# ПРИЛОЖЕНИЕ

## CTCVoCoder1.m

Основная программа, реализующая кодирование и декодирование речевого сообщения.

```
s=myload('wav');

tic

Segments=floor(length(s)/160);
if Segments*160<length(s),
    Segments=Segments+1;
    s=[s zeros(1,Segments*160-length(s))];
end

max_L =length(s);
begin_frame = 1;
R1=zeros(1,max_L);
R2=zeros(1,max_L);
LARmas=zeros(1,max_L/160*8);
Gmas=zeros(1,max_L/160*4);
MaxAplphamas=zeros(1,max_L/160*4);

s_last=zeros(1,160);
for ii = 1:Segments,
  end_frame = 160*ii;
  if max_L < end_frame, end_frame = max_L; end;

  beg4=(ii-1)*4+1;
  end4=(ii-1)*4+4;
  [LARmas((ii-1)*8+1:(ii-1)*8+8), Gmas(beg4:end4), MaxAlphamas(beg4:end4),
R1(begin_frame:end_frame), R2(begin_frame:end_frame)] =
myfilter2lar(s(begin_frame:end_frame),s_last,8);
  s_last=s(begin_frame:end_frame);

  begin_frame = end_frame + 1;
end;

max_LARL=max_L/160*8;
max_LAR1=max(abs(LARmas(1:8:max_LARL)));
max_LAR2=max(abs(LARmas(2:8:max_LARL)));
max_LAR3=max(abs(LARmas(3:8:max_LARL)));
max_LAR4=max(abs(LARmas(4:8:max_LARL)));
max_LAR5=max(abs(LARmas(5:8:max_LARL)));
max_LAR6=max(abs(LARmas(6:8:max_LARL)));
max_LAR7=max(abs(LARmas(7:8:max_LARL)));
max_LAR8=max(abs(LARmas(8:8:max_LARL)));
max_LAR12=max([max_LAR1 max_LAR2]);
max_LAR34=max([max_LAR3 max_LAR4]);
max_LAR56=max([max_LAR5 max_LAR6]);
max_LAR78=max([max_LAR7 max_LAR8]);

GC_LAR12=31;
GC_LAR34=15;
GC_LAR56=7;
GC_LAR78=3;
LARmas(1:8:max_LARL)=round(LARmas(1:8:max_LARL)/max_LAR12*GC_LAR12);
LARmas(2:8:max_LARL)=round(LARmas(2:8:max_LARL)/max_LAR12*GC_LAR12);
LARmas(3:8:max_LARL)=round(LARmas(3:8:max_LARL)/max_LAR34*GC_LAR34);
LARmas(4:8:max_LARL)=round(LARmas(4:8:max_LARL)/max_LAR34*GC_LAR34);
LARmas(5:8:max_LARL)=round(LARmas(5:8:max_LARL)/max_LAR56*GC_LAR56);
LARmas(6:8:max_LARL)=round(LARmas(6:8:max_LARL)/max_LAR56*GC_LAR56);
LARmas(7:8:max_LARL)=round(LARmas(7:8:max_LARL)/max_LAR78*GC_LAR78);
LARmas(8:8:max_LARL)=round(LARmas(8:8:max_LARL)/max_LAR78*GC_LAR78);
```

```
Gmas=round(Gmas*7);
b=fir1(8,0.48);
R2_tmp1=filtfilt(b,1,R2);
R2_tmp2=R2_tmp1(1:2:max_L);
R2ADPCM=imaadpcm_en(int16(round(R2_tmp2*32766)));
clear R2_tmp1 R2_tmp2
R2_tmp2=double(imaadpcm_de(R2ADPCM))/32766;
R2_vos=zeros(1,max_L);
R2_vos(1:2:max_L)=R2_tmp2;
f=[0 .5 1];
m=[0.5 4 0.5];
b=fir2(8,f,m);
R2_vos=filtfilt(b,1,R2_vos);

figure(1),pwelch(s,[],[],[],1);
figure(2),pwelch(R1,[],[],[],1);
figure(3),pwelch(R2,[],[],[],1);
clear A G MaxAlpha n R2_tmp1 R2_tmp2

Gmas=Gmas/7;

LARmas(1:8:max_LARL)=LARmas(1:8:max_LARL)/GC_LAR12*max_LAR12;
LARmas(2:8:max_LARL)=LARmas(2:8:max_LARL)/GC_LAR12*max_LAR12;
LARmas(3:8:max_LARL)=LARmas(3:8:max_LARL)/GC_LAR34*max_LAR34;
LARmas(4:8:max_LARL)=LARmas(4:8:max_LARL)/GC_LAR34*max_LAR34;
LARmas(5:8:max_LARL)=LARmas(5:8:max_LARL)/GC_LAR56*max_LAR56;
LARmas(6:8:max_LARL)=LARmas(6:8:max_LARL)/GC_LAR56*max_LAR56;
LARmas(7:8:max_LARL)=LARmas(7:8:max_LARL)/GC_LAR78*max_LAR78;
LARmas(8:8:max_LARL)=LARmas(8:8:max_LARL)/GC_LAR78*max_LAR78;
begin_frame = 1;
R1_vos=zeros(1,max_L);
S_vos=zeros(1,max_L);

for ii = 1:Segments,
  end_frame = 160*ii;
  if max_L < end_frame, end_frame = max_L; end;

  beg4=(ii-1)*4+1;
  end4=(ii-1)*4+4;
  LAR=LARmas((ii-1)*8+1:(ii-1)*8+8);

  RC=zeros(1,8);
  for kk=1:8
   RC(kk)=-(1-exp(LAR(kk)))/(1+exp(LAR(kk)));
end

  A=myrc2poly(RC,8);

  G=Gmas(beg4:end4);
  MaxAlpha=MaxAlphamas(beg4:end4);

  for n = begin_frame:end_frame
    k = ceil(4*(n-begin_frame+1)/(end_frame-begin_frame+1));
    if (n-MaxAlpha(k)) < begin_frame, Temp = 0;
    else Temp = R1_vos(n-MaxAlpha(k)); end;
    R1_vos(n) = R2_vos(n) + G(k)*Temp;
  end;

 if begin_frame<9,

  for n = begin_frame:begin_frame+7, S_vos(n) = R1_vos(n); end;
  for n = begin_frame+8:end_frame
    Temp = 0;
    for k = 1:8
      Temp = Temp + A(k)*S_vos(n-k);
    end;
```

```
    S_vos(n) =  R1_vos(n) + Temp;
  end;
 else

  for n = begin_frame:end_frame
    Temp = 0;
    for k = 1:8
      Temp = Temp + A(k)*S_vos(n-k);
    end;
    S_vos(n) =  R1_vos(n) + Temp;
  end;

 end;

  begin_frame = end_frame + 1;
end;

b=fir1(8,0.9);
S_vos=filtfilt(b,1,S_vos);
toc

figure(4),pwelch(S_vos,[],[],[],1);
figure(5),pwelch(R1_vos,[],[],[],1);
figure(6),pwelch(R2_vos,[],[],[],1);
after;
```

