TR-I-0135

波形重ね合わせ法による合成音の品質について

Quality evaluation for synthesized speech using wave overlap adding

安部勝雄、匂坂芳典、桑原尚夫†

Katsuo ABE, Yoshinori SAGISAKA and Hisao KUWABARA

1990. 2

内容梗概

ケプストラムから波形重ね合わせ法によって生成した合成音の品質について述べる。ケプストラムを用いた分析合成系の合成フィルタの代わりに波形重ね合わせ法を導入することにより、女声音合成時に現れる音質の劣化を避けることができたので、受聴試験を行いその有効性を示す。巻末に分析合成プログラムリストを付した。

ATR 自動翻訳電話研究所
ATR Interpreting Telephony Research Laboratories
†NHK放送技術研究所音響聴覚研究部

1. はじめに

音声の分析合成系は規則合成の音質を左右する重要な技術の一つであり、様々な手法が考案されている。

直接パラメータに手を加える規則合成において分析合成系に要求される条件は、まず原音を忠実に再生できること、パラメータの操作が容易であること、原音ピッチと使用ピッチの違いによる音質の劣化が小さいこと、等があげられる。ケプストラムを用いた分析合成法[1][2]は、パラメータの操作の容易さから規則合成に有効な分析合成系として用いられてきた。しかしこの分析合成法では女声音の有声区間で男声音のときには現れない『ゴロゴロ』した音質の劣化が生じることがある。

本報告では、ケプストラムを用いた分析合成法の合成フィルタの代わりに波形重ね合わせ法[3][4]を導入することにより、女声音合成時に現れる音質の劣化を避けることができたので、受聴試験を行い波形重ね合わせ法の有効性を示す。

2.合成方法

2.1 分析パラメータの作成

分析法として改良ケプストラム法[1]を用いた。サンプリング周波数12kHz語長16ビットの音声データをフレーム長21.3ms、フレーム周期5msで分析し、30次の改良ケプストラムを得た。ピッチ情報は、フレーム長32msで求めたケプストラムから抽出した。

2.2 合成法

(1)LMAフィルタ法[2]

ケプストラムを係数とする対数振幅特性近似(LMA)フィルタを構成し、ピッチ情報に従ってパルスまたは雑音でLMAフィルタを駆動する。

(2)波形重ね合わせ法[4]

ケプストラムをフーリエ変換して対数スペクトル包絡を求め、この振幅を線形変換した後に再び逆フーリエ変換を行い零位相化したインパルス応答波形とする。インパルス応答波形をピッチ間隔でずらしながら波形の重ね合わせを行い合成音とする。

3.受聴試験

3.1 実験方法

規則合成への適用を考え、3種類のピッチパタンで合成音を作成し、LMAフィルタと波形重ね合わせ法とのプリファレンスをとった。用いたピッチパタンは次の3種類である。

(A)原音ピッチ(10単語)
(B)非一様ピッチ変換(10単語): 例えばスペクトル情報として『こうじょう(工場)』のケプストラムを用い、ピッチ情報として『こうじょう(向上)』のピッチパタンを使用することを言う。
(C)一様ピッチ変換(1単語): 単語中の有声区間のピッチ周波数を原音に対し一様に定数倍することで、男声音では0.4~2.2倍までを0.2きざみで、女声音では0.6~1.5倍までを0.1きざみで各々10段階のピッチパタンを用いた。

原音サンプルは男性ナレータ、女性アナウンサ各1名発声の21単語を用いた。合成音の提示は各ピッチパタン毎のグループ内で各単語について、(波形重ね合わせ法)-(LMAフィルタ法)、またその順番をいれかえたものをペアとして計20組をランダムに配し、プリファレンスをとった。被験者は男性3名、女性8名の計11名である。なお波形重ね合わせ法による合成は、有声区間のみとし無声区間では、LMAフィルタを用いた。

3.2 実験結果

各ピッチパタングループ毎のプリファレンス・スコアを図1に示す。女声音の場合には、各グループとも波形重ね合わせ法が好まれる強い傾向にあり、男声音では(A)原音ピッチを用いた場合LMAフィルタ法が好まれるが、原音とは異なるピッチを用いた(B)非一様ピッチ変換、(C)一様ピッチ変換の場合には若干波形重ね合わせ法が好まれている。図2には一様ピッチ変換の場合の各ピッチ変換率毎のプリファレンス・スコアを示す。女声音では原音ピッチと近いピッチで波形重ね合わせ法の優位さが見られるが、原音ピッチから離れるにつれLMAフィルタ法との差はなくなる。男声音ではピッチとの関連した差は見られない。

4.考察

原音ピッチで合成したとき、女声音の場合にはLMAフィルタ法では有声区間で多少『ゴロゴロ』した感じがあるが、波形重ね合わせ法により滑らかになる。男声音では劣化という観点からはいずれの合成法も品質の差は小さい。

図1のプリファレンス・スコアは女声音では波形重ね合わせ法が優位であるという結果を示したが、男声音では波形重ね合わせ法の優位さは明確でない。さらにピッチ変換率毎にスコアを示した図2によれば、高いピッチでは男女声ともスコアに差はなくなっている。

ピッチを変えたときのピッチとスペクトル歪との関係を見るために、一様ピッチ変換で合成したときの合成音と原音声とのケプストラム距離(20次の改良ケプストラムを使用)をとり図3に示した。男女声とも波形重ね合わせ法のケプストラム距離は小さく、この合成法のスペクトル包絡の再現性の高さを表している。

図3の関係を図2に対応させると、女声音では原音ピッチのときの波形重ね合わせ法の滑らかさがプリファレンス・スコアに反映されているが、ピッチを上げた

ときにはピッチ上昇により発生するスペクトル歪の影響が大きく二つの合成法の品質は接近する。男声音の場合にはピッチ上昇によるケプストラム距離に変化はないが、音質の劣化は明らかであり、ケプストラム距離で表現しえないスペクトル歪が生じていると思われる。

5. おわりに

従来のケプストラム分析合成系に波形重ね合わせ法を導入しその有効性を確認した。波形重ね合わせ法は女声音の原音ピッチに近いピッチ周波数の音を合成するときにとくに有効であり、使用ピッチを限定すれば規則合成への適用も有効であることが明らかになった。

謝辞

日頃御指導頂く樽松社長、鹿野室長、並びに討論頂いたATR 自動翻訳電話研究所、ATR 視聴覚機構研究所の皆様に深謝致します。

文献

[1]今井、阿部「改良ケプストラム法によるスペクトル包絡の抽出」、信学論(A)、J62-A、4、p.217, 1979

[2]今井「対数振幅近似(LMA)フィルタ」、信学論(A)、J63-A、12、p.886, 1980

[3]A.V.Oppenheim, "Speech analysis-synthesis system based on homomorphic filtering," J. Acoust. Soc. Am. 45, 458-464(1969)

