

Department of Electronic and Information Engineering
電子及資訊工程學系

The Hong Kong Polytechnic University

Center for Multimedia Signal Processing

CELP Decoder

FS1016 CELP Codec

Dr. M. W. Mak

enmwak@polyu.edu.hk

Tel: 27666257

Fax: 23628439

URL: www.en.polyu.edu.hk/~mwak/mypage.htm

3 Aug., 2000

Summary

This document describes the C implementation of the Code-Excited Linear Prediction (CELP 3.2a) decoder available from ftp://svr-ftp.eng.cam.ac.uk/comp.speech/coding/celp_3.2a.tar.gz.

1. FS-1016 Coding Characteristics

	Spectrum	Pitch	Code Book
	-----	-----	-----
Update	30 ms l1=240	30/4 = 7.5 ms lp=60	30/4 = 7.5 ms l=60
Order	10	256 (max) x 60 1 gain	512 (max) x 60 1 gain
Analysis	Open loop Correlation 30 ms Hamming no preemphasis 15 Hz BW exp	Closed loop Modified MSPE VQ, weight=0.8 range 20 to 147 (w/ fractions)	Closed loop MSPE VQ weight=0.8 shift by 2 samples 77% sparsely
Bits per Frame	34 indep LSP [34444433333]	index: 8+6+8+6 gain(-1,2): 5*4	index: 9*4 gain(+/-): 5*4
Bit Rate	1133.3 bps	1600 bps	1866.67 bps

NOTE: The remaining 200 bps are used as follows: 1 bit per frame for synchronization, 4 bits per frame for forward error correction and 1 bit per frame to provide future expansion(s) of the coder.

2. FS-1016 Bit Assignment

	Bit	Bit		Bit
	-----	-----		-----
lsp 1	1-3	lsp 6	20-22	
lsp 2	4-7	lsp 7	23-25	
lsp 3	8-11	lsp 8	26-28	
lsp 4	12-15	lsp 9	29-31	
lsp 5	16-19	lsp 10	32-34	
Subframe:	1	2	3	4
	-----	-----	-----	-----
pitch delay	35-42	87-94
delta delay	62-67	114-119
pitch gain	43-47	68-72	95-99	120-124
cbindex	48-56	73-81	100-108	125-133
cbgain	57-61	82-86	109-113	134-138

```

future bit      139
error control   140-143
sync           144

```

The sync bit (144) begins with 0 in the first frame, then alternates between 1 and 0 on successive frames.

3. CELP Decoding Algorithm

```

int    nLsp=10;           // No. of LSP coefficients per frame
int    nSubFrms=4;       // No. of subframes per analysis frame
int    lspIndex[nLsp];   // Index to LSP coding table
float  lspFreq[nLsp];    // LSP coefficients
int    lspTbl[nLsp][16]; // LSP quantization table
float  lspSubframe[4][10]; // Interpolated LSP for 4 subframes
int    nBitsPitchDelay=8; // 8 bits for pitch delay (D)
int    nBitsPitchDDelay=6; // 6 bits for delta delay (f)
int    nBitsPitchGain=5; // 5 bits for pitch gain
float  pitchDelay[4];    // Pitch delay for each subframe
float  pitchGain[4];     // Pitch gain for each frame
int    lspAlloc[10]={3, 4, 4, 4, 4, 3, 3, 3, 3, 3}; // Bit allocation for LSP coefficients
int    cbkIndex[4];      // Codebook index for each subframe
float  cbkGain[4];       // Codebook gain for each subframe
int    nBitsCbkIndex=9;  // 9 bits for codebook index
int    nBitsCbkGain=5;   // 5 bits for codebook gain
short  synSpeech[240];   // Buffer storing synthetic speech
int    cbk[1083];        // Stochastic codebook

```

celp_decode()

```

{
    Open(celpBitStream); // Open bitstream or look for sync bit
    while ((stream=Read(celpBitStream, 144))!=eof) { // Read 144 bits from file or bitstream
        for (i=0; i<nLsp; i++) { // For each LSP coefficient
            UnPack(stream, lspAlloc[i]; &lspIndex[i]); // Get LSP index from stream[]
        }
        // Decode lspIndex[] to lsp frequency using lspTbl[]
        LspDecode(lspIndex, lspFreq);

        // Interpolate lspFreq[] to form 4 interpolated subframe lsp
        InterpolateLsp(lspFreq, lspSubFrame);

        // Decode pitch delay for the 4 subframes
        DecodePitchDelay(stream, nBitsPitchDelay, nBitsPitchDDelay, pitchDelay);

        // Decode pitch gain for the 4 subframes
        DecodePitchGain(stream, nBitsPitchGain, pitchGain);
    }
}

