telecommunications & signal processing laboratory

ITU-T G.723.1 Speech Coder: A Matlab Implementation

P. Kabal

Department of Electrical & Computer Engineering

McGill University

McGill

Version 1: 2003-11-21

Table of Contents

ITU-T G.723.1 Speech Coder:
A Matlab Implementation

This report documents the details of the processing steps in the ITU-T G.723.1 Speech Coder. This report accompanies an implantation of that coder in Matlab. The Matlab implementation was designed to facilitate experimentation and research using a practical speech coder as a base.

1 Introduction

This document describes the ITU-T G.723.1 speech coder [1] and a Matlab implementation of that coder. The implementation is based on the ITU-T floating-point reference code. G.723.1 is a CELP (Code-Excited Linear Prediction Coder). The Matlab implementation supports the multipulse coding mode (operating at 6.3 kb/s). This document gives an overview of the coder and decoder, and fills in details of the processing steps.

The implementation gives results that are close but not identical to those from the reference code. Careful checking of several input files has shown that the differences are in isolated frames. The first subframe in the first frame is problematic, because that subframe is all zeros. The placement of pulses in that subframe influences future subframes (since the pulse amplitudes are always non-zero). Fortunately, differences in the first frame tend to stay isolated to that frame. Occasionally, the quantized predictor coefficients will differ from those generated by the reference code. The effect of that difference tends to propagate for several frames. In all cases, the output coded speech is indistinguishable from that produced by the reference code.

Given a bitstream file, the decoder implemented in Matlab generates the same speech file as the reference code.

Unimplemented Features

At the moment, the coder works only at 6.3 kb/s. It does not implement the voice activity detector (VAD) feature. The decoder also works only at 6.3 kb/s and does not handle lost frames or inactive frames. In addition, the postfiltering operations have not been implemented in the decoder.

2 Generalities

2.1 Input / Output

The input to the Matlab version of the coder is a speech file (8 kHz sampling rate) in one of several file formats (WAVE, raw, AU, Sphere). The output of the coder is either a bitstream file (compatible with the reference code) or a Matlab data file. The former contains codes and corresponds to the compressed bitstream from the coder. The latter stores values, rather then codes, and can be used to pass, for instance, unquantized values to the decoder.

The decoder takes as input either a bitstream file (from the Matlab implementation of the coder or the reference code implementation of the coder) or a data file (from the Matlab implementation of the coder). Its output is a WAVE file.

2.2 Filters

Many of the filters are implemented using a custom Matlab routine, `PZFilter`. This routine is an interface to the Matlab routine `filter`, which uses a direct form II structure. If a filter starts in zero-state and has constant coefficients, then the Matlab routine `filter` can be used directly. This is the case for the highpass filter used as a preprocessing step in the coder. However, for other filtering operations in which the coefficients change from frame-to-frame, the structure of the filter affects the results. The routine `PZFilter` does the extra operations (using the Matlab routine `filtic`) necessary to use the past input data and past output data as state values, thereby duplicating the results of the filtering as done in the reference code.

2.3 Data Values and Tabulated Value

This implementation uses the Matlab convention of audio data normalized to a full-scale value of unity. This means that internal data values will be 1/32768 of the values in the reference code. This scaling is kept throughout the program.

There are a number of tables in the reference code. The Matlab code can read these tables, but can also generate the data for many of the tables on the fly. For instance, the analysis window for LP analysis can either be read from a table (taken from the reference code and stored with 6 digits of accuracy) or generated at startup with full precision. Quantizer tables are read in from files.

3 Frame Level Processing

The coder and decoder operate on two time scales: a frame of 240 samples that is divided into subframes of length 60. The frame level operations involve the following steps.

- Highpass filter the input signal.
- Form an extended (highpass filtered) signal consisting of three parts: look-back samples, current frame samples, and look-ahead samples. The current frame samples are divided into 4 subframes.
- Linear prediction analysis is done on each subframe. This creates four sets of LP coefficients, one for each subframe.
- The LP coefficients for the last subframe (subframe number 3) are quantized (in the LSF domain, as described below).
- The quantized LP coefficients are linearly interpolated (in the LSF domain) using the quantized LP coefficients from the previous frame. This creates four sets of quantized LP coefficients, one for each subframe, the last of these being the quantized LP coefficients for subframe 3. These quantized coefficients will be used for the synthesis filter.
- The unquantized LP coefficients (4 sets) are used to form a formant perceptual weighting filter. This filter is used to weight the error signal during the search for the best excitation parameters, in effect directing the search procedure to take into account psycho-acoustic properties.
- The input signal (after highpass filtering) is processed with the formant perceptual weighting filter. The output of this filter is used to form an initial estimate of the pitch lag. This is termed the open-loop pitch estimate. This estimate is based on two subframes at a time, giving two open-loop pitch estimates per frame: one for subframes 0 and 1, and another for subframes 2 and 3.
- The open-loop pitch estimate is used to generate a second weighting filter, the harmonic noise weighting filter, which tends to emphasize the harmonic peaks during voiced speech.
- The input signal, processed by the combination of highpass filter, formant weighting filter and harmonic noise weighting filter forms the so-called target signal.

3.1 LP Analysis

The linear prediction (LP) analysis operates on the highpass filtered signal. LP analysis is carried out for each subframe. A 180 sample Hamming window is applied for each subframe. The window is centred on a subframe and so extends on either side of the subframe (60 samples back, 60 samples over the subframe, and 60 samples ahead). The look-back for the frame is 60 samples to accommodate the backwards extent of the window when processing the first subframe. The look-ahead for the frame is 60 samples to accommodate the forward extent of the window when processing the last subframe. The positions of the windows for the subframes are shown in Fig. 1

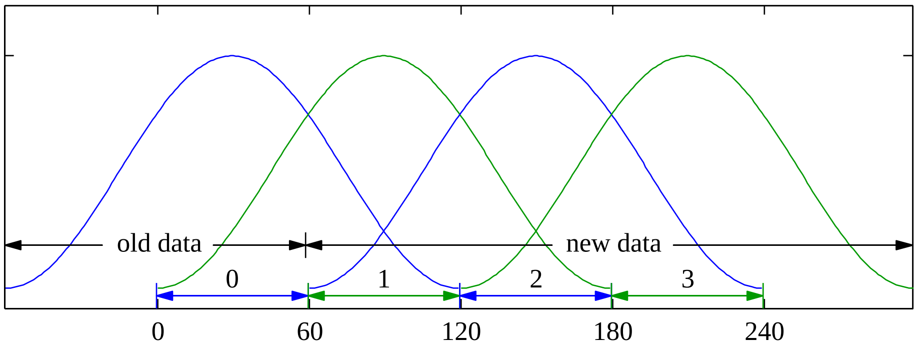

Fig. 1 LP windows.