[4]中島、鈴木「パワースペクトル包絡(PSE)音声分析·合成系」、日本音響学会誌、VOL.44 NO.11, 1988

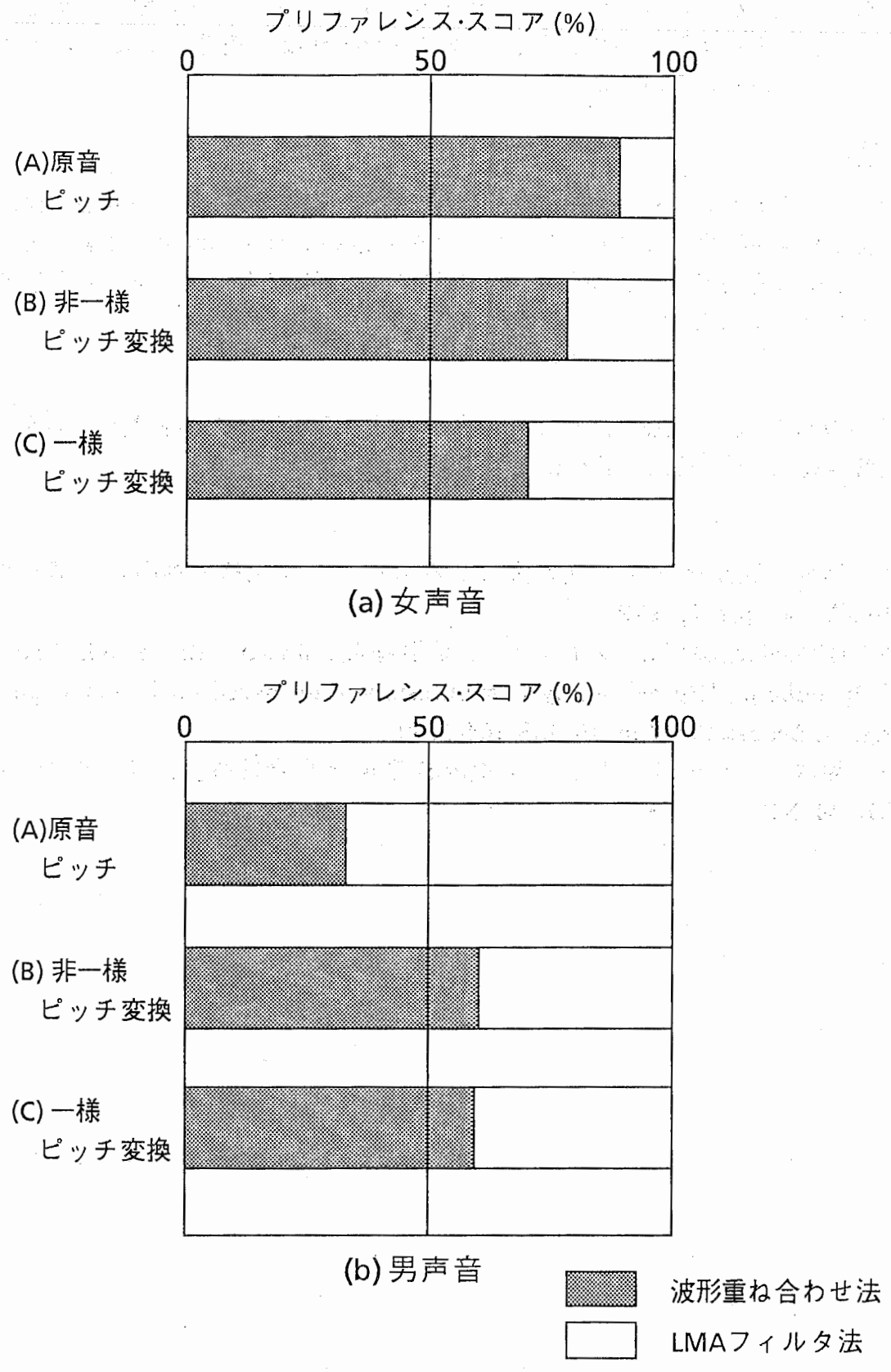


図1 プリファレンス･スコア

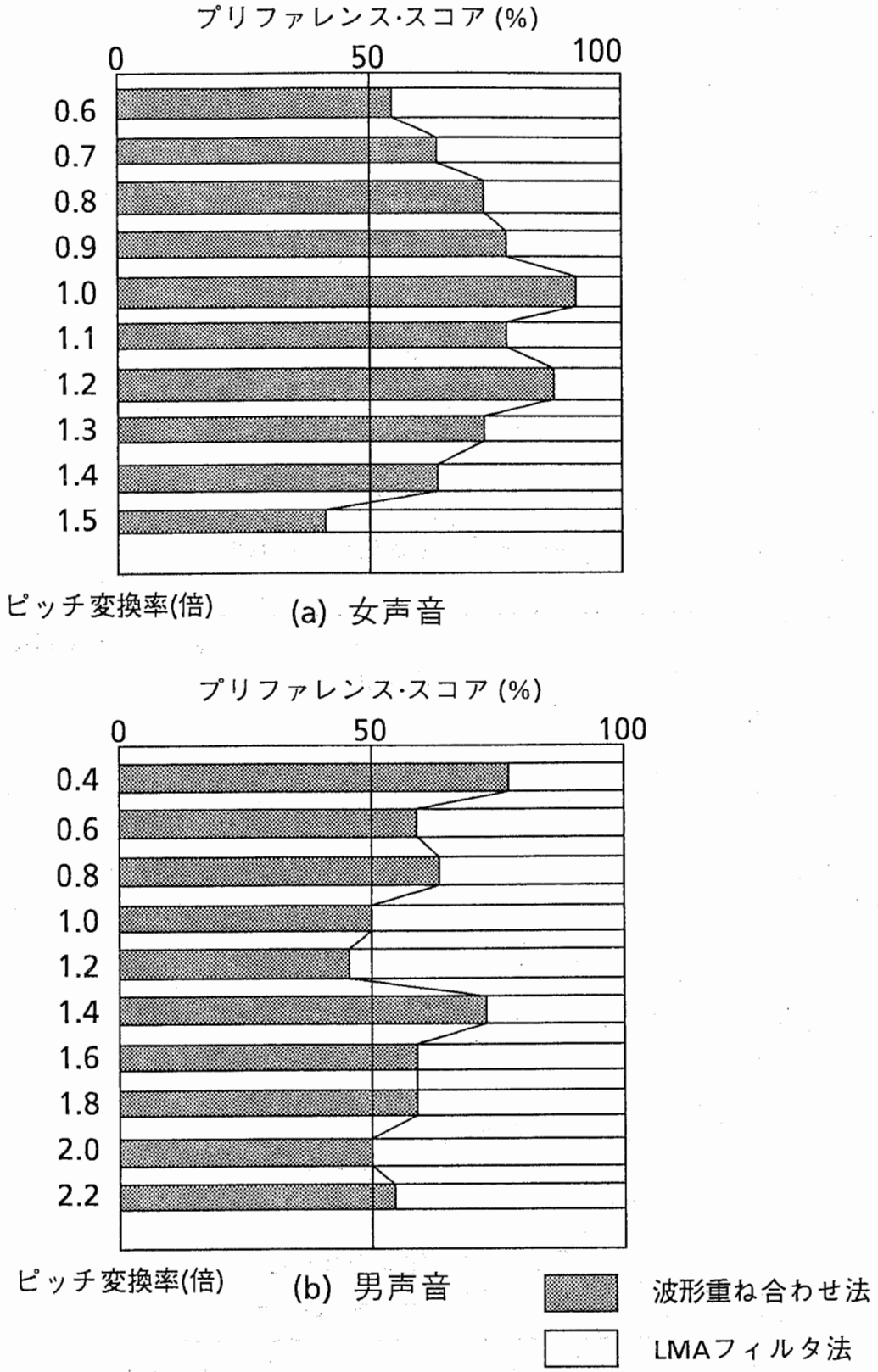
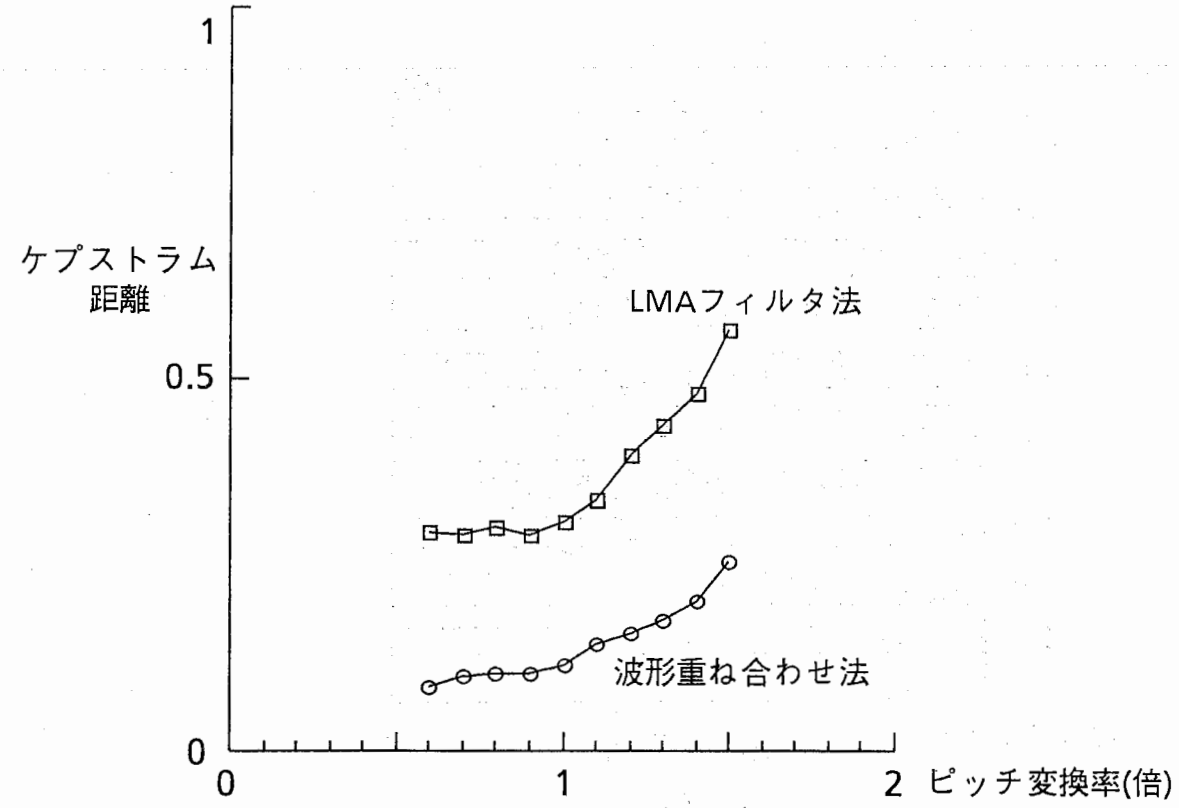
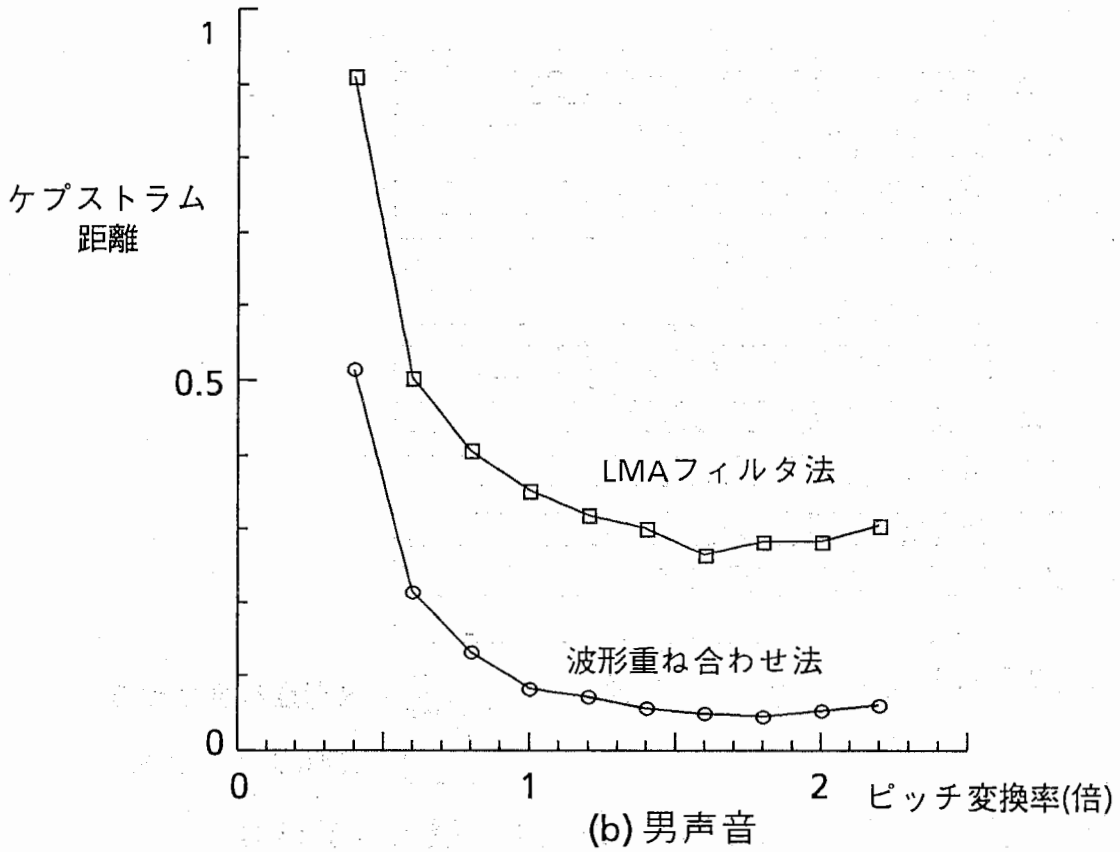


図2 プリファレンス･スコア (一様ピッチ変換)



