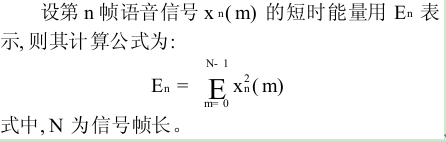
基于语音端点检测的方法有很多，从历史的发展来看。

首先是基于短时能量和短视过零率的端点检测=〉各变换域=〉人工神经网络=〉基于倒谱距离的检测算法=〉基于谱熵的方法=〉几何门限的方法=〉sigma函数=〉近些年的从分形技术和混沌理论引入的端点检测。

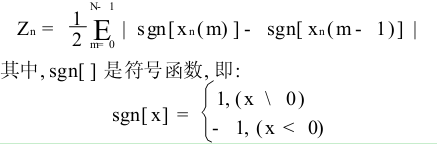
作为最早的短时能量与过零率的检测方法，比较简单，当然也有很大的缺陷：在SNR比较低的情况下，准确度大打折扣，严重的话会失效。以目前而言，先研究这个简单的再说。

**1，**     **实现原理。**

**1.1基于两个公式**，即短时能量和短时过零率。



过零率公式：

****

**1.2理论基础**：语音信号一般可分为无声段、清音段和浊音段。无声段是背景噪声段, 平均能量最低; 浊音段为声带振动发出对应的语音信号段, 平均能量最高; 清音段是空气在口腔中的摩擦、冲击或爆破而发出的语音信号段, 平均能量居于前两者之间。清音段和无声段的波形特点有明显的不同, 无声段信号变化较为缓慢, 而清音段信号在幅度上变化剧烈, 穿越零电平次数也多。经验表明, 通常清音段过零率最大。端点检测就是首先判断/ 有声0还是/ 无声0, 如果有声,则还要判断是/ 清音0还是/ 浊音0。为正确地实现端点检测, 一般综合利用短时能量和过零率两个特征,采用/ 双门限检测法0。

**1.3基本思路**：根据信号设置三个阈值：能量阈值，TL，TH；过零率阈值ZCR,当某帧信号大于TL或者大于ZCR时，认为信号的开始、起点，当大于TH时，则认为正式的语音信号，如果保持一段时间，则确认这信号即所需信号。

**2，matlab实现**

**2.1matlab语句分析**。

从网上下载了一份程序如下：

% 语音信号的端点检测matlab实现:

[x,fs,nbits]=wavread('E:\西瓜\koushao.wav');%首先打开经录好的信号，一段口哨声。

x = x / max(abs(x));%幅度归一化到[-1,1]

%参数设置

FrameLen = 256;     %帧长

inc = 90;           %未重叠部分，这里涉及到信号分帧的问题，在后边再解释。

amp1 = 10;          %短时能量阈值

amp2 = 2;           %即设定能量的两个阈值。

zcr1 = 10;          %过零率阈值

zcr2 = 5;                 %过零率的两个阈值，感觉第一个没有用到。

minsilence = 6;   %用无声的长度来判断语音是否结束

minlen  = 15;    %判断是语音的最小长度

status  = 0;      %记录语音段的状态

count   = 0;     %语音序列的长度

silence = 0;      %无声的长度

%计算过零率

tmp1  = enframe(x(1:end-1), FrameLen,inc);

tmp2  = enframe(x(2:end)  , FrameLen,inc);

signs = (tmp1.\*tmp2)<0;

diffs = (tmp1 - tmp2)>0.02;

zcr   = sum(signs.\*diffs,2);%虽然没搞懂上边的原理，但是可以推测存的是各桢的过零率。上边计算过零率的放到后边分析，这里只要了解通过这几句得到了信号各帧的过零率值，放到zcr矩阵中。

%计算短时能量

 %amp = sum((abs(enframe(filter([1 -0.9375], 1, x), FrameLen, inc))).^2, 2);%不知道这里的filter是干啥的？但的出来的是各贞的能量了。

amp = sum((abs(enframe( x, FrameLen, inc))).^2, 2);%通过把filter给去掉，发现结果差不多，所以个人感觉没必要加一个滤波器，上边出现的enframe函数放到后边分析。这里知道是求出x各帧的能量值就行。