**myfilter2lar.m**

Подпрограмма, реализующая фильтры кратковременного и долговременного предсказания.

```
function [LAR, G, Maxalpha, R1, R2] = myfilter2lar(input,input_last,p)

L = length(input);

%kratkovr
for jj = 0:p R(jj+1)=autokorell(input,jj); end;
[A,RC]=mylevinson(R,p);
LAR=zeros(1,8);
for k=1:p
 LAR(k)=log((1+RC(k))/(1-RC(k)));
end

for n = 1:L
  for k = 1:p
    if (n-k) <= 0 S1(k) = input_last(160+n-k);
    else S1(k) = input(n-k);
    end
  end

  R1(n) = input(n) - sum(A.*S1);
end

clear jj k n S1 Rt R

%dolgovr
Max = zeros(1,4);
Maxalpha = zeros(1,4);
G = zeros(1,4);

for jj = 1:4
  jj4=jj*L/4;
  RR1 = R1(1:floor(jj4));
  for alpha = 20:147

    if alpha > jj4 break; end
    X=zeros(1,jj4);
    X(alpha+1:jj4)=RR1(1:jj4-alpha);
```

```
    Temp1 = sum(X.^2);
    if Temp1 == 0 Temp = 0; else Temp2 = sum(RR1.*X); Temp = (Temp2^2)/(Temp1); end
    if Temp > Max(jj)
       Max(jj) = Temp;
       Maxalpha(jj) = alpha;
       G(jj) = Temp2/Temp1;
       if abs(G(jj)) > 1 G(jj) = sign(G(jj)); end
    end;
  end

end

for n = 1:L
  k = ceil(4*n/L);
   X=zeros(1,160);
   X(Maxalpha(k)+1:160)=RR1(1:160-Maxalpha(k));

  R2(n) = R1(n) - G(k)*X(n);
end
```

