 将预测的数据（解码数据）写入文件
        f = wave.open("DPCM\\2_" + str(bit) + "bit.pcm", "wb")
        f.setparams((1, 2, 16000, len(x), 'NONE', 'HIT-1190201215'))
        f.writeframes(x.astype(np.short))
        f.close()
        #将 c(n) 写入文件
        lengh = int(np.ceil((c.shape[0]+1) / 2)) # 计算两个采集点为一个 byte
的数组长度
        c_dpc = np.zeros(lengh) # 初始化新数组，为即将写入文件的编码
        g = open("DPCM\\2_" + str(bit) + "bit.dpc", "wb")
        c_dpc[0] = c[0]
        g.write(np.short(c_dpc[0])) # 首位仍然采用 16bit 写入文件
        for i in range(1, len(c)-1, 2): # 步长为 2，为放置溢出，终止条件为原
c(n)-1
            n = int(np.ceil(i/2))
            c_dpc[n] = c[i]*16 + c[i+1]
            g.write(np.int8(n))
        if len(c) % 2 == 0: # 当原数组是 2 的整数倍也就是除去首位后无法被 2 整除，
则最后一位单独处理
            n = lengh -1
            c_dpc[n] = c[len(c)-1]*16
```

```
            g.write(np.int8(c_dpc[n]))
        g.close()

        return x, c_dpc

    # 4-bit 量化因子
    elif method == 1 and bit == 4:
        if d[0] > 7 * b:
            c[0] = 7
        elif d[0] < -8 * b:
            c[0] = -8
        else:
            for j in range(-8, 8):
                if (j-1) * b < d[0] <= j * b:
                    c[0] = j
                    break
        for i in range(1, n):
            d[i] = data[i] - x[i-1]
            if d[i] > 7 * b:
                c[i] = 7
            elif d[i] < -8 * b:
                c[i] = -8
            else:
                for j in range(-8, 8):
                    if (j - 1) * b < d[i] <= j * b:
                        c[i] = j
                        break
            c[i] += 8
            x[i] = x[i-1] + (c[i] - 8) * b # 解码过程
        print(cal_snr(wave_data, x))

        f = wave.open("DPCM\\2_" + str(bit) + "bit.pcm", "wb")
        f.setparams((1, 2, 16000, len(x), 'NONE', 'HIT-1190201215'))
        f.writeframes(x.astype(np.short))
        f.close()

        lengh = int(np.ceil((c.shape[0]+1) / 2))
        c_dpc = np.zeros(lengh)
        g = open("DPCM\\2_" + str(bit) + "bit.dpc", "wb")
        c_dpc[0] = c[0]
        g.write(np.short(c_dpc[0]))
        for i in range(1, len(c)-1, 2):
            n = int(np.ceil(i/2))
            c_dpc[n] = c[i]*16 + c[i+1]
            g.write(np.int8(n))
        if len(c) % 2 == 0:
            n = lengh -1
            c_dpc[n] = c[len(c)-1]*16
            g.write(np.int8(c_dpc[n]))
        g.close()

        return x, c_dpc

    # 4-bit 自己优化的方法，结合了对数变换和量化因子法
    elif method == 2 and bit == 4:
        c[0] = np.log(abs(d[0])) / np.log(p)
        if c[0] < -7*q:
```

```
            c[0] = -7
        elif c[0] > 8*q:
            c[0] = 8
        else:
            for j in range(-7, 7):
                if (j-1)*q < c[0] <= j*q:
                    c[0] = j
                    break
        for i in range(1, n):
            d[i] = data[i] - x[i-1]
            if d[i] == 0:
                d[i] = 1
            c[i] = np.log(abs(d[i])) # 先对 dn 取对数，然后对对数进行量化因子法
            if c[i] < -7 * q:
                c[i] = -7
            elif c[i] > 8 * q:
                c[i] = 8
            else:
                for j in range(-7, 7):
                    if (j - 1) * q < c[i] <= j * q:
                        c[i] = j
                        break
            c[i] += 8
            # 在解码的过程中要注意 dn 的正负
            if d[i] > 0:
                x[i] = x[i-1] + pow(p, (c[i]-8) * q)
            else:
                x[i] = x[i-1] - pow(p, (c[i]-8) * q)

        print(cal_snr(wave_data, x))

        f = wave.open("DPCM\\2_" + str(bit) + "bit.pcm", "wb")
        f.setparams((1, 2, 16000, len(x), 'NONE', 'HIT-1190201215'))
        f.writeframes(x.astype(np.short))
        f.close()

        lengh = int(np.ceil((c.shape[0]+1) / 2))
        c_dpc = np.zeros(lengh)
        g = open("DPCM\\2_" + str(bit) + "bit.dpc", "wb")
        c_dpc[0] = c[0]
        g.write(np.short(c_dpc[0]))
        for i in range(1, len(c)-1, 2):
            n = int(np.ceil(i/2))
            c_dpc[n] = c[i]*16 + c[i+1]
            g.write(np.int8(n))
        if len(c) % 2 == 0:
            n = lengh -1
            c_dpc[n] = c[len(c)-1]*16
            g.write(np.int8(c_dpc[n]))
        g.close()

        return x, c_dpc


    else:
        print('error!')
        return -1, -1
```

```
def sgn(x):
    if x >= 0:
        return 1
    return -1

def cal_snr(x, X):
    lengh = len(x)
    s1 = np.dot(x/lengh, x.T/lengh) # 除以 lengh 归一化处理，防止溢出
    s2 = np.dot((x-X)/lengh, (x-X).T/lengh)
    return 10 * np.log10(s1/s2)


file_path = "语料\\2.wav"
# 从文件读取语音数据，返回采样点和采样点时间
wave_data, time, nchannels, sampwidth, framerate =
readWaveData(file_path)
# print(wave_data, len(wave_data))
# method: 0 直接量化法，1 量化因子法，2 自己优化的方法，结合了对数变换和量化因子法
# bit: 4-4bit 量化，8-8bit 量化
print("信噪比如下：")
x, c = DPCM(wave_data, method=2, bit=4)
# print(x, c, len(x))
```