哈爾濱工業大學

实验报告

实 验（二）

题　　目 DPCM 编解码

专　　业 _____

学　　号 _____

班　　级 _____

学　　生 _____

指 导 教 师 _____

实 验 地 点 _____

实 验 日 期 _____

计算机科学与技术学院

一、 实验基本信息

1.1 实验环境

Windows10; python 3.9.12; Visual Studio Code; Anaconda.

1.2 开发工具

主要使用 Python 工具完成实验，其中，调用 wave、os 和 numpy 库对音频文件进行处理。

1.3 语料的格式

采样频率　= 16000Hz
量化比特数= 16bits
声道个数　= 1（Mono 单声道）

1.4 读入语料的存储格式

在编解码之前，读入的原始语料量化比特数为 16bits，因此每一个采样值占 2bytes. 每个采样值存储在一个 numpy.short 格式的整数中，整体的音频数据 wave_data 组成一个 dtype = numpy.short 的 numpy.array 数组，该数组的长度即为采样值的总数。代码如下所示：

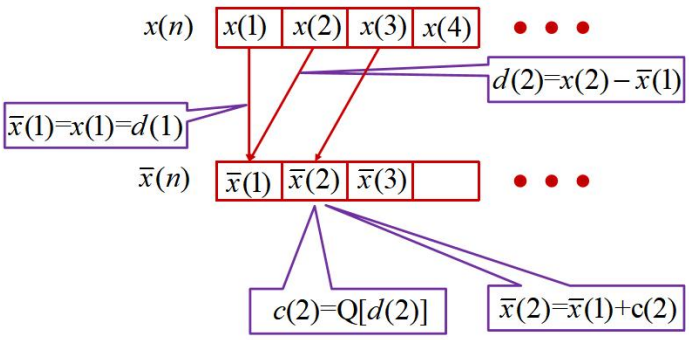
```
filepath = '语料'
filename = os.listdir(filepath)
# 打开一个声音文件，返回一个声音的实例
f = wave.open(filepath + '/' + filename[i], 'r')
# 一次性返回所有的音频参数，返回的是一个元组
params = f.getparams()
# 声道数，量化位数(byte 单位)，采样频率，采样点数
# 量化位数为用多少 bit 表达一次采样所采集的数据，1byte=8bits
#print (params)
nchannels, sampwidth, framerate, nframes = params[:4]
# readframes 返回的是二进制数据（一大堆 bytes），在 Python 中用字符串表示二进制数据。
str_data = f.readframes(nframes)
# 转成二字节数组形式（每个采样点占两个字节，即 short）
wave_data = np.frombuffer(str_data, dtype=np.short)
f.close()
```

二、DPCM 算法

2.1 量化因子法（因子为 1 时相当于直接量化法）

预测编码：

预测编码利用的是信源相邻符号间的相关性。根据某一模型利用以往的样本值对新的样本值进行预测。然后将样本的实际值与预测值相减得到一个误差值，最后对这一误值进行编码。如果模型足够好且样本序列在时间上相关性比较强，则误差信号的幅度将远远小于原始信号，从而得到较大的数据压缩。编码过程如下图所示：



量化：

量化是用一个很小的集合表示一个大集合的值，是有失真压缩的有效工具。也就是上图所示的 Q 函数，表示量化过程。

①8 比特量化编码算法：

量化函数 Q_8bit 如下所示，其中 a 为量化因子：

$$c(n)=\begin{cases}127, & d(n)>127a\\ -128, & d(n)<-128a\\ i, & (i-1)a<d(n)\le ia,\ -128\le i\le 127\end{cases}$$

代码如下所示：

```
# a 为正数
def Q_8bit(x, a):
    if x > 127 * a:
        return 127
    elif x < -128 * a:
        return -128
    else:
        return np.ceil(x/a).astype(np.int8)
```