**mylevinson.m**

Подпрограмма, вычисляющая коэффициенты отражения.

```
function [lpc,Pk,E]=mylevinson(acf,lpc_order);

%Calculation of LPC filter koeffs using Levinson - Durbin recursion.
%*acf = ponter to array of autocorrelation function values acf[0],
acf[1]....acf[lpc_order]
%*lpc = pointer to filter koeffs calculated by function  a[0],a[1]...a[lpc_order-1]
%lpc_order = the order of LPC

Pk=0;
E=acf(1);
for i=0:lpc_order-1,
    lpc(i+1)=0;
end

for i=0:lpc_order-1,
    tmp0=acf(i+2);
    for j=0:i-1,
        tmp0=tmp0-lpc(j+1)*acf(i-j+1);
    end
    if abs(tmp0)>=E
        break
    end
    Pk(i+1)=tmp0/E;
    lpc(i+1)=Pk(i+1);
    E=E-tmp0*Pk(i+1);
    for j=0:i-1,
        tmp(j+1)=lpc(j+1);
    end
    for j=0:i-1,
        lpc(j+1)=lpc(j+1)-Pk(i+1)*tmp(i-j);
    end
end
```

**myrc2poly.m**

Подпрограмма, восстанавливающая коэффициенты фильтра кратковременного предсказания.

```
function [lpc]=myrc2poly(Pk,lpc_order);

for i=0:lpc_order-1,
    lpc(i+1)=0;
end

for i=0:lpc_order-1,
    lpc(i+1)=Pk(i+1);
    for j=0:i-1,
```

```
        tmp(j+1)=lpc(j+1);
    end
    for j=0:i-1,
        lpc(j+1)=lpc(j+1)-Pk(i+1)*tmp(i-j);
    end
end
```

**myload.m**

Подпрограмма загрузки звукового файла.

```
function out = myload(f_type)
%
%
%

jj = 1;
while jj,
  [filename, pathname] = uigetfile(strcat('*.',f_type), 'Выберите файл');
  if isequal(filename,0)|isequal(pathname,0), continue;end;
  jj = 0;
%if strcmp(f_type,'dat'),
  temp = (importdata(filename));
  if strcmp(f_type,'wav'),
      if temp.fs ~= 8000, jj = 1; uiwait(errordlg('Fd must be 8 kHz'));
      else out = (temp.data)';
      end;
  else out = temp';
  end;
end;
```

**autokorell.m**

Подпрограмма, вычисляющая автокорелляционную функцию входного сигнала для фильтра кратковременного предсказания.

```
function R = autokorell(input,J)
L = length(input);
S1 = input(J+1:L);
S2 = input(1:L-J);
R = sum(S1.*S2);
```