```

```

// Decode stochastic codebook indexes for the 4 subframes
DecodeCbkJndex(stream, nBitsCbkJndex, cbkJndex);

// Decode stochastic codebook gain for the 4 subframes
DecodeCbkJGain(stream, nBitsCbkJGain, cbkJGain);

// Synthesize each subframe
for (sf = 0; sf < 4; sf++) {

    // Generate a gain scaled stochastic code vector based on codebook index and gain
    // (see the definition of the stochastic codebook cbkJ[] below)
    CoodBookSynthesis(cbkJndex[sf], cbkJGain[sf], &synSpeech[sf*60]);

    // Feed the scaled stochastic code vector to the long-term synthesis filter
    PitchSynthesis(pitchDelay[sf], pitchGain[sf], &synSpeech[sf*60]);

    // Feed the LTP output to the LSP synthesis filter (see diagram in the .ppt file)
    StpSynthesis(lspSubFrame[sf], &synSpeech[sf*60]);
}
// Write synthetic speech to output file
Write(synSpeech);
}
// End while loop
}

```

LspDecode

This function looks up the LSP table `lspTbl[][]` based on the LSP index, and returns 10 LSP frequencies

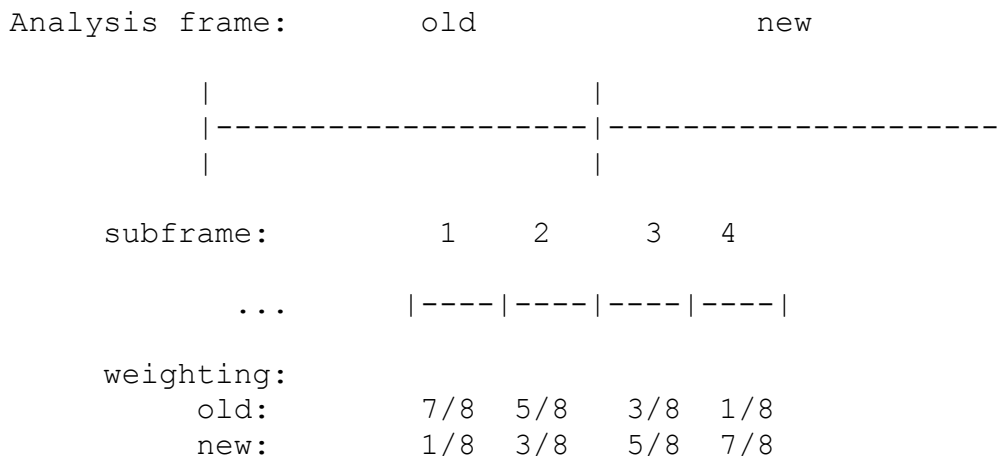
```
LspDecode(int lspIndex[], float lspFreq[])
{
    int i;
    for (i = 0; i < 10; i++)
        lspFreq[i] = lspTbl[i][lspIndex[i]] / 8000.0;
}
```

The LSP quantization table `lspTbl[0...9][0...15]` is defined as

```
lspTbl[0] = {100, 170, 225, 250, 280, 340, 420, 500, 0, 0, 0, 0, 0, 0, 0}
lspTbl[1] = {210, 235, 265, 295, 325, 360, 400, 440, 480, 520, 560, 610, 670, 740, 810, 880}
lspTbl[2] = {420, 460, 500, 540, 585, 640, 705, 775, 850, 950, 1050, 1150, 1250, 1350, 1450, 1550}
lspTbl[3] = {620, 660, 720, 795, 880, 970, 1080, 1170, 1270, 1370, 1470, 1570, 1670, 1770, 1870, 1970}
lspTbl[4] = {1000, 1050, 1130, 1210, 1285, 1350, 1430, 1510, 1590, 1670, 1750, 1850, 1950, 2050, 2150, 2250}
lspTbl[5] = {1470, 1570, 1690, 1830, 2000, 2200, 2400, 2600, 0, 0, 0, 0, 0, 0, 0}
lspTbl[6] = {1800, 1880, 1960, 2100, 2300, 2480, 2700, 2900, 0, 0, 0, 0, 0, 0, 0}
lspTbl[7] = {2225, 2400, 2525, 2650, 2800, 2950, 3150, 3350, 0, 0, 0, 0, 0, 0, 0}
lspTbl[8] = {2760, 2880, 3000, 3100, 3200, 3310, 3430, 3550, 0, 0, 0, 0, 0, 0, 0}
lspTbl[9] = {3190, 3270, 3350, 3420, 3490, 3590, 3710, 3830, 0, 0, 0, 0, 0, 0, 0}
```