此外，第一个采样值不进行压缩，占 16bit，在存储压缩值的 numpy.int8 的数组中国，占两个位置。通过对原始音频第一个采样值 wave_data[0]化为正数后对 256 取余和整除，拆分成两个 numpy.int8 格式的数存储，分别存储在 new_data[-1]和 new_data[0]中。

```
def compress_to_8bit(wave_data, a):
    # 第一个采样值占 16bits
    new_data = np.zeros(len(wave_data) + 1, dtype=np.int8)
    # 把第一个采样值按二进制的前 8 位和后 8 位分别存储在 new_data[-1]和 new_data[0]中
    new_data[-1] = np.int8((wave_data[0] + 32768) // 256 - 128)
    new_data[0] = np.int8((wave_data[0] + 32768) % 256 - 128)
    x_ = int(wave_data[0])
    for i in range(1, len(wave_data)):
        d = int(wave_data[i]) - int(x_)
        new_data[i] = Q_8bit(d, a)
        x_ += new_data[i] * a
    return new_data
```

解码时，对第一个采样值数据进行复原后，乘量化因子后累加。

```
def decompress_8bit(wave_data, a):
    decompress_data = np.zeros(len(wave_data) - 1, dtype=np.int16)
    decompress_data[0] = (wave_data[0] + 128) + (wave_data[-1] + 128) * 256 - 32768
    for i in range(1, len(decompress_data)):
        decompress_data[i] = decompress_data[i-1] + wave_data[i] * a
    return decompress_data
```

①4 比特量化编码算法：

量化函数 Q_4bit 如下所示，其中 a 为量化因子：

$$c(n)=\begin{cases}7, & d(n)>7a\\ -8, & d(n)<-8a\\ i, & (i-1)a<d(n)\le ia,-8\le i\le 7\end{cases}$$

为方便存储，令 c(n)不为负数，c(n) = c(n) + 8. 不过在实际的编码过程中，当−8a ≤ d(n) ≤ 7a 时取 ceil(x/a)容易把噪音放大，改为取 round(x/a)能避免这一问题。代码如下所示：

```
# a 为正数，要返回正数
def Q_4bit(x, a):
    if x > 7 * a:
        return 15
    elif x < -8 * a:
        return 0
    else:
        return round(x/a) + 8
```

由于没有只占 4bit 的数据类型，因此存储采样值压缩到 4bit 后数据，每两个采样值占一个 numpy.unit8. 此外，第一个采样值不进行压缩，占 16bit，在存储压缩值的 numpy.int8 的数组中国，占两个位置。通过对原始音频第一个采样值 wave_data[0]化为正数后对 256 取余和整除，拆分成两个 numpy.uint8 格式的数存储，分别存储在 new_data[-1]和 new_data[0]中。代码如下所示，mehod = 'a'是量化因子法的情况：

```
# 两种压缩成 4bit 的方式：量化因子法或指数映射的方法
def compress_to_4bit(wave_data, a_or_c, Q, method):
    # 第一个采样值占 16bits，如果不整的话多 4bit，值为 0
    new_data = np.zeros(int(np.ceil((len(wave_data)-1)/2) + 2), dtype=np.uint8)
    # 把第一个采样值按二进制的前 8 位和后 8 位分别存储在 new_data[-1]和 new_data[0]中
    new_data[-1] = np.int8((wave_data[0] + 32768) // 256)
    new_data[0] = np.int8((wave_data[0] + 32768) % 256)
    # wave_data 的索引 i，new_data 的索引 j，ceil(i // 2) = j
    x_ = int(wave_data[0])
    for i in range(1, len(wave_data)):
        j = int(np.ceil(i/2))
        d = int(wave_data[i]) - int(x_)
        c = Q(d, a_or_c)
        if method == 'a':
            x_ += (c - 8) * a_or_c
        elif method == 'e':
            if c >= 8:
                x_ += round(-np.log((7.5 - np.abs(c - 8)) / 8) / a_or_c)
            else:
                x_ += round(np.log((7.5 - np.abs(c)) / 8) / a_or_c)
        if i % 2 == 1:
            new_data[j] = c * 16
        else:
            new_data[j] += c
    if (len(wave_data)-1)/2 % 2 != 0:
        new_data[-2] += 8
    return new_data
```