Let the highpass filtered data for a subframe be $x[n]$. This is the data lying under the analysis window for that subframe. The windowed data is

$$x_w[n] = x[n]w[n] \qquad 0 \le n \le N_w - 1. \tag{1}$$

The prediction error is

$$e[n] = x_w[n] - \sum_{k=1}^{N_p} c_k x_w[n-k]. \tag{2}$$

For this equation, we can let n take on all values if we define $x_w[n]$ to be zero for $n < 0$ and $n \ge N_w$. Then writing the error in vector-matrix form,

$$\begin{aligned} \mathbf{e} &= \mathbf{x}_w^{(0)} - \left[\mathbf{x}_w^{(1)} \cdots \mathbf{x}_w^{(N_p-1)} \right] \mathbf{c} \\ &= \mathbf{x}_w^{(0)} - \mathbf{X}_w \mathbf{c}. \end{aligned} \tag{3}$$

where $\mathbf{x}_w^{(k)}$ is a shifted version of the windowed data vector,

$$\mathbf{x}_w^{(0)} = \begin{bmatrix} \vdots \\ 0 \\ x_w[0] \\ x_w[1] \\ \vdots \\ x_w[N_w-2] \\ x_w[N_w-1] \\ 0 \\ \vdots \end{bmatrix}, \quad \mathbf{x}_w^{(1)} = \begin{bmatrix} \vdots \\ 0 \\ 0 \\ x_w[0] \\ \vdots \\ x_w[N_w-3] \\ x_w[N_w-2] \\ x_w[N_w-1] \\ \vdots \end{bmatrix}, \cdots \tag{4}$$

The vector are shown as infinite in extent, but have only N_w non-zero elements. LP analysis chooses a set of predictor coefficients to minimize the squared-error,

$$\begin{aligned} \varepsilon &= \mathbf{e}^T\mathbf{e} \\ &= \mathbf{x}_w^{(0)T}\mathbf{x}_w^{(0)} - 2\mathbf{x}_w^{(0)}\mathbf{X}_w\mathbf{c} + \mathbf{c}^T\mathbf{X}_w^T\mathbf{X}_w\mathbf{c} \\ &= r[0] - 2\mathbf{c}^T\mathbf{r} + \mathbf{c}^T\mathbf{R}\mathbf{c}, \end{aligned} \tag{5}$$

where the correlation values are functions of the time differences,

$$r[k] = \mathbf{x}_w^{(n)T}\mathbf{x}_w^{(n-k)}, \quad \mathbf{r} = \begin{bmatrix} r[1] \\ r[2] \\ \vdots \\ r[N_p] \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} r[0] & r[1] & \cdots & r[N_p-1] \\ r[1] & r[0] & \cdots & r[N_p-2] \\ \vdots & \vdots & \ddots & \vdots \\ r[N_p-1] & r[N_p-2] & \cdots & r[0] \end{bmatrix}. \tag{6}$$

In the LP analysis for G.723.1, the correlation values are lag windowed (see [2]),

$$r'(k) = r(k)w_l(k). \tag{7}$$

The lag window is often chosen to have a Gaussian shape. Multiplying the correlation values by the Gaussian window is equivalent to convolving the power spectrum of the windowed signal by another Gaussian shape. The values of the lag window used in G.723.1 correspond approximately to a Gaussian frequency response with a one-sided bandwidth of 43 Hz.

The LP analysis also uses white noise compensation,

$$r''[k] = r'[k](1 + \mu\delta[k]), \tag{8}$$

where $\mu = 1/1024$. The use of this white noise is equivalent to using a modified error criterion

$$\varepsilon' = r[0] - 2\mathbf{c}^T\mathbf{r} + \mathbf{c}^T\mathbf{R}\mathbf{c} + \mu\mathbf{c}^T\mathbf{c}. \tag{9}$$

In this form, we can see that the error criterion has been augmented with a Lagrange multiplier that controls the sum of the squares of the coefficient vector.

The LP equations can be efficiently solved using the Durbin-Levinson recursion to give the optimal predictor coefficients,

$$\mathbf{R}\mathbf{c}_{opt} = \mathbf{r}. \tag{10}$$

Differences from the Reference Code

- The linear prediction analysis uses the built-in Matlab routine `levinson`. This may cause some small differences in the resulting values compared to the reference code.
- The reference code (function `Comp_Lpc`) scales the autocorrelation values by $1/N_f^2$, where N_f is the frame length (240). This is not done in the Matlab code since the LP analysis is insensitive to scaling.
- The correlation is lag windowed. In the reference code, the lag window is defined in a table. The documentation indicates that the lag window is based on binomial coefficients. The Matlab code can read the table of lag window values or calculate a Gaussian lag window,

  $$w[k] = \exp(-\frac{1}{2}(2\pi f_{\mathrm{BW}} k)^2). \tag{11}$$

  The reference code defines the lag window to 6 digits (values near unity). The value of f_{BW} which best matches the tabulated values is 42.4869/8000 (42.5 Hz for a 8000 Hz sampling rate). The maximum error in window values is 5×10^{-7}. An attempt was to find a true binomial window that matches the tabulated data. The closest binomial window is a binomial window of length $N = 4001$. Normalizing the window with respect to the middle coefficient and selecting the 11 coefficients starting with the middle coefficient gave an error of more than 0.001 with respect to the tabulated values.
- The reference code checks for zero energy, and if found, short-circuits the calculation of the remaining correlation coefficients by setting them to zero. The Matlab implementation does not do this check.

- The reference code for the Levinson-Durbin recursion (function `Durbin`) checks for numerical problems by checking for decreasing absolute error with order.[1] If an increase in error is found, the order recursion is aborted. The predictor coefficients have been initialized to zero, so in effect, any higher order predictor coefficients are set to zero. In addition, the second reflection coefficient is set to 0.99. This value is used outside the routine as a sine detector (used by the pitch prediction routine).

3.2 Sine Detector

After the Levinson recursion, a sine detector based on the second reflection coefficient is applied. The theory behind this scheme is described in Appendix A. The presence of a sine is signalled by a second reflection coefficient larger than 0.95. The sine detector pushes a one or a zero onto a 15-bit word. If 14 or more bits are set, a flag (the 16th bit) is set. In the Matlab code, an array is used to store the binary values. The sine detector is used in the adaptive codebook search and in the Voice Activity detector feature.

3.3 LSF Quantization

The quantization of the LP parameters is done in the line spectral frequency (LSF) domain. One set of LP parameters per frame is quantized. Before quantization, the LP parameters have an additional bandwidth expansion applied

$$c'[k] = \gamma^k c[k], \tag{12}$$

where $\gamma = 0.994$, corresponding to a one-sided bandwidth expansion of 25 Hz.

The LP coefficients are converted to LSF parameters. Let the LSF parameters be denoted by ω_i, $1 \le i \le N_p$. The LSF parameters are an ordered set of values between 0 and π. A vector of fixed average values is subtracted from the LSFs to give a set of mean-removed LSFs,

$$\tilde{\boldsymbol{\omega}} = \boldsymbol{\omega} - \bar{\boldsymbol{\omega}}. \tag{13}$$

The quantized LSFs from the previous frame are used to predict the LSFs for the current frame. The prediction error is

[1] This check ignores small negative errors, which also signal numerical problems.

$$\begin{aligned}\tilde{\boldsymbol{\omega}} &= (\boldsymbol{\omega} - \bar{\boldsymbol{\omega}}) - b(\hat{\boldsymbol{\omega}}_P - \bar{\boldsymbol{\omega}}) \\ &= \boldsymbol{\omega} - b\hat{\boldsymbol{\omega}}_P - (1-b)\bar{\boldsymbol{\omega}},\end{aligned} \tag{14}$$

where $b = 12/32$. This formulation will give a zero error when the current and the previous quantized LSFs are equal to the mean values. The prediction error vector is then quantized.