InterpolateLsp

This function interpolates the LSP coefficients of the previous and the current analysis frames, and produces 40 LSP coefficients for 4 sub-frames.



```

InterpolateLsp(float lspFreq[], float lspSubFrame[])
{
  for (i = 0; i < 4; i++)
  {
    for (j = 0; j < 10; j++)
      lspSubFrame[i][j] = w[0][i] * lspOld[j] + w[1][i] * lspFreq[j];
  }
  // Update interpolated LSP history
  for (i = 0; i < 10; i++)
    lspOld[i] = lspFreq[i];
}
    
```

The LSP interpolation matrix $w[0...1][0...3]$ is defined as

```

w[2][4] = {
  0.875, 0.625, 0.375, 0.125,
  0.125, 0.375, 0.625, 0.875
};
    
```

The initial value of `lspOld` is defined as

```

lspOld[10] = { .03, .05, .09, .13, .19, .23, .29, .33, .39, .44 };
    
```

DecodePitchDelay

```

DecodePitchDelay(int stream[], int nBitsPitchDelay, int nBitsPitchDDelay, float pitchDelay[])
{
    int i, tptr, mxptr, mnptr;
    static int lptr = 0;

    for (i = 0; i < 4; i++)
    {
        // Process subframes 1 and 3
        if (((i + 1) % 2) != 0)
        {
            unpack(stream, nBitsPitchDelay, &tptr);
            pitchDelay[i] = pddecode[tptr];
        }
        else
        // Process subframe 2 and 4
        {
            unpack(stream, nBitsPitchDDelay, &tptr);
            mnptr = lptr - 31;
            mxptr = lptr + 32;
            if (mnptr < 0)
                mnptr = 0;
            if (mxptr > 255)
                mnptr = 192;
            pitchDelay[i] = pdelay[tptr + mnptr];
        }
        lptr = pdtabi[tptr];
    }
}

```

The pitch delay coding index pdtabi[0...255] is defined as

```

pdtabi[256] = {
112, 29, 23, 25, 218, 121, 114, 117, 113, 28, 22, 24, 219, 120, 115, 116, 217, 31,
111, 27, 222, 123, 220, 118, 216, 30, 110, 26, 223, 122, 221, 119, 169, 176, 172, 177,
189, 181, 188, 179, 170, 175, 173, 178, 187, 182, 186, 180, 59, 62, 60, 63, 69, 184,
68, 66, 171, 174, 61, 64, 185, 183, 67, 65, 196, 197, 0, 231, 194, 195, 1, 2, 191,
190, 70, 71, 192, 193, 72, 73, 80, 81, 241, 242, 78, 79, 4, 3, 6, 5, 236, 235, 76, 77,
75, 74, 138, 40, 238, 237, 139, 41, 243, 244, 146, 46, 150, 50, 144, 44, 151, 152,
141, 43, 143, 142, 140, 42, 240, 239, 147, 47, 148, 48, 145, 45, 149, 49, 105, 210,
108, 214, 15, 97, 20, 207, 102, 100, 107, 213, 16, 99, 19, 209, 104, 211, 109, 215,
14, 96, 21, 206, 103, 101, 106, 212, 17, 98, 18, 208, 93, 205, 84, 199, 92, 204, 83,
198, 13, 94, 82, 247, 12, 95, 7, 255, 88, 202, 86, 201, 90, 203, 85, 200, 10, 89, 9,
87, 11, 91, 8, 251, 52, 51, 54, 53, 153, 154, 159, 157, 156, 155, 160, 158, 245, 246,
249, 248, 58, 57, 55, 56, 168, 166, 161, 164, 167, 165, 162, 163, 254, 253, 250, 252,
135, 37, 133, 35, 134, 36, 132, 34, 229, 227, 129, 126, 131, 128, 130, 127, 137, 38,
232, 32, 136, 39, 233, 33, 234, 224, 225, 124, 230, 226, 228, 125
}

