ITU-T G.723.1 Speech Coder:

A Matlab Implementation

P. Kabal

Department of Electrical & Computer Engineering

McGill University

Version 2a: 2009-10-29

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 implementation of that coder/decoder 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 [2]. G.723.1 is a Code-Excited Linear Prediction Coder. The Matlab version is a partial implementation of the coder and supports only the multipulse coding mode (operating at 6.3 kb/s). The Matlab implementation of the decoder supports all modes (multipulse decoding, ACELP decoding, comfort noise generation (silence compression mode [3]), and packet loss concealment). This document gives details of the processing steps in the coder and decoder.

The implementation of the *coder* 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 in the generated quantized values occur in isolated frames, but differences in one frame can propagate for several frames. In all cases, the output coded speech is indistinguishable in quality from that produced by the reference code.

The *decoder* implemented in Matlab generates exactly the same sample values as the reference code for all of the modes using all of the test files which accompany the reference code.

2 General Comments on the Matlab Implementation

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 or a data file (from the Matlab implementation of the coder). Its output is a WAVE file. The decoder also accepts as input an auxiliary frame error file (of the same format as used by the reference code) which specifies which frames are lost.

2.2 Filters

Some 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 Indexing

In this document, we use zero-based indexing as in the C-language reference code, even though Matlab itself uses 1-based indexing of arrays. For instance, the first subframe within a frame is referred to as subframe 0.

The Matlab code has been written to have clear interfaces between functions (no globals). There is also a clean separation between system parameters (fixed values that define the behaviour of the coder or decoder) and system memory (values that keep the state of the coder or decoder).

It is hoped that the Matlab code for the G.723.1 standard will be a useful vehicle for research, making it much easier to test out changes to the coding strategy.

Differences: Matlab / Reference Code

- 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 programs.
- There are a number of tables in the reference code. The Matlab code can read these tables, but can also generate the data for some of the tables on the fly. For instance, the 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. In order to exactly match the results of the reference code, the reduced precision values in the tables taken from the reference code must be used.

3 G.723.1 Coder

3.1 Frame Level Processing

The coder operates on two time scales: a frame of 240 samples that is divided into subframes of length 60. The frame level operations in the coder consist of 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 are 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 allowing the search procedure to take into account psychoacoustic properties.
- The input signal (after highpass filtering) is processed with the formant perceptual weighting filter.
- The output of formant weighting 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 tracks 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.

The block diagram of these processing steps is shown in Fig. 1. This figure shows the arrangement to generate the target signal.

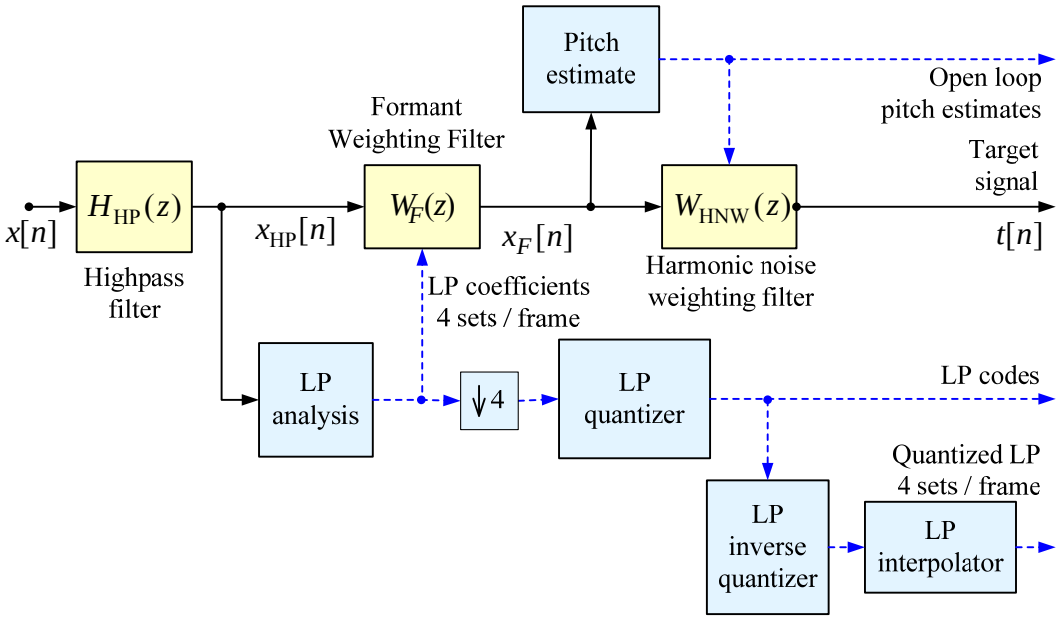

Fig. 1 Target signal generation.

3.1.1 Highpass Filter

The highpass filter used to remove the DC components is of the form

$$H_{HP}(z) = \frac{1 - z^{-1}}{1 - az^{-1}}, \tag{1}$$

where $a = 127/128$. The frequency response of this filter is plotted in Fig. 2. The input to this filter is $x[n]$ and the output of the filter is $x_{HP}[n]$.

3.1.2 LP Analysis

The linear prediction 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. 3. The figure shows that the processing requires 120 samples of past signal. The new 240 samples for a

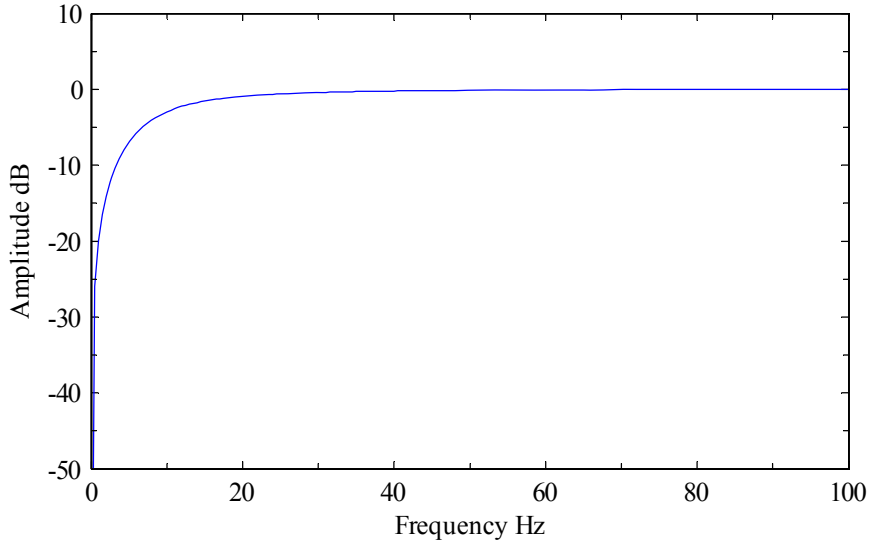

Fig. 2 Highpass filter frequency response.

frame are appended to give the full 360 samples needed. After LP analysis, the top 120 samples become the memory for the next frame.

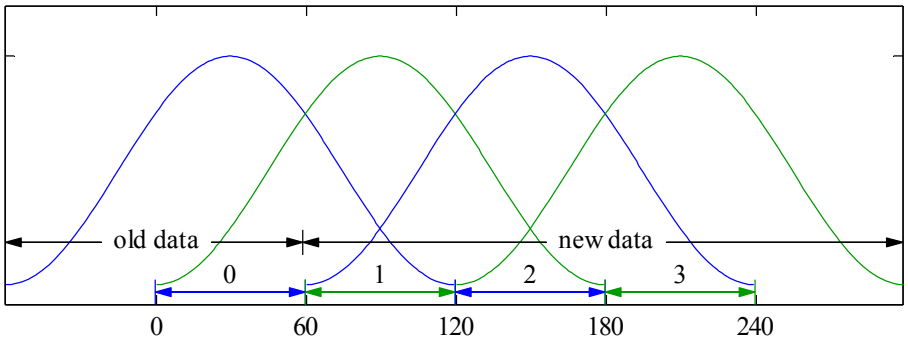

Fig. 3 LP windows.