%调整能量门限

amp1 = min(amp1, max(amp)/4);

amp2 = min(amp2, max(amp)/8);%min函数是求最小值的，没必要说了。

%开始端点检测

for n=1:length(zcr)%从这里开始才是整个程序的思路。Length（zcr）得到的是整个信号的帧数。

   goto = 0;

   switch status

   case {0,1}                   % 0 = 静音, 1 = 可能开始

      if amp(n) > amp1          % 确信进入语音段

         x1 = max(n-count-1,1); % 记录语音段的起始点

         status  = 2;

         silence = 0;

         count   = count + 1;

      elseif amp(n) > amp2 || zcr(n) > zcr2 % 可能处于语音段

         status = 1;

         count  = count + 1;

      else                       % 静音状态

         status  = 0;

         count   = 0;

      end

   case 2,                       % 2 = 语音段

      if amp(n) > amp2 ||zcr(n) > zcr2     % 保持在语音段

         count = count + 1;

      else                       % 语音将结束

         silence = silence+1;

         if silence < minsilence % 静音还不够长，尚未结束

            count  = count + 1;

         elseif count < minlen   % 语音长度太短，认为是噪声

            status  = 0;

            silence = 0;

            count   = 0;

         else                    % 语音结束

            status  = 3;

         end

      end

   case 3,

      break;

   end

end

count = count-silence/2;

x2 = x1 + count -1;              %记录语音段结束点

%后边的程序是找出语音端，然后用红线给标出来，没多少技术含量，就不多说了。

subplot(3,1,1)

plot(x)

axis([1 length(x) -1 1])%限制x轴与y轴的范围。

ylabel('Speech');

line([x1\*inc x1\*inc], [-1 1], 'Color', 'red');

line([x2\*inc x2\*inc], [-1 1], 'Color', 'red');%注意下line函数的用法：基于两点连成一条直线，就清楚了。

subplot(3,1,2)

plot(amp);

axis([1 length(amp) 0 max(amp)])

ylabel('Energy');

line([x1 x1], [min(amp),max(amp)], 'Color', 'red');

line([x2 x2], [min(amp),max(amp)], 'Color', 'red');

subplot(3,1,3)

plot(zcr);

axis([1 length(zcr) 0 max(zcr)])

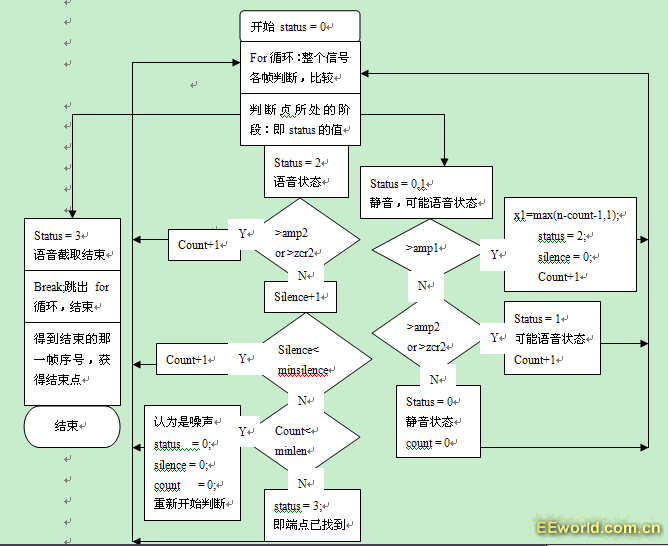
ylabel('ZCR');

line([x1 x1], [min(zcr),max(zcr)], 'Color', 'red');

line([x2 x2], [min(zcr),max(zcr)], 'Color', 'red');

**2.2由语句提出的程序流程**。

由上边的程序可以看出程序流程为：



**3，mat lab程序中的部分解释说明**

**3.1流程图的说明**

       amp2,amp1为能量的两个阈值，前者为小的，后者为大的，zcr2为过零率的阈值小值，当>amp2 or >zcr2，count开始加1，在此期间如果有不满足该条件的话，count立即为0，回到0状态。如果>amp1 时，count加1，然后进入2状态。在2状态里边，当>amp2 or >zcr2，count也1，如果不满足条件，则Silence+1，如果Silence即在2状态期间处于静音状态满足结束时的静音条件，则判断所有计数的信号即count的值是否满足最小的语音信号长度值，如果满足，则找到结束点，否则认为是噪声，重新开始。如果Silence即在2状态期间处于静音状态不足结束时的静音条件，则count继续加1。

**3.2起点和终点的判断**

即判断X1,X2。根据程序x1=max(n-count-1,1);n为找到>amp1时，此时的贞的序列值。而count为在这之前的>amp2 or >zcr2，的贞的个数。一般突发信号从无慢慢到有，如果N=6，COUNT=2，则起点从第三帧开始。很好理解。

对于X2，有count = count-silence/2;x2 = x1 + count - 1;总count值为>amp2 or >zcr2开始，到判断超过最小静音为止。而当进入silence加的时候，已经为静音阶段了。当silence=6时，便结束了，然后X2的计算方法，基本了解，但silence/2感觉可以不要也行，后边的-1也感觉可以不要。