The quantizer finds the best codebook entries in the sense of a weighted squared-error. The weighting function is a diagonal matrix with entries equal to inverse of the distance of a particular LSF to its closest neighbour. The entries of the diagonal of the weighting matrix are

$$\mathbf{w} = \frac{1}{\min\left(\begin{bmatrix}\Delta\omega_1 \\ \Delta\boldsymbol{\omega}\end{bmatrix}, \begin{bmatrix}\Delta\boldsymbol{\omega} \\ \Delta\omega_{N_p-1}\end{bmatrix}\right)}, \quad \Delta\boldsymbol{\omega} = \begin{bmatrix}\omega_2 - \omega_1 \\ \omega_3 - \omega_2 \\ \vdots \\ \omega_{N_p-1} - \omega_{N_p-2}\end{bmatrix}. \tag{15}$$

The weighted error is

$$\varepsilon = (\tilde{\boldsymbol{\omega}} - \hat{\tilde{\boldsymbol{\omega}}})^T \mathbf{W} (\tilde{\boldsymbol{\omega}} - \hat{\tilde{\boldsymbol{\omega}}}). \tag{16}$$

The quantizer is a 3-split quantizer with dimensions 3-3-4. The error computation is spit by dimension with separate quantization of each subvector. The reconstructed LSF vector is given by

$$\begin{aligned}\hat{\boldsymbol{\omega}}_C &= \hat{\tilde{\boldsymbol{\omega}}} + b(\hat{\boldsymbol{\omega}}_P - \bar{\boldsymbol{\omega}}) + \bar{\boldsymbol{\omega}} \\ &= \hat{\tilde{\boldsymbol{\omega}}} + b\hat{\boldsymbol{\omega}}_P + (1-b)\bar{\boldsymbol{\omega}}.\end{aligned} \tag{17}$$

Since the subvectors are quantized independently, the final vector may have closely spaced or improperly ordered LSFs. Measures are taken to correct these cases. The minimum separation of two LSFs is to be $\Delta\omega_{\min}$. If any differences are less than this value, the offending LSFs are each moved to $\pm\Delta\omega_{\min}/2$ of their average value. This process may have to be repeated several times.

After quantization and imposing a minimum separation, the LSF values for each frame are linearly interpolated to give LSF values for each subframe,

$$\hat{\boldsymbol{\omega}}_k = \alpha_k \hat{\boldsymbol{\omega}}_C + (1-\alpha_k)\hat{\boldsymbol{\omega}}_P, \quad \alpha_k = \frac{k+1}{N_s}, \quad 0 \le k \le N_s - 1. \tag{18}$$

Difference with the Reference Code

- The conversion to LSFs is done using the Matlab routine `poly2lsf`. This difference occasionally gives rise to different quantized values.

- If the reference code fails to find all of the LSFs (due to numerical problems), it reverts to the previous set of quantized LSFs.
- The LSF values in the reference code take on values from 0 to 256, corresponding to the radian frequencies 0 to π. The values in the Matlab implementation are in radians.
- The LSF means (tabulated) have been converted to radian values.
- The LSF quantizer tables have been converted to radian values as have the parameters controlling the minimum separation of LSFs.
- The conversion of the interpolated, quantized LSF values back to LP parameters is done using the Matlab routine `lsf2poly`. This was found to give small differences with respect to the reference code. The reference code first transforms the LSFs to the $x = \cos(\omega)$ domain using linear interpolation into a table of cosine values. To compensate for the approximations in this transformation, the Matlab code does the same transformation and then brings back the values (accurately) to the LSF domain, before converting the LSF values with `lsf2poly`.

3.4 Formant Weighting Filter

As part of the process of forming a target signal, the highpass filtered speech is passed through a formant perceptual weighting filter. This a pole-zero filter with coefficients changing every subframe. The coefficients are taken from the unquantized LP parameters after bandwidth expansion. The filter is implemented using the Matlab routine `PZFilter` in order to mimic the operation of the reference code.

Let the unquantized LP parameters for a particular subframe be represented in terms of the all-pole LP synthesis filter $1/A(z)$. The formant weighting filter is

$$W_F(z) = \frac{A(\gamma_1 z)}{A(\gamma_2 z)}, \qquad \gamma_1 = 0.9, \quad \gamma_2 = 0.5. \tag{19}$$

The effect of this weighting filter is to deemphasize those regions of the spectrum in which the LP spectrum has peaks and to emphasize those regions in between peaks. The idea is that the peaks of the LP spectrum will tend to mask the noise at those frequencies, while the noise in the valleys is more audible. An example of the weighting filter response is shown in Fig. 1.

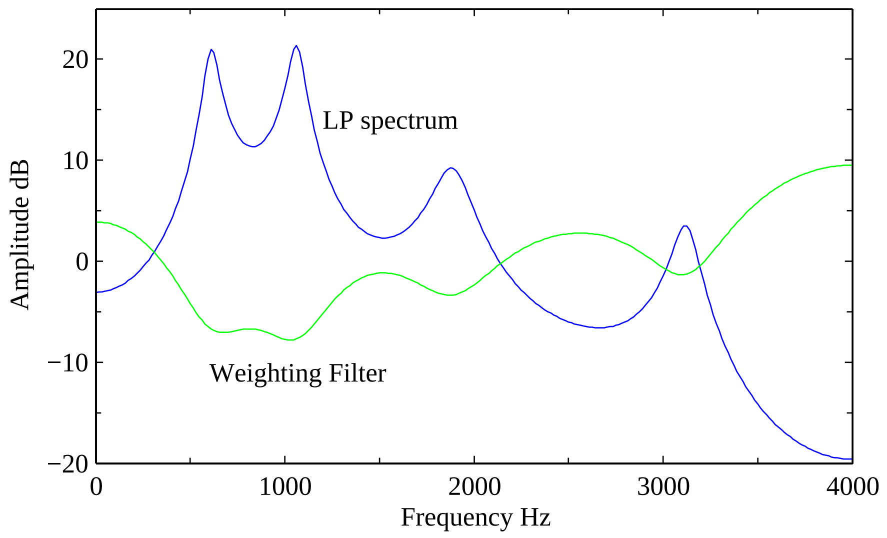

Fig. 2 Formant weighting filter response.

3.5 Open-Loop Pitch Estimation

The open loop pitch estimate finds the pitch lag and pitch gain values that minimize the mean-square estimation error. The open-loop pitch is determined from the output of the perceptually weighted filter with the original signal as input. The prediction error is

$$e[n] = x[n] - gx[n-L]. \tag{20}$$

The squared-error for a frame can be written as

$$\varepsilon_L = R[0,0] - 2gR[0,L] + g^2 R[L,L], \tag{21}$$

where the correlation terms are defined as

$$R[i,j] = \sum_{n=0}^{N-1} x[n-i]x[n-j]. \tag{22}$$

The open-loop pitch is determined for two subframes at a time. This means that the summation is over 120 samples. The optimum value of gain for a given lag is

$$g_{opt} = \frac{R[0,L]}{R[L,L]}. \tag{23}$$

With this value of gain the squared error for a frame is

$$\varepsilon_{opt} = R[0,0] - \frac{R[0,L]^2}{R[L,L]}. \tag{24}$$

The best lag value L is chosen by maximizing the reduction in error as given by the second term in the equation above,

$$L_o = \max_L \frac{R[0,L]^2}{R[L,L]}. \tag{25}$$

The denominator of the second term can be computed recursively,

$$R[l+1,l+1] = R[l,l] + x^2[-i-1] - x^2[N-1-i]. \tag{26}$$

The search is done from small lags to large lags. Only lags with positive values of $R[0,L]$ are pitch candidates. Given a current lag candidate L_o, a close-by lag giving a reduced squared error becomes the next lag candidate, i.e., the new lag is chosen if

$$\frac{R^2[0,L]}{R[L,L]} > \frac{R^2[0,L_o]}{R[L_o,L_o]}, \qquad L - L_o < dL_{\min}. \tag{27}$$

However, if the search encounters a reduced squared error that is not close by the current candidate, that new reduction in error must be substantially greater than that for the current candidate,

$$\frac{R^2[0,L]}{R[L,L]} > A\frac{R^2[0,L_o]}{R[L_o,L_o]}, \quad L - L_o \geq dL_{\min}, \tag{28}$$

where A is 4/3. This additional check is done to try to avoid choosing pitch multiples.