**imaadpcm_en.m**

Подпрограмма, реализующая кодер IMA ADPCM.

```
function [y]=imaadpcm_en(x)
 global state_index state_previousValue indexAdjustTable stepSizeTable
 indexAdjustTable = [-1,-1,-1,-1,2,4,6,8,-1,-1,-1,-1,2,4,6,8];
 stepSizeTable = [
7,8,9,10,11,12,13,14,16,17,19,21,23,25,28,31,34,37,41,45,50,55,60,66,73,80,88,97,107,
118,130,143,157,173,190,209,230,253,279,307, ...
337,371,408,449,494,544,598,658,724,796,876,963,1060,1166,1282,1411,1552,1707,1878,2066
,2272,2499,2749,3024,3327,3660,4026,4428, ...
4871,5358,5894,6484,7132,7845,8630,9493,10442,11487,12635,13899,15289,16818,18500,20350
,22385,24623,27086,29794,32767];
 state_index=44;
 state_previousValue=x(1);
 y=zeros(1,length(x));
 y(1)=state_previousValue;
 for k=2:length(x),
     y(k)=ImaAdpcmEncode(x(k));
 end

function [AudioSample]=ImaAdpcmDecode(deltaCode)
 global state_index state_previousValue indexAdjustTable stepSizeTable
 step=stepSizeTable(state_index);
```

```
 difference = bitshift(step,-3,32);

 if bitand(deltaCode,1)==1
     difference=difference+bitshift(step,-2,32);
 end

 if bitand(deltaCode,2)==2
     difference=difference+bitshift(step,-1,32);
 end

 if bitand(deltaCode,4)==4
     difference=difference+step;
 end

 if bitand(deltaCode,8)==8
     difference=-double(difference);
 end

 state_previousValue = int32(double(state_previousValue)+double(difference));

 if state_previousValue > 32767
     state_previousValue = 32767;
 elseif state_previousValue < -32768
     state_previousValue = -32768;
 end

 state_index =state_index+ indexAdjustTable(deltaCode+1);
 if state_index < 1
     state_index = 1;
 elseif state_index > 89
     state_index = 89;
 end

 AudioSample = int16(state_previousValue);

function [AudioByte]=ImaAdpcmEncode(sample)
 global state_index state_previousValue indexAdjustTable stepSizeTable
 diff = double(sample) - double(state_previousValue);
 step = stepSizeTable(state_index);
 deltaCode = 0;

 if diff < 0
      deltaCode = 8; diff = -diff;
 end

 if  diff >= step
      deltaCode=bitor(deltaCode,4);  diff =diff- step;
 end
 step=bitshift(step, -1, 32);

 if  diff >= step
      deltaCode=bitor(deltaCode,2);  diff =diff- step;
 end

 ImaAdpcmDecode(deltaCode);
 AudioByte=uint8(deltaCode);
```

**imaadpcm_de.m**

Подпрограмма, реализующая декодер IMA ADPCM.

```
function [y]=imaadpcm_de(x)
 global state_index state_previousValue indexAdjustTable stepSizeTable
 indexAdjustTable = [-1,-1,-1,-1,2,4,6,8,-1,-1,-1,-1,2,4,6,8];
 stepSizeTable = [
7,8,9,10,11,12,13,14,16,17,19,21,23,25,28,31,34,37,41,45,50,55,60,66,73,80,88,97,107,
118,130,143,157,173,190,209,230,253,279,307, ...
337,371,408,449,494,544,598,658,724,796,876,963,1060,1166,1282,1411,1552,1707,1878,2066
,2272,2499,2749,3024,3327,3660,4026,4428, ...
4871,5358,5894,6484,7132,7845,8630,9493,10442,11487,12635,13899,15289,16818,18500,20350
,22385,24623,27086,29794,32767];
 state_index=44;
 state_previousValue=x(1);
 y=zeros(1,length(x));
 y(1)=state_previousValue;
 for k=2:length(x),
     y(k)=ImaAdpcmDecode(x(k));
 end

function [AudioSample]=ImaAdpcmDecode(deltaCode)
 global state_index state_previousValue indexAdjustTable stepSizeTable
 step=stepSizeTable(state_index);
 difference = bitshift(step,-3,32);

 if bitand(deltaCode,1)==1
     difference=difference+bitshift(step,-2,32);
 end

 if bitand(deltaCode,2)==2
     difference=difference+bitshift(step,-1,32);
 end

 if bitand(deltaCode,4)==4
     difference=difference+step;
 end

 if bitand(deltaCode,8)==8
     difference=-double(difference);
 end

 state_previousValue = int32(double(state_previousValue)+double(difference));

 if state_previousValue > 32767
     state_previousValue = 32767;
 elseif state_previousValue < -32768
     state_previousValue = -32768;
 end

 state_index =state_index+ indexAdjustTable(deltaCode+1);
 if state_index < 1
     state_index = 1;
 elseif state_index > 89
     state_index = 89;
 end

 AudioSample = int16(state_previousValue);
```