Let the highpass filtered data for a subframe be $x_{\mathrm{HP}}[n]$. This is the data lying under the analysis window for a subframe. The windowed data is

$$x_w[n] = x_{\mathrm{HP}}[n]w[n] \qquad 0 \le n \le N_w - 1. \tag{2}$$

The prediction error is

$$e[n] = x_w[n] - \sum_{k=1}^{N_p} c_k x_w[n-k]. \tag{3}$$

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,

$$\begin{aligned}\mathbf{e} &= \mathbf{x}_w^{(0)} - \left[\mathbf{x}_w^{(1)} \cdots \mathbf{x}_w^{(N_p-1)}\right]\mathbf{c} \\ &= \mathbf{x}_w^{(0)} - \mathbf{X}_w\mathbf{c}.\end{aligned} \tag{4}$$

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

$$\mathbf{x}_w^{(0)} = \begin{bmatrix} \vdots \\ 0 \\ x_w[0] \\ x_w[1] \\ \vdots \\ x_w[N_w-2] \\ x_w[N_w-1] \\ 0 \\ \vdots \end{bmatrix}, \quad \mathbf{x}_w^{(1)} = \begin{bmatrix} \vdots \\ 0 \\ 0 \\ x_w[0] \\ \vdots \\ x_w[N_w-3] \\ x_w[N_w-2] \\ x_w[N_w-1] \\ \vdots \end{bmatrix}, \ldots \tag{5}$$

The vectors 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,

$$\begin{aligned}\varepsilon &= \mathbf{e}^T\mathbf{e} \\ &= \mathbf{x}_w^{(0)T}\mathbf{x}_w^{(0)} - 2\mathbf{x}_w^{(0)}\mathbf{X}_w\mathbf{c} + \mathbf{c}^T\mathbf{X}_w^T\mathbf{X}_w\mathbf{c} \\ &= r[0] - 2\mathbf{c}^T\mathbf{r} + \mathbf{c}^T\mathbf{R}\mathbf{c},\end{aligned} \tag{6}$$

where the correlation values are functions of time differences,

$$r[k] = \mathbf{x}_w^{(n)T}\mathbf{x}_w^{(n-k)}, \quad \mathbf{r} = \begin{bmatrix} r[1] \\ r[2] \\ \vdots \\ r[N_p] \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} r[0] & r[1] & \cdots & r[N_p-1] \\ r[1] & r[0] & \cdots & r[N_p-2] \\ \vdots & \vdots & \ddots & \vdots \\ r[N_p-1] & r[N_p-2] & \cdots & r[0] \end{bmatrix}. \tag{7}$$

In the LP analysis for G.723.1, the correlation values are lag windowed (see [4]),

$$r'[k] = r[k]w_l[k]. \tag{8}$$

The lag window has 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 f_{BW} equal to 42.5 Hz,

$$w_l[k] = \exp\left(-\frac{1}{2}(2\pi f_{\mathrm{BW}}k)^2\right). \tag{9}$$

The LP analysis also uses white noise compensation,

$$r''[k] = r'[k](1 + \mu\delta[k]), \tag{10}$$

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

$$\varepsilon' = r[0] - 2\mathbf{c}^T\mathbf{r} + \mathbf{c}^T\mathbf{R}\mathbf{c} + \mu r[0]\mathbf{c}^T\mathbf{c}. \tag{11}$$

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,

$$\mathbf{R}\mathbf{c}_{\text{opt}} = \mathbf{r}. \tag{12}$$

Differences: Matlab / Reference Code

- The linear prediction analysis uses the built-in Matlab routine `levinson`. This causes 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.
- In the reference code, the lag window is defined in a table. The documentation in the standard indicates that the lag window is based on binomial coefficients. The reference code defines the lag window with 6 digits of precision (values near unity). The value of f_{BW} (one-sided 1-σ bandwidth) which best matches the tabulated values is 42.4869/8000 or about 42.5 Hz for an 8000 Hz sampling rate. The maximum error in window values is 5×10^{-7} . An attempt was made 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).

3.1.3 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 shifts a one or a zero into 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.

3.1.4 LSF Quantization

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

$$c'[k] = \gamma^k c[k], \tag{13}$$

where $\gamma = 0.994$, corresponding to a two-sided 3-dB bandwidth expansion of 15 Hz.

The LP coefficients are converted to LSF parameters. In the reference code, this is done by searching for roots between discrete values, and then using linear interpolation between those discrete values. The Matlab code uses the routine `poly2lsf` which generally returns more accurate values.

Differential Coding of the LSFs

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 $\bar{\boldsymbol{\omega}}$ is subtracted from the LSFs to give a set of mean-removed LSFs,

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

$$\boldsymbol{\omega}' = \boldsymbol{\omega} - \bar{\boldsymbol{\omega}}. \tag{14}$$

The quantized LSFs from the previous frame $\hat{\boldsymbol{\omega}}_P$ are used to predict the LSFs for the current frame. The prediction error on the mean-removed LSFs is

$$\tilde{\boldsymbol{\omega}} = \left(\boldsymbol{\omega} - \bar{\boldsymbol{\omega}}\right) - b\left(\hat{\boldsymbol{\omega}}_P - \bar{\boldsymbol{\omega}}\right), \tag{15}$$

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 $\tilde{\boldsymbol{\omega}}$ is then quantized.

LSF Quantizer

The quantizer finds the best codebook entries in the sense of a weighted squared-error. The weighting function is a diagonal matrix $\mathbf{W}$ 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

$$\mathbf{w} = \frac{1}{\min\left(\begin{bmatrix}\omega_1 \\ \Delta\boldsymbol{\omega}\end{bmatrix}, \begin{bmatrix}\Delta\boldsymbol{\omega} \\ \pi - \omega_{N_p-1}\end{bmatrix}\right)}, \quad \Delta\boldsymbol{\omega} = \begin{bmatrix}\omega_2 - \omega_1 \\ \omega_3 - \omega_2 \\ \vdots \\ \omega_{N_p-1} - \omega_{N_p-2}\end{bmatrix}. \tag{16}$$

For a candidate quantized vector $\hat{\tilde{\boldsymbol{\omega}}}$, the weighted error is

$$\varepsilon = (\tilde{\boldsymbol{\omega}} - \hat{\tilde{\boldsymbol{\omega}}})^T \mathbf{W} (\tilde{\boldsymbol{\omega}} - \hat{\tilde{\boldsymbol{\omega}}}). \tag{17}$$

The quantizer is a 3-split quantizer with subvectors of dimensions 3-3-4. The error computation is split by dimension, with independent quantization of each subvector. Each component is coded as one of 256 values, determined by an exhaustive search of the corresponding codebook. The codebook indices are transmitted and used locally to reconstruct quantized LSFs.

Quantized LSFs to Interpolated LP Coefficients

The quantized subvectors as determined by the quantizer indices are reassembled into the vector $\hat{\tilde{\boldsymbol{\omega}}}$. The reconstructed LSF vector is given by

$$\begin{aligned}\hat{\boldsymbol{\omega}} &= \hat{\tilde{\boldsymbol{\omega}}} + b(\hat{\boldsymbol{\omega}}_P - \bar{\boldsymbol{\omega}}) + \bar{\boldsymbol{\omega}} \\ &= \hat{\tilde{\boldsymbol{\omega}}} + b\,\hat{\boldsymbol{\omega}}_P + (1-b)\bar{\boldsymbol{\omega}}.\end{aligned} \tag{18}$$

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 difference between $\hat{\omega}_i$ and $\hat{\omega}_{i+1}$ is less than this value, they are forced apart,

$$\omega_i' = \frac{\omega_i + \omega_{i+1}}{2} - \frac{\Delta\omega_{\min}}{2};\ \omega_{i+1}' = \frac{\omega_i + \omega_{i+1}}{2} + \frac{\Delta\omega_{\min}}{2}. \tag{19}$$

Moving one pair apart may cause the spacing to one of the neighbours to become too small. The process of moving pairs apart is repeated until no more violations occur or a maximum of number of trials is exceeded.

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

$$\hat{\boldsymbol{\omega}}_k = \alpha_k \hat{\boldsymbol{\omega}} + (1-\alpha_k)\hat{\boldsymbol{\omega}}_P, \qquad \alpha_k = \frac{k+1}{N_s}, \quad 0 \le k \le N_s - 1. \tag{20}$$

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`.

Differences: Matlab / Reference Code

- The conversion to LSFs is done using the Matlab routine poly2lsf. This difference occasionally gives rise to different quantized values.
- The conversion from LSFs back to LP coefficients uses the Matlab routine lsf2poly together with the cosine table compensation described above. This procedure is clearly effective, since at the decoder, the same procedure results in outputs which are exactly the same as for the reference code.
- 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 directly in radians.

3.1.5 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 is 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 filtering operation in 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

$$W_F(z) = \frac{A(\gamma_1 z)}{A(\gamma_2 z)}, \qquad \gamma_1 = 0.9, \quad \gamma_2 = 0.5. \tag{21}$$

The input to this filter is $x_{\mathrm{HP}}[n]$ and the output of the filter is $x_F[n]$.

The effect of the formant 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. 4.