3.6 Harmonic Noise Weighting

Another component of the overall perceptual filtering to form the target signal is a harmonic noise-weighting (HNW) filter. This is a single tap FIR filter of the form

$$y[n] = x[n] - gx[n-L]. \tag{29}$$

This is a gain reduced gain version of a pitch predictor.

The lag L is chosen by searching around the open-loop pitch value. The HNW is found for every subframe even though the open-loop pitch values are for two subframes at a time. The choice of lag is governed by the same equations as for the open-loop pitch search, but a separate set of lags and coefficients is determined for each subframe. This means that the summation for the determining the correlation values is over 60 samples.

As a final check, the HNW filter is only used if the prediction gain is sufficiently high. The prediction gain is the ratio of the input energy to the output energy,

$$\begin{aligned} P_G &= \frac{R[0,0]}{\varepsilon_{opt}} \\ &= \frac{1}{1-\dfrac{R^2[0,L]}{R[0,0]R[L,L]}}. \end{aligned} \tag{30}$$

The prediction gain should be larger than a given minimum value,

$$\begin{aligned} P_G &> P_{G\min} \\ R^2[0,L] &> \frac{P_{G\min}-1}{P_{G\min}} R[0,0]R[L,L]. \end{aligned} \tag{31}$$

The gain is set to zero if the gain value is negative or the prediction gain condition ($P_{G\min} = 1.6$) is not satisfied. In addition, the gain is limited to at most one. The HNW filter is formed by reducing the gain so found by a multiplicative factor (10/32). This gives a filter that deemphasizes the pitch harmonics and emphasizes the spectrum between pitch harmonics.

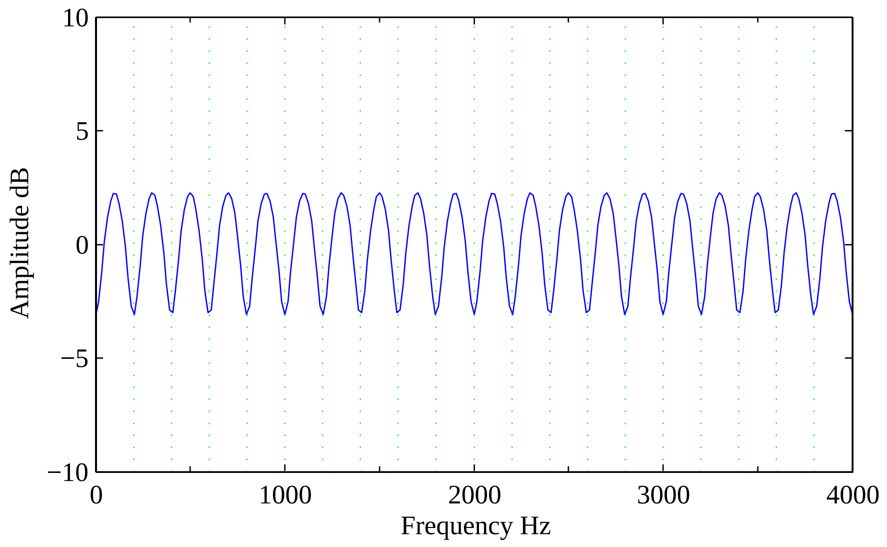

Fig. 3 Harmonic noise weighting filter response ($g = 0.2$, $L = 40$ (200 Hz))

4 Analysis-by-Synthesis Target Signal

The concept in analysis-by-synthesis (AbS) is to generate outputs corresponding to different choices of excitation signal parameters. Conceptually, a candidate excitation signal is passed through the LP synthesis filter and compared to the input speech. The combination of parameters that create the best reconstructed speech is chosen. A perceptually motivated weighting filter is used to weight the error between the input signal and the reconstructed signal. The weighting filter is the formant-weighting filter in cascade with the harmonic noise-weighting filter.

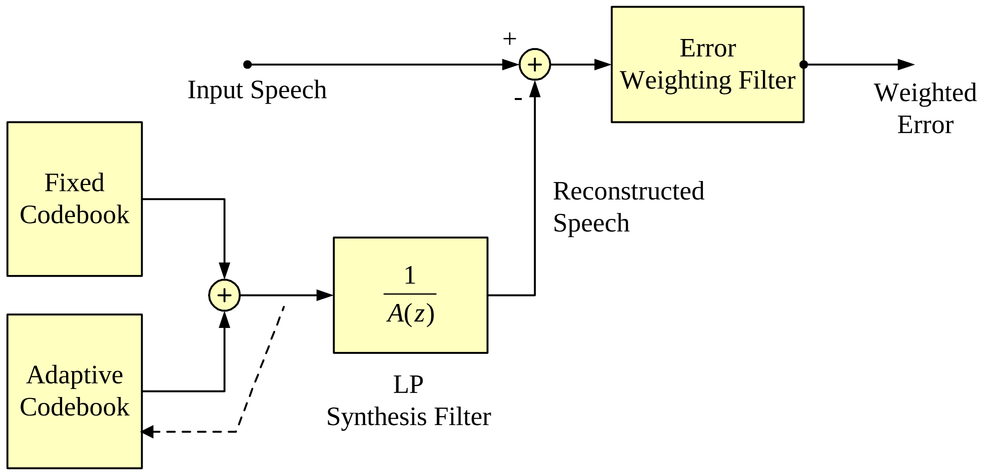

Fig. 4 Analysis-by-Synthesis CELP coding.

With some manipulation of the block diagram, the weighting filter can be moved into the input signal path and into the excitation signal path as shown in Fig. 5. This rearrangement has computational advantages since the input signal only has to be filtered once for each subframe. The input signal filtered by the weighting filter is termed the target signal.

The excitation signal has to be filtered many times in the AbS procedure. The output of the excitation branch is the sum of two parts, a zero-state response and a zero-input response. The zero-state response is the same for all candidate excitation signals for a particular subframe. As such, it can be calculated once. For convenience, this zero-input response can be subtracted from the target signal. The zero-state output of the excitation branch is then compared with the modified target signal. Furthermore, the composite filter in the excitation signal path can be represented by the overall impulse response of a weighted synthesis filter.

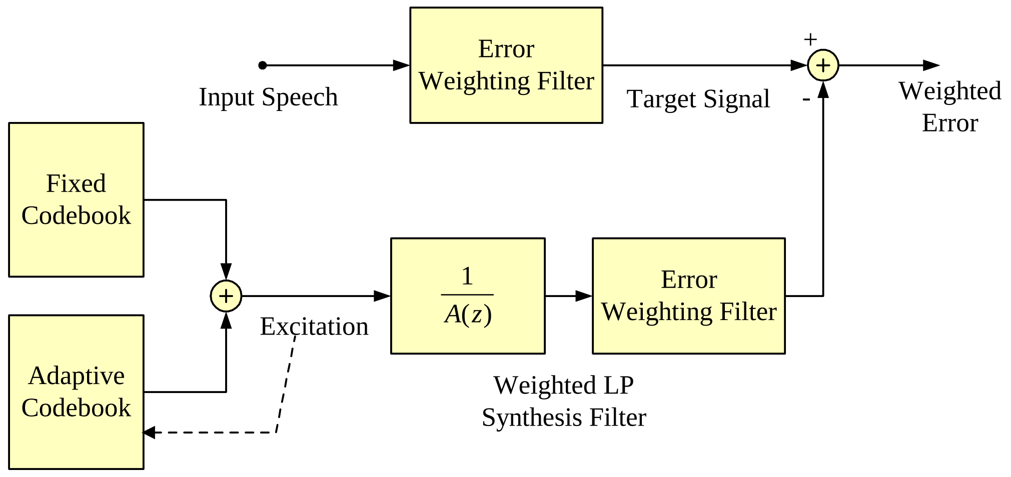

Fig. 5 Restructured Analysis-by-Synthesis CELP Coding

The excitation consists of two components. An adaptive codebook contribution that is in essence is a segment of the past excitation. This supplies the pitch-like components by placing repetitions of past pitch pulses into the correct position in the excitation. The second excitation component is the fixed codebook contribution. This supplies missing details in the excitation. The search procedure for the best excitation is done sequentially. First an adaptive codebook contribution (pitch contribution) is determined assuming the fixed codebook contribution is zero. Then the fixed codebook contribution is added.