(a) 女声音

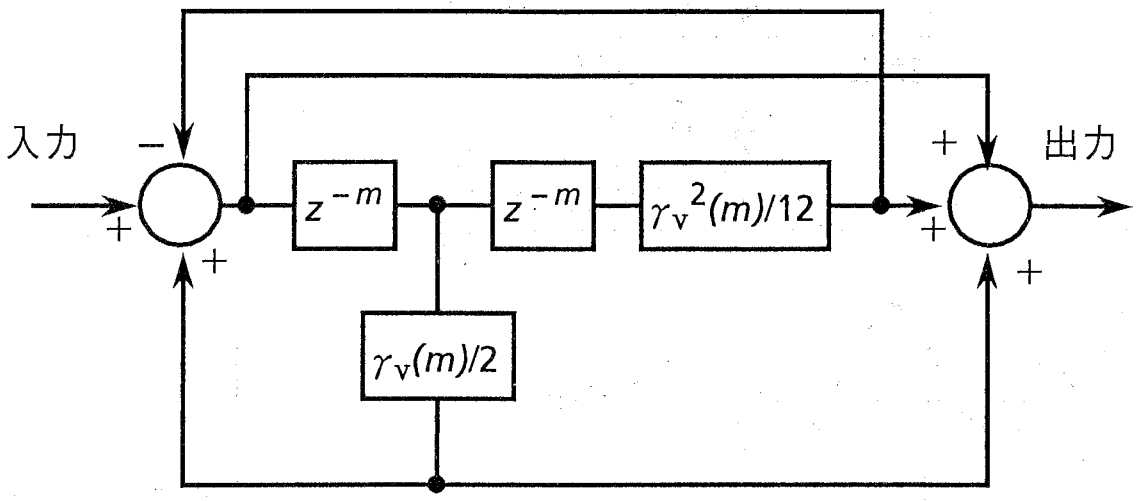


(b) 男声音

図3 ケプストラム距離 (一様ピッチ変換)

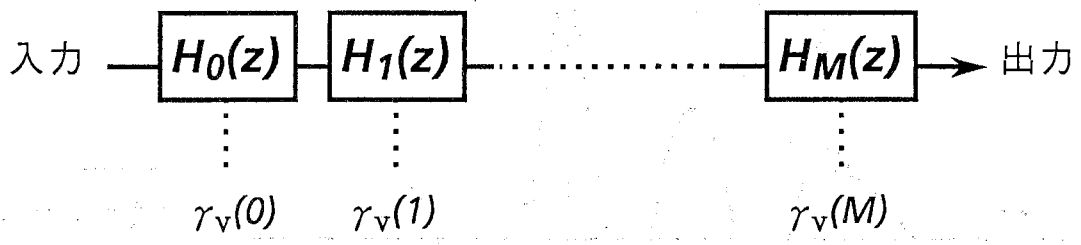
付録1

対数振幅特性近似(LMA)フィルタのブロック図を図4に、波形重ね合わせ法の説明図を図5に示す。



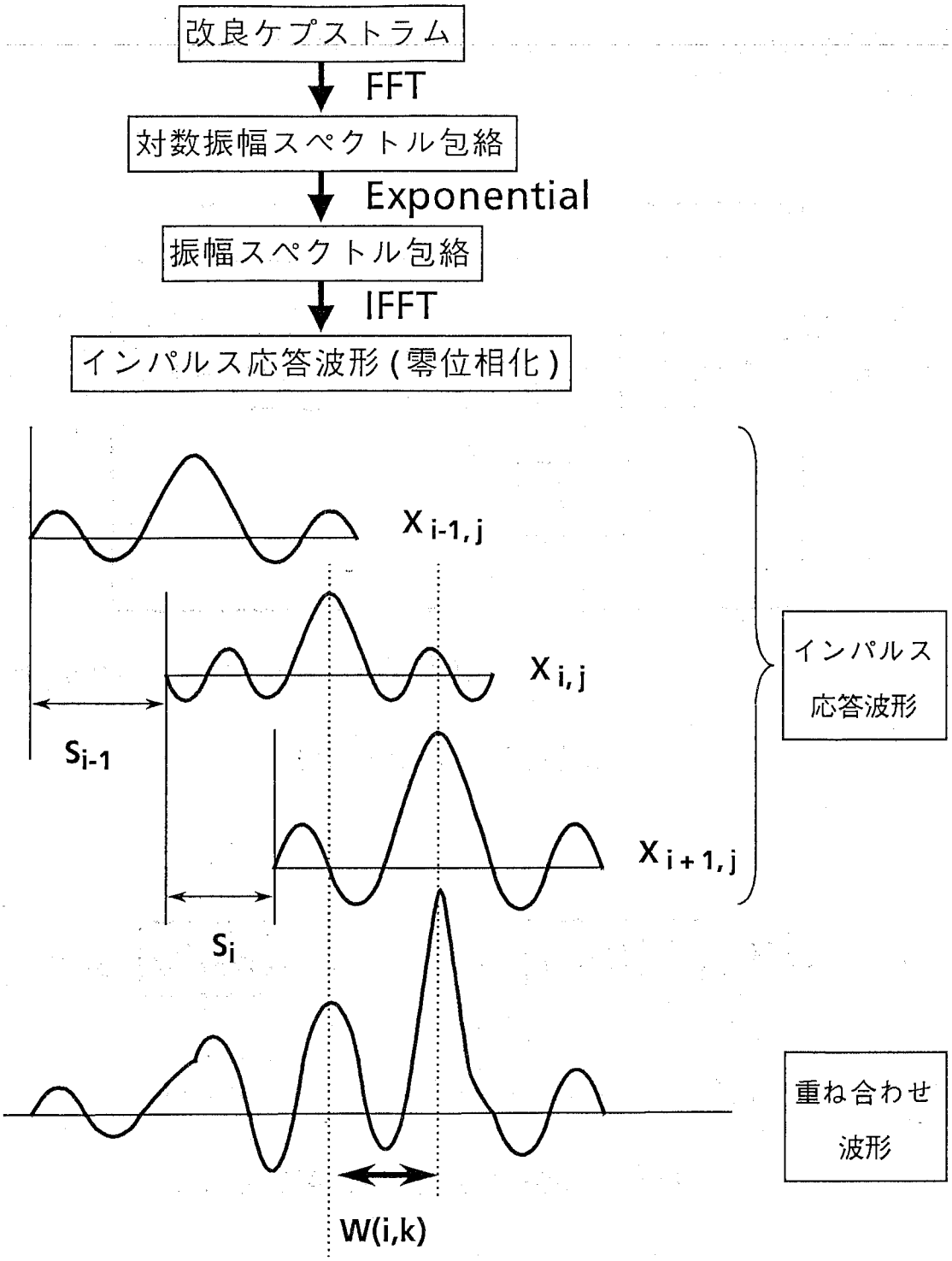
$\gamma_v(m)$: 改良ケプストラム, $(m=0,1,\ldots,M)$

(a)基礎フィルタ $H_m(z)$



(b)対数振幅特性近似(LMA)フィルタ

図4　LMAフィルタの構成



$$W(i,k) = \cdots + (X_{i-1,k+S_{i-1}}) + (X_{i,k}) + (X_{i+1,k-S_i}) + \cdots$$

where $k = 0,1,2,\cdots,Si-1$

図5　波形重ね合わせ法の説明図

付録2

実験に用いた3種類のプログラムのリストを掲載する。これらのプログラムを用いて合成音を作成するときの手順を図6に示す。

1. 改良ケプストラム分析プログラム　　cep__ana.c

改良ケプストラム分析、及びケプストラムによるピッチ抽出。

[原音](short integer)
サンプリング周波数12kHz、語長16bit

[改良ケプストラムの抽出](floating)
フレーム長21.3ms、フレーム周期5ms
改良繰り返し回数3、改良加速度係数1.0
ケプストラム次数30(0次から30次まで計31個/フレーム)