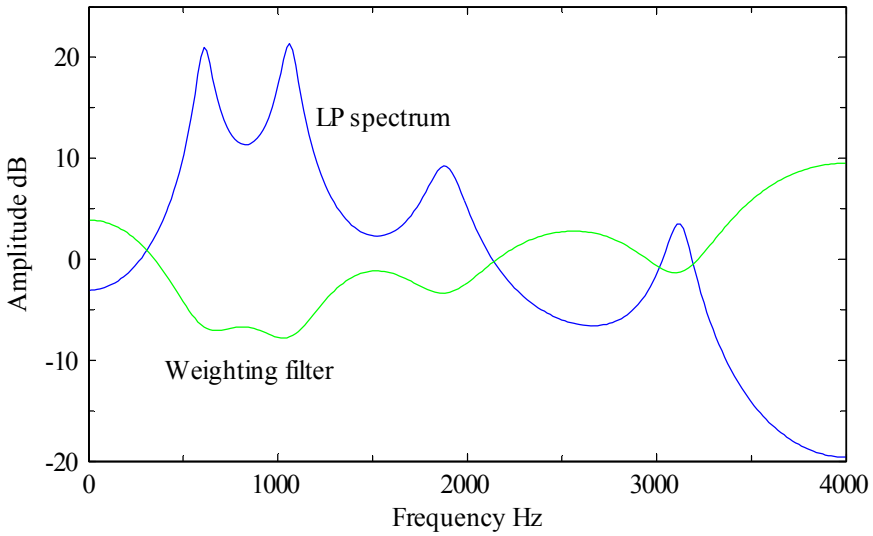

Fig. 4 Formant weighting filter response.

3.1.6 Open-Loop Pitch Estimation

The open loop pitch estimate finds the pitch lag and pitch gain values that minimize the mean-square prediction error. The open-loop pitch is determined from the output of the perceptually weighting filter $x_F[n]$. The prediction error is

$$e[n] = x_F[n] - g\,x_F[n-L]. \tag{22}$$

The squared prediction error for a frame can be written as

$$\varepsilon_L = R[0,0] - 2gR[0,L] + g^2 R[L,L], \tag{23}$$

where the correlation terms are defined as

$$R[i,j] = \sum_{n=0}^{N-1} x_F[n-i]\,x_F[n-j]. \tag{24}$$

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

$$g_{\text{opt}} = \frac{R[0,L]}{R[L,L]}. \tag{25}$$

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

$$\varepsilon_{\text{opt}} = R[0,0] - \frac{R[0,L]^2}{R[L,L]}. \tag{26}$$

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

$$L_o = \max_L \frac{R[0,L]^2}{R[L,L]}. \tag{27}$$

The denominator can be computed recursively,

$$R[l+1,l+1] = R[l,l] + x^2[-i-1] - x^2[N-1-i]. \tag{28}$$

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

$$\frac{R^2[0,L]}{R[L,L]} > \frac{R^2[0,L_o]}{R[L_o,L_o]}, \qquad L < L_o + dL_{\min}. \tag{29}$$

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

$$\frac{R^2[0,L]}{R[L,L]} > A\frac{R^2[0,L_o]}{R[L_o,L_o]}, \qquad L \geq L_o + dL_{\min}, \tag{30}$$

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

3.1.7 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

$$y[n] = x[n] - g_{\mathrm{HNW}} x[n-L]. \tag{31}$$

This is a gain reduced version of a pitch predictor. The response of the filter is

$$W_{\mathrm{HNW}}(z) = 1 - g_{\mathrm{HNW}} z^{-L}. \tag{32}$$

The input to this filter is $x_F[n]$ and the output is the target signal $t[n]$. The magnitude-squared of the frequency response of the filter is

$$\left|W_{\mathrm{HNW}}(\omega)\right|^2 = 1 - 2g_{\mathrm{HNW}} \cos(\omega L) + g_{\mathrm{HNW}}^2. \tag{33}$$

This response oscillates between $(1-g_{\mathrm{HNW}})^2$ and $(1+g_{\mathrm{HNW}})^2$. For positive g_{HNW}, the dips occur at $\omega = 2\pi k / L$, or $f = kF_s / L$, where F_s is the sampling rate.

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 value is found 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 the subframe length of 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 of the HNW filter,

$$\begin{aligned} P_G &= \frac{R[0,0]}{\varepsilon_{\mathrm{opt}}} \\ &= \frac{1}{1 - \dfrac{R^2[0,L_o]}{R[0,0]R[L_o,L_o]}}. \end{aligned} \tag{34}$$

The prediction gain should be larger than a given minimum value, $P_{G\min} = 1.6$,

$$\begin{aligned} P_G &> P_{G\min} \\ R^2[0,L] &> \frac{P_{G\min} - 1}{P_{G\min}} R[0,0]R[L,L]. \end{aligned} \tag{35}$$

The HNW gain value is set to zero if the prediction gain condition is not satisfied.

The optimal predictor gain is

$$g_{\text{opt}} = \frac{R[0, L]}{R[L, L]}. \tag{36}$$

The HNW gain is limited to lie between 0 and 1,

$$g_{\text{HNW}} = \begin{cases} a, & 1 > g_{\text{opt}}, \\ a\, g_{\text{opt},} & 0 \le g_{\text{opt}} \le 1, \\ 0, & g_{\text{opt}} < 0, \end{cases} \tag{37}$$

where a is a multiplicative factor (10/32) to reduce the peak/valley ratio of the frequency response. An example of a HNW response is shown in Fig. 5. Note that the filter emphasizes the regions between pitch harmonics.

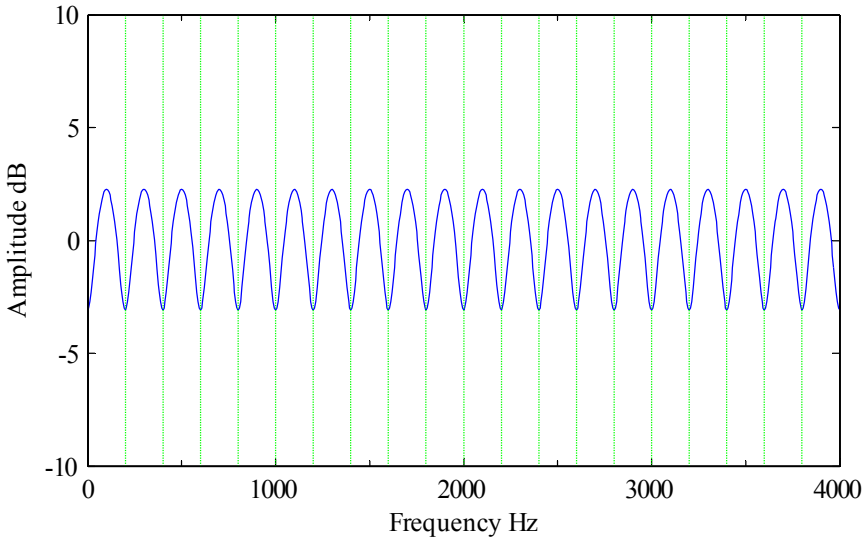

Fig. 5 Harmonic noise weighting filter response ($g_{\text{HNW}} = 0.2, L = 40$ (200 Hz)).

3.2 Analysis-by-Synthesis Target Signal

The concept in analysis-by-synthesis (AbS) is to generate outputs corresponding to different choices of excitation signal parameters. Each candidate excitation signal is passed through the LP synthesis filter and compared to the input speech, see Fig. 6. 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,

$$H_W(z) = H_F(z) H_{\text{HNW}}(z). \tag{38}$$

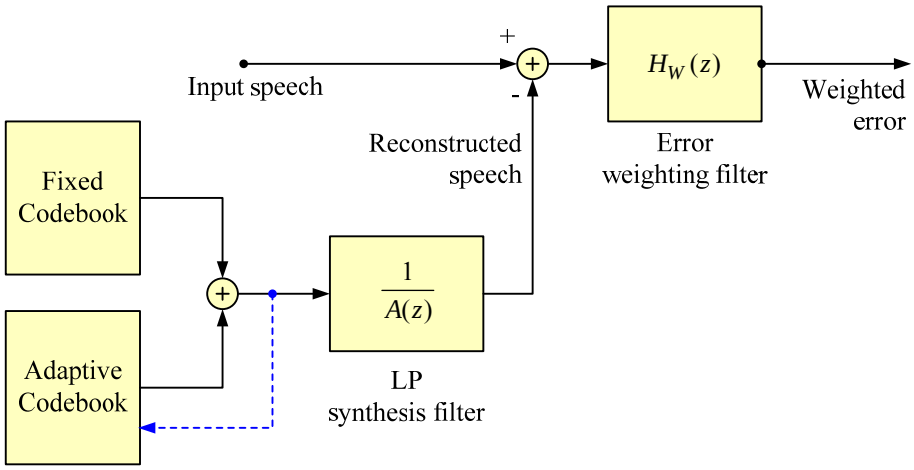

Fig. 6 Analysis-by-synthesis coding.

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. 7. 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.