解码时，对第一个采样值数据进行复原后，乘量化因子后累加。由于两个采样值共用数组的一位，要通过对 16 整除和取余分离两个数。并且当总采样数为偶数是，会解码多一个采样值，在最后一位，与原本最后一位采样值相同，对听觉几乎没有影响，不进行特殊处理。代码如下所示，mehod = 'a'是量化因子法的情况：

```
def decompress_4bit(wave_data, a_or_c, method):
    decompress_data = np.zeros(2 * (len(wave_data) - 2) + 1, dtype=np.int16)
    decompress_data[0] = wave_data[0] + wave_data[-1] * 256 - 32768
    for i in range(1, len(decompress_data)):
        index = int(np.ceil(i/2))
        if i % 2 == 1:
            c = wave_data[index] // 16
        else:
            c = wave_data[index] % 16
        if method == 'a':
            decompress_data[i] = decompress_data[i-1] + (c - 8) * a_or_c
        elif method == 'e':
```

```
            if c >= 8:
                d = round(-np.log((7.5 - np.abs(c - 8)) / 8) / a_or_c)
            else:
                d = round(np.log((7.5 - np.abs(c)) / 8) / a_or_c)
            decompress_data[i] = decompress_data[i-1] + d
    return decompress_data
```

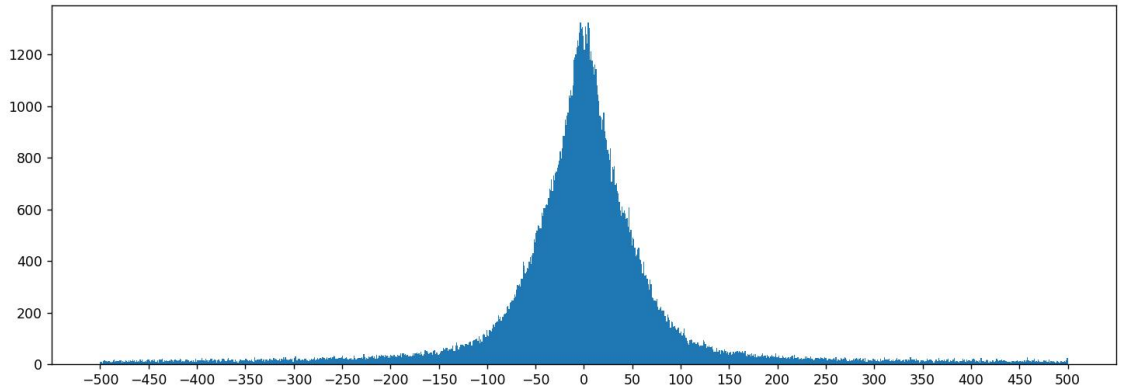
2.2 改进的算法（一种通过指数函数映射的方法）

首先，通过统计采样差分值的分布，观测采样值的特征，并绘制直方图。代码如下所示：

```
def Compute_diff(wave_data):
    diff_lst = np.zeros(len(wave_data) - 1)
    for i in range(1, len(wave_data)):
        diff_lst[i-1] = wave_data[i] - wave_data[i-1]

    d = np.linspace(-500,500,1001).astype(int)
    plt.xticks(d[::50])
    plt.hist(diff_lst, d)
    plt.show()
```

得到的关于 2.wav 的统计结果，其中 x 轴坐标表示与上一个采样值的差值，y 轴坐标 h(x)表示采样差值 x 对应出现的次数（经实验，其他音频得出的结果也与之高度相近）：



