**江 西 理 工 大 学**

**语音信号处理 实验报告**

**实验名称 实验二 语音信号时域分析**

**日 期 2023.03 专 业 智能科学与技术 班 级 智科201班**

**实验人 吴佳龙 学 号 2420203236 同组人**

**一、实验目的**

1、了解语音信号短时时域分析的原理。

2、了解短时时域分析的一些参数计算方法。

3、根据原理能编程实现短时时域分析的参数计算。

**二、实验原理**

语音信号的时域分析就是分析和提取语音信号的时域参数。语音信号本身就是时域信号，这种分析方法直接利用语音信号的时域波形。

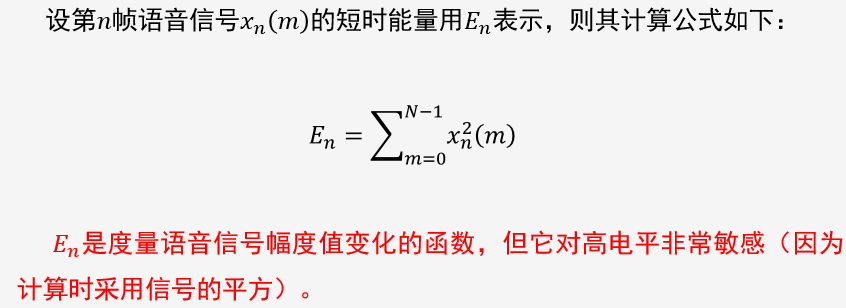
时域分析方法的特点是：

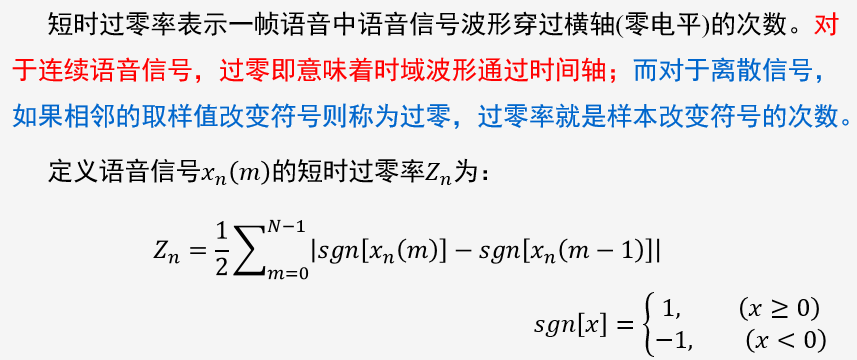
①表示语音信号比较直观、物理意义明确。

②实现起来比较简单、运算量少。

③可以得到语音的一些重要的参数。

④只使用示波器等通用设备，使用较为简单等。





**三、实验设备及环境**

1、设备：笔记本、台式电脑

2、实验环境：Windows 7、Windows10或Linux操作系统

MATLAB/Python3.6+，Anaconda3，Pycharm2017+

**四、实验内容与步骤**

**1、实验内容：**

1）为了显示方便，编程实现FrameTimeC函数，函数的功能为计算分帧后每帧语音中点处对应的时间。函数定义如下：

输入参数：frameNum帧的个数，framelen帧长，inc帧移，fs采样频率；

输出参数：frametime分针后每帧对应的时间。

2）根据reference.wav，编程实现短时能量，短时过零率并利用python或matlab绘图。

**五、实验注意事项**

1、实验中应遵守实验室实验规则，爱护实验室设备，保持卫生清洁；各种实验仪器设备在观察使用后放归原处，不得损坏或随意放置。

**六、思考题**

1、语音信号预处理中预加重和分帧的目的是什么？

预加重的目的是为了对语音的高频部分进行加重，去除口唇辐射的影响，增加语音的高频分辨率。因为高频端大约在800Hz以上按6dB/oct (倍频程)衰减，频率越高相应的成分越小，为此要在对语音信号进行分析之前对其高频部分加以提升。