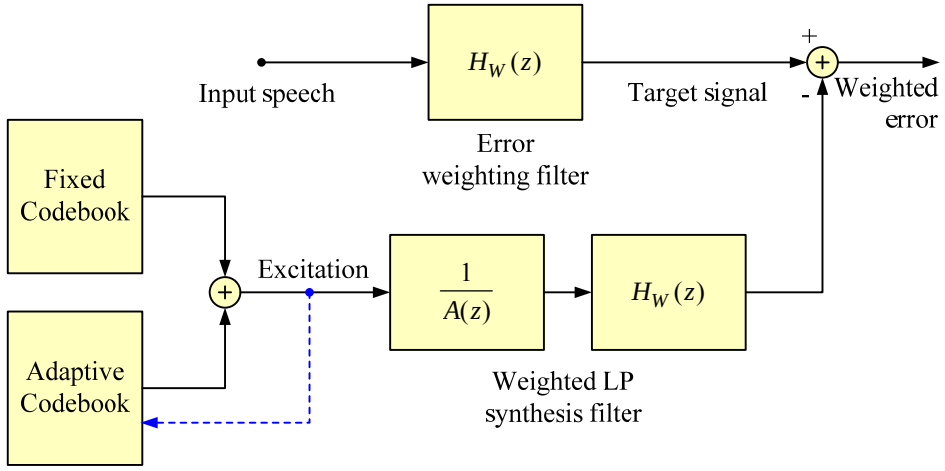

Fig. 7 Restructured Analysis-by-Synthesis Coding

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-input 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 modi-

fied target signal. Furthermore, the composite filter in the excitation signal path can be represented by the impulse response of a weighted synthesis filter.

The excitation consists of two components. The adaptive codebook contribution is taken from 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. 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, given the adaptive codebook contribution, the appropriate fixed codebook contribution is found.

3.2.1 Weighted LP Synthesis Filter

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

$$H(z) = \frac{1}{A(z)} H_F(z) H_{\mathrm{HNW}}(z). \tag{39}$$

Let the weighted synthesis filter have impulse response $h[n]$. This is a causal infinite length 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).

3.3 Subframe Level Processing

The excitation signal is created subframe by subframe from the adaptive codebook contribution and the fixed codebook contribution.

3.3.1 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

$$e_p[n] = \sum_{k=K_L}^{K_U} b_k \tilde{e}[n-k-L], \qquad 0 \le n \le N-1, \tag{40}$$

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

$$\tilde{e}[n] = \begin{cases} e[n], & n < 0, \\ e[\mathrm{mod}(n, L) - L], & n \ge 0. \end{cases} \tag{41}$$

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 ($n \ge 0$) 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 pitch lag L takes on 128 values from 18 to 145 inclusive. In our notation, the lag L refers to the delay to the reference coefficient. Of the 128 values, the last four values are "forbidden", so the effective lag range is 18 to 141. Only these lag values are generated by the coder. If a forbidden lag is detected by the decoder, that frame is flagged as received in error.

The ACB coefficients are taken from one of two codebooks. The first has 85 vector 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 indexing 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 2 and 3 depends on the lag chosen for subframe 2.

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 around the open-loop lag (open-loop lag ±1) determined earlier. This limited search range reduces 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

$$\mathbf{e}_p = \tilde{\mathbf{E}}_L \mathbf{b}, \tag{42}$$

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,

$$\tilde{\mathbf{E}}_L = \begin{bmatrix} \tilde{e}[-L-K_L] & \cdots & \tilde{e}[-L-K_U] \\ \vdots & \ddots & \vdots \\ \tilde{e}[N-1-L-K_L] & \cdots & \tilde{e}[N-1-L-K_U] \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} b_{K_L} \\ \vdots \\ b_{K_U} \end{bmatrix}. \tag{43}$$

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

$$\mathbf{s}_p = \mathbf{S}_L \mathbf{b}, \tag{44}$$

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 weighted LP synthesis filter,

$$\mathbf{S}_L = \begin{bmatrix} h_o & 0 & \cdots & 0 \\ h_1 & h_o & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ h_{N-1} & h_{N-2} & \cdots & h_o \end{bmatrix} \tilde{\mathbf{E}}_L. \tag{45}$$

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

$$\tilde{\mathbf{e}}_k = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \tilde{\mathbf{e}}_{k-1} + \begin{bmatrix} \tilde{e}[-L-k] \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{46}$$

The filtered version of this column can be expressed as

$$\begin{aligned} \mathbf{s}_k &= \mathbf{H}\tilde{\mathbf{e}}_k \\ &= \mathbf{HP}\tilde{\mathbf{e}}_{k-1} + \mathbf{H}\begin{bmatrix} \tilde{e}[-L-k] \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \end{aligned} \tag{47}$$

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 the recursive update to $\mathbf{s}_k$ is

$$\mathbf{s}_k = \mathbf{P}\mathbf{s}_{k-1} + \tilde{e}[-L-k]\begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_{N-1} \end{bmatrix}. \tag{48}$$

Optimal Pitch Contribution

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

$$\begin{aligned} \varepsilon &= (\mathbf{t} - \mathbf{s}_p)^T(\mathbf{t} - \mathbf{s}_p) \\ &= \mathbf{t}^T\mathbf{t} - 2\mathbf{t}^T\mathbf{S}_L\mathbf{b} + \mathbf{b}^T\mathbf{S}_L^T\mathbf{S}_L\mathbf{b} \\ &= \mathbf{t}^T\mathbf{t} - 2\mathbf{R}_{Lts}\mathbf{b} + \mathbf{b}^T\mathbf{R}_{Lss}\mathbf{b}. \end{aligned} \tag{49}$$

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,

$$\mathbf{b}_{\text{opt}} = \max_{\mathbf{b}}(2\mathbf{R}_{Lts}\mathbf{b} - \mathbf{b}^T\mathbf{R}_{Lss}\mathbf{b}). \tag{50}$$

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.

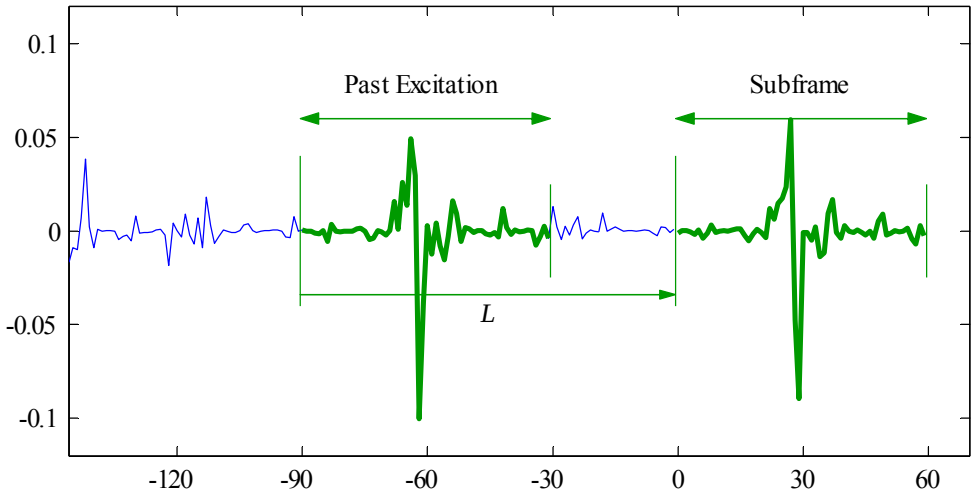

Fig. 8 Adaptive codebook contribution, $L = 90$, $\mathbf{b} = [0.06 \quad -0.17 \quad 0.63 \quad 0.59 \quad -0.17]$.

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,

$$\begin{aligned} \mathbf{b}^T \mathbf{R}_{Lss} \mathbf{b} &= \sum_{k=K_L}^{K_U} \sum_{m=K_L}^{K_U} b_k b_m R_{Lss}[k,m] \\ &= \sum_{k=K_L}^{K_U} b_k^2 R_{Lss}[k,k] + 2 \sum_{k=K_L}^{K_U} \sum_{m=K_L}^{k-1} b_k b_m R_{Lss}[k,m]. \end{aligned} \tag{51}$$

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 coefficients $2\mathbf{b}$, 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 in Eq. (50) uses 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,

$$\mathbf{t}' = \mathbf{t} - \mathbf{H}\tilde{\mathbf{E}}_L \mathbf{b}_{\text{opt}}. \tag{52}$$

Pitch Taming Procedure

The adaptive codebook search uses a "taming" procedure that 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.

3.3.2 Fixed Codebook

The fixed codebook contribution is from a multipulse coding or an ACELP coding procedure. These procedures are similar at a broad level, but differ in detail. Both coding options place a limited number of pulses in a frame. Multipulse uses 6 or 5 pulses per subframe, while ACELP uses 4 pulses per subframe. Both options consider two separate grids for placing the pulses. One grid contains only odd-numbered positions and the other grid contains only even-numbered positions. Both options use the same amplitude for all pulses, but each pulse can take on an arbitrary sign.

The multipulse coding procedure places the pulses sequentially in any of the possible positions on a particular grid (see Table 1). The sequential procedure is suboptimal in that it does not check all possible combinations of positions. Consider a hypothetical case in which the best location for a single pulse is at location 4. The multipulse search will never check the case of pulses at locations 2 and 6 (without one at position 4).