```

The pitch delay coding table `pdelay[0...255]` is defined as follows:

```

pdelay[256] = {
20.00000,    20.33334,    20.66667,    21.00000,    21.33334,    21.66667,
22.00000,    22.33334,    22.66667,    23.00000,    23.33334,    23.66667,
24.00000,    24.33334,    24.66667,    25.00000,    25.33334,    25.66667,
26.00000,    26.25000,    26.50000,    26.75000,    27.00000,    27.25000,
27.50000,    27.75000,    28.00000,    28.25000,    28.50000,    28.75000,
29.00000,    29.25000,    29.50000,    29.75000,    30.00000,    30.25000,
30.50000,    30.75000,    31.00000,    31.25000,    31.50000,    31.75000,
32.00000,    32.25000,    32.50000,    32.75000,    33.00000,    33.25000,
33.50000,    33.75000,    34.00000,    34.33334,    34.66667,    35.00000,
35.33334,    35.66667,    36.00000,    36.33334,    36.66667,    37.00000,
37.33334,    37.66667,    38.00000,    38.33334,    38.66667,    39.00000,
39.33334,    39.66667,    40.00000,    40.33334,    40.66667,    41.00000,
41.33334,    41.66667,    42.00000,    42.33334,    42.66667,    43.00000,
43.33334,    43.66667,    44.00000,    44.33334,    44.66667,    45.00000,
45.33334,    45.66667,    46.00000,    46.33334,    46.66667,    47.00000,
47.33334,    47.66667,    48.00000,    48.33334,    48.66667,    49.00000,
49.33334,    49.66667,    50.00000,    50.33334,    50.66667,    51.00000,
51.33334,    51.66667,    52.00000,    52.33334,    52.66667,    53.00000,
53.33334,    53.66667,    54.00000,    54.33334,    54.66667,    55.00000,
55.33334,    55.66667,    56.00000,    56.33334,    56.66667,    57.00000,
57.33334,    57.66667,    58.00000,    58.33334,    58.66667,    59.00000,
59.33334,    59.66667,    60.00000,    60.33334,    60.66667,    61.00000,
61.33334,    61.66667,    62.00000,    62.33334,    62.66667,    63.00000,
63.33334,    63.66667,    64.00000,    64.33334,    64.66667,    65.00000,
65.33334,    65.66667,    66.00000,    66.33334,    66.66667,    67.00000,
67.33334,    67.66667,    68.00000,    68.33334,    68.66667,    69.00000,
69.33334,    69.66667,    70.00000,    70.33334,    70.66667,    71.00000,
71.33334,    71.66667,    72.00000,    72.33334,    72.66667,    73.00000,
73.33334,    73.66667,    74.00000,    74.33334,    74.66667,    75.00000,
75.33334,    75.66667,    76.00000,    76.33334,    76.66667,    77.00000,
77.33334,    77.66667,    78.00000,    78.33334,    78.66667,    79.00000,
79.33334,    79.66667,    80.00000,    81.00000,    82.00000,    83.00000,
84.00000,    85.00000,    86.00000,    87.00000,    88.00000,    89.00000,
90.00000,    91.00000,    92.00000,    93.00000,    94.00000,    95.00000,
96.00000,    97.00000,    98.00000,    99.00000,    100.00000,    101.00000,
102.00000,    103.00000,    104.00000,    105.00000,    106.00000,    107.00000,
108.00000,    109.00000,    110.00000,    111.00000,    112.00000,    113.00000,
114.00000,    115.00000,    116.00000,    117.00000,    118.00000,    119.00000,
120.00000,    121.00000,    122.00000,    123.00000,    124.00000,    125.00000,
126.00000,    127.00000,    128.00000,    129.00000,    130.00000,    131.00000,
132.00000,    133.00000,    134.00000,    135.00000,    136.00000,    137.00000,
138.00000,    139.00000,    140.00000,    141.00000,    142.00000,    143.00000,
144.00000,    145.00000,    146.00000,    147.00000
}

```

The pitch decode permutation table `pddecode[]` is defined as:

```

pddecode[256] = {
54.66667, 28.75000, 27.25000, 27.75000, 110.00000, 57.66667, 55.33334,
56.33334, 55.00000, 28.50000, 27.00000, 27.50000, 111.00000, 57.33334,
55.66667, 56.00000, 109.00000, 29.25000, 54.33334, 28.25000, 114.00000,
58.33334, 112.00000, 56.66667, 108.00000, 29.00000, 54.00000, 28.00000,
115.00000, 58.00000, 113.00000, 57.00000, 73.66667, 76.00000, 74.66667,