[ピッチ抽出](integer)
フレーム長32ms

2. 対数振幅特性近似(LMA)フィルタプログラム　　cep__lma.c

3. 波形重ね合わせ(OLA)プログラム　　cp__h__ol.c

波形重ね合わせ法とLMAフィルタのハイブリッド形の合成プログラム。有声音区間は波形重ね合わせ法によって合成し、無声音区間はLMAフィルタによって合成する。

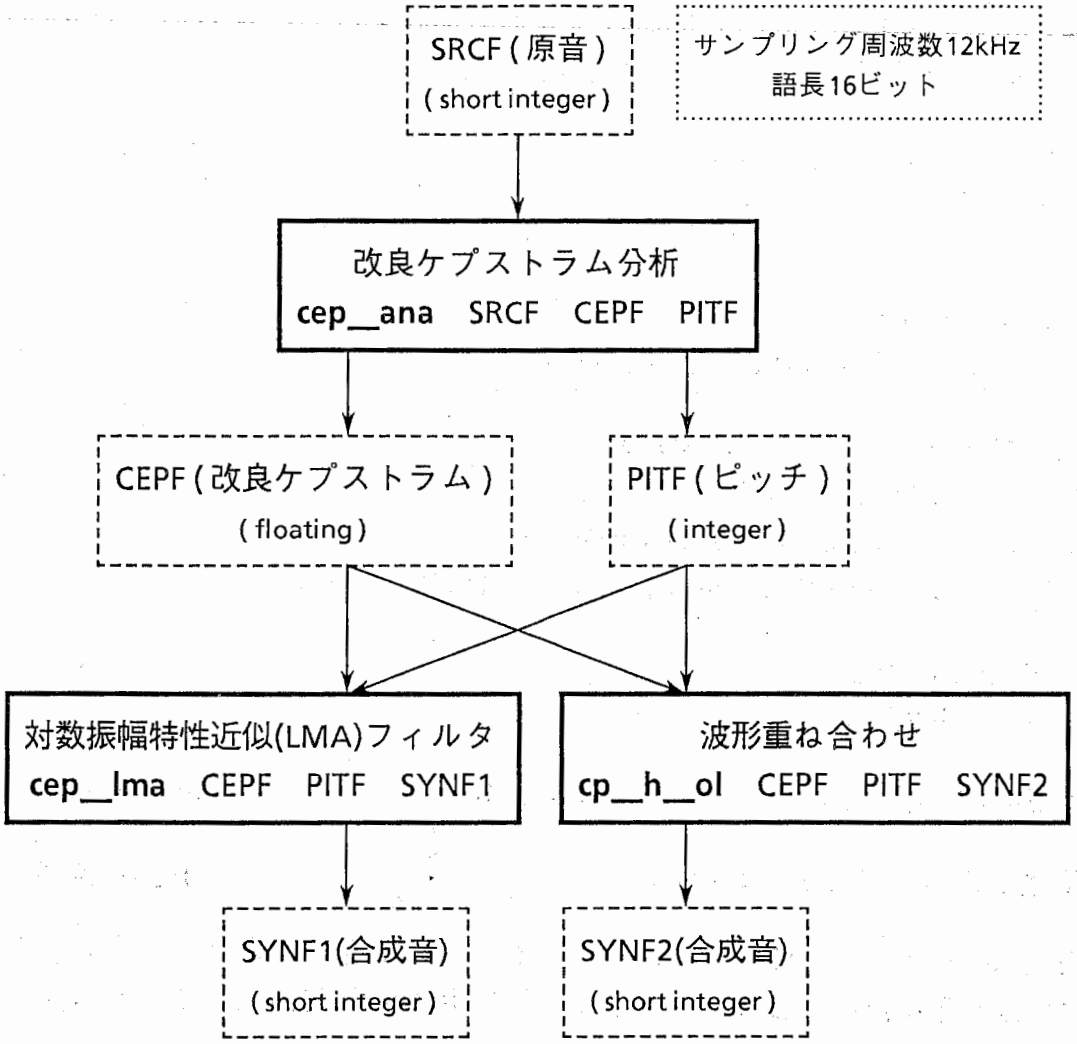


図6　合成音作成のフローチャート

```
/*****************************************************************************/
/*                                                                           */
/*      CEPSTRUM Analysis ( cep_ana.c )                                      */
/*                                                                           */
/*              - create cepstrum(f) file and pitch(i) file -                */
/*                                                                           */
/*      Compiled by:                                                         */
/*              cc -o cep_ana cep_ana.c -lm                                  */
/*                                                                           */
/*      [ USAGE ]                                                            */
/*              cep_ana SRCF CEPF PITF THR(6.)                               */
/*                                                                           */
/*        Input:                                                             */
/*              SRCF:   input speech source file                             */
/*              CEPF:   output cepstrum(f) file                              */
/*              PITF:   output pitch(i) file                                 */
/*              THR:    input pitch decision threshold value                 */
/*                                                                           */
/*---------------------------------------------------------------------------*/
/*      Version 1.0     coded by k.abe                  12/08/89             */
/*****************************************************************************/

#include <stdio.h>
#include <math.h>
#include <sys/types.h>
#include <sys/times.h>

#define PI      3.14159265358979323846

struct  tms buff;

main( argc, argv )
int     argc;
char    *argv[];
{
  double        x[512];
  double        xp[512];
  double        y[512];
  double        peak[3];
  float         cep[64];                /* cepstrum */
  float         coef_icep[16];          /* improved cepstrum coefficients */
  int           ad_int[512];            /* A/D data */
  int           pitch_some[3];
  short         ad[512];                /* A/D data */

  double        bn;
  float         thr_vuv;
  int           i;
  int           fd_src;                 /* speech source file descriptor */
  int           fd_cep;                 /* cepstrum(f) file descriptor */
  int           fd_pit;                 /* pitch(i) file descriptor */
  int           rb_src;
  int           nframe;                 /* frame number */
  int           pitch;                  /* pitch period */
  int           flag_vuv;               /* voice/unvoice decision flag */

  int           sframe = 60;            /* frame shift = 5ms */
  int           len_frame = 256;        /* frame length for spec. estimation
                                           = 21.3ms */
  int           len_fft = 256;          /* fft frame length for spec. estimation */
  int           size_fft = 8;           /* fft frame size for spec. estimation */
  int           ord_icep = 33;          /* improved cepstrum order */
  int           ref_icep = 3;           /* refrain number of improvement */
  int           ord_cep = 30;           /* cepstrum order */
  int           len_pframe = 384;       /* frame length for pitch extraction
```

```
                                                = 32ms */
  int           len_pfft = 512;         /* fft frame length for pitch extraction */
  int           size_pfft = 9;          /* fft frame size for pitch extraction */

/*------- check arguement consistency ----------------------------------------*/
  if( argc == 4 ) thr_vuv = 6.;
  else if( argc == 5 ) thr_vuv = atof( argv[4] );
  else
  {
    printf( " - create cepstrum(f) file and pitch(i) file -\n" );
    printf( "[ USAGE ] cep_ana SRCF CEPF PITF         or\n" );
    printf( "[ USAGE ] cep_ana SRCF CEPF PITF THR(6.)\n" );
    exit();
  }

/*------- start time ---------------------------------------------------------*/
  t_start();

/*------- initial setting of analysis conditions -----------------------------*/
  for( i=0; i<ref_icep; i++ )
       coef_icep[i] = 1.5;              /* improved cepstrum coefficient */

/*------- open files ---------------------------------------------------------*/
  if( ( fd_src=open(argv[1], 0) ) == -1 )
  {
    printf( " * File Open Error ! SRCF = %s\n", argv[1] );
    exit();
  }
  fd_cep = creat( argv[2], 0644 );
  fd_pit = creat( argv[3], 0644 );

/*------- read source data ---------------------------------------------------*/
  nframe = 0;
  if( ( rb_src=read(fd_src, ad, 2*(len_pframe-128-sframe)) ) == 0 )
       eof( argv[1], nframe );
  for( i=0; i<rb_src/2; i++ ) ad_int[128+sframe+i] = (int)ad[i];
  if( rb_src/2 < sframe )
       for( i=rb_src/2; i<sframe; i++ ) ad_int[i] = 0;

/*------- read source data ---------------------------------------------------*/
  while( 1 )
  {
    for( i=0; i<len_pframe-sframe; i++ ) ad_int[i] = ad_int[i+sframe];
    if( ( rb_src=read(fd_src, ad, 2*sframe) ) == 0 ) eof( argv[1], nframe );
    for( i=0; i<rb_src/2; i++ ) ad_int[len_pframe-sframe+i] = (int)ad[i];
    if( rb_src/2 < sframe )
         for( i=rb_src/2; i<sframe; i++ ) ad_int[i] = 0;
    d_zero( x, len_fft );
    d_zero( xp, len_pfft );
    for( i=0; i<len_frame; i++ ) x[i] = (double)ad_int[i];
    for( i=0; i<len_pframe; i++ ) xp[i] = (double)ad_int[i];

/*------- improved cepstrum --------------------------------------------------*/
    black( x, len_frame );
    d_zero( y, len_fft );
    fft( x, y, size_fft, -1.0 );
    logspec( x, y, len_fft/2+1 );
    imcps( x, size_fft, ord_icep, coef_icep, ref_icep );
    x[0] *= 0.5;
    if( x[0] < 1. ) x[0] = 1.;

/*------- pitch extraction ---------------------------------------------------*/
    sinew( xp, len_pframe );
    cps( xp, size_pfft );
    apitch( pitch_some, &flag_vuv, xp, thr_vuv, peak, &bn );
```

```
    pitch = 0;
    if( flag_vuv == 1 ) pitch = pitch_some[0];

/*------- write cepstrum ------------------------------------------------------*/
    for( i=0; i<=ord_cep; i++ ) cep[i] = (float)x[i];
    write( fd_pit, &pitch, 4 );
    write( fd_cep, cep, 4*ord_cep+4 );

/*------- increment counter ---------------------------------------------------*/
    ++nframe;
  }
}


/******** clear double array ********************************************/
/*                                                                      */
/*      x:      [out] cleared double array                              */
/*      n:      [in ] array length                                      */
/*                                                                      */
/************************************************************************/
d_zero( x, n )
double  x[];
int     n;
{
  int   i;

  for( i=0; i<n; i++ )          x[i] = 0.;
}

/******** blackman window ***********************************************/
/*                                                                      */
/*      x:      [in ] source array                                      */
/*              [out] blackman windowed array                           */
/*      n:      [in ] window length                                     */
/*                                                                      */
/************************************************************************/
black( x, n )
double  x[];
int     n;
{
  double        t;
  int           i;

  for( i=0; i<n; i++ )
  {
    t = 2.*PI*(double)i/(n-1.);
    x[i] *= ( 0.42 - 0.5*cos(t) + 0.08*cos(2.*t) );
  }
}

/******* sine window ****************************************************/
/*                                                                      */
/*      x:      [in ] source array                                      */
/*              [out] sine windowed array                               */
/*      n:      [in ] window length                                     */
/*                                                                      */
/************************************************************************/
sinew( x, n )
double  x[];
int     n;
{
  int   i;

  for( i=0; i<n; i++ )
       x[i] *= sin( (double)i*PI/(n-1.) );
```

```
}

/******** copy x from y ( double ) ************************************************/
/*                                                                                */
/*      x:      [out] copied double array                                         */
/*      y:      [in ] source double array                                         */
/*      n:      [in ] copy length                                                 */
/*                                                                                */
/**********************************************************************************/
d_copy( x, y, n )
double  x[], y[];
int     n;
{
  int   i;

  for( i=0; i<n; i++ )          x[i] = y[i];
}

/******** fast fourier transform ( FFT.c ) ****************************************/
/*                                                                                */
/*      x:      [i/o] real part array                                             */
/*      y:      [i/o] imaginary part array                                        */
/*      l:      [in ] FFT size ( 2^l )                                            */
/*      f:      [in ] FFT flag ( -1.0:FFT  1.0:IFFT )                             */
/*                                                                                */
/**********************************************************************************/
int fft( x, y, l, f )
double  *x, *y, f;
int     l;
{
  double        s, c, s1, c1, sc, x1, y1, t;
  int           i, i0, i1, j, l1, n, ns, n1, k, ipow2();

  n = ipow2( l );                       /* n = 2^l */
  n1 = n/2;     sc = PI;

  j = 0;
  for( i=0; i<n-1; i++ )                /* bit reverse counter */
  {
    if( i <= j )
    {
      t = x[i];         x[i] = x[j];            x[j] = t;
      t = y[i];         y[i] = y[j];            y[j] = t;
    }
    k = n/2;
    while( k <= j )
    {
      j = j - k;        k /= 2;
    }
    j = j + k;
  }

  ns = 1;
  while( ns <= n/2 )            /* main loop */
  {
    c1 = cos( sc );     s1 = sin( f*sc );
    c = 1.0;            s = 0.0;
    for( l1=0; l1<ns; l1++ )
    {
      for( i0=l1; i0<n; i0+=(2*ns) )
      {
        i1 = i0 + ns;
        x1 = x[i1]*c - y[i1]*s;         y1 = y[i1]*c + x[i1]*s;
        x[i1] = x[i0] - x1;             y[i1] = y[i0] - y1;
        x[i0] = x[i0] + x1;             y[i0] = y[i0] + y1;
```

```
      }
      t = c1*c - s1*s;  s = s1*c + c1*s;        c = t;
    }
    ns = ns*2;          sc = sc/2.0;
  }

  if( f > 0.0 )                 /* judge fft or ifft (update) */
       for( i=0; i<n; i++ )
       {
         x[i] /= (double)n;
         y[i] /= (double)n;
       }
}

/******** power ********************************************************************/
/*                                                                                  */
/*      l:      [in ] 2^l                                                           */
/*      n:      [out] n = 2^l                                                       */
/*                                                                                  */
/************************************************************************************/
int     ipow2( l )
int     l;
{
  int   n;

  n = 1;
  while( l != 0 )
  {
    n *= 2;
    l--;
  }
  return n;
}

/******** cepstrum *****************************************************************/
/*                                                                                  */
/*      x:              [in ] windowed array                                        */
/*                      [out] cepstrum                                              */