**3.3 enframe函数**的说明

前边算能量，过零率都是基于帧来计算的，而enframe函数是用来把信号进行分帧的。

其代码如下：

function f=enframe(x,win,inc)%定义函数。

nx=length(x(:)); %x(:)的作用是把x给弄成一个向量，x为一行，则变成一列，如果为矩阵，则按每一列的顺序排成一列。得出的nx为序列的数据个数。

nwin=length(win);

if (nwin == 1)

    len = win; %如果win中就一个数，则len就=该数，此例中为256个点。即每帧长

else

    len = nwin; %如果有多个数，则len=个数。

end

if (nargin < 3) % nargin返回的是函数输入的个数，如果中间有变量，返回的是负值。

    inc = len; %也就是说，如果函数enframe的输入只有两个的话，系统就自动赋inc

end

nf = fix((nx-len+inc)/inc);%这个比较关键，nf为分帧的组数，结合下边的可以分析这里

                                    各参数的意义，len为帧长，inc为未覆盖的数据，nx为整个数

                     据量。假设数据为1：30，len为10，未覆盖为5，则nf=5，5

                     组，第一组为1，2，……10，第二组为6，7，……15，依次列

推，便可知其缘由，即（nx-len ）/inc + 1；

f=zeros(nf,len);             %构成以组数为行，帧长为列的矩阵。

indf= inc\*(0:(nf-1)).'; %indf为一列nf个数据，即0到nf-1的inc倍，即分好了每幀起点。

inds = (1:len);      %构成了长度为len的一行。

f(:) = x(indf(:,ones(1,len))+inds(ones(nf,1),:));

                            %上一条语句为整个算法的核心部分了，indf(:,ones(1,len))把indf的

                         第一列扩展了nf\*len的矩阵。同理inds(ones(nf,1),:)把inds第一行扩

展为nf\*len的矩阵，相加得到

[  1    2    3 …… len

inc+1 inc+2 inc+3 …… inc+len

2\*inc+1………… ……2\*inc+len

。

。…………………………………]然后就按照这个矩阵从x中把数据

给选出来，达到分帧的目的。

if (nwin > 1)    %nwin大于1的情况就不说了。

w = win(:)';

f = f .\* w(ones(nf,1),:);

end

**3.4过零率**的计算

其语句如下：

tmp1  = enframe(x(1:end-1), FrameLen,inc);

tmp2  = enframe(x(2:end)  , FrameLen,inc);

signs = (tmp1.\*tmp2)<0;%对于tmp1.\*tmp2算出来的矩阵，矩阵中<0的数都为1，其他

为0，后边也是一样的。

diffs = (tmp1 - tmp2)>0.02;

zcr = sum(signs.\*diffs,2);

假设数据量为1，2……21，帧长为10，inc为5，则

tmp1为[1 2 3 4 5 6 7 8 9 10

               6 7 8 9 10 11 12 13 14 15

                        11 12 13 14 15 16 17 18 19 20]

tmp2为[2 3 4 5 6 7 8 9 10 11

               7 8 9 10 11 12 13 14 15 16

                         12 13 14 15 16 17 18 19 20 21]

在这里注意一个问题：即数组的乘法与矩阵的乘法是不一样的。数组乘法：A.\*A,矩阵乘法：A\*A’。前者有‘.’号，算出来的结果是在矩阵A中每一个数据平方，而后者成为另一个数组，行与列相乘然后相加作为一个值。

在这里tmp1.\*tmp2为数组相乘，第一个数乘以第二个数，第二个数乘以第三个数，依次，从而判断两者的符号，<0的为1。

然后进行相减，第一个减第二个数，第二个减第三个数……，>0.02,为什么>0.02了？首先得到的signs是真正的过零率，但得限制能量，因为对于噪音的话，也会在0点附近上下摆动，但噪声能量显然是没有语音大的，根据实际情况，所以选择>0.02，其次感觉diffs = abs((tmp1 - tmp2))>0.02;%¸就是加个绝对值，因为对于负值-正值也会满足条件，结果验证也不影响。

**3.5能量**的计算

语句为：

amp = sum((abs(enframe( x, FrameLen, inc))).^2, 2);

通过对enframe函数的分析，就比较容易了，enframe对x分帧后，绝对值然后平方，最后是sum（x，2）2代表是各列相加最后得到的是一列数据，即各帧的平方和。从中可以看出矩阵处理数据的方便性，一个矩阵就把各帧的结果给弄出来了。

最后运行结果为图