The contribution to the reconstructed signal is determined by passing the excitation through a weighted synthesis filter. This has an all-pole synthesis filter (as will be used in the decoder) based on the quantized LP parameters, a formant weighting filter based on the unquantized LP parameters, and a harmonic noise-weighting filter. Let the weighted synthesis filter have impulse response $h_w[n]$. This is a causal infinite response, but we will only need the first N values of the response, where N is the subframe length.

For convenience, a custom routine `WSyn` is used to implement the weighted synthesis filter (it uses `PZFilter` internally). It has as input the filter state and the input signal, and has as output the output signal and the updated state. The routine `WSyn` is used in three ways:

- Before processing a subframe, `WSyn` is used to calculate the impulse response (zero-state, a unit impulse sequence as input; impulse response as output, discard the updated state).
- Before processing a subframe, `WSyn` is used to calculate the zero-input response to be subtracted from the target signal (previous state, an all-zero sequence as input; zero-state response as output, discard the updated state).

- After generating the excitation for a subframe, `WSyn` is used to update the filter state (previous state, excitation as input; discard the filter output, save the updated filter state).

5 Subframe Level Processing

5.1 Adaptive Codebook

The adaptive codebook (ACB) supplies the pitch contribution to the excitation signal. The pitch filter is a multi-tap IIR filter of the form

$$e_p[n] = \sum_{k=K_L}^{K_U} b_k \tilde{e}[n-k-L], \tag{32}$$

where $\tilde{e}[n]$ is a pitch repeated version of the past excitation,

$$\tilde{e}[n] = \begin{cases} e[n], & n < 0, \\ e[\operatorname{mod}(n,L)-L], & n \geq 0. \end{cases} \tag{33}$$

The past excitation contains both the ACB and fixed codebook contributions. The ACB generates only the pitch-like contribution to the current excitation. The pitch repetition is necessary for short pitch lags since the full excitation for the current subframe ($n \geq 0$) has not been generated yet. With the large subframe size used in G.723.1 (60 samples), this repetition is called into play quite often. The pitch filter uses 5 taps, with the reference tap being in the middle ($K_L = -2$ and $K_U = 2$). However, contrary to one's expectations, the tabulated vectors of ACB coefficients do not always have the largest coefficients near the middle of the filter.

The ACB coefficients are taken from one of two codebooks. The first has 85 entries; the second has 170 entries. The first codebook is used for short pitch lags, while the second is used for larger pitch lags. When the first codebook is used, the bit saved in using the shorter codebook is reserved for use by the multipulse coding procedure. It is to be noted that the switch of codebooks depends on the lag chosen in the even-numbered subframes (0 and 2). Thus the codebook used for subframes 0 and 1 depends on the lag chosen for subframe 0 and the codebook used for subframes 1 and 2 depends on the lag chosen for subframe 1.

The adaptive codebook has two modes. In the even-numbered subframes (0 and 2), the lag is sent as an absolute value. The search for lags in the even-numbered subframes is done in a limited range (-1 to +1) around the open-loop lag determined earlier. This is done to reduce computations. In the odd-numbered subframes (1 and 3), the lag is coded relative to the previous subframe. The lag offset is coded with 2 bits, allowing the lag for odd-numbered subframes to have lags offset from –1 to +2 relative to the lag of the previous subframe.

In vector-matrix notation, the pitch contribution to the excitation is

$$\mathbf{e}_p = \tilde{\mathbf{E}}_L \mathbf{b}, \tag{34}$$

where $\mathbf{e}_p$ is an $N \times 1$ vector of pitch contributions, $\tilde{\mathbf{E}}_L$ is an $N \times N_b$ matrix of repeated excitation signals, and $\mathbf{b}$ is an $N_b \times 1$ vector of pitch coefficients,

$$\tilde{\mathbf{E}}_L = \begin{bmatrix} \tilde{e}[-L-K_L] & \cdots & \tilde{e}[-L-K_U] \\ \vdots & \ddots & \vdots \\ \tilde{e}[N-1-L-K_L] & \cdots & \tilde{e}[N-1-L-K_U] \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} b_{K_L} \\ \vdots \\ b_{K_U} \end{bmatrix}. \tag{35}$$

The contribution to the reconstructed signal is obtained by passing $\mathbf{e}_p$ through the weighted synthesis filter. One has to be careful here: we are interested in the zero-state response, so the past excitation is implicitly zero. Filtering $\mathbf{e}_p$, we get

$$\mathbf{s}_p = \mathbf{S}_L \mathbf{b}, \tag{36}$$

where $\mathbf{S}_L$ is an $N \times N_b$ matrix formed by convolving the columns of $\tilde{\mathbf{E}}_L$ with the convolution matrix containing the impulse response coefficients of the filter,

$$\mathbf{S}_L = \begin{bmatrix} h_o & 0 & \cdots & 0 \\ h_1 & h_o & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ h_{N-1} & h_{N-2} & \cdots & h_o \end{bmatrix} \tilde{\mathbf{E}}_L. \tag{37}$$

In the Matlab code, this operation is carried out column-by-column using the Matlab filtering routine.

Recursive Computation

In the reference code, a recursive approach is used to reduce computations. Since the columns of $\tilde{\mathbf{E}}_L$ are shifted versions of each other, a column $\tilde{\mathbf{e}}_k$ (columns numbered from K_L to K_U) can be expressed as

$$\tilde{\mathbf{e}}_k = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \tilde{\mathbf{e}}_{k-1} + \begin{bmatrix} \tilde{e}[-L-k] \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{38}$$

The filtered version of this column can be expressed as

$$\begin{aligned}\mathbf{s}_k &= \mathbf{H}\tilde{\mathbf{e}}_k \\ &= \mathbf{H}\mathbf{P}\tilde{\mathbf{e}}_{k-1} + \mathbf{H}\begin{bmatrix}\tilde{e}[-L-k] \\ 0 \\ 0 \\ \vdots \\ 0\end{bmatrix},\end{aligned} \tag{39}$$

where $\mathbf{P}$ is the shift matrix and $\mathbf{H}$ is the impulse response matrix. The shift matrix when used as a premultiplier, shifts the elements of a column down; when used as a postmultiplier, it shifts rows to the left. Since $\mathbf{H}$ is Toeplitz, these operations are the same and $\mathbf{HP} = \mathbf{PH}$. Then recursive update to $\mathbf{s}_k$ is

$$\mathbf{s}_k = \mathbf{P}\mathbf{s}_{k-1} + \tilde{e}[-L-k]\begin{bmatrix}h_0 \\ h_1 \\ \vdots \\ h_{N-1}\end{bmatrix}. \tag{40}$$