/*      size_fft:       [in ] fft frame size                                        */
/*                                                                                  */
/************************************************************************************/
cps( x, size_fft )
double  x[];
int     size_fft;
{
  double        y[512];
  int           i, len_fft;

  len_fft = ipow2( size_fft );

  d_zero( y, len_fft );
  fft( x, y, size_fft, -1.0 );
  logspec( x, y, len_fft/2+1 );
  for( i=len_fft/2+1; i<len_fft; i++ ) x[i] = x[len_fft-i];
  d_zero( y, len_fft );
  fft( x, y, size_fft, 1.0 );
}

/******** log magnitude spectrum ***************************************************/
/*                                                                                  */
/*      x:      [in ] real part of spectrum                                         */
/*              [out] log magnitude spectrum                                        */
/*      y:      [in ] imaginary part of spectrum                                    */
/*      n:      [in ] array length                                                  */
/*                                                                                  */
```

```
/*****************************************************************************/
logspec( x, y, n )
double  *x, *y;
int     n;
{
  double        amp;
  int           i;

  for( i=0; i<n; i++ )
  {
    amp = x[i]*x[i] + y[i]*y[i];
    if( amp == 0.0 ) x[i] = 0.0;
    else x[i] = log( amp );
  }
}

/******** improved cepstrum ************************************************/
/*                                                                         */
/*      x:              [in ] log magnitude spectrum                       */
/*                      [out] improved cepstrum                            */
/*      size_fft:       [in ] fft frame size                               */
/*      ord_icep:       [in ] improved cepstrum order                      */
/*      coef_icep:      [in ] improvement coefficient                      */
/*      ref_icep:       [in ] refrain number of improvement                */
/*                                                                         */
/*****************************************************************************/
imcps( x, size_fft, ord_icep, coef_icep, ref_icep )
double  x[];
int     size_fft, ord_icep, ref_icep;
float   coef_icep[];
{
  double        w[512], z[512], y[512];
  int           i, len_fft;

  len_fft = ipow2( size_fft );

  for( i=len_fft/2+1; i<len_fft; i++ ) x[i] = x[len_fft-i];
  d_copy( w, x, len_fft );
  d_zero( y, len_fft );
  fft( x, y, size_fft, 1.0 );                   /* FFT */

  if( ref_icep > 0 )
  {
    for( i=0; i<ref_icep; i++ )
    {
      d_copy( z, x, ord_icep+1 );
      hamm( z, len_fft, ord_icep+1, 1 );
      d_zero( y, len_fft );
      fft( z, y, size_fft, -1.0 );
      subar( z, w, z, len_fft );
      d_zero( y, len_fft );
      fft( z, y, size_fft, 1.0 );
      mult( z, coef_icep[i], ord_icep+1 );
      addar( x, z, ord_icep+1 );
    }
    hamm( x, len_fft, ord_icep+1, 1 );
  }
}

/******** hamming window ***************************************************/
/*                                                                         */
/*      cep:    [in ] cepstrum                                             */
/*              [out] hamming windowed cepstrum                            */
/*      n:      [in ] array length                                         */
/*      m:      [in ] window length                                        */
```

```
/*      flag:   [in ] window type flag                                      */
/*                    [ 1:hamming , others:rectangular ]                    */
/*                                                                          */
/**************************************************************************/
hamm( cep, n, m, flag )
double  cep[];
int     n, m, flag;
{
  int   i;

  if( flag != 1 )
       for( i=0; i<m; i++ )
            cep[i] *= ( 0.54 + 0.46*cos((double)(PI*i)/(m-1.)) );

  for( i=m; i<=n-m; i++ )               cep[i] = 0.;

  for( i=1; i<m; i++ )          cep[n-i] = cep[i];
}

/******** subtract array ***************************************************/
/*                                                                          */
/*      z:      [out] subtracted array                                      */
/*      x:      [in ] source array                                          */
/*      y:      [in ] subtract array                                        */
/*      n:      [in ] array length                                          */
/*                                                                          */
/**************************************************************************/
subar( z, x, y, n )
double  z[], x[], y[];
int     n;
{
  int   i;

  for( i=0; i<n; i++ )
  {
    z[i] = x[i] - y[i];
    if( z[i] <= 0.0 ) z[i] = 0.0;
  }
}

/******** add array ********************************************************/
/*                                                                          */
/*      x:      [in ] source array                                          */
/*              [out] added array                                           */
/*      y:      [in ] add array                                             */
/*      n:      [in ] array length                                          */
/*                                                                          */
/**************************************************************************/
addar( x, y, n )
double  x[], y[];
int     n;
{
  int   i;

  for( i=0; i<n; i++ )          x[i] += y[i];
}

/******** multiply constant ************************************************/
/*                                                                          */
/*      x:      [in ] source array                                          */
/*              [out] multiplied array                                      */
/*      tl:     [in ] multiple constant value                               */
/*      n:      [in ] array length                                          */
/*                                                                          */
/**************************************************************************/
```

```
mult( x, tl, n )
double  x[];
float   tl;
int     n;
{
  int   i;

  for( i=0; i<n; i++ )          x[i] *= tl;
}

/******** extract pitch ************************************************/
/*                                                                     */
/*      pitch_some:     [out] pitch period                             */
/*      flag_vuv:       [out] v/uv flag                                */
/*      cep:            [in ] cepstrum                                 */
/*      thr_vuv:        [in ] threshold value of v/uv decision         */
/*      peak:           [out] cepstrum peak values                     */
/*      bn:             [out] parameter of v/uv decision               */
/*                                                                     */
/***********************************************************************/
apitch( pitch_some, flag_vuv, cep, thr_vuv, peak, bn )
double  cep[], peak[], *bn;
int     pitch_some[], *flag_vuv;
float   thr_vuv;
{
  double        w12, w21, w122, w212, sum;
  int           i, j;
  static float  freq_lower = 50.;       /* lower limit freq. for v/uv decision */
  static float  freq_upper = 350.;      /* upper limit freq. for v/uv decision */
  static int    ord_cep_vuv = 20;       /* cepstrum order for v/uv decision */

/*------- extract peak ------------------------------------------------*/
  qfrw( cep, 384. );
  for( j=0; j<2; j++ )
  {
    peak[j] = 0.;
    for( i=28; i<255; i++ )
         if( cep[i] > peak[j] )
         {
           pitch_some[j] = i;
           peak[j] = cep[i];
         }
    cep[pitch_some[j]] = 0.;
  }

/*------- extract v/uv decision parameter -----------------------------*/
  w12 = 2.*PI*(freq_lower+freq_upper)/12000.;
  w21 = 2.*PI*(freq_upper-freq_lower)/12000.;
  w122 = w12/2.;
  w212 = w21/2.;
  sum = 0.;
  for( i=1; i<=ord_cep_vuv; i++ )
       sum += (0.5*cep[i]*cos((double)i*w122)*sin((double)i*w212))/w21/i;
  *bn = 0.5*cep[0] + 4.*sum;

/*------- decide v/uv flag --------------------------------------------*/
  *flag_vuv = 0;
  if( *bn > thr_vuv ) *flag_vuv = 1;
}

/******** quefrency window *********************************************/
/*                                                                     */
/*      cep:    [in ] cepstrum                                         */
/*              [out] quefrency windowed cepstrum                      */
/*      rate:   [in ] weighted rate                                    */
```

```
/*                                                                              */
/********************************************************************************/
qfrw( cep, rate )
double  cep[], rate;
{
  double        x[512];
  int           i;

  for( i=28; i<256; i++ ) x[i] = cep[i] * (1.+(double)i/rate);
  for( i=28; i<255; i++ ) cep[i] = x[i-1] + 2.*x[i] + x[i+1];
}

/******** End Of File ************************************************************/
/*                                                                              */
/*      file:   [in ] filename which is on End Of File                          */
/*      nframe: [in ] frame number                                              */
/*                                                                              */
/********************************************************************************/
eof( file, nframe )
char    *file;
int     nframe;
{
  printf( " * End Of File ! %s [ %d ]\n", file, nframe );
  t_end();
  exit();
}

/******** display start time *****************************************************/
/*                                                                              */
/********************************************************************************/
t_start()
{
  int   ts;

  ts = time( 0 );
  printf( "\n ---> program start %s\n", ctime(&ts) );
  fflush( stdout );
}

/******** display end time *******************************************************/
/*                                                                              */
/********************************************************************************/
t_end()
{
  int   te;

  te = time( 0 );
  printf( "\n <--- program end %s", ctime(&te) );
  times( &buff );
  printf( " My user time.....%f sec\n\n", buff.tms_utime/60. );
  fflush( stdout );
}
```

```
/*****************************************************************************/
/*                                                                           */
/*      CEPSTRUM synthesis ( cep_lma.c )                                     */
/*                                                                           */
/*              - synthesize speech from cepstrum using LMA filter -         */
/*              - using cepstrum(f) file and pitch(i) file -                 */
/*                                                                           */
/*      Compiled by:                                                         */
/*              cc -o cep_lma cep_lma.c -lm                                  */
/*                                                                           */
/*      [ USAGE ]                                                            */
/*              cep_lma CEPF PITF SYNF CEP0S                                 */
/*                                                                           */
/*        Input:                                                             */
/*              CEPF:   input cepstrum(f) file                               */
/*              PITF:   input pitch(i) file                                  */
/*              SYNF:   output synthesized speech file                       */
/*              CEP0S:  input cepstrum[0] shift value                        */
/*                                                                           */
/*---------------------------------------------------------------------------*/