```

76.33334, 81.00000, 77.66667, 80.00000, 77.00000, 74.00000, 75.66667,
75.00000, 76.66667, 79.66667, 78.00000, 79.33334, 77.33334, 37.00000,
38.00000, 37.33334, 38.33334, 40.33334, 78.66667, 40.00000, 39.33334,
74.33334, 75.33334, 37.66667, 38.66667, 79.00000, 78.33334, 39.66667,
39.00000, 88.00000, 89.00000, 20.00000, 123.00000, 86.00000, 87.00000,
20.33334, 20.66667, 83.00000, 82.00000, 40.66667, 41.00000, 84.00000,
85.00000, 41.33334, 41.66667, 44.00000, 44.33334, 133.00000, 134.00000,
43.33334, 43.66667, 21.33334, 21.00000, 22.00000, 21.66667, 128.00000,
127.00000, 42.66667, 43.00000, 42.33334, 42.00000, 63.33334, 31.50000,
130.00000, 129.00000, 63.66667, 31.75000, 135.00000, 136.00000, 66.00000,
33.00000, 67.33334, 34.00000, 65.33334, 32.50000, 67.66667, 68.00000,
64.33334, 32.25000, 65.00000, 64.66667, 64.00000, 32.00000, 132.00000,
131.00000, 66.33334, 33.25000, 66.66667, 33.50000, 65.66667, 32.75000,
67.00000, 33.75000, 52.33334, 102.00000, 53.33334, 106.00000, 25.00000,
49.66667, 26.50000, 99.00000, 51.33334, 50.66667, 53.00000, 105.00000,
25.33334, 50.33334, 26.25000, 101.00000, 52.00000, 103.00000, 53.66667,
107.00000, 24.66667, 49.33334, 26.75000, 98.00000, 51.66667, 51.00000,
52.66667, 104.00000, 25.66667, 50.00000, 26.00000, 100.00000, 48.33334,
97.00000, 45.33334, 91.00000, 48.00000, 96.00000, 45.00000, 90.00000,
24.33334, 48.66667, 44.66667, 139.00000, 24.00000, 49.00000, 22.33334,
147.00000, 46.66667, 94.00000, 46.00000, 93.00000, 47.33334, 95.00000,
45.66667, 92.00000, 23.33334, 47.00000, 23.00000, 46.33334, 23.66667,
47.66667, 22.66667, 143.00000, 34.66667, 34.33334, 35.33334, 35.00000,
68.33334, 68.66667, 70.33334, 69.66667, 69.33334, 69.00000, 70.66667,
70.00000, 137.00000, 138.00000, 141.00000, 140.00000, 36.66667, 36.33334,
35.66667, 36.00000, 73.33334, 72.66667, 71.00000, 72.00000, 73.00000,
72.33334, 71.33334, 71.66667, 146.00000, 145.00000, 142.00000, 144.00000,
62.33334, 30.75000, 61.66667, 30.25000, 62.00000, 30.50000, 61.33334,
30.00000, 121.00000, 119.00000, 60.33334, 59.33334, 61.00000, 60.00000,
60.66667, 59.66667, 63.00000, 31.00000, 124.00000, 29.50000, 62.66667,
31.25000, 125.00000, 29.75000, 126.00000, 116.00000, 117.00000, 58.66667,
122.00000, 118.00000, 120.00000, 59.00000
}

DecodePitchGain

```

DecodePitchGain(int stream[], int nBitsPitchGain, float pitchGain)
{
    int pitchGainIndex;
    for (i = 0; i < 4; i++) {

        // Extract pitch gain index from input stream
        Unpack(stream, nBitsPitchGain, &pitchGainIndex);

        // Look up pitch gain from pitch gain table
        pitchGain = pitchGainTbl[pitchGainIndex];
    }
}

```

The pitch gain table pitchGainTbl[0...31] is defined as follows:

```

pitchGainTbl[32] =
{
    -0.993, -0.831, -0.693, -0.555, -0.414, -0.229, 0.0, 0.139,
    0.255, 0.368, 0.457, 0.531, 0.601, 0.653, 0.702, 0.745,
    0.780, 0.816, 0.850, 0.881, 0.915, 0.948, 0.983, 1.020,
    1.062, 1.117, 1.193, 1.289, 1.394, 1.540, 1.765, 1.991
};

```

DecodeCbkIndex

```
DecodeCbkIndex(int stream[], int nBitsCbkIndex, int cbkIndex[])
{
    int i;
    for (i = 0; i < 4; i++) {
        Unpack(stream, nBitsCbkIndex, &cbkIndex[i]);
        cbkIndex[i] = cbkIndex[i] + 1;
    }
}
```

DecodeCbkGain

```
DecodeCbkGain(int stream[], int nBitsCbkGain, int cbkGain)
{
    int i;
    int cbkGainIndex;
    for (i = 0; i < 4; i++) {
        Unpack(stream, nBitsCbkIndex, &cbkGainIndex);
        cbkGain[i] = cbkGainTbl[cbkGainIndex];
    }
}
```

```
cbkGainTbl[32] =
{
    -1330., -870., -660., -520., -418., -340., -278., -224.,
    -178., -136., -98., -64., -35., -13., -3., -1.,
    1., 3., 13., 35., 64., 98., 136., 178.,
    224., 278., 340., 418., 520., 660., 870., 1330.
};
```

CodeBookSynthesis

```
CodeBookSynthesis(int cbkIndex, float cbkGain, float y[])
{
    int i;

    // For each sample in the subframe
    for (i = 0; i < 60; i++)
        y[i] = cbk[i + 2*(512-cbkIndex)] * cbkGain;
}
```