Optimal Pitch Contribution

Give a target vector $\mathbf{t}$, the squared error is

$$\begin{aligned}\varepsilon &= (\mathbf{t}-\mathbf{s}_p)^T(\mathbf{t}-\mathbf{s}_p) \\ &= \mathbf{t}^T\mathbf{t} - 2\mathbf{t}^T\mathbf{S}_L\mathbf{b} + \mathbf{b}^T\mathbf{S}_L^T\mathbf{S}_L\mathbf{b} \\ &= \mathbf{t}^T\mathbf{t} - 2\mathbf{R}_{Lts}\mathbf{b} + \mathbf{b}^T\mathbf{R}_{Lss}\mathbf{b}.\end{aligned} \tag{41}$$

The cross-correlation matrix $\mathbf{R}_{Lts}$ is $1 \times N_b$; the correlation matrix $\mathbf{R}_{Lss}$ is $N_b \times N_b$. The search for the best ACB is over lags and over coefficient vectors. The lag search range is limited as described earlier. For a given lag, the search for the best coefficient vectors amounts to maximizing the sum of the last two terms in the previous expression,

$$\mathbf{b}_{opt} = \max_{\mathbf{b}}(2\mathbf{R}_{Lts}\mathbf{b} - \mathbf{b}^T\mathbf{R}_{Lss}\mathbf{b}). \tag{42}$$

This is the formulation used in the Matlab routine.

The figure below shows an example of the action of the adaptive codebook. In this case, the two almost equal coefficients in the coefficient vector serve to interpolate between integer lag values.

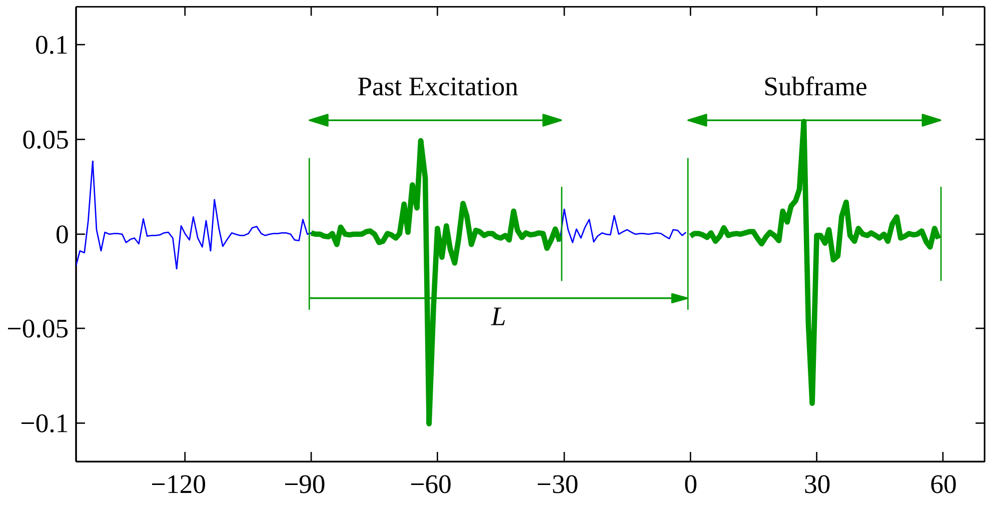

Fig. 6 Adaptive codebook contribution, $L = 90$, $\mathbf{b} = [0.06 \quad -0.17 \quad 0.63 \quad 0.59 \quad -0.17]$.

Streamlined Comparison

In the reference code, a streamlined version of this computation is used. The quadratic form $\mathbf{b}^T \mathbf{R}_{Lss} \mathbf{b}$ is symmetric and so nearly half of the computations can be avoided,

$$\begin{aligned} \mathbf{b}^T \mathbf{R}_{Lss} \mathbf{b} &= \sum_{k=K_L}^{K_U} \sum_{m=K_L}^{K_U} b_k b_m R_{Lss}[k,m] \\ &= \sum_{k=K_L}^{K_U} b_k^2 R_{Lss}[k,k] + 2 \sum_{k=K_L}^{K_U} \sum_{m=K_L}^{k-1} b_k b_m R_{Lss}[k,m]. \end{aligned} \tag{43}$$

The product terms $b_k b_m$ can be pre-computed.

The final step in the streamlining is to make two vectors of the components in the computation; one with correlation terms and one with coefficient terms. The correlation terms are the N_b cross-correlations $\mathbf{R}_{Lts}$, the N_b diagonal terms $R_{Lss}[k,k]$, and the $N_b(N_b-1)/2$ off-diagonal terms of $R_{Lss}[k,m]$. The coefficient terms are the N_b coefficient $2\mathbf{b}$, the N_b diagonal terms $-b_k^2$, and the $N_b(N_b-1)/2$ off-diagonal terms $-2b_k b_m$. The test for the best vector of coefficients is then the dot product of the vector of correlation terms and the pre-computed vector of coefficient terms.

Target Signal Update

After determining the ACB contribution, this contribution is subtracted from the target signal to give a modified target signal that will be the target for the multipulse search,

$$\mathbf{t}' = \mathbf{t} - \mathbf{H}\tilde{\mathbf{E}}_L \mathbf{b}_{opt} . \tag{44}$$

Pitch Taming Procedure

The adaptive codebook search uses some “taming” procedures that in some still obscure ways are meant to help reduce distortion in the case of frame loss. The adaptive codebook gain values are ordered from small to large. The range of ACB gain indices to be searched is limited by the “taming” procedure to include only the small gain values. The taming procedure uses the sine detector described earlier. Since the taming has no effect during normal speech, the efficacy and correctness of this process is still in question.

5.2 Multipulse Coding

The multipulse excitation uses 6 pulses per subframe in subframes 0 and 2 and 5 pulses per subframe in subframes 1 and 3. All the pulses for a subframe must be placed on one of two grids: even-numbered positions or odd-numbered positions. One bit is used to specify which of the grids should be used. There are then 30 pulse positions in which to place either 6 or 5 pulses. The pulses for a subframe all have the same amplitude (one of 24 quantized values), but the signs are specified separately with 6 or 5 bits per subframe.

Pitch Repetition

For short pitch lags, the multipulse excitation is subject to a pitch repetition. A single bit is used to determine whether to use this pitch repetition or not. The option to use pitch repetition is available for subframes 0 and 1 whenever the pitch lag for subframe 0 is less than 58. The option to use pitch repetition is available for subframes 2 and 3 whenever the pitch lag for subframe 2 is less than 58. The bit to signal whether to use pitch repetition is available for short pitch lags because for these short pitch lags, the ACB gain table is smaller by a factor of 2 compared to longer pitch lags.

The multipulse repetition can be achieved in one of two ways: by repeating the pulses or by repeating the impulse response. During the multipulse search, the second method is used. During speech synthesis, the first method is used. The pitch repeated impulse response is given by

$$\tilde{h}[n] = \sum_{k=0}^{K_u} h\left[n + kL\right], \qquad 0 \le n \le N-1, \tag{45}$$

where the upper limit is $K_u = \lfloor (N-1)/L \rfloor$.

Pulse Positions and Amplitudes

The search for the pulse positions and amplitudes is done in nested loops. The outermost loop tests whether to use pitch repetition or not. The next loop is over the two possible grids. The next loop is a search over pulse amplitudes. The innermost loop, generates finds the pulse locations sequentially.

