

**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)

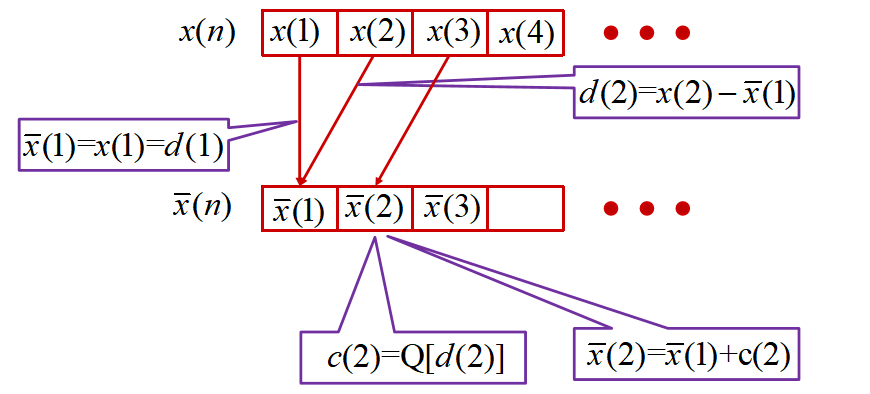
        f.close()

# 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：



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：



为方便存储，令c(n)不为负数，. 不过在实际的编码过程中，当时取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]

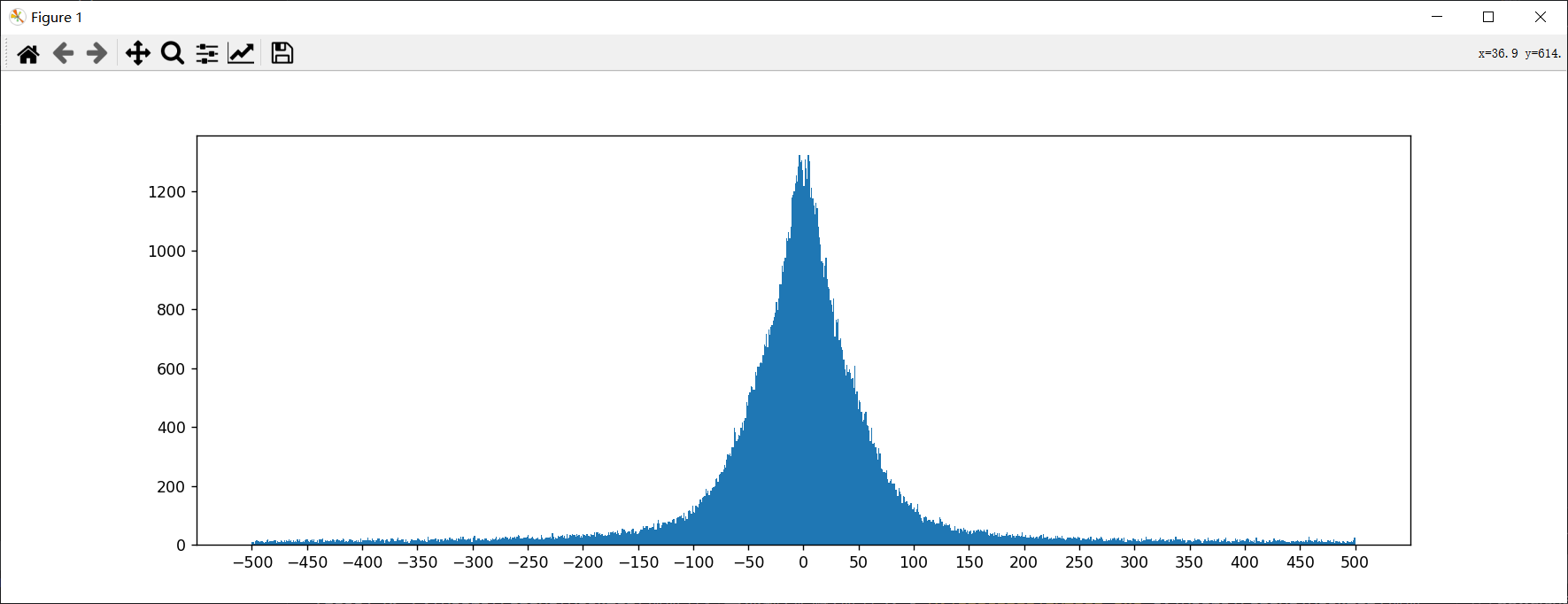
    d = np.linspace(-500,500,1001).astype(int)