Table 1 Multipulse pulse locations

Grid 0										Grid 1									
0	2	4	6	8	10	12	14	16	18	1	3	5	7	9	11	13	15	17	19
20	22	24	26	28	30	32	34	36	38	21	23	25	27	29	31	33	35	37	39
40	42	44	46	48	50	52	54	56	58	41	43	45	47	49	51	53	55	57	59

The ACELP procedure has 4 tracks in which to place one pulse each (see Table 2). The locations marked with a "–" indicate a missing pulse, thus providing for an opportunity to have fewer than 4 pulses per subframe. The possible pulse locations are more restrictive than for multipulse coding, since only a single location can be used from any track. It does take interaction of the pulses into account more fully than multipulse. To reduce computational complexity, searches involving unpromising combinations of pulse positions are aborted prematurely.

Table 2 ACELP pulse locations

	Grid 0								Grid 1							
Track 0	0	8	16	24	32	40	48	56	1	9	17	25	33	41	49	57
Track 1	2	10	18	26	34	42	50	58	3	11	19	27	35	43	51	59
Track 2	4	12	20	28	36	44	52	–	5	13	21	29	37	45	53	–
Track 3	6	14	22	30	38	46	54	–	7	15	23	31	39	47	55	–

3.3.3 *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 is to 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. The pulse contribution to the excitation is

$$e_f[n] = \sum_{k=1}^{N_f} g_k \delta[n - m_k], \tag{53}$$

where the magnitudes of the pulses $|g_k|$ are the same, but the signs can differ. This contribution is subject to pitch repetition as noted below.

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 indicate 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 pulse repetition procedure can be considered as processing by a causal IIR filter with zero initial state,

$$G_R(z) = \frac{1}{1 - z^{-L}} = \sum_{k=0}^{\infty} z^{-kL}. \tag{54}$$

The impulse response of this filter is

$$h_R[n] = \sum_{k=0}^{\infty} \delta[n - kL]. \tag{55}$$

The pitch-repeated contribution to the weighted signal is

$$\begin{aligned} \tilde{s}_f[n] &= e_f[n] * h_R[n] * h[n] \\ &= \tilde{e}_f[n] * h[n] \\ &= e_f[n] * \tilde{h}[n]. \end{aligned} \tag{56}$$

The second line of this equation uses the pitch-repeated excitation,

$$\tilde{e}_f[n] = \sum_{k=0}^{K} e_f[n - kL], \quad 0 \le n \le N - 1, \tag{57}$$

where the upper limit is $K = \lfloor (N-1)/L \rfloor$. The third line of Eq. (56) uses the pitch-repeated impulse response,

$$\tilde{h}[n] = \sum_{k=0}^{K} h\big[n - kL\big], \qquad 0 \le n \le N - 1. \tag{58}$$

Pulse Positions and Amplitudes

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

The analysis for choosing the next best pulse position can be formulated as follows. Let the target vector be $t[n]$ (the modified target vector, less 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

$$E[m] = E_t - 2g_m R_{th}[m] + g_m^2 R_{hh}[0,0], \tag{59}$$

where

$$\begin{aligned} R_{th}[m] &= \sum_{n=0}^{N-1} t[n]h[n-m] \\ &= \sum_{n=m}^{N-1} t[n]h[n-m], \end{aligned} \tag{60}$$

and

$$\begin{aligned} R_{th}[m] &= \sum_{n=0}^{N-1} h[n]h[n-m] \\ &= \sum_{n=m}^{N-1} h[n]h[n-m]. \end{aligned} \tag{61}$$

The value of gain which minimizes Eq. (59) is

$$g_{\text{opt}} = \frac{R_{th}[m]}{R_{hh}[0,0]}. \tag{62}$$

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. (59) is g_{opt}, the error in using another value of gain can be expressed as

$$E[m] = E_{\min}[m] + (g_m - g_{\text{opt}})^2 R_{hh}[0]. \tag{63}$$

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

Estimating the Pulse Amplitude

The same pulse amplitude is used for all pulses. The amplitude of the first pulse is used as an initial estimate of the pulse amplitude to be used for all pulses. The position of the first pulse that gives the biggest reduction in squared error is found as

$$\begin{aligned} m_{\text{opt}} &= \max_m (2 g_{\text{opt}} R_{th}[m] - g_{\text{opt}}^2 R_{hh}[0,0]) \\ &= \max_m \left(\frac{R_{th}^2[m]}{R_{hh}[0,0]} \right) \\ &= \max_m \left(\left| R_{rh}[m] \right| \right). \end{aligned} \tag{64}$$

Once the best position is found, g_{opt} for that pulse is given by Eq. (62). 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

$$\begin{aligned} i_g &= \min_i \left(\left\| g_{\text{opt}} \right| - A_g(i) \right| \right) \\ &= \min_i \left(\left| A_g(i) R_{hh}[0,0] - \left| R_{th}[m] \right| \right| \right). \end{aligned} \tag{65}$$

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). The best pulse positions will be found for each of these gain values.

Pulse Positions

Given the trial quantized gain, the error for a trial position is (from Eq. (59)),

$$E[m] = E_t \mp 2 A_g(i) R_{th}[m] + A_g^2(i) R_{hh}[0,0], \tag{66}$$

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

$$M = \max_m \left(\left| R_{th}[m] \right| \right). \tag{67}$$

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

$$g_M = \text{sign}(R_{th}[M]) A_g(i). \tag{68}$$

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

$$\begin{aligned} t'[n] &= t[n] - g_M \delta[n - M] * h[n] \\ &= t[n] - g_M h[n - M]. \end{aligned} \tag{69}$$

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

$$
\begin{aligned}
R_{t'h}[n] &= \sum_{n=m}^{N-1} t'[n]h[n-m] \\
&= R_{th}[n] - g_M \sum_{n=m}^{N-1} h[n-m]h[n-M] \\
&= R_{th}[n] - g_M R_{hh}[|n-M|].
\end{aligned} \tag{70}
$$

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 Multipulse 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 $\binom{30}{5} = 142506$ values for odd subframes. The minimum number of bits required to code the pulse positions is 73 bits per frame. This is accomplished by splitting the coding as follows.

- Express the code for subframe i 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 M_i is 2^{16} for even subframes and 2^{14} for odd subframes.
- The q_i values are coded with 16 or 14 bits per subframe, for a total of 60 bits per frame.
- The p_i values are coded as $C_x = p_3 + 9p_2 + 90p_1 + 810p_0$. This value can be represented with 13 bits per frame.

Differences: Matlab / Reference Code

- The multipulse coding differs in small details from the reference code. 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. For the Matlab search, the order of the grid positions is reversed to get the same behaviour as 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.

3.3.4 Combined Gain Coding

The ACB coefficient code and the code for the pulse gain are combined into a single 12-bit value. The ACB coefficient code can take on either 85 or 170 values. The former is used only for the multipulse mode when short pitch lags occur. When the 85 element code table is used, one-bit becomes available to specify whether pitch repetition is to be used or not. The combined code is calculated as follows

$$C_{\text{comb}} = \begin{cases} C_{\text{pulse}} + 24 \cdot C_{\text{ACB}}, & \text{if } C_{\text{ACB}} \text{ takes on 170 values,} \\ C_{\text{pulse}} + 24 \cdot C_{\text{ACB}} + 2048 \cdot C_{pr,} & \text{if } C_{\text{ACB}} \text{ takes on 85 values,} \end{cases} \tag{71}$$

where C_{pulse} is the code for the pulse gain ($0,\ldots,23$), C_{pitch} is the code for the ACB vector ($0,\ldots,84$ or $0,\ldots,169$), and C_{pr} is a one-bit code ($0,1$) to control pitch repetition.

3.3.5 State Update

The combined excitation signal is the sum of the ACB contribution and the fixed codebook (multipulse or ACELP) contribution,

$$e[n] = e_p[n] + e_f[n]. \tag{72}$$

This signal is clipped to prevent overloads,

$$e'[n] = \begin{cases} 1-\upsilon & e[n] > 1 - 3\upsilon/2, \\ e[n] & -(1-\upsilon/2) \le e[n] \le 1 - 3\upsilon/2, \\ -1 & e[n] < -(1-\upsilon/2), \end{cases} \tag{73}$$

where $\upsilon = 1/32768$. This slightly unorthodox clipping strategy leaves small "holes" of size $\upsilon/2$ near the limits. The clipping is done every subframe.

At the *coder*, this clipped excitation is used for the "past" excitation for the ACB contribution, but the unclipped excitation is used to update the state of the LP synthesis filter (see Section 3.2.1).

At the *decoder*, the clipping is done once per frame. As such, the past excitation consists of some clipped values (from previous frames) and some unclipped values (from previous subframes in the current frame). This results in a mistracking between the coder and decoder if the excitation signal is large enough to require clipping.

4 BitStream

The coder generates a bitstream, stored as binary data in a file. Each frame of data is preceded by a two-bit code, indicating the type of frame. For the different types of frames, the number of data bits changes. The number of data in a frame of data is as follows (including the 2-bit frame-type indicator).