从中可以看出，差分值关于 x = 0 对称分布，在 x = 0 附近 h(x)取得最大值 1300 左右。且绝大部分差分值集中在-100~100 的范围内，且 $\lim_{|x|\to\infty} h(x) = 0$。

为了使编解码后的音频损失最小(信噪比最大)，实际上应使：

$$\sum (Q(x) - x)^2 = \int_{-32768}^{32767} (Q(x) - x)^2 \cdot h(x)\, dx$$

取最小值。而编码位数是固定为 4bit 的，因此应当给出现次数多的差分值附近分配更多的编码位数，令编码值变化较大，减小因为编码舍入而产生的误差；而

给出现次数少的差分值附近分配更少的编码位数，令编码值变化较小，能够减小累积误差值。**即，h(x)越大，Q(x)的变化率（导数）应当越大，应当使更少的 x_0 附近的 x 被映射到 $Q(x_0)$。**

由于差分值关于 x = 0 对称分布，因此正的差分值和负的差分值应当各自分配 3bit(8 个值)进行存储，且从差分值绝对值映射到存储值的函数应当相同。另外 1bit 用于存储差分值符号的正负。

以 $x \geq 0$ 为例，$\lim_{x\to\infty} h(x) = 0$，$h(0) \approx 1300$，且从直方图 h(x)的变化趋势来看，变化趋势非常接近指数函数，假设 $h(x) \approx ke^{-mx} + n(m, n, k > 0)$。

虽然 Q(x)是$[0, +\infty) \to \{0,1,2,3,4,5,6,7\}$的映射，但是在 Q(x)的设计上，先按照 Q(x)是连续函数进行分析，再四舍五入进行取整。有 h(x)的假设可知，令 Q(x)为指数函数时能够满足：Q(x)的变化率（导数）越大，h(x)越大。因此，设

$$Q(x) = -ae^{-cx} + b \qquad (a, b, c > 0)$$

考虑到 Q(x)的离散属性，令 $Q(x):[0, +\infty) \to [-0.5, 7.5)$，实际 Q(x) = round(Q(x))。所以有，

$$\begin{cases} Q(0) = -ae^{-c\cdot 0} + b = -0.5 \\ Q(+\infty) = -ae^{-c\cdot+\infty} + b = 7.5 \end{cases}$$

解得 $a = 8, b = 7.5$.所以 $Q(x) = \text{round}(-8e^{-cx} + 7.5),\ c > 0$. 其中 c 是唯一需要调节的超参数。

为了方便存储，令 Q(x)始终为正数，所以 $x \geq 0$ 时，$(x) = \text{round}(-8e^{-cx} + 7.5)+8$. 解码时 $x = \text{round}[-\frac{1}{c}\ln(\frac{7.5-(Q(x)-8)}{8})]$.

同理，$x < 0$ 时，$Q(x) = \text{round}(-8e^{cx} + 7.5),\ c > 0$.解码时 $x = \text{round}[\frac{1}{c}\ln(\frac{7.5-Q(x)}{8})]$.