/*      Version 1.0     coded by k.abe                  12/14/89             */
/*****************************************************************************/

#include <stdio.h>
#include <math.h>
#include <sys/types.h>
#include <sys/times.h>

struct  tms buff;

main( argc, argv )
int     argc;
char    *argv[];
{
  float         cep[64];                /* current frame cepstrum */
  float         cep_n[64];              /* next frame cepstrum */
  short         out_sp[512];            /* synthesized output speech */

  float         shift_cep0;             /* cepstrum[0] shift value */
  float         rnd();                  /* random noise generation routine */
  int           i;
  int           fd_cep;                 /* cepstrum(f) file descriptor */
  int           fd_pit;                 /* pitch(i) file descriptor */
  int           fd_syn;                 /* synthesized speech file descriptor */
  int           nframe;                 /* frame number */
  int           ptr;                    /* pulse pointer */
  int           pitch;                  /* current frame pitch period */
  int           pitch_n;                /* next frame pitch period */

  static double filt_io[256];           /* LMA filter input/output */
  static double filt_buf[1024];         /* LMA filter buffer */

  int           sframe = 60;            /* frame shift = 5ms */
  int           ord_cep = 30;           /* cepstrum order */

  rnd( 9 );

/*------- check arguement consistency ----------------------------------------*/
  if( argc == 4 ) shift_cep0 = 0.;
  else if( argc == 5 ) shift_cep0 = atof( argv[4] );
  else
  {
    printf( " - synthesize speech from cepstrum using LMA filter -\n" );
    printf( " - using cepstrum(f) file and pitch(i) file -\n" );
    printf( "[ USAGE ] cep_lma CEPF PITF SYNF          or\n" );
```

```
    printf( "[ USAGE ] cep_lma CEPF PITF SYNF CEP0S(0.)\n" );
    exit();
  }

/*------- start time --------------------------------------------------------*/
  t_start();

/*------- open files --------------------------------------------------------*/
  if( ( fd_cep=open(argv[1], 0) ) == -1 )
  {
    printf( " * File Open Error ! CEPF = %s\n", argv[1] );
    exit();
  }
  if( ( fd_pit=open(argv[2], 0) ) == -1 )
  {
    printf( " * File Open Error ! PITF = %s\n", argv[2] );
    exit();
  }
  fd_syn = creat( argv[3], 0644 );

/*------- print conditions --------------------------------------------------*/
  printf( " ************************************\n" );
  printf( " ***** CEPSTRUM -> LMA synthesis ***** ( cep_lma.c )\n" );
  printf( " ************************************\n\n" );
  printf( " Cepstrum(f) File        = %s\n", argv[1] );
  printf( " Pitch(i) File           = %s\n", argv[2] );
  printf( " Synthesized Speech File = %s\n\n", argv[3] );
  printf( " frame shift=%3d   cepstrum order=%2d  ", sframe, ord_cep );
  printf( " cep[0] shift=%-4.1f\n\n", shift_cep0 );

/*---------------------------------------------------------------------------*/
/*                                                                           */
/*              process                                                      */
/*                                                                           */
/*---------------------------------------------------------------------------*/
/*------- initialize counters -----------------------------------------------*/
  nframe = 1;
  ptr = 0;

/*------- read cepstrum data ( current frame ) ------------------------------*/
  if( read(fd_cep, cep, 4*ord_cep+4) == 0 ) eof( argv[1], nframe );
  if( read(fd_pit, &pitch, 4) == 0 ) eof( argv[2], nframe );
  cep[0] -= shift_cep0;

/*------- read cepstrum data ( next frame ) ---------------------------------*/
  while( 1 )
  {
    if( read(fd_cep, cep_n, 4*ord_cep+4) == 0 ) eof( argv[1], nframe );
    if( read(fd_pit, &pitch_n, 4) == 0 ) eof( argv[2], nframe );
    cep_n[0] -= shift_cep0;

/*------- generate filter exciting function ---------------------------------*/
    plsg( filt_io, &ptr, sframe, pitch );

/*------- digital filtering -------------------------------------------------*/
    digfi( filt_io, filt_buf, cep, cep_n, ord_cep, sframe );

/*------- write speech data -------------------------------------------------*/
    for( i=0; i<sframe; i++ ) out_sp[i] = (short)filt_io[i];
    write( fd_syn, out_sp, 2*sframe );

/*------- shift data --------------------------------------------------------*/
    f_copy( cep, cep_n, ord_cep+1 );
    pitch = pitch_n;
```

```
/*------- increment counter ----------------------------------------------------*/
    ++(nframe);
    fflush( stdout );
  }
}


/******** clear double array ***************************************************/
/*                                                                             */
/*      x:      [out] cleared double array                                     */
/*      n:      [in ] array length                                             */
/*                                                                             */
/*******************************************************************************/
d_zero( x, n )
double  x[];
int     n;
{
  int   i;

  for( i=0; i<n; i++ )          x[i] = 0.;
}

/******** copy x from y ( floating ) *******************************************/
/*                                                                             */
/*      x:      [out] copied float array                                       */
/*      y:      [in ] source float array                                       */
/*      n:      [in ] copy length                                              */
/*                                                                             */
/*******************************************************************************/
f_copy( x, y, n )
float   x[], y[];
int     n;
{
  int   i;

  for( i=0; i<n; i++ )          x[i] = y[i];
}

/******** generate pulse train *************************************************/
/*                                                                             */
/*      filt_io:        [out] filter exciting function                         */
/*      ptr:            [i/o] pulse pointer                                    */
/*      sframe:         [in ] frame shift                                      */
/*      pitch:          [in ] pitch period                                     */
/*                                                                             */
/*******************************************************************************/
plsg( filt_io, ptr, sframe, pitch )
double  filt_io[];
int     *ptr, sframe, pitch;
{
  int           i;
  static double pulse_amp = 1.0;        /* pulse amplitude */
  static double noise_amp = 0.125;      /* noise amplitude */

  d_zero( filt_io, sframe );

/*------- generate random noise ------------------------------------------------*/
  if( pitch == 0 )
  {
    for( i=(*ptr); i<sframe; i++ )
    {
      filt_io[i] = 2.*rnd(1) - 1.;
      if( filt_io[i] >= 0.0 ) filt_io[i] = noise_amp;
      else if( filt_io[i] < 0.0 ) filt_io[i] = -noise_amp;
    }
```

```
    *ptr = 0;
    return;
  }

/*------- generate pulse train ------------------------------------------------*/
  while( *ptr < sframe )
  {
    filt_io[*ptr] = pulse_amp;
    *ptr += pitch;
  }
  *ptr -= sframe;
}

/******** digital filter ***************************************************/
/*                                                                         */
/*      filt_io:        [in ] filter exciting function                     */
/*                      [out] synthesized speech                           */
/*      filt_buf:       [i/o] filter buffer                                */
/*      cep:            [in ] current frame cepstrum                       */
/*      cep_n:          [in ] next frame cepstrum                          */
/*      ord_cep:        [in ] cepstrum order                               */
/*      sframe:         [in ] frame shift                                  */
/*                                                                         */
/***************************************************************************/
digfi( filt_io, filt_buf, cep, cep_n, ord_cep, sframe )
double  filt_io[], filt_buf[];
float   cep[], cep_n[];
int     ord_cep, sframe;
{
  double        cep_diff[64];           /* difference between cep_n & cep */
  double        cep_itpl[64];           /* interpolated cepstrum */
  int           i, j, lv;
  static int    itvl_itpl = 15;         /* interpolation interval */

  if( ord_cep < 0 ) return;
  lv = sframe/itvl_itpl;

  for( i=0; i<=ord_cep; i++ )
       cep_diff[i] = ((double)cep_n[i] - (double)cep[i])/(double)lv;

  for( j=1; j<=lv; j++ )
  {
    for( i=0; i<=ord_cep; i++ )
         cep_itpl[i] = (double)cep[i] + cep_diff[i]*(double)j;
    for( i=(j-1)*itvl_itpl; i<j*itvl_itpl; i++ )
    {
      lma( &(filt_io[i]), filt_buf, cep_itpl, ord_cep );
      if( fabs(filt_io[i]) > 32767. )
           printf( "\n !!! overflow !!! %6.0f\n", filt_io[i] );
    }
  }
}

/******** log magnitude approximate filter *********************************/
/*                                                                         */
/*      x:              [i/o] input/output of LMA filter                   */
/*      filt_buf:       [i/o] filter buffer                                */
/*      cep:            [in ] cepstrum                                     */
/*      ord_cep:        [in ] cepstrum order                               */
/*                                                                         */
/***************************************************************************/
lma( x, filt_buf, cep, ord_cep )
double  *x, filt_buf[], cep[];
int     ord_cep;
{
```

```
  double        w0, x0, wn1[3];
  int           i, i0, j, k, ms, ms0;
  static double fk1 = 0.999071235/2.;
  static double fk2[3] = { 0., 0.999870613/2., 0.994034994/2. };
  static int    ll[64] = { 1, 3, 3, 2, 2, 2, 2, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1 };

  *x *= exp( cep[0] );

  ms = 0;

  for( i0=1; i0<=ord_cep; i0++ )
       for( i=1; i<=ll[i0]; i++ )
       {
         x0 = *x;
         ms0 = ms;
         for( j=1; j<=2; j++ )
         {
           if( i0 != 1 )
                for( k=1; k<i0; k++ )
                {
                  w0 = filt_buf[++ms];
                  filt_buf[ms] = *x;
                  *x = w0;
                }
           w0 = filt_buf[++ms];
           filt_buf[ms] = *x;
           *x = w0*cep[i0];
           wn1[j] = *x * fk2[j] / pow((double)ll[i0], (double)j);
           if( fabs(wn1[j]) <= 1.0e-15 ) wn1[j] = 0.;
         }
         filt_buf[ms0+1] = filt_buf[ms0+1] + wn1[1] - wn1[2];
         *x = x0 + 2.*wn1[1];
       }
}

/******** generate random noise ************************************************/
/*                                                                             */
/*      x:      [i/o] input/output of random routine                           */
/*                                                                             */
/*******************************************************************************/
float   rnd( x )
short   x;
{
  static short  ix=1, init_on=0;

  if( ((x % 2)!=0) && (init_on==0) )
  {
    ix = x;
    init_on = 1;
  }                     /* set init flag */

  ix *=  899;
  if( ix < 0 ) ix = ix + 32767 + 1;

  return( (float)ix/32768.0 );
}

/******** End Of File **********************************************************/
/*                                                                             */
```

```
/*      file:   [in ] filename which is on End Of File                          */
/*      nframe: [in ] frame number                                              */
/*                                                                              */
/********************************************************************************/
eof( file, nframe )
char    file[];
int     nframe;
{
  printf( " * End Of File ! %s [ %d ]\n", file, nframe );
  t_end();
  exit();
}

/******** display start time ***************************************************/