- Multipulse mode: The multipulse mode uses 24 bytes (192 bits) per frame.
- ACELP mode: The ACELP mode uses 20 bytes (160 bits) per frame.
- SID mode: A Silence Insertion Descriptor frame uses 4 bytes (32 bits) per frame in the bitstream file.
- Null mode: A null frame uses 1 byte (of which 2 bits are used) per frame in the bitstream file.

The number of data bits (following the 2-bit frame-type indictor) is summarized in the table below.

Table 3 Bitstream format

Frame Code	Frame Type	Number of Data Bits
00	Multipulse	190
01	ACELP	158
10	SID	30
11	Null	0

There are common threads for the modes. The first three modes use the same LP parameter coding. Multipulse and ACELP use the same adaptive codebook lag coding and the same combined gain coding.

4.1 Multipulse Mode

The data bits for the multipulse mode are as follows.

- LP parameters: codes for the split VQ LSF parameters, total 24 bits
- Adaptive codebook lags: coded with [7, 2, 7, 2] bits per subframe, total 18 bits.
- Combined gain codes: 12 bits per subframe, total 48 bits.
- Multipulse grid codes: 1 bit per subframe, 4 bits total.
- Reserved bit: One unused bit.
- Multipulse position codes: 73 bits total.

- Multipulse signs: [6, 5, 6, 5] bits per subframe, total 22 bits.

4.2 ACELP Mode

The data bits for the ACELP mode are as follows.

- LP parameters: codes for the split VQ LSF parameters, total 24 bits.
- Adaptive codebook lags: coded with [7, 2, 7, 2] bits per subframe, total 18 bits.
- Combined gain codes: 12 bits per subframe, total 48 bits.
- ACELP grid codes: 1 bit per subframe, 4 bits total.
- ACELP pulse position codes: 12 bits (3 bits for each of 4 tracks) per subframe, 48 bits total.
- ACELP pulse signs: 4 bits (one bit per track) per subframe, total 16 bits.

4.3 SID Mode

The data bits for the SID mode are as follows.

- LP parameters: codes for the split VQ LSF parameters, total 24 bits.
- SID gain code: 6 bits per frame.

5 G.723.1 Decoder

The decoder is shown in Fig. 9. The decoder does no searching, so is computationally much less onerous than the coder.

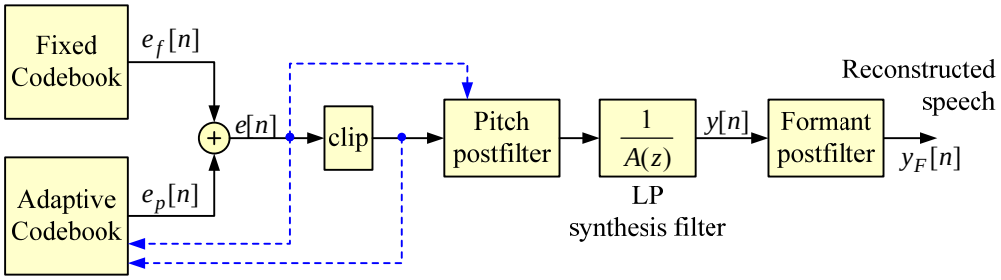

Fig. 9 Decoder.

The decoder handles five modes.

1. Multipulse mode
2. ACELP mode
3. SID (silence insertion descriptor) mode
4. Null (silence) frame mode
5. Packet loss mode

Consider the first four modes for now. Active speech is coded with either multipulse or ACELP. SID frames always follow an active speech frame. An SID frame updates the LP parameters for a "silence" frame and sets the comfort noise level for that frame. Null frames generate comfort noise using the parameters sent during the previous SID frame. As such an SID frame must always precede null frames. The allowable mode sequences are shown in the state transition diagram shown in Fig. 10.

5.1 Excitation Generation

The excitation has two components, one from the adaptive codebook and one from the fixed codebook.

5.1.1 Adaptive Codebook Contribution

For the active speech modes (multipulse and ACELP), the processing of the adaptive codebook lags and gains is the same. For the comfort noise modes, the ACB contribution is not trans-

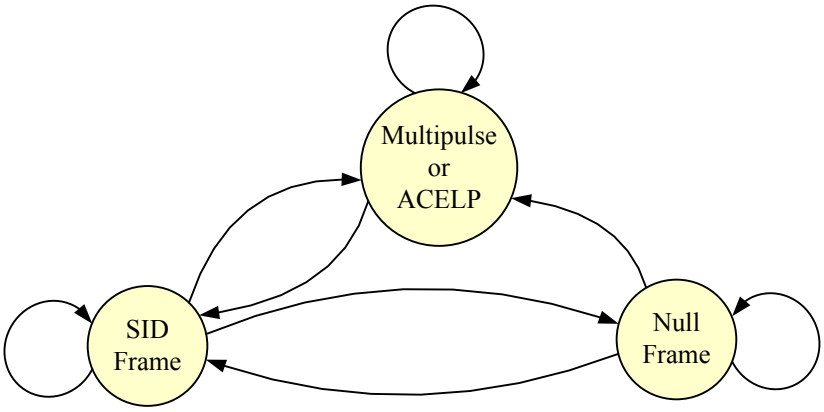

Fig. 10 Mode transitions.

mitted and is not used to generate a “pitch” contribution. Instead, it is used to add a random component to enhance richness of the excitation.

For the active speech modes, the ACB contribution is decoded as follows.

- Decode the ACB lags. These are absolute lags for subframes 0 and 2 and relative lags for subframes 1 and 3. Note that the 4 largest lags for subframes 0 and 2 are “forbidden” lag values. The presence of these values triggers the packet loss mode (more on this later).
- The 24 level pulse gain code is extracted from the combined gain code.
- For multipulse coding, the ACB coefficient code can take on either 85 or 170 values. The 85-element codebook is used for short pitch lags (less than 58 samples). For these short pitch lags, an extra bit is thus available to signal whether to use pitch lag repetition of the pulses or not. The pitch lag specified for subframe 0 which triggers which codebook to use for subframes 0 and 1. Likewise for subframes 2 and 3, it is the pitch lag for subframe 2 determines which codebook to use. The combined gain coding leaves some combined gain codes unused. If these appear, a packet loss is declared.
- For ACELP coding, the ACB coefficient code is always taken from the 170-element table. The combined gain coding leaves some combined gain codes unused. If these appear, a packet loss is declared.

The ACB contribution is determined by taking the past excitation, repeating and shifting if necessary, and filtering using the vector ACB filter coefficients. The pitch filter is a multi-tap IIR filter of the form

$$e_p[n] = \sum_{k=K_L}^{K_U} b_k \tilde{e}[n-k-L], \tag{74}$$

where $\tilde{e}[n]$ is a pitch repeated version of the past excitation (containing both the ACB and fixed codebook contributions),

$$\tilde{e}[n] = \begin{cases} e[n], & n < 0, \\ e[\text{mod}(n, L) - L], & n \geq 0. \end{cases} \tag{75}$$

The pitch filter uses 5 taps, with the reference tap being in the middle ($K_L = -2$ and $K_U = 2$).

5.1.2 Multipulse Excitation

The multipulse contribution is determined as follows for each subframe.

- The pulse grid bit selects the pulse grid (odd or even samples in a subframe), see Table 1.
- The pulse position codes select the pulse positions (6 positions for subframes 0 and 2, and 5 positions for subframes 1 and 3).
- Set the pulse amplitudes together with the pulse signs in the pulse positions to form the excitation.
- Pitch repetition is an option for subframes 0 and 1, if the pitch lag for subframe 0 is less than 58. Similarly, for subframes, 2 and 3, pitch repetition can be used if the pitch lag for subframe 2 is less than 58. If pitch repetition is enabled and the pitch repetition bit is set, take the pulse excitation and form the shifted and repeated excitation.

The multipulse contribution to the excitation is

$$e_f[n] = \sum_{k=1}^{N_f} g_k \delta[n - m_k], \tag{76}$$

where the magnitudes of the pulses $|g_k|$ are the same, but the signs can differ. If the pitch repetition bit is to be applied, the pitch repeated excitation contribution is

$$\tilde{e}_f[n] = \sum_{k=0}^{K} e_f[n - kL], \quad 0 \leq n \leq N-1, \tag{77}$$

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

5.1.3 ACELP Excitation

The ACELP contribution is determined as follows, subframe by subframe.

- The pulse grid bit selects the pulse grid (odd or even samples in a subframe), see Table 2.
- Place a pulse from each of the 4 tracks in the position determined by the ACELP position codes.
- Set the pulse amplitudes,
- Set the sign of each pulse according to the ACELP sign bits.
- Repeat a scaled version of the pulse excitation. The repetition lag is determined from the pitch lag for a subframe, together with an offset valued from a table addressed by the ACB coefficient code. If the repetition lag is less than 58, the pulse excitation is shifted by that lag and scaled by a gain determined from the ACB coefficient code.