The stochastic codebook cbk[0...1083] is defined as follows:

```
cbk[1084] = {
    0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 1., 0., 0., 0., 0., -1.,
    0., 0., -1., 0., -1., 0., 1., 0., 0., 0., 0., 0., 0., -1., 0., -1., -1., 0., 0.,
-1., 0., -1., 0., -1., 0., -1., 0., 0., 0., 0., 0., 0., 0., -1., 0., 0., 0., 1., 0.,
    0., 0., 0., -1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., -1., 0.,
    1., 0., 1., 0., -1., 1., 0., 0., 0., 0., 0., -1., 0., 0., 1., 0., 0., 0., -1., 0.,
    0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
    0., -1., -1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., -1., -1.,
    0., 1., -1., 0., 0., -1., -1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
    0., 0., 0., -1., 0., -1., -1., 0., 0., 0., -1., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
    0., -1., 0., 0., 0., -1., 0., -1., 0., 0., 0., -1., 0., 0., 0., 0., 0., 0., 0., 0.,
    0., 0., -1., 1., 0., 0., 0., -1., 1., 0., 0., 1., 0., 0., -1., 0., 0., 0., 0., 0.,
    0., -1., 0., 0., 1., -1., 0., 0., 1., 0., 0., 0., 0., 0., 1., 1., 0., 1., 1., 0., 0.,
    1., 1., 0., 0., 0., -1., 0., 0., 0., -1., 0., 0., -1., 1., 0., 0., 1., 1., 0., 0., -1.,
-1., 0., 0., 0., 0., -1., 0., 0., 0., 0., 0., -1., 0., 0., 0., 0., 0., 0., 0., 0.,
    0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0., -1., -1., 0., 0., 1., 0.,
    0., 0., 0., 0., 0., -1., 0., 0., -1., 1., 0., 0., 0., 0., 0., 0., -1., 0., -1.,
    0., 0., 0., 0., 0., 0., -1., 0., 0., -1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
    0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., -1., 0.,
    0., 0., 0., 1., -1., 0., 0., -1., 0., -1., 0., 0., -1., 0., 0., 1., 0., 0., 0., 0.,
-1., -1., 0., -1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
-1., 0., 1., 0., 0., 0., -1., 0., 0., 0., -1., 0., 0., 0., 0., 0., 0., 0., 0., -1., 0.,
    0., -1., -1., 0., 0., 0., 1., 0., -1., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
    0., 0., 1., 1., 0., 0., 0., 1., 1., 0., 0., -1., 0., 0., 0., 0., 0., -1., 1.,
    1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
    0., 0., 0., 0., 0., 0., 0., 0., -1., 0., 1., 1., 0., 0., 0., 1., 0., 0., 1.,
    0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., -1.,
-1., 1., 0., 0., 0., 1., 0., 1., 0., 1., 0., -1., 0., 0., -1., 0., 0., 0., 0., 0.,
    0., -1., 0., -1., 0., 0., 0., 0., -1., 0., 0., -1., 0., 1., 1., 0., 0., 0., 0.,
    0., 0., 0., 0., 0., 0., 0., -1., -1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
-1., 0., -1., -1., 0., 0., -1., 0., -1., 0., 0., 0., 0., 0., 0., 1., 0., 1., 0., 0.,
    0., 0., 0., 1., -1., 0., 1., 0., 0., 0., -1., 0., 0., 0., 0., -1., 0., 0., 1., 0.,
    0., -1., -1., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
    0., 1., 0., 0., 0., 1., 0., 0., 0., 0., -1., 0., 0., 0., -1., 0., 0., 0., 0., 0.,
    0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., -1., 0., 0.,
    0., 0., 0., -1., 0., 0., 0., 0., 0., 0., -1., 0., -1., 1., 1., 0., 0., 0., 0., 1.,
    0., 0., 0., 0., -1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
    0., 0., -1., 0., 0., 0., 1., 0., -1., 0., 0., 1., -1., 0., 1., 0., 0., 0., 0.,
    0., 0., -1., 0., 0., 0., 0., 0., -1., 0., 0., 1., 0., 0., 0., 0., 0., -1., 0.,
    0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 1., -1., 0., 0., 1., 0., 0., 0., 0., -1.,
    0., 0., 0., 0., 0., 0., 0., 0., -1., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0.,
    1., 0., 0., -1., 0., 0., -1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
    0., 0., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0., -1., 1., -1., 0., 1., 0., 0., 0.,
```

```

0., 0., 0., 0., 1., -1., 0., 0., 0., 0., 0., 0., 0., -1., 1., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., -1., 0., 0., 0., 0., -1., 0.,
0., 0., -1., 0., -1., 0., 0., 0., 0., 0., 0., -1., 0., 0., -1., 0., 0., 0., 0.,
0., 0., 0., 1., -1., 0., 0., 0., 0., -1., 0., 1., 0., 0., 1., 0., 0., 0., -1.,
0., 0., 0., 0., 1., 0., 0., 0., 0., -1., 0., 0., -1., 0., 0., 0., 0., 0., 1., 0.,
0.
}

```

The stochastic codebook contains $2*512+60=1084$ samples of $-1, 0,$ and $+1$ The k -th ($k=0,\dots,512$) codevector c_k is formed by $cbk[2(512-k)\dots 2(512-k)+59]$. For example,

$$c_{512} = cbk[0\dots 59]$$

$$c_{511} = cbk[2\dots 61]$$

$$c_{510} = cbk[4\dots 63]$$

.