在编程过程中，为防止机器计算的舍入误差，令 Q(x)不超过[-0.5+1e-8, 7.5-1e-8]的范围。代码如下，method = 'e' 是该指数映射方法编码的情况。

```
# c 为正数，要返回正数
def Q_e(x, c):
    if x >= 0:
        return round(np.clip(-8 * np.exp(-c * x) + 7.5, -0.5+1e-8, 7.5-1e-8)) + 8
    else:
        return round(np.clip(-8 * np.exp(c * x) + 7.5, -0.5+1e-8, 7.5-1e-8))


# 两种压缩成 4bit 的方式：量化因子法或指数映射的方法
def compress_to_4bit(wave_data, a_or_c, Q, method):
    # 第一个采样值占 16bits，如果不整的话多 4bit，值为 0
```

```
    new_data = np.zeros(int(np.ceil((len(wave_data)-1)/2) + 2), dtype=np.uint8)
    # 把第一个采样值按二进制的前 8 位和后 8 位分别存储在 new_data[-1]和 new_data[0]中
    new_data[-1] = np.int8((wave_data[0] + 32768) // 256)
    new_data[0] = np.int8((wave_data[0] + 32768) % 256)
    # wave_data 的索引 i, new_data 的索引 j, ceil(i // 2) = j
    x_ = int(wave_data[0])
    for i in range(1, len(wave_data)):
        j = int(np.ceil(i/2))
        d = int(wave_data[i]) - int(x_)
        c = Q(d, a_or_c)
        if method == 'a':
            x_ += (c - 8) * a_or_c
        elif method == 'e':
            if c >= 8:
                x_ += round(-np.log((7.5 - np.abs(c - 8)) / 8) / a_or_c)
            else:
                x_ += round(np.log((7.5 - np.abs(c)) / 8) / a_or_c)
        if i % 2 == 1:
            new_data[j] = c * 16
        else:
            new_data[j] += c
    if (len(wave_data)-1)/2 % 2 != 0:
        new_data[-2] += 8
    return new_data
```

```
def decompress_4bit(wave_data, a_or_c, method):
    decompress_data = np.zeros(2 * (len(wave_data) - 2) + 1, dtype=np.int16)
    decompress_data[0] = wave_data[0] + wave_data[-1] * 256 - 32768
    for i in range(1, len(decompress_data)):
        index = int(np.ceil(i/2))
        if i % 2 == 1:
            c = wave_data[index] // 16
        else:
            c = wave_data[index] % 16
        if method == 'a':
            decompress_data[i] = decompress_data[i-1] + (c - 8) * a_or_c
        elif method == 'e':
            if c >= 8:
                d = round(-np.log((7.5 - np.abs(c - 8)) / 8) / a_or_c)
            else:
                d = round(np.log((7.5 - np.abs(c)) / 8) / a_or_c)
            decompress_data[i] = decompress_data[i-1] + d
    return decompress_data
```

三、 计算检测正确率

3.1 信噪比(SNR)算法

信噪比，英文名称叫做 SNR 或 S/N（SIGNAL-NOISE RATIO）。是指一个电子设备或者电子系统中信号与噪声的比例。这里面的信号指的是来自设备外部需要通过这台设备进行处理的电子信号，噪声是指经过该设备后产生的原信号中并不存在的无规则的额外信号（或信息），并且该种信号并不随原信号的变化而变化。计算公式如下：

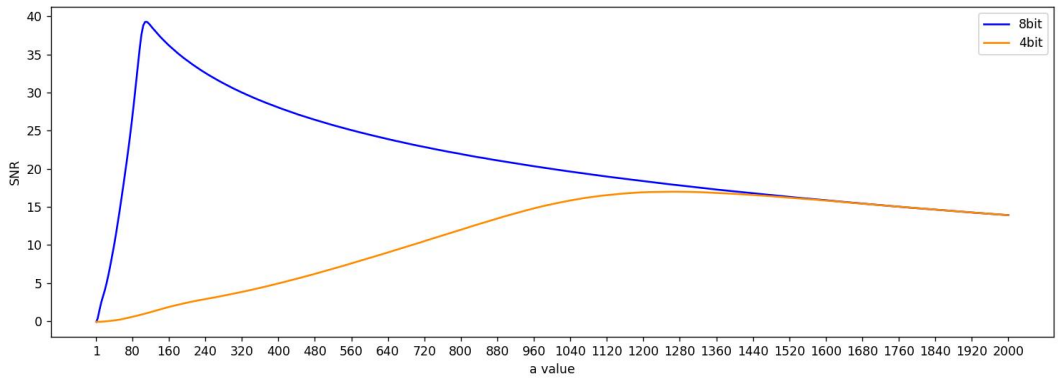
$$SNR = 10 * \log_{10}\left\{\frac{\sum_{n=0}^{M}(x(n))^2}{\sum_{n=0}^{M}(\overline{x}(n) - x(n))^2}\right\}$$