    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。

为了使编解码后的音频损失最小(信噪比最大)，实际上应使：

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x0附近的x被映射到Q(x0)。**

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。

以为例，，，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。

虽然Q(x)是的映射，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

考虑到Q(x)的离散属性，令Q(x):，实际Q(x) = round(Q(x))。所以有，

解得所以. 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，所以时，+8. 解码时)].

同理，时，.解码时)].

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

# 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：



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]

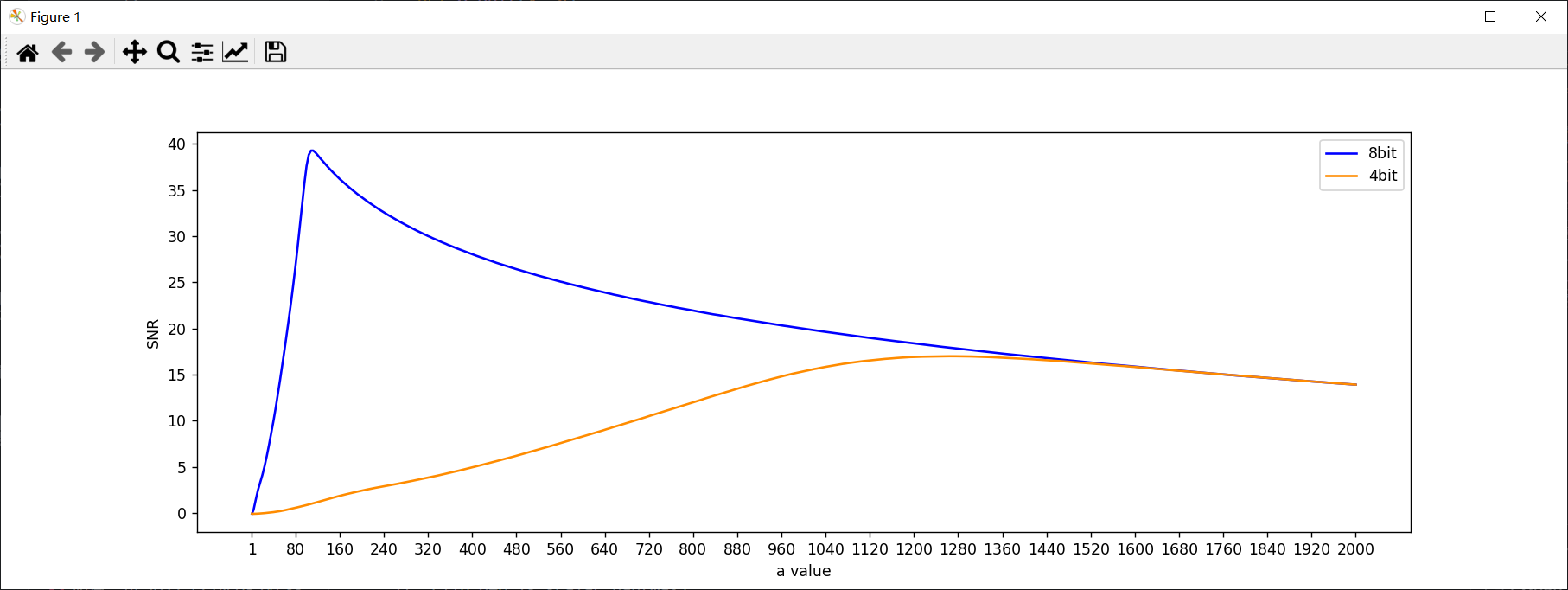
    result = 10 \* np.log10((np.linalg.norm(wave\_data) \*\* 2) /