/*                                                                              */
/********************************************************************************/
t_start()
{
  int   ts;

  ts = time( 0 );
  printf( "\n ---> program start %s\n", ctime(&ts) );
  fflush( stdout );
}

/******** display end time *****************************************************/
/*                                                                              */
/********************************************************************************/
t_end()
{
  int   te;

  te = time( 0 );
  printf( "\n <--- program end %s", ctime(&te) );
  times( &buff );
  printf( " My user time.....%f sec\n\n", buff.tms_utime/60. );
  fflush( stdout );
}
```

```
/*****************************************************************************/
/*                                                                           */
/*      CEPSTRUM Synthesis ( cp_h_ol.c )                                     */
/*                                                                           */
/*              - hybrid method using OLA and LMAF -                         */
/*                                                                           */
/*      Compiled by:                                                         */
/*              cc -o cp_h_ol cp_h_ol.c -lm                                  */
/*                                                                           */
/*      [ USAGE ]                                                            */
/*              cp_h_ol CEPF PITF SYNF CEP0S                                 */
/*                                                                           */
/*        Input:                                                             */
/*              CEPF:   input cepstrum(f) file                               */
/*              PITF:   input pitch(i) file                                  */
/*              SYNF:   output synthesized speech file                       */
/*              CEP0S:  input cepstrum[0] shift value                        */
/*                                                                           */
/*---------------------------------------------------------------------------*/
/*      Version 1.0     coded by k.abe                  12/06/88             */
/*****************************************************************************/

#include <stdio.h>
#include <math.h>
#include <sys/types.h>
#include <sys/times.h>

#define PI      3.14159265358979323846

struct  tms buff;

main( argc, argv )
int     argc;
char    *argv[];
{
  float         pse[512];               /* current frame power spectrum envelope */
  float         pse_n[512];             /* next frame power spectrum envelope */
  float         cep[64];                /* current frame cepstrum */
  float         cep_n[64];              /* next frame cepstrum */

  float         shift_cep0;             /* cepstrum[0] shift value */
  float         rnd();                  /* random noise generation routine */
  long          fsize();
  int           i;
  int           fd_cep;                 /* cepstrum(f) file descriptor */
  int           fd_pit;                 /* pitch(i) file descriptor */
  int           fd_syn;                 /* synthesized speech file descriptor */
  int           file_size;              /* file size */
  int           nframe;                 /* frame number */
  int           ptr;                    /* pulse pointer */
  int           pitch;                  /* current frame pitch period */
  int           pitch_n;                /* next frame pitch period */

  static double *buf;
  static short  *out_sp;

  int           sframe = 60;            /* frame shift = 5ms */
  int           ord_cep = 30;           /* cepstrum order */

  rnd( 9 );

/*------- check arguement consistency ----------------------------------------*/
  if( argc == 4 ) shift_cep0 = 0.;
  else if( argc == 5 ) shift_cep0 = atof( argv[4] );
  else
```

```
  {
    printf( "[ USAGE ] cp_h_ol CEPF PITF SYNF           or\n" );
    printf( "[ USAGE ] cp_h_ol CEPF PITF SYNF CEP0S(0.3)\n" );
    exit();
  }

/*------- start time ------------------------------------------------------------*/
  t_start();

/*------- open files ------------------------------------------------------------*/
  if( ( fd_cep=open(argv[1], 0) ) == -1 )
  {
    printf( " * File Open Error ! CEPF = %s\n", argv[1] );
    exit();
  }
  if( ( fd_pit=open(argv[2], 0) ) == -1 )
  {
    printf( " * File Open Error ! PITF = %s\n", argv[2] );
    exit();
  }
  fd_syn = creat( argv[3], 0644 );

/*------- set file size ---------------------------------------------------------*/
  file_size = (fsize(fd_pit)/4-1)*sframe + 512;
  lseek( fd_pit, 0, 0 );
  if( NULL == (buf=(double *)malloc(8*file_size)) )
  {
    printf( "!!! No Memories !!! KORA_1 !!!\n" );
    exit();
  }
  if( NULL == (out_sp=(short *)malloc(2*file_size)) )
  {
    printf( "!!! No Memories !!! KORA_2 !!!\n" );
    exit();
  }

/*-----------------------------------------------------------------------------*/
/*                                                                             */
/*              process                                                        */
/*                                                                             */
/*-----------------------------------------------------------------------------*/
/*------- initialize counters ---------------------------------------------------*/
  nframe = 0;
  ptr = 0;

/*------- read cepstrum data ( current frame ) ----------------------------------*/
  if( read(fd_cep, cep, 4*ord_cep+4) == 0 ) eof( argv[1], nframe );
  if( read(fd_pit, &pitch, 4) == 0 ) eof( argv[2], nframe );
  cep[0] -= shift_cep0;
  env( pse, cep, ord_cep, 512 );

/*------- read cepstrum data ( next frame ) -------------------------------------*/
  while( 1 )
  {
    if( read(fd_cep, cep_n, 4*ord_cep+4) == 0 )
    {
      for( i=0; i<file_size; i++ ) out_sp[i] = (short)buf[i];
      write( fd_syn, out_sp, 2*file_size );
      eof( argv[1], nframe );
    }
    if( read(fd_pit, &pitch_n, 4) == 0 )
    {
      for( i=0; i<file_size; i++ ) out_sp[i] = (short)buf[i];
      write( fd_syn, out_sp, 2*file_size );
      eof( argv[2], nframe );
```

```
    }
    cep_n[0] -= shift_cep0;
    env( pse_n, cep_n, ord_cep, 512 );

/*------- process voiced frame ----------------------------------------------*/
    if( pitch != 0 ) ola( buf, pse, pse_n, sframe, &ptr, pitch, nframe );

/*------- process unvoiced frame --------------------------------------------*/
    else if( pitch == 0 )
         flt( buf, cep, cep_n, ord_cep, sframe, &ptr, pitch_n, nframe );

/*------- shift data --------------------------------------------------------*/
    f_copy( pse, pse_n, 257 );
    f_copy( cep, cep_n, ord_cep+1 );
    pitch = pitch_n;

/*------- increment counter -------------------------------------------------*/
    ++(nframe);
    fflush( stdout );
  }
}


/******** file size ***************************************************************/
/*                                                                                */
/*      fd:     [i/o] file descriptor                                             */
/*                                                                                */
/**********************************************************************************/
long    fsize( fd )
int     fd;
{
  long  lseek();

  return( lseek(fd, 0, 2) );
}

/******** clear double array ******************************************************/
/*                                                                                */
/*      x:      [out] cleared double array                                        */
/*      n:      [in ] array length                                                */
/*                                                                                */
/**********************************************************************************/
d_zero( x, n )
double  x[];
int     n;
{
  int   i;

  for( i=0; i<n; i++ )          x[i] = 0.;
}

/******** copy x from y ( float -> double ) ***************************************/
/*                                                                                */
/*      x:      [out] copied double array                                         */
/*      y:      [in ] source float array                                          */
/*      n:      [in ] copy length                                                 */
/*                                                                                */
/**********************************************************************************/
fd_copy( x, y, n )
double  x[];
float   y[];
int     n;
{
  int   i;
```

```
  for( i=0; i<n; i++ )          x[i] = (double)y[i];
}

/******** copy x from y ( floating ) ******************************************/
/*                                                                            */
/*      x:      [out] copied float array                                      */
/*      y:      [in ] source float array                                      */
/*      n:      [in ] copy length                                             */
/*                                                                            */
/******************************************************************************/
f_copy( x, y, n )
float   x[], y[];
int     n;
{
  int   i;

  for( i=0; i<n; i++ )          x[i] = y[i];
}

/******** power ***************************************************************/
/*                                                                            */
/*      l:      [in ] 2^l                                                     */
/*      n:      [out] n = 2^l                                                 */
/*                                                                            */
/******************************************************************************/
int     ipow2( l )
int     l;
{
  int   n;

  n = 1;
  while( l != 0 )
  {
    n *= 2;
    l--;
  }
  return n;
}

/******** fast fourier transform ( FFT.c ) ************************************/
/*                                                                            */
/*      x:      [i/o] real part array                                         */
/*      y:      [i/o] imaginary part array                                    */
/*      l:      [in ] FFT size ( 2^l )                                        */
/*      f:      [in ] FFT flag ( -1.0:FFT  1.0:IFFT )                         */
/*                                                                            */
/******************************************************************************/
int fft( x, y, l, f )
double  *x, *y, f;
int     l;
{
  double        s, c, s1, c1, sc, x1, y1, t;
  int           i, i0, i1, j, l1, n, ns, n1, k, ipow2();

  n = ipow2( l );                       /* n = 2^l */
  n1 = n/2;     sc = PI;

  j = 0;
  for( i=0; i<n-1; i++ )                /* bit reverse counter */
  {
    if( i <= j )
    {
      t = x[i];         x[i] = x[j];            x[j] = t;
      t = y[i];         y[i] = y[j];            y[j] = t;
    }
```

```
    k = n/2;
    while( k <= j )
    {
      j = j - k;        k /= 2;
    }
    j = j + k;
  }

  ns = 1;
  while( ns <= n/2 )            /* main loop */
  {
    c1 = cos( sc );     s1 = sin( f*sc );
    c = 1.0;            s = 0.0;
    for( l1=0; l1<ns; l1++ )
    {
      for( i0=l1; i0<n; i0+=(2*ns) )
      {
        i1 = i0 + ns;
        x1 = x[i1]*c - y[i1]*s;         y1 = y[i1]*c + x[i1]*s;
        x[i1] = x[i0] - x1;             y[i1] = y[i0] - y1;
        x[i0] = x[i0] + x1;             y[i0] = y[i0] + y1;
      }
      t = c1*c - s1*s;  s = s1*c + c1*s;        c = t;
    }
    ns = ns*2;          sc = sc/2.0;
  }

  if( f > 0.0 )                 /* judge fft or ifft (update) */
       for( i=0; i<n; i++ )
       {
         x[i] /= (double)n;
         y[i] /= (double)n;
       }
}

/******** spectrum envelope ********************************************************/