由于计算平方和计算过慢，采用向量 2-范数的平方替代，代码如下：

```
def ComputeSNR(wave_data, decompress_data):
    if len(wave_data) != len(decompress_data):
        decompress_data = decompress_data[:-1]
    result = 10 * np.log10((np.linalg.norm(wave_data) ** 2) /
                           (np.linalg.norm(np.abs(wave_data - decompress_data)) ** 2))
    return result
```

3.2 信噪比与因子取值

①量化因子法信噪比随因子取值的变化



以 2.wav 为例，对于量化因子法而言：压缩为 8bit 时，a = 106 时 SNR 取最大值 39.31；压缩为 4bit 时，a = 1280 时 SNR 取最大值 17.06.

因此在压缩过程中，压缩为 8bit 时取 a = 106，压缩为 4bit 时取 a = 1280.

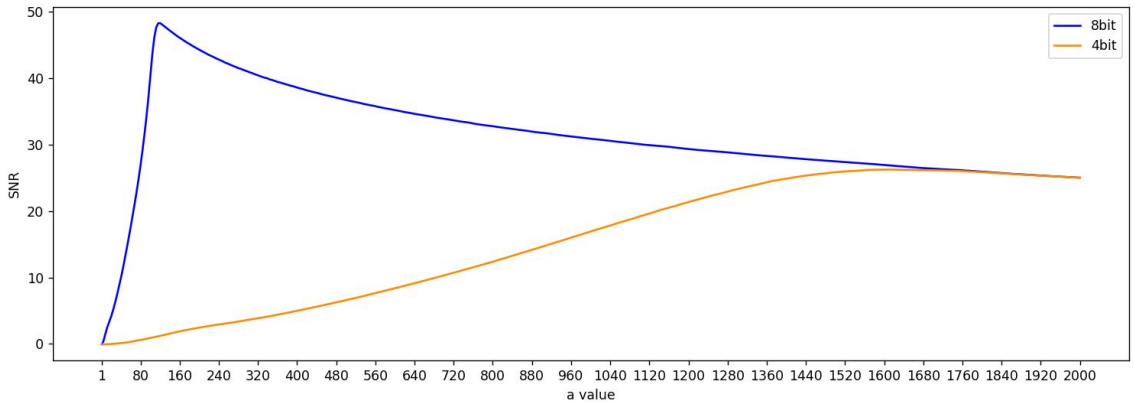
②改进量化因子法信噪比随因子取值的变化

这里的改进方法指的是对 Q(x)的改进，Q(x)由

$$c(n)=\begin{cases}7 & d(n)>7a\\ -8 & d(n)<-8a\\ i & (i-1)a<d(n)\le ia,-8\le i\le 7\end{cases} \quad \Rightarrow \quad c(n)=c(n)+8$$

改进为（即$-8a \le x \le 7a$ 时，不直接取 i，而取 i 和(i-1)中距离较近的一个）：

```
# a 为正数，要返回正数
def Q_4bit(x, a):
    if x > 7 * a:
        return 15
    elif x < -8 * a:
        return 0
    else:
        return round(x/a) + 8
```