The pulse contribution to the excitation is

$$e_f[n] = \sum_{k=1}^{N_f} g_k \delta[n - m_k], \tag{78}$$

where the magnitudes of the pulses $|g_k|$ are the same, but the signs can differ. This contribution is subject to pitch repetition The pitch repeated version of the ACELP excitation is

$$\tilde{e}_f[n] = \sum_{k=0}^{K} g_r^k e_f[n - kL'], \qquad 0 \le n \le N-1, \tag{79}$$

where the upper limit is $K = \lfloor (N-1)/L' \rfloor$. The pitch repetition uses a single coefficient g_r based on the current ACB gain vector index. The lag L' is offset from the ACB lag (the offset is based on the current ACB gain index), presumably to try to account for the fact that the maximum gain in the ACB gain vector is not necessarily in the middle of the gain vector.

5.1.4 SID Mode and Null Mode

The SID and Null modes generate comfort noise. The SID mode sets up the LP parameters and the gain to be used for the comfort noise generation (CNG). The parameters transmitted during an SID frame are the LP parameters and a gain value. No information is transmitted during a null frame. During a null frame, the information from the previous SID frame is used.

The level for the excitation is controlled by two state variables, the SID gain and the CNG gain. For the first SID frame following an active frame, the comfort noise level is initialized to received SID gain value,

$$g_{\mathrm{CNG}} = g_{\mathrm{SID}}. \tag{80}$$

For each null frame and SID frame, the comfort noise gain is updated subframe-by-subframe,

$$g_{\mathrm{CNG}} \leftarrow (1-\alpha) g_{\mathrm{CNG}} + \alpha g_{\mathrm{SID}}, \tag{81}$$

Where α is 1/8. For the first group of comfort noise frames, since the SID gain and CNG gain was initialized to the SID gain, the gain remains constant. When a subsequent SID frame follows a non-active frame, the SID gain is updated, and the gain smoothing comes into play.

Comfort Noise Excitation

The excitation in both the SID and Null modes is generated randomly. The random excitation is generated two subframes (120 samples) at a time. There are two parts to the excitation, an adaptive codebook contribution and a multipulse contribution.

The ACB contribution to the excitation is generated with a random lag in the range 123 to 143. This lag is used for the first subframe of the two subframes. The lag for the second subframe is offset by a fixed number from this value. The lag is larger than the number of samples in the pair of subframes and thus guarantees that only previous excitation values are accessed — there is no need for pitch repetition. The ACB coefficient vector is chosen randomly from a 50 element (low amplitude) subset of the 170 element ACB table of ACB coefficient vectors. The ACB contribution thus supplies a random component to the excitation for comfort noise.

The multipulse contribution is generated by randomly choosing a grid (odd or even positions), randomly choosing the pulse positions in the grid (6 pulses for the first of the two subframes and 5 pulses for the second of the two subframes), and randomly choosing a sign for each pulse. The common amplitude of the pulses for both subframes is initially set to unity.

Scaled Excitation

The excitation consists of the ACB contribution and a scaled multipulse contribution,

$$e[n] = e_p[n] + G_f e_f[n]. \tag{82}$$

The gain G_f is chosen to match a target average energy for the frame. The target average energy is the square of the CNG gain value,

$$E_T = g_{\mathrm{CNG}}^2. \tag{83}$$

In vector notation, the goal is to minimize the difference in average energies,

$$\begin{aligned} E_T - \frac{1}{N}\mathbf{e}^T\mathbf{e} &= E_T - \frac{1}{N}\left(\mathbf{e}_p^T\mathbf{e}_p - 2G_f\mathbf{e}_p^T\mathbf{e}_f + G_f^2\mathbf{e}_f^T\mathbf{e}_f\right) \\ &= aG_f^2 + 2bG_f + c. \end{aligned} \tag{84}$$

We can solve the quadratic equation for the multipulse gain G_f.

$$G_f = \begin{cases} \dfrac{-b \pm \sqrt{b^2 - ac}}{a}, & \text{if } b^2 > ac, \\ -\dfrac{b}{a}, & \text{if } b^2 \le ac. \end{cases} \tag{85}$$

The first case occurs if the energy of the ACB contribution is less than the target energy. The gain is used to scale the multipulse contribution to bring the total energy to the target energy. In that case, solving the quadratic above will give two real solutions. We take the solution of smaller absolute value (or the positive one if they have equal magnitudes). The second case occurs when It may no real value of gain will allow matching to the target energy. This can happen when the energy of the ACB contribution is already too large. In that case, the gain sets the energy of the excitation as close as possible to the target energy. The two cases are shown in Fig. 11.

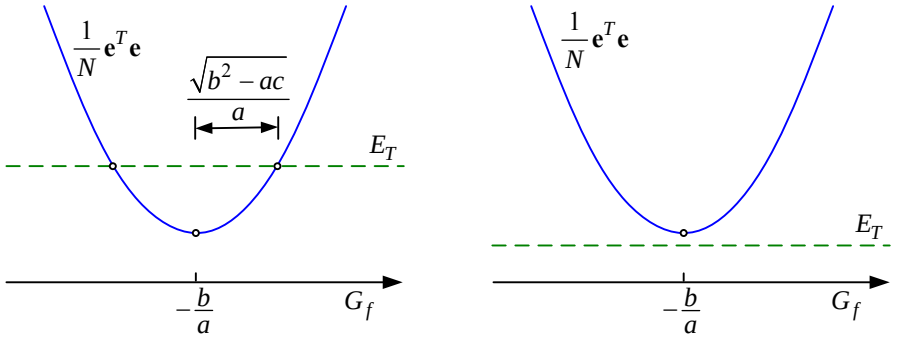

Fig. 11 Two cases for the pulse amplitude determination.

Random Number Generator

The comfort noise is generated using a mixed congruential method,

$$x_{k+1} = (ax_k + c) \bmod m, \tag{86}$$

where the multiplier a is 521, the increment c is 259, and the modulus m is 2^{16}. The modulus is of the form 2^b; the increment c is relatively prime to the modulus; and $a \bmod 4$ is 1. These properties guarantee that the sequence of pseudo-random numbers has a period of m [6]. The initial value x_0 (the seed) is 12345. This pseudo-random number generator is used to generate random integer values. In the C reference code, a 16-bit variable is used to store the random number. This value is masked to 15-bits and scaled to give random integer values. In the Matlab code, this behaviour is achieved as follows,

$$i = \left\lfloor (x_{k+1} \bmod 32768) \frac{N}{32768} \right\rfloor, \tag{87}$$

resulting in i taking on integer values $[0,1,\ldots,N-1]$.

In some cases, the random integer is further dissected to give random bits. For instance, to generate 13 random bits (2 grid values, 11 pulse signs for a pair of subframes), N is set to 2^{13}.

The random number generator is reset back to the initial seed, whenever a non-comfort noise frame is processed.

5.1.5 Excitation Clipping

The excitation is formed as the sum of the ACB contribution and the fixed codebook contribution,

$$e[n] = e_p[n] + e_f[n]. \tag{88}$$

As the excitation is formed subframe by subframe, the ACB uses past values to create the pitch contribution to the excitation.

The excitation is clipped once per frame. This means that in the middle of processing a frame, some samples of the past excitation are clipped and some are not. For instance when processing the second subframe in a frame, the immediate past excitation of 60 samples (one subframe) is unclipped, but samples further back are clipped. This affects the ACB calculations.

5.2 Pitch Postfilter

During active frames, the clipped excitation is processed through a pitch postfilter. However it is the unclipped excitation that is used to determine the parameters for the pitch postfilter. The pitch postfilter is not used for the comfort noise (or the frame loss) modes. The output of the pitch postfilter drives the LP synthesis filter.[2]

If both the pitch postfilter and the LP synthesis filter were time-invariant, the order of the filters would not matter. However, there seems to be an advantage to having the pitch filter ahead of the LP synthesis filter when the parameters of the pitch filter (both the lag and the coefficient) change — the LP synthesis filter tends to smooth out the transitions.

---

[2] The reference code provides a flag to disable the pitch postfilter. If the pitch postfilter is disabled, the unclipped excitation is passed through to synthesize the output signal. This behaviour has been retained in the Matlab code.

The pitch postfilter uses a formulation similar to that of the harmonic noise weighting filter encountered in the coder (see Section 3.1.7). However, now we want dips at the pitch harmonics — the HNW filter has dips at the pitch harmonics. This change is accomplished by changing the sign before the gain in the filter. The pitch postfilter is of the form,

$$y[n] = g_E(x[n] + g_{\text{ppf}}[n + L]), \tag{89}$$

where g_E is an overall gain chosen to make the energy of the output signal equal to the energy of the input signal.