/*                                                                                  */
/*      pse:            [out] log power spectrum envelope                           */
/*      cep:            [in ] cepstrum                                              */
/*      ord_cep:        [in ] cepstrum order                                        */
/*      n:              [in ] array length                                          */
/*                                                                                  */
/************************************************************************************/
env( pse, cep, ord_cep, n )
float   pse[], cep[];
int     ord_cep, n;
{
  double        x[512], y[512];
  int           i;

  d_zero( x, 512 );
  for( i=0; i<=ord_cep; i++ ) x[i] = 0.5*cep[i];
  for( i=n-ord_cep; i<n; i++ ) x[i] = 0.5*cep[n-i];
  x[0] *= 2.0;
  d_zero( y, 512 );
  fft( x, y, 9, -1.0 );
  for( i=0; i<n; i++ ) pse[i] = (float)x[i];
}

/******** wave overlapping add *****************************************************/
/*                                                                                  */
/*      buf:    [i/o] working buffer of output speech                               */
/*      pse:    [in ] current frame log pse                                         */
/*      pse_n:  [in ] next frame log pse                                            */
/*      sframe: [in ] frame shift                                                   */
```

```
/*      ptr:    [i/o] pulse pointer                                             */
/*      pitch:  [in ] pitch period                                              */
/*      nframe: [in ] frame number                                              */
/*                                                                              */
/********************************************************************************/
ola( buf, pse, pse_n, sframe, ptr, pitch, nframe )
double  buf[];
float   pse[], pse_n[];
int     sframe, *ptr, pitch, nframe;
{
  double        xi[512], y[512];
  int           i, offset_ptr;

/*------- check pointer ---------------------------------------------------*/
  while( *ptr < sframe )
  {

/*------- calculate interpolated pse --------------------------------------*/
    for( i=0; i<257; i++ )
         xi[i] = (double)((sframe-*ptr)*pse[i] + *ptr*pse_n[i])/(double)sframe;

/*------- calculate impulse response --------------------------------------*/
    d_zero( y, 512 );
    spec( xi, 512 );
    fft( xi, y, 9, 1.0 );

/*------- add working buff ------------------------------------------------*/
    offset_ptr = nframe*sframe + *ptr;
    for( i=0; i<256; i++ ) buf[offset_ptr+i] += xi[256-i];
    for( i=256; i<512; i++ ) buf[offset_ptr+i] += xi[i-256];

/*------- set current frame pointer of pulse center -----------------------*/
    *ptr += pitch;
  }

/*------- set next frame pointer of pulse center --------------------------*/
  *ptr -= sframe;
}

/******** spectrum ***************************************************************/
/*                                                                               */
/*      x:      [in ] log power spectrum envelope                                */
/*              [out] power spectrum envelope                                    */
/*      n:      [in ] array length                                               */
/*                                                                               */
/*********************************************************************************/
spec( x, n )
double  x[];
int     n;
{
  int   i;

  for( i=0; i<n/2+1; i++ ) x[i] = exp( x[i] );
  for( i=n/2+1; i<n; i++ ) x[i] = x[n-i];
}

/******** lma filtering **********************************************************/
/*                                                                               */
/*      buf:            [i/o] working buffer of output speech                    */
/*      cep:            [in ] current frame cepstrum                             */
/*      cep_n:          [in ] next frame cepstrum                                */
/*      ord_cep:        [in ] cepstrum order                                     */
/*      sframe:         [in ] frame shift                                        */
/*      ptr:            [i/o] pulse pointer                                      */
/*      pitch_n:        [in ] next frame pitch period                            */
```

```
/*      nframe:         [in ] frame number                                      */
/*                                                                              */
/********************************************************************************/
flt( buf, cep, cep_n, ord_cep, sframe, ptr, pitch_n, nframe )
double  buf[];
float   cep[], cep_n[];
int     ord_cep, sframe, *ptr, pitch_n, nframe;
{
  double        filt_io[256];
  int           i, offset_ptr;
  static double filt_buf[1024];

/*------- generate random noise -------------------------------------------------*/
  rndg( filt_io, ptr, sframe );

/*------- digital filtering -----------------------------------------------------*/
  digfi( filt_io, filt_buf, cep, cep_n, ord_cep, sframe );

/*------- overlap adding --------------------------------------------------------*/
  offset_ptr = nframe*sframe + 256;
  for( i=0; i<sframe; i++ ) buf[offset_ptr+i] += filt_io[i];

  if( pitch_n != 0 ) d_zero( filt_buf, 1024 );
}

/******** generate random noise ************************************************/
/*                                                                              */
/*      filt_io:        [out] filter exciting function                          */
/*      ptr:            [i/o] pulse pointer                                     */
/*      sframe:         [in ] frame shift                                       */
/*                                                                              */
/********************************************************************************/
rndg( filt_io, ptr, sframe )
double  filt_io[];
int     *ptr, sframe;
{
  int           i;
  static double noise_amp = 0.125;      /* noise amplitude */

  d_zero( filt_io, sframe );

/*------- generate random noise -------------------------------------------------*/
  for( i=(*ptr); i<sframe; i++ )
  {
    filt_io[i] = 2.*rnd(1) - 1.;
    if( filt_io[i] >= 0.0 ) filt_io[i] = noise_amp;
    else if( filt_io[i] < 0.0 ) filt_io[i] = -noise_amp;
  }
  *ptr = 0;
}

/******** digital filter *******************************************************/
/*                                                                              */
/*      filt_io:        [in ] filter exciting function                          */
/*                      [out] synthesized speech                                */
/*      filt_buf:       [i/o] filter buffer                                     */
/*      cep:            [in ] current frame cepstrum                            */
/*      cep_n:          [in ] next frame cepstrum                               */
/*      ord_cep:        [in ] cepstrum order                                    */
/*      sframe:         [in ] frame shift                                       */
/*                                                                              */
/********************************************************************************/
digfi( filt_io, filt_buf, cep, cep_n, ord_cep, sframe )
double  filt_io[], filt_buf[];
float   cep[], cep_n[];
```

```
int     ord_cep, sframe;
{
  double        cep_diff[64];           /* difference between cep_n & cep */
  double        cep_itpl[64];           /* interpolated cepstrum */
  int           i, j, lv;
  static int    itvl_itpl = 15;         /* interpolation interval */

  if( ord_cep < 0 ) return;
  lv = sframe/itvl_itpl;

  for( i=0; i<=ord_cep; i++ )
       cep_diff[i] = ((double)cep_n[i] - (double)cep[i])/(double)lv;

  for( j=1; j<=lv; j++ )
  {
    for( i=0; i<=ord_cep; i++ )
         cep_itpl[i] = (double)cep[i] + cep_diff[i]*(double)j;
    for( i=(j-1)*itvl_itpl; i<j*itvl_itpl; i++ )
    {
      lma( &(filt_io[i]), filt_buf, cep_itpl, ord_cep );
      if( fabs(filt_io[i]) > 32767. )
           printf( "\n !!! overflow !!! %6.0f\n", filt_io[i] );
    }
  }
}

/******** log magnitude approximate filter ************************************/
/*                                                                            */
/*      x:              [i/o] input/output of LMA filter                      */
/*      filt_buf:       [i/o] filter buffer                                   */
/*      cep:            [in ] cepstrum                                        */
/*      ord_cep:        [in ] cepstrum order                                  */
/*                                                                            */
/******************************************************************************/
lma( x, filt_buf, cep, ord_cep )
double  *x, filt_buf[], cep[];
int     ord_cep;
{
  double        w0, x0, wn1[3];
  int           i, i0, j, k, ms, ms0;
  static double fk1 = 0.999071235/2.;
  static double fk2[3] = { 0., 0.999870613/2., 0.994034994/2. };
  static int    ll[64] = { 1, 3, 3, 2, 2, 2, 2, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1 };

  *x *= exp( cep[0] );

  ms = 0;

  for( i0=1; i0<=ord_cep; i0++ )
       for( i=1; i<=ll[i0]; i++ )
       {
         x0 = *x;
         ms0 = ms;
         for( j=1; j<=2; j++ )
         {
           if( i0 != 1 )
                for( k=1; k<i0; k++ )
                {
                  w0 = filt_buf[++ms];
```

```
                    filt_buf[ms] = *x;
                    *x = w0;
                  }
              w0 = filt_buf[++ms];
              filt_buf[ms] = *x;
              *x = w0*cep[i0];
              wn1[j] = *x * fk2[j] / pow((double)ll[i0], (double)j);
              if( fabs(wn1[j]) <= 1.0e-15 ) wn1[j] = 0.;
            }
            filt_buf[ms0+1] = filt_buf[ms0+1] + wn1[1] - wn1[2];
            *x = x0 + 2.*wn1[1];
          }
}

/******** generate random noise ***************************************************/
/*                                                                                */
/*      x:      [i/o] input/output of random routine                              */
/*                                                                                */
/**********************************************************************************/
float   rnd( x )
short   x;
{
  static short  ix=1, init_on=0;

  if( ((x % 2)!=0) && (init_on==0) )
  {
    ix = x;
    init_on = 1;
  }                     /* set init flag */

  ix *=  899;
  if( ix < 0 ) ix = ix + 32767 + 1;

  return( (float)ix/32768.0 );
}

/******** End Of File *************************************************************/
/*                                                                                */
/*      file:   [in ] filename which is on End Of File                            */
/*      nframe: [in ] frame number                                                */
/*                                                                                */
/**********************************************************************************/
eof( file, nframe )
char    file[];
int     nframe;
{
  printf( " * End Of File ! %s [ %d ]\n", file, nframe );
  t_end();
  exit();
}

/******** display start time ******************************************************/
/*                                                                                */
/**********************************************************************************/
t_start()
{
  int   ts;

  ts = time( 0 );
  printf( "\n ---> program start %s\n", ctime(&ts) );
  fflush( stdout );
}

/******** display end time ********************************************************/
/*                                                                                */
```

```
/*****************************************************************************/
t_end()
{
  int   te;

  te = time( 0 );
  printf( "\n <--- program end %s", ctime(&te) );
  times( &buff );
  printf( " My user time.....%f sec\n\n", buff.tms_utime/60. );
  fflush( stdout );
}
```