                           (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**

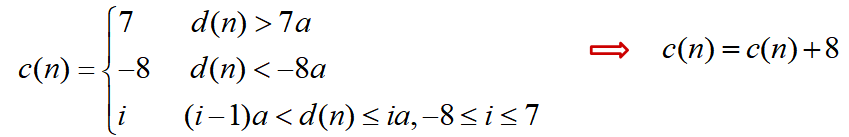


以2.wav为例，对于量化因子法而言：压缩为8bit时，a = 106时SNR取最大值39.31；压缩为4bit时，a = 1280时SNR取最大值17.06.

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



改进为（即时，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:

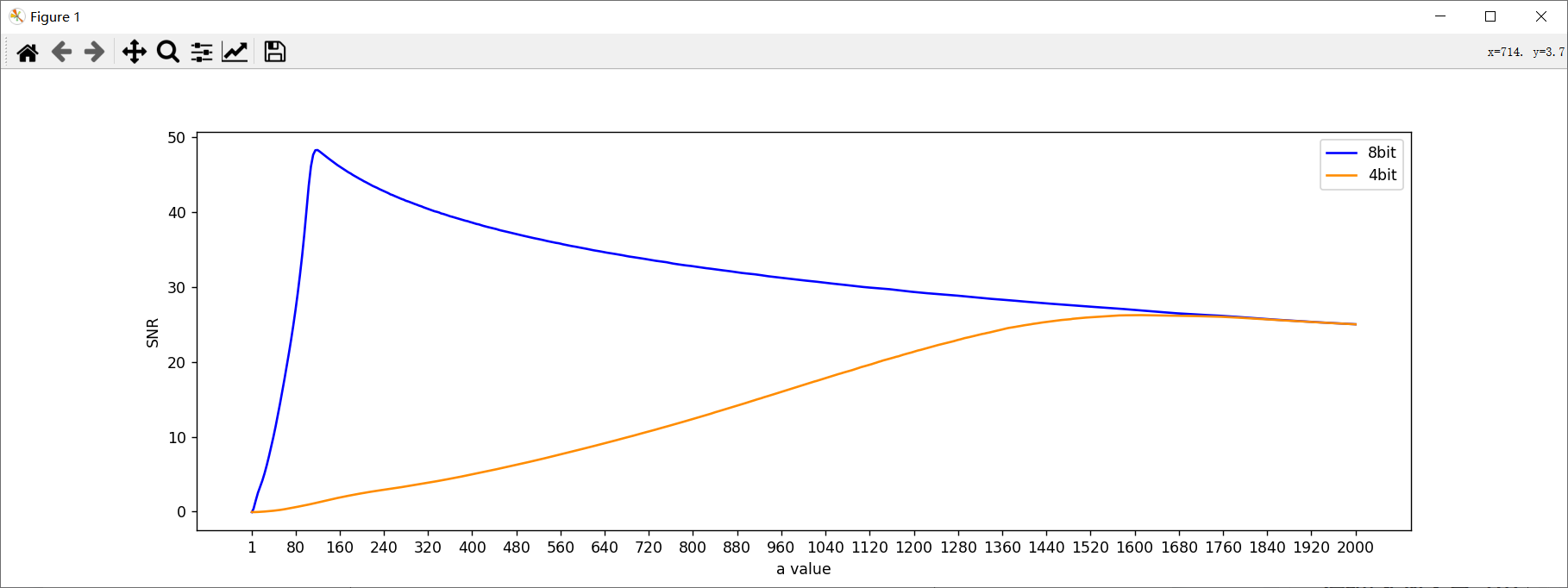
        return 15

    elif x < -8 \* a:

        return 0

    else:

        return round(x/a) + 8

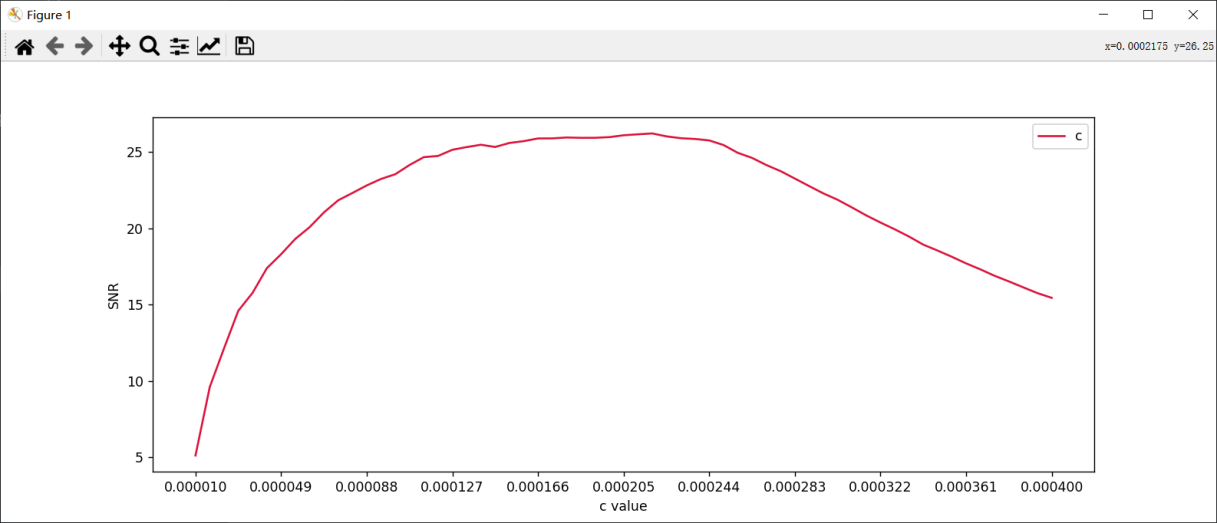


以2.wav为例，对于改进量化因子法而言：压缩为8bit时，a = 115时SNR取最大值48.40；压缩为4bit时，a = 1600时SNR取最大值26.10. 从中可以看出，改进量化因子法的压缩效果更好，信噪比有明显的提升。

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**

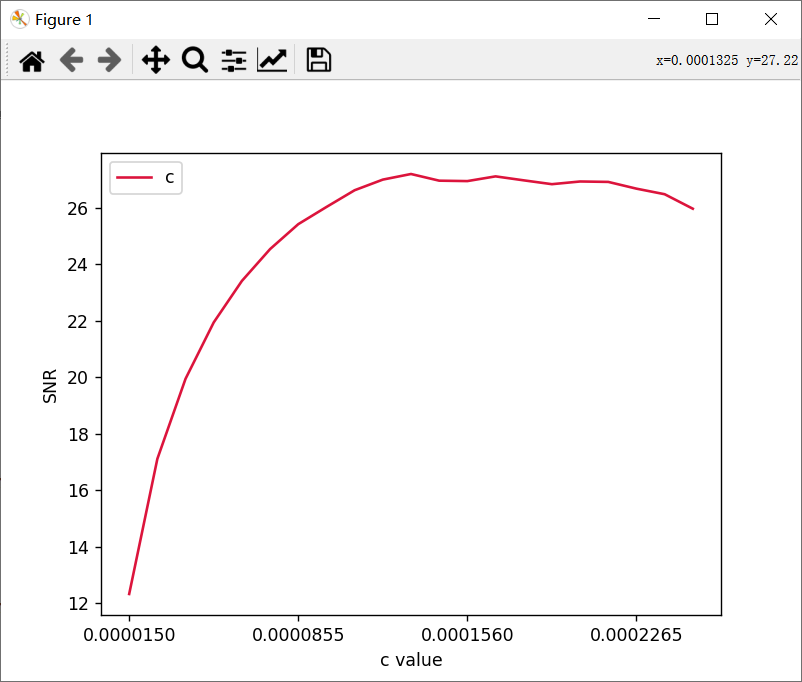
在本方案改进的指数映射方法中，因子指的是映射函数中的c.



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。

# 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。