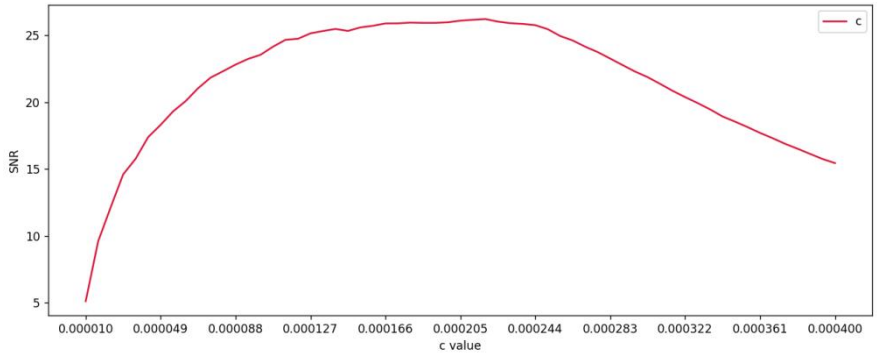


以 2.wav 为例，对于改进量化因子法而言：压缩为 8bit 时，a = 115 时 SNR 取最大值 48.40；压缩为 4bit 时，a = 1600 时 SNR 取最大值 26.10. 从中可以看出，改进量化因子法的压缩效果更好，信噪比有明显的提升。

因此在压缩过程中，压缩为 8bit 时取 a = 115，压缩为 4bit 时取 a = 1600.

③指数映射法信噪比随因子取值的变化

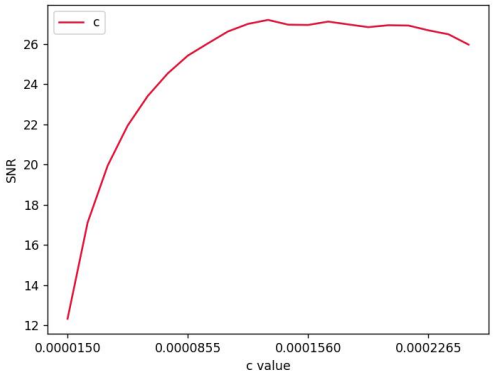
在本方案改进的指数映射方法中，因子指的是映射函数 $Q(x) = round(-8e^{-cx} + 7.5)$中的 c.



以 2.wav 为例，对于指数映射法而言：c = 2.18e-4 时 SNR 取最大值 26.25. 从中可以看出，该指数映射方法相比于量化因子法的压缩效果更好，信噪比有明显的提升。 但是和改进的量化因子法压缩效果和信噪比相近，略有提高。

因此在压缩过程中，取 c = 2.18e-4.

指数映射方法信噪比没有大幅度提高可能是因为：0 的解码值并非是 0，而是一个有符号的较小的数，虽然避免了这个区间较大的数被解码成 0，但是也容易使得静音段时间较长的音频，静音被放大出现噪声。因此，用实验 1 端点检测去除静音后的 2.wav 进行测试，结果如下：



从中可以看出，c = 1.33e-4 时 SNR 取最大值 27.22，信噪比确实有所提高，能够高于改进量化因子法的效果。

3.2 正确率听辨判断

正确检出文件的个数：30

正确率= 100 %

对每个音频文件，8bit、4bit 量化因子法和指数映射方法编解码后，都能够清晰听辨语音说话内容。其中 8bit 编解码变化少、接近原声。指数映射方法相比 4bit 量化因子法，主观上听起来声音更自然一点，语音段的机械感和刺啦声少一些，但是静音段会对环境噪声保留和轻微放大。

四、 局限性、收获与建议

4.1 局限性分析

指数映射方法在静音段的表现有待提高，会轻微放大一些环境噪声。可能是因为在这个方法中：0 的解码值并非是 0，而是一个有符号的较小的数，虽然避免了这个区间较大的数被解码成 0，但是也容易使得静音段时间较长的音频，静音被放大出现噪声。

4.2 请总结本次实验的收获

在本次实验中，我使用 python 软件进行 DPCM 编解码，体会了相邻音频采样值的高度相关性，理解了充分运用音频文件特征进行压缩的思想。此外，通过信噪比的多次计算，我通过不断实验得出了最佳因子取值，掌握了超参数确定的一种方法。同时，在实验中通过观测差分值特征并建立数学模型，自己设计了一种压缩方法，拓宽了思路，增强了理论运用能力和理解的深度，收获很大。

4.3 请给出对本次实验内容的建议

本实验难度适中，设计任务合理，编解码方式让自己发挥并验证效果很有意思，感谢老师的实验设计和答疑解惑。