分帧是为了将不平稳的语音信号分成若干个短时平稳的小段，以便进行傅里叶变换或其他处理。

2、请写出汉明窗和汉宁窗的窗函数表达式？

汉明窗：

汉宁窗：

**七、实验程序及结果展示**

|  |
| --- |
| 1) #-coding:utf-8-\*-  from scipy.io import wavfile  import matplotlib.pyplot as plt  from windows import \*  from timefeature import \*  from soundBase import \*  import matplotlib as mpl  plt.rcParams['font.family'] = ['sans-serif']  plt.rcParams['font.sans-serif'] = ['SimHei']  mpl.rcParams['axes.unicode\_minus'] = False # 解决保存图像是负号'-'显示为方块的问题  data, fs, bits = soundBase('reference.wav').audioread()  inc = 100  wlen = 200  win = hanning\_window(wlen)  N = len(data)  time = [i / fs for i in range(N)]  EN = STEn(data, win, inc) # 短时能量  Mn = STMn(data, win, inc) # 短时平均幅度  Zcr = STZcr(data, win, inc) # 短时过零率  X = enframe(data, win, inc)  X = X.T  Ac = STAc(X)  Ac = Ac.T  Ac = Ac.flatten()  Amdf = STAmdf(X)  Amdf = Amdf.flatten()  fig = plt.figure(figsize=(14, 13))  plt.subplot(3, 1, 1)  plt.plot(time, data)  plt.title('(a)语音波形')  plt.subplot(3, 1, 2)  frameTime = FrameTimeC(len(EN), wlen, inc, fs)  plt.plot(frameTime, Mn)  plt.title('(b)短时幅值')  plt.subplot(3, 1, 3)  plt.plot(frameTime, EN)  plt.title('(c)短时能量')  plt.savefig('images/energy.png')  fig1 = plt.figure(figsize=(10, 13))  plt.subplot(2, 1, 1)  plt.plot(time, data)  plt.title('(a)语音波形')  plt.subplot(2, 1, 2)  plt.plot(frameTime, Zcr)  plt.title('(b)短时过零率')  plt.savefig('images/Zcr.png')  import pyaudio  import wave  import librosa  import librosa.display  import matplotlib.pyplot as plt  from scipy.io import wavfile  import numpy as np  import pandas as pd  from scipy.signal import lfilter  class soundBase:  def \_\_init\_\_(self, path):  self.path = path  def audiorecorder(self, len=2, formater=pyaudio.paInt16, rate=16000, frames\_per\_buffer=1024, channels=2):  """  使用麦克风进行录音  2020-2-25 Jie Y. Init  :param len: 录制时间长度(秒)  :param formater: 格式  :param rate: 采样率  :param frames\_per\_buffer:  :param channels: 通道数  :return:  """  p = pyaudio.PyAudio()  stream = p.open(format=formater, channels=channels, rate=rate, input=True, frames\_per\_buffer=frames\_per\_buffer)  print("start recording......")  frames = []  for i in range(0, int(rate / frames\_per\_buffer \* len)):  data = stream.read(frames\_per\_buffer)  frames.append(data)  print("stop recording......")  stream.stop\_stream()  stream.close()  p.terminate()  wf = wave.open(self.path, 'wb')  wf.setnchannels(channels)  wf.setsampwidth(p.get\_sample\_size(formater))  wf.setframerate(rate)  wf.writeframes(b''.join(frames))  wf.close()  def audioplayer(self, frames\_per\_buffer=1024):  """  播放语音文件  2020-2-25 Jie Y. Init  :param frames\_per\_buffer:  :return:  """  wf = wave.open(self.path, 'rb')  p = pyaudio.PyAudio()  stream = p.open(format=p.get\_format\_from\_width(wf.getsampwidth()),  channels=wf.getnchannels(),  rate=wf.getframerate(),  output=True)  data = wf.readframes(frames\_per\_buffer)  while data != b'':  stream.write(data)  data = wf.readframes(frames\_per\_buffer)  stream.stop\_stream()  stream.close()  p.terminate()  def audiowrite(self, data, fs, binary=True, channel=1, path=[]):  """  信息写入到.wav文件中  :param data: 语音信息数据  :param fs: 采样率(Hz)  :param binary: 是否写成二进制文件(只有在写成二进制文件才能用audioplayer播放)  :param channel: 通道数  :param path: 文件路径，默认为self.path的路径  :return:  """  if len(path) == 0:  path = self.path  if binary:  wf = wave.open(path, 'wb')  wf.setframerate(fs)  wf.setnchannels(channel)  wf.setsampwidth(2)  wf.writeframes(b''.join(data))  else:  wavfile.write(path, fs, data)  def audioread(self, return\_nbits=False, formater='sample'):  """  读取语音文件  这里的wavfile.read()函数修改了里面的代码，返回项return fs, data 改为了return fs, data, bit\_depth  如果这里报错，可以将wavfile.read()修改。  :param formater: 获取数据的格式，为sample时，数据为loat32的，[-1,1]，同matlab同名函数. 否则为文件本身的数据格式  指定formater为任意非sample字符串，则返回原始数据。  :return: 语音数据data, 采样率fs，数据位数bits  """  fs, data, bits = wavfile.read(self.path)  if formater == 'sample':  data = data / (2 \*\* (bits - 1))  if return\_nbits:  return data, fs, bits  else:  return data, fs, bits  def soundplot(self, data=[], sr=16000, size=(14, 5)):  """  将语音数据/或读取语音数据并绘制出来  2020-2-25 Jie Y. Init  :param data: 语音数据  :param sr: 采样率  :param size: 绘图窗口大小  :return:  """  if len(data) == 0:  data, fs, \_ = self.audioread()  plt.figure(figsize=size)  x = [i / sr for i in range(len(data))]  plt.plot(x, data)  plt.xlim([0, len(data) / sr])  plt.xlabel('s')  plt.show()  def sound\_add(self, data1, data2):  """  将两个信号序列相加，若长短不一，在短的序列后端补零  :param data1: 序列1  :param data2: 序列2  :return:  """  if len(data1) < len(data2):  tmp = np.zeros([len(data2)])  for i in range(len(data1)):  tmp[i] += data1[i]  return tmp + data2  elif len(data1) > len(data2):  tmp = np.zeros([len(data1)])  for i in range(len(data2)):  tmp[i] += data2[i]  return tmp + data1  else:  return data1 + data2  def SPL(self, data, fs, frameLen=100, isplot=True):  """  计算声压曲线  2020-2-26 Jie Y. Init  :param data: 语音信号数据  :param fs: 采样率  :param frameLen: 计算声压的时间长度(ms单位)  :param isplot: 是否绘图，默认是  :return: 返回声压列表spls  """  def spl\_cal(s, fs, frameLen):  """  根据数学公式计算单个声压值  $y=\sqrt(\sum\_{i=1}^Nx^2(i))$  2020-2-26 Jie Y. Init  :param s: 输入数据  :param fs: 采样率  :param frameLen: 计算声压的时间长度(ms单位)  :return: 单个声压数值  """  l = len(s)  M = frameLen \* fs / 1000  if not l == M:  exit('输入信号长度与所定义帧长不等！')  # 计算有效声压  pp = 0  for i in range(int(M)):  pp += (s[i] \* s[i])  pa = np.sqrt(pp / M)  p0 = 2e-5  spl = 20 \* np.log10(pa / p0)  return spl  length = len(data)  M = fs \* frameLen // 1000  m = length % M  if not m < M // 2:  # 最后一帧长度不小于M的一半  data = np.hstack((data, np.zeros(M - m)))  else:  # 最后一帧长度小于M的一半  data = data[:M \* (length // M)]  spls = np.zeros(len(data) // M)  for i in range(len(data)// M - 1):  s = data[i \* M:(i + 1) \* M]  spls[i] = spl\_cal(s, fs, frameLen)  if isplot:  plt.subplot(211)  plt.plot(data)  plt.subplot(212)  plt.step([i for i in range(len(spls))], spls)  plt.show()  return spls  def iso226(self, phon, isplot=True):  """  绘制等响度曲线，输入响度phon  2020-2-26 Jie Y. Init  :param phon: 响度值0~90  :param isplot: 是否绘图，默认是  :return:  """  ## 参数来源: 语音信号处理试验教程，梁瑞宇P36-P37  f = [20, 25, 31.5, 40, 50, 63, 80, 100, 125, 160, 200, 250, 315, 400, 500, 630, 800, \  1000, 1250, 1600, 2000, 2500, 3150, 4000, 5000, 6300, 8000, 10000, 12500]  af = [0.532, 0.506, 0.480, 0.455, 0.432, 0.409, 0.387, 0.367, 0.349, 0.330, 0.315, \  0.301, 0.288, 0.276, 0.267, 0.259, 0.253, 0.250, 0.246, 0.244, 0.243, 0.243, \  0.243, 0.242, 0.242, 0.245, 0.254, 0.271, 0.301]  Lu = [-31.6, - 27.2, - 23.0, - 19.1, - 15.9, - 13.0, - 10.3, - 8.1, - 6.2, - 4.5, - 3.1, \  - 2.0, - 1.1, - 0.4, 0.0, 0.3, 0.5, 0.0, - 2.7, - 4.1, - 1.0, 1.7, \  2.5, 1.2, - 2.1, - 7.1, - 11.2, - 10.7, - 3.1]  Tf = [78.5, 68.7, 59.5, 51.1, 44.0, 37.5, 31.5, 26.5, 22.1, 17.9, 14.4, \  11.4, 8.6, 6.2, 4.4, 3.0, 2.2, 2.4, 3.5, 1.7, - 1.3, - 4.2, \  - 6.0, - 5.4, - 1.5, 6.0, 12.6, 13.9, 12.3]  if phon < 0 or phon > 90:  print('Phon value out of range!')  spl = 0  freq = 0  else:  Ln = phon  # 从响度级计算声压级  Af = 4.47E-3 \* (10 \*\* (0.025 \* Ln) - 1.15) + np.power(0.4 \* np.power(10, np.add(Tf, Lu) / 10 - 9), af)  Lp = np.multiply(np.divide(10, af), np.log10(Af)) - Lu + 94  spl = Lp  freq = f  if isplot:  plt.semilogx(freq, spl, ':k')  plt.axis([20, 20000, -10, 130])  plt.title('Phon={}'.format(phon))  plt.grid()  plt.show()  return spl, freq  def vowel\_generate(self, len, pitch=100, sr=16000, f=[730, 1090, 2440]):  """  生成一个元音片段  2020-2-26 Jie Y. Init  :param len: 长度，点数  :param pitch:  :param sr: 采样率  :param f: 前3个共振峰，默认为元音a的  :return: 生成的序列  """  f1, f2, f3 = f[0], f[1], f[2]  y = np.zeros(len)  points = [i for i in range(0, len, sr // pitch)]  indices = np.array(list(map(int, np.floor(points))))  y[indices] = (indices + 1) - points  y[indices + 1] = points - indices  a = np.exp(-250 \* 2 \* np.pi / sr)  y = lfilter([1], [1, 0, -a \* a], y)  if f1 > 0:  cft = f1 / sr  bw = 50  q = f1 / bw  rho = np.exp(-np.pi \* cft / q)  theta = 2 \* np.pi \* cft \* np.sqrt(1 - 1 / (4 \* q \* q))  a2 = -2 \* rho \* np.cos(theta)  a3 = rho \* rho  y = lfilter([1 + a2 + a3], [1, a2, a3], y)  if f2 > 0:  cft = f2 / sr  bw = 50  q = f2 / bw  rho = np.exp(-np.pi \* cft / q)  theta = 2 \* np.pi \* cft \* np.sqrt(1 - 1 / (4 \* q \* q))  a2 = -2 \* rho \* np.cos(theta)  a3 = rho \* rho  y = lfilter([1 + a2 + a3], [1, a2, a3], y)  if f3 > 0:  cft = f3 / sr  bw = 50  q = f3 / bw  rho = np.exp(-np.pi \* cft / q)  theta = 2 \* np.pi \* cft \* np.sqrt(1 - 1 / (4 \* q \* q))  a2 = -2 \* rho \* np.cos(theta)  a3 = rho \* rho  y = lfilter([1 + a2 + a3], [1, a2, a3], y)  plt.plot(y)  plt.show()  return y  import numpy as np  def enframe(x, win, inc=None):  nx = len(x)  if isinstance(win, list) or isinstance(win, np.ndarray):  nwin = len(win)  nlen = nwin # 帧长=窗长  elif isinstance(win, int):  nwin = 1  nlen = win # 设置为帧长  if inc is None:  inc = nlen  nf = (nx - nlen + inc) // inc  frameout = np.zeros((nf, nlen))  indf = np.multiply(inc, np.array([i for i in range(nf)]))  for i in range(nf):  frameout[i, :] = x[indf[i]:indf[i] + nlen]  if isinstance(win, list) or isinstance(win, np.ndarray):  frameout = np.multiply(frameout, np.array(win))  return frameout  def STAc(x):  """  计算短时相关函数  :param x:  :return:  """  para = np.zeros(x.shape)  fn = x.shape[1]  for i in range(fn):  R = np.correlate(x[:, i], x[:, i], 'valid')  para[:, i] = R  return para  def STEn(x, win, inc):  """  计算短时能量函数  :param x:  :param win:  :param inc:  :return:  """  X = enframe(x, win, inc)  s = np.multiply(X, X)  return np.sum(s, axis=1)  def STMn(x, win, inc):  """  计算短时平均幅度计算函数  :param x:  :param win:  :param inc:  :return:  """  X = enframe(x, win, inc)  s = np.abs(X)  return np.mean(s, axis=1)  def STZcr(x, win, inc, delta=0):  """  计算短时过零率  :param x:  :param win:  :param inc:  :return:  """  absx = np.abs(x)  x = np.where(absx < delta, 0, x)  X = enframe(x, win, inc)  X1 = X[:, :-1]  X2 = X[:, 1:]  s = np.multiply(X1, X2)  sgn = np.where(s < 0, 1, 0)  return np.sum(sgn, axis=1)  def STAmdf(X):  """  计算短时幅度差，好像有点问题  :param X:  :return:  """  # para = np.zeros(X.shape)  fn = X.shape[1]  wlen = X.shape[0]  para = np.zeros((wlen, wlen))  for i in range(fn):  u = X[:, i]  for k in range(wlen):  en = len(u)  para[k, :] = np.sum(np.abs(u[k:] - u[:en - k]))  return para  def FrameTimeC(frameNum, frameLen, inc, fs):  ll = np.array([i for i in range(frameNum)])  return ((ll - 1) \* inc + frameLen / 2) / fs  import numpy as np  def reg\_window(N):  return np.ones(N)  def hanning\_window(N):  nn = [i for i in range(N)]  return 0.5 \* (1 - np.cos(np.multiply(nn, 2 \* np.pi) / (N - 1)))  def hamming\_window(N):  nn = [i for i in range(N)]  return 0.54 - 0.46 \* np.cos(np.multiply(nn, 2 \* np.pi) / (N - 1))  结果图    实验结果分析  从图中可以看出，短时能量和短时过零率都随着时间变化而波动，反映了语音信号的动态特性。在有话段时，短时能量较高，短时过零率较低；在无话段或噪声段时，短时能量较低，短时过零率较高。这与语音信号的特点是一致的。因此，可以利用这两种特征来区分有话段和无话段，或者清音和浊音。 |