.

$$c_1 = cbk[1022\dots 1081]$$

$$c_0 = cbk[1024\dots 1083]$$

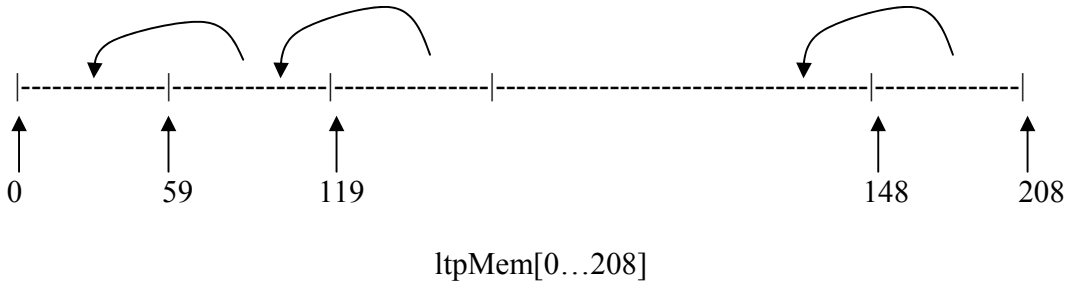
Therefore, the synthetic speech due to the k -th stochastic code vector is

$$synSpeech[i] = cbk[i+2*(512-k)] * cbkGain \quad i = 0,\dots,59 \text{ and } k = 0,\dots,512$$

PitchSynthesis

This procedure involves three steps: (1) update the LTP memory, (2) compute the excitation signals for the current subframe, and (3) synthesis speech using the LTP filter.

(1) Update LTP memory: the memory is updated by shifting memory elements as follows:



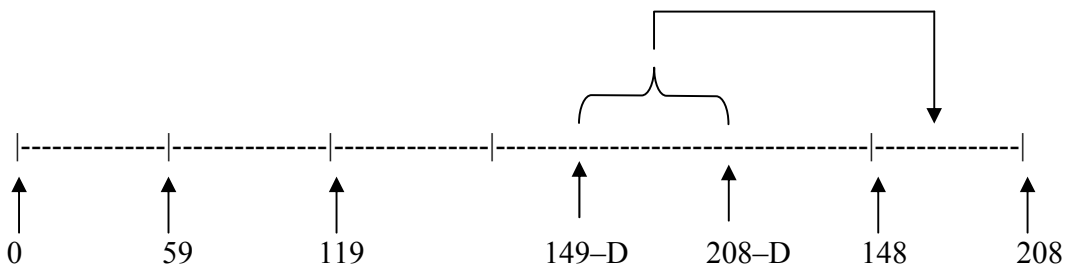
```
extern float ltpMem[209];
for (i = 0; i < 149; i++)
    ltpMem[i] = ltpMem[i+60];
```

Note that before this update, the buffer ltpMem[] contains the LTP synthetic speech of the previous sub-frames.

(2) We synthesize the speech for the current subframe using the LTP filter with transfer function $H(z) = 1/(1 - bz^{-D})$. Therefore, if $x[n]$ is the excitation signals and $y[n]$ the LTP filter's output, we have

$$y[n] = x[n] + by[n - D] \tag{Eqn. 1}$$

This means that we need to access $y[n-D]$ to generate $y[n]$. When $y[0]$ corresponds to ltpMem[149], what we need to do is to copy the previous LTP memory (delayed by D samples) to ltpMem[149...209], as follows:



```
for (i = 149; i < 209; i++)
    ltpMem[i] = ltpMem[i-D];
```

For fractional delay, the procedure calls FracDelay() to interpolate the LTP memory. The interpolated signals are again stored in ltpMem[149...208].

(3) To implement Eqn. 1, we do the following

```

for (i = 0; i < 60; i++) {
    y[i] = x[i] + b*ltpMem[149+i];
    ltpMem[149+i] = y[i];
}

```

Note that $x[0...59]$ is the gain scaled stochastic code vector produced by CodeBookSynthesis().