The formalism for choosing the next best pulse position can be formulated as follows. Let the target vector be $t[n]$ (after being compensated for by the adaptive codebook contribution) and the impulse response of the weighted synthesis filter be $h[n]$ (actual impulse response or the pitch repeated impulse response). If we place a pulse of amplitude g_m in position m, the error is

$$E[m] = E_t - 2g_m R_{th}[m] + g_m^2 R_{hh}[0,0], \tag{46}$$

where

$$\begin{aligned} R_{th}[m] &= \sum_{n=0}^{N-1} t[n]h[n-m] \\ &= \sum_{n=m}^{N-1} t[n]h[n-m], \end{aligned} \tag{47}$$

and

$$\begin{aligned} R_{th}[m] &= \sum_{n=0}^{N-1} h[n]h[n-m] \\ &= \sum_{n=m}^{N-1} h[n]h[n-m]. \end{aligned} \tag{48}$$

The optimum value of gain can be found as

$$g_{opt} = \frac{R_{th}[m]}{R_{hh}[0,0]}. \tag{49}$$

We will choose g_m from a fixed set of quantized amplitudes, but allowing g_m to take on either sign. To reduce complexity, we use a quantized estimate of the gain and search over quantized gain amplitudes nearby the estimated gain. If the gain which minimizes Eq. (46) is denotes as g_{opt}, the error in using another value of gain can be expressed as

$$E[m] = E_{\min}[m] + (g_m - g_{opt})^2 R_{hh}[0]. \tag{50}$$

The quantized gain that minimizes the mean-square error is that value which is closest to g_{opt}.

Estimating the Pulse Amplitude

The pulse position that gives the biggest reduction in squared error is found as

$$\begin{aligned} m_{opt} &= \max_m (2 g_{opt} R_{th}[m] - g_{opt}^2 R_{hh}[0,0]) \\ &= \max_m \left(\frac{R_{th}^2[m]}{R_{hh}[0,0]} \right) \\ &= \max_m \left(\left| R_{rh}[m] \right| \right). \end{aligned} \tag{51}$$

Once the best position is found, g_{opt} for that pulse is given by Eq. (49). The quantized value of gain $\hat{g}_m$ nearest g_{opt} is found. If the table of quantized amplitudes has elements $A_g[i]$, the index of the quantized amplitude is found as

$$\begin{aligned} i_g &= \min_i \left(\left| \, | g_{opt} | - A_g(i) \right| \right) \\ &= \min_i \left(\left| A_g(i) R_{hh}[0,0] - \left| R_{th}[m] \right| \right| \right). \end{aligned} \tag{52}$$

The search for the gain that is used for all of the pulses is limited to quantized gain values near $A_g[i]$ (relative indices -2 to +1).

Now we can loop over the reduced set of quantized gain amplitudes. The same amplitude will be used for all pulses in a subframe.

Pulse Positions

Given the trial quantized gain, the error for a trial position is (from Eq. (46)),

$$E[m] = E_t \mp 2 A_g(i) R_{th}[m] + A_g^2(i) R_{hh}[0,0], \tag{53}$$

where the upper sign is used if the pulse is positive and the lower sign is used if the pulse is negative. The position that gives the lowest squared error is

$$M = \max_m \left(\left|R_{th}[m]\right|\right). \tag{54}$$

The sign of the pulse is determined by the sign of $R_{th}[M]$,

$$g_M = \text{sign}(R_{th}[M])A_g(i). \tag{55}$$

Once a pulse position and pulse gain has been found, the effect of that pulse can be subtracted from the target signal,

$$\begin{aligned} t'[n] &= t[n] - g_M \delta[n-M] * h[n] \\ &= t[n] - g_M h[n-M]. \end{aligned} \tag{56}$$

Using this expression, the cross-correlation can be updated,

$$\begin{aligned} R_{t'h}[n] &= \sum_{n=m}^{N-1} t'[n]h[n-m] \\ &= R_{th}[n] - g_M \sum_{n=m}^{N-1} h[n-m]h[n-M] \\ &= R_{th}[n] - g_M R_{hh}[|n-M|]. \end{aligned} \tag{57}$$

With the updated cross-correlation, the next pulse can be placed. As each pulse is placed, the position it occupies is marked as occupied to prevent a subsequent pulse being placed in the same position.

Coding the Pulse Positions

The pulse positions are coded using the combinatorial coding procedure described in Appendix B. The procedure generates a code for each subframe. The codes take on $\binom{30}{6} = 593775$ values for even subframes and $\binom{30}{6} = 142506$ values for odd subframes. The minimum number of bits required to code the pulse positions is 73 bits. This is accomplished by splitting the coding as follows.

- Express the code for a subframe as $C_i = p_i M_i + q_i$, where $p_i = \lfloor C_i / M_i \rfloor$ and $q_i = \text{mod}(C_i, M_i)$. The modulus value is 2^{16} for even subframes and 2^{14} for odd subframes.
- The q_i values are coded with 16 or 14 bits.

- The excess values are coded as $C_x = p_3 + 9p_2 + 90p_1 + 810p_0$. This value can be represented with 13 bits.

Differences from the Reference Code

The multipulse coding differs in small details from the reference code. As an example, if several pulse positions are equally good, the reference code uses the last of the equally good positions. The Matlab implementation uses a `max` operation that returns the first of the equally good positions. Reversing the order of the grid positions during the search has reduced the differences with respect to the reference code.

The combinatorial coding procedure used in the Matlab code differs from that in the reference code. In the terms of the analysis of Appendix B, the reference code generates a code that marks the positions of non-pulses. The Matlab code uses the more efficient process of coding positions of the pulses and then “reverses” the code to the give exactly the same result as the reference code.

6 Decoder

The decoder is shown in Fig. 7. The decoder does no searching, so is computationally much less onerous than the coder. The steps involved in generating the reconstructed speech are as follows.

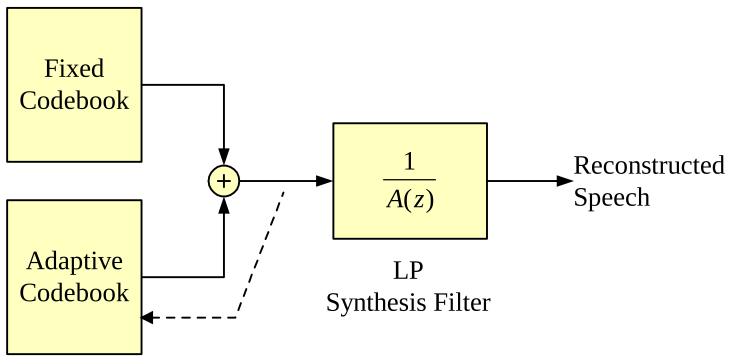

Fig. 7 Decoder

- Generate the quantized LP parameters for each subframe.
  - Decode the quantized LSF values.
  - Correct for improperly ordered or closely spaced LSF values.
  - Interpolate the quantized LSF values from the quantized LSF values from the last frame to create a vector of quantized LSF values for each subframe.
  - Convert the LSF values to the corresponding LP coefficient values.
- Decode ACB and multipulse parameters.
- Generate the excitation for each subframe.
  - Generate the ACB contribution to the excitation.
  - Generate the multipulse contribution to the excitation.
  - Filter the total excitation with the LP synthesis filter.

Notes

The combined excitation signal is saved for use in future subframes. The excitation signal that is saved is clipped. The clipping level is just below full scale (1 for the Matlab version and 32768 for the reference code). The output signal, however, is created from the unclipped excitation.