Pitch Postfilter Lag

The lag L can be negative (looking backward in time) or positive (looking forward time). The pitch postfilter is applied after a whole frame of excitation signal has been formed. For the first subframe in the frame, the pitch lag can look forward up to 3 subframes; for the last subframe in the frame, no future samples are available, so the pitch lag can only look backward. The search for pitch lags is done around the pitch lags transmitted for subframe 0 (used to find the pitch for subframes 0 and 1) and subframe 2 (used for subframes 2 and 3). If the transmitted pitch lag is L_o, the search for pitch predictors is done for lags $-L_o - 3$ to $-L_o + 3$ (backwards) and $L_o - 3$ to $L_o + 3$ (forward). The lag is found as the lag which maximizes the correlation $R[0, L]$,

$$L_p = \max_L R[0, L], \tag{90}$$

The correlation is computed from

$$R[i, j] = \sum_{n}^{N-1} e[n - i]e[n - j], \tag{91}$$

where N is the length of a subframe. Only positive correlations are considered in the search for the maximum lag.

The lag which maximizes the correlation is used to compute the optimal predictor coefficient,

$$g_{\text{opt}} = \frac{R[0, L_p]}{R[L_p, L_p]}. \tag{92}$$

The pitch postfilter will be turned off unless the optimal predictor has a sufficiently large prediction gain. The error with the optimal pitch coefficient is (compare with the development in Section 3.1.6)

$$\varepsilon_{\text{opt}} = R[0,0] - \frac{R[0,L_p]^2}{R[L_p,L_p]}. \tag{93}$$

Pitch Postfilter Gain

The prediction gain is,

$$\begin{aligned} P_G &= \frac{R[0,0]}{\varepsilon_{opt}} \\ &= \frac{1}{1-\dfrac{R^2[0,L]}{R[0,0]R[L,L]}}. \end{aligned} \tag{94}$$

The prediction gain should be larger than a given minimum value, $P_{G\min} = 4/3$,

$$\begin{aligned} P_G &> P_{G\min} \\ R^2[0,L] &> \frac{P_{G\min}-1}{P_{G\min}} R[0,0]R[L,L]. \end{aligned} \tag{95}$$

The pitch postfilter gain value is set to zero if the prediction gain condition is not satisfied.

The pitch postfilter gain value is found from the predictor value as,

$$g_{\text{ppf}} = \begin{cases} a, & 1 > g_{\text{opt}}, \\ a\,g_{\text{opt,}} & 0 \le g_{\text{opt}} \le 1, \\ 0, & g_{\text{opt}} < 0, \end{cases} \tag{96}$$

where a is a multiplicative factor (0.1875 for the multipulse mode and 0.25 for the ACELP mode). The energy of the output signal from the pitch postfilter for this value of gain (without further scaling) can be calculated as

$$\varepsilon = R[0,0] + 2g_{\text{ppf}}R[0,L] + g_{\text{ppf}}^2 R[L,L]. \tag{97}$$

The correlation values are already available from the calculation of the prediction gain. The energy of the output signal can be matched to the energy of the input signal by choosing the scaling value,

$$g_E = \sqrt{\frac{R[0,0]}{\varepsilon}}. \tag{98}$$

An example of the frequency response of the pitch postfilter is shown in Fig. 12. The scaling gain is not included in the response. Note that this filter has valleys between the harmonics (in contrast to the harmonic noise weighting filter). This example uses the largest gain value for the multipulse mode.

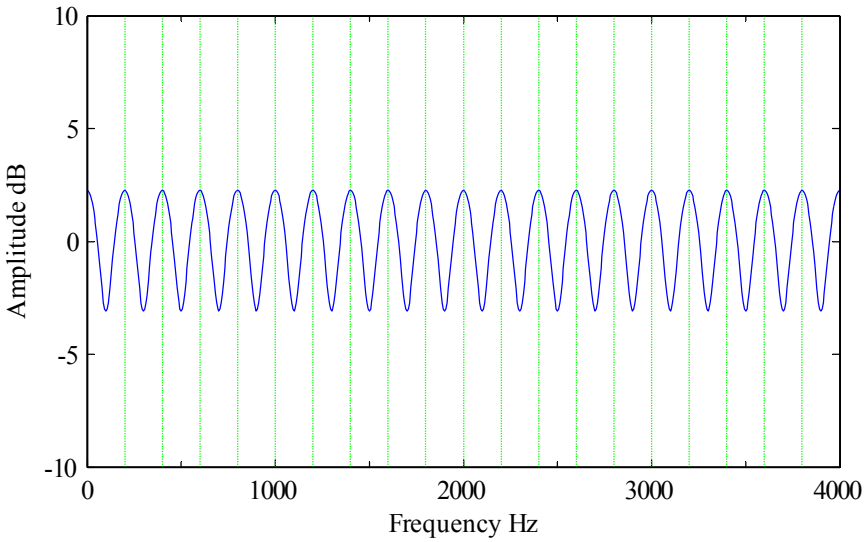

Fig. 12 Pitch postfilter ($g_{\text{ppf}} = 0.1875$, $L = 40$ (200 Hz)).

5.3 LP Parameters

For the active modes (multipulse and ACELP) and the SID mode, LP parameters are sent in the bitstream. These are processed as follows.

- Decode the quantized LSF values from the bitstream.
- 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 interpolated LSF values to the corresponding LP coefficient values.

The excitation signal is filtered with the all-pole filter specified by the LP parameters. The coefficients are updated subframe by subframe.

5.4 Formant Postfilter

The formant postfilter is applied for all modes after the LP synthesis filter. The format post-filter is updated each subframe and is of the form,

$$F_F(z) = \frac{A(\gamma_n z)}{A(\gamma_d z)}(1 - a r_y z^{-1}), \tag{99}$$

where the bandwidth expansion factors are $\gamma_n = 0.65$ and $\gamma_d = 0.75$. The second term in the filter is an adaptive "tilt" compensation which depends on the 1-lag correlation. The parameter a is 0.25. The one-lag correlation is a time-averaged value which is updated from subframe to subframe as

$$r_y \leftarrow (1-\beta) r_y + \beta r[1], \tag{100}$$

where the parameter β is 0.25, and r[1] is the normalized autocorrelation at lag 1 for the current subframe, where

$$r[k] = \sum_{l=0}^{N-k} y[l] y[l+k]. \tag{101}$$

The first term in formant postfilter is the ratio of two terms. Each of the terms by themselves gives a frequency response which gets less peaky as the gamma factor is decreased. However, if the gamma factors are equal, the response is flat.

An example of the formant postfilter is shown in Fig. 13. The LP spectrum is the same as used in the example of a formant weighting filter (see Fig. 4). For this plot, the 1-lag autocorrelation was taken from the correlation values used to generate the LP coefficients. This example has quite a peaky LP response, but the postfilter is quite flat. The dashed line in the figure shows the postfilter without the tilt compensation.

Gain Scaling

The output of the formant postfilter is scaled to match the energy of its output to the energy of its input. The scaling gain required to achieve this is

$$g_s = \sqrt{\frac{\mathbf{y}^T \mathbf{y}}{\mathbf{y}_F^T \mathbf{y}_F}}, \tag{102}$$

where $\mathbf{y}$ is the vector of outputs from the LP synthesis filter for a subframe, and $\mathbf{y}_F$ is the corresponding vector of outputs from the formant postfilter. Rather than use a constant gain for the subframe, the implementation changes the gain sample-by-sample,

$$g_A[n] = (1-\alpha) g_A[n-1] + \alpha g_s. \tag{103}$$

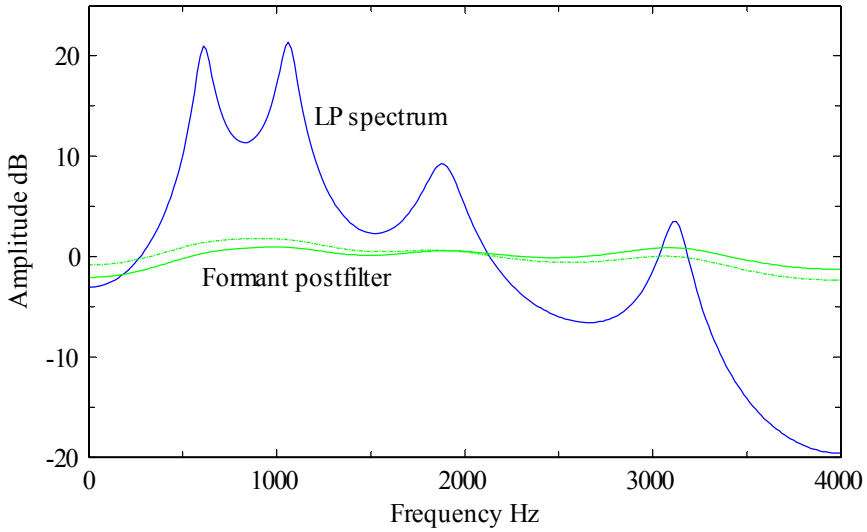

Fig. 13 Formant postfilter. The dashed line shows the postfilter filter response with no tilt compensation.

6 Packet Loss Concealment

The G.723