Example Implementation

PitchSynthesis(float pitchDelay, float pitchGain, float y[])

```

{
    float *x;
    extern float ltpMem[209];
    int i;
    float b = pitchGain;

    // Before running this function, y[] is the input excitation
    x = y;
    for (i = 0; i < 149; i++)                // Step 1
        ltpMem[i] = ltpMem[i+60];

    if (integerDelay==TRUE) {                // Step 2
        for (i = 149; i < 209; i++)
            ltpMem[i] = ltpMem[i-D];
    }
    else {
        FracDelay(ltpMem, pitchDelay);
    }
    for (i = 0; i < 60; i++) {                // Step 3
        y[i] = x[i] + b*ltpMem[149+i];        // Produce synthetic speech and store it in y[]
        ltpMem[149+i] = y[i];
    }
}

```

FracDelay

FracDelay(float ltpMem[], float pitchDelay)

This procedure computes the interpolated excitation signals based on the fractional part of pitchDelay and LTP memory ltpMem[]. The fractional part is first compared with the constant array f[0..4]={0.25, 0.33333, 0.5, 0.666667, 0.75}. The entry of f[] closest to the fractional part is identified as “n”. Then the filter member ltpMem[] (denoted as r[]) is updated as follows:

$$r[149 + i] = \sum_{j=0}^{39} r[149 - D + (j - 20) + i] * w[j][n]$$

where *D* is the integer delay and w[][] is defined as

w[40][5] = {

-0.0009157528	-0.0011301822	-0.0013290340	-0.0011766120	-0.0009726766
0.0010664382	0.0013298646	0.0015951599	0.0014385806	0.0011996499
-0.0013749709	-0.0017250286	-0.0020920311	-0.0019047754	-0.0015951820
0.0018631391	0.0023425117	0.0028510056	0.0026026941	0.0021818711
-0.0025545254	-0.0032114214	-0.0039062425	-0.0035624271	-0.0029845031
0.0034753943	0.0043642647	0.0052961037	0.0048179631	0.0040311604
-0.0046560653	-0.0058386908	-0.0070653162	-0.0064092022	-0.0053549218
0.0061330274	0.0076801768	0.0092683081	0.0083850641	0.0069964975
-0.0079522254	-0.0099462019	-0.0119744064	-0.0108083375	-0.0090083424
0.0101742521	0.0127128689	0.0152761098	0.0137634557	0.0114612682
-0.0128828110	-0.0160857141	-0.0193026867	-0.0173693206	-0.0144553734
0.0161990318	0.0202180520	0.0242434125	0.0218014307	0.0181390028
-0.0203068778	-0.0253437161	-0.0303894430	-0.0273320973	-0.0227433555
0.0255011376	0.0318392329	0.0382142961	0.0344087631	0.0286503248
-0.0322854556	-0.0403517857	-0.0485420339	-0.0438203216	-0.0365377814
0.0415940434	0.0520915091	0.0629395396	0.0570935421	0.0477298014
-0.0553679951	-0.0696018860	-0.0847808793	-0.0776027218	-0.0651931837
0.0783885419	0.0992630646	0.1228656545	0.1145323291	0.0971909687
-0.1263953149	-0.1628061831	-0.2095094770	-0.2046700865	-0.1784717441
0.2991485298	0.4124546945	0.6357170343	0.8264719844	0.8999969959
0.8999969959	0.8264719844	0.6357169747	0.4124546349	0.2991485298
-0.1784717441	-0.2046701014	-0.2095094770	-0.1628061682	-0.1263953000
0.0971909612	0.1145323217	0.1228656545	0.0992630497	0.0783885419
-0.0651931763	-0.0776027218	-0.0847808719	-0.0696018785	-0.0553679913
0.0477297977	0.0570935383	0.0629395321	0.0520915054	0.0415940396
-0.0365377814	-0.0438203216	-0.0485420302	-0.0403517783	-0.0322854482
0.0286503229	0.0344087631	0.0382142924	0.0318392292	0.0255011357
-0.0227433518	-0.0273320954	-0.0303894412	-0.0253437087	-0.0203068759
0.0181390010	0.0218014307	0.0242434107	0.0202180464	0.0161990318
-0.0144553725	-0.0173693206	-0.0193026830	-0.0160857085	-0.0128828101
0.0114612654	0.0137634547	0.0152761079	0.0127128661	0.0101742502
-0.0090083415	-0.0108083356	-0.0119744046	-0.0099461991	-0.0079522235
0.0069964956	0.0083850622	0.0092683071	0.0076801744	0.0061330264
-0.0053549209	-0.0064092013	-0.0070653148	-0.0058386894	-0.0046560639
0.0040311590	0.0048179622	0.0052961023	0.0043642633	0.0034753936
-0.0029845024	-0.0035624264	-0.0039062414	-0.0032114205	-0.0025545249
0.0021818704	0.0026026936	0.0028510047	0.0023425107	0.0018631388
-0.0015951816	-0.0019047750	-0.0020920306	-0.0017250280	-0.0013749705
0.0011996495	0.0014385802	0.0015951595	0.0013298644	0.0010664379

CMSP, HKPolyU

-0.0009726765 -0.0011766119 -0.0013290339 -0.0011301821 -0.0009157527
}