References

1. ITU-T Recommendation G.723.1, *Dual Rate Speech Coder for Multimedia Communications Transmitting at 5.3 and 6.3 kbit/s*, March 1996.
2. P. Kabal, “Ill-Conditioning and Bandwidth Expansion in Linear Prediction of Speech”, *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing* (Hong Kong), pp. I-824–I-827, April 2003.
   See also P. Kabal, “Ill-Conditioning and Bandwidth Expansion in Linear Prediction of Speech”, Technical Report, Electrical & Computer Engineering, McGill University, Feb. 2003: http://WWW.TSP.ECE.McGill.CA/Documents.
3. J. P. M. Schalkwijk, “An Algorithm for Source Coding”, *IEEE Trans. Inform. Theory*, vol. 18, pp. 395–399, May 1972.

Appendix A Sine Detector

Consider samples from a discrete-time sine wave. This signal has a spectrum with two poles on the unit circle,

$$S(z) = \frac{1}{1 - 2\cos\omega_o z^{-1} + z^{-2}}. \tag{58}$$

The optimal prediction error filter will generate zeros to cancel the poles,

$$H(z) = 1 - 2\cos\omega_o z^{-1} + z^{-2}. \tag{59}$$

The corresponding predictor has coefficients $2\cos\omega_o$ and -1.

This analysis shows that a 2-tap predictor can exactly predict the next value of a sine from the previous two samples. This can be written as

$$s[n] = 2\cos\omega_o s[n-1] - s[n-2], \tag{60}$$

where $s[n] = \cos(\omega_o n + \phi)$ and can be verified using standard trigonometric identities. Note that for an optimal predictor, the last coefficient is equal to the negative of the last reflection coefficient. Thus a second reflection coefficient of 1 indicates a sine wave.

In a speech coder, the windowing of the signal will mean that the calculated correlations are not exactly that for a sine wave and the predictor coefficients are not exactly those needed to predict a sine. However, for reasonable size frames, the second reflection coefficient will be near unity and hence this value can be used to detect the presence of a sine.

Appendix B Combinatorial Coding

The problem under consideration is how to code the pulse positions. Combinatorial coding assigns a code for each combination of pulse positions. The procedure for doing this coding has been reinvented many times, possibly described first in the engineering literature in [3].

K pulses are placed in N possible positions. Let the pulse positions be ordered and let the positions be numbered from 1 to N. The minimum number of bits for coding the positions of randomly placed pulses can be determined from a combinatorial argument. The total number of different combinations of K pulses in N positions is $\binom{N}{K}$. If each possibility is equally likely, the coding requires at least

$$N_{\min} = \log_2 \binom{N}{K} \tag{61}$$

bits. This is the lower bound on the number of bits required to code unconstrained random pulse positions. Direct coding of the pulse positions is much more expensive, requiring $K \log_2 N$ bits. For G.723.1, the number of positions is 30 and the number of pulses is either 6 (even subframes) or 5 (odd subframes). For the even subframes, $N_{\min}$ is 19.2 bits, while direct coding would require 29.4 bits.

The pulse positions for a block can be coded by directly tabulating the $\binom{N}{K}$ possible combinations of pulse positions. Consider a binary N-vector of weight K. Ones mark the positions of the pulses, while zeros mark the empty positions. Index the vectors from 0 to $\binom{N}{K} - 1$, with the vectors arranged in lexicographic order. For this purpose, number the data positions from $N-1$ for the most significant position to 0 for the least significant position.

The following algorithm can be used to determine the index for a given data vector. The vector is searched from the most significant (lexicographic) position to the least significant position, n running from $N-1$ to 0. Whenever a one is encountered in position n, the index is increased by $\binom{n-1}{m}$ where m is the number of ones yet to be found. The resulting value is the index to the lexicographic ordering. The complementary problem of determining a data vector containing pulse positions can be solved by comparing the index value with the same combinatorial values to determine if a one or zero should be placed in a given position.

The pulse coding problem may be viewed as finding a path through the trellis shown in Fig. 8. Moving diagonally downward in the trellis decreases the number of pulses remaining. Moving across in the trellis decreases the number of positions remaining. Each node in the trellis is labelled with a combinatorial term. If a pulse (a one) is encountered, the value at the node is added to the index and the downward diagonal path is taken. Now solve the sub-problem with one fewer position and one fewer pulse. If no pulse (a zero) is encountered, the index remains unchanged and a horizontal path is taken. Now solve the sub-problem with one fewer position.

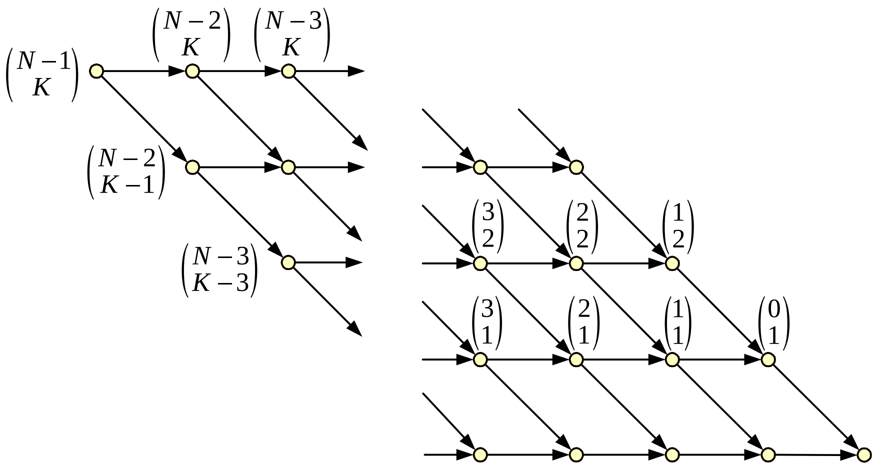

Fig. 8 Combinatorial coding trellis.

At a given horizontal level in the trellis, the values on the nodes decrease in moving to the right,

$$\binom{n}{m} > \binom{n-k}{m} \qquad \text{for } k \geq 1. \tag{62}$$

The index determined as described will reflect a strict lexicographic ordering—moving a pulse from a more significant position to a less significant position can only decrease the index value. Furthermore, the index value corresponding to a pulse configuration is unique.

The largest index value results when the leftmost diagonals in the trellis are traversed,

$$\begin{aligned} i_{\max} &= \sum_{k=1}^{K} \binom{N-k}{K-k+1} \\ &= \binom{N}{K} - 1. \end{aligned} \tag{63}$$

The smallest index value results when the rightmost diagonals are traversed,

$$i_{\min} = 0. \tag{64}$$

Therefore, the index determined by the algorithm takes on all $\binom{N}{K}$ integer values.

The decoding problem can also be viewed as traversing the trellis. This time the index value is compared to the value at the node. If it is as large, a pulse is placed in that position. The justification for this comes from Eq. (63)

The contribution to the index value of all subsequent pulses is strictly less than that for a single pulse at the position under consideration. If a pulse is placed at that position, the index value is then decreased by the value at that node and the algorithm can be repeated on a subproblem with one less position and one less pulse. The total number of comparisons to be made is $n-1$.

The combinatorial terms on the nodes of the trellis can be stored as a table of values. If the table size is considered excessive, these terms can be computed recursively using a single multiply and divide. This is possible since $\binom{n}{m}$, is followed by either $\binom{n}{m-1}$ or $\binom{n-1}{m-1}$ on a path through the trellis. A strategy that falls between a full table and full computation stores the combinatorial terms for the top row of nodes. When the path traversed descends from the top row, the values for the next row of nodes can be determined from $\binom{n-1}{m-1} = \binom{n}{m} - \binom{n-1}{m}$, avoiding multiplications and divisions.

This coding procedure requires K additions and K references to combinatorial terms. The decoding requires N references to the combinatorial terms, but still only K additions. The combinatorial terms can be stored in a table of less than KN values. Alternately several algorithms are available to generate the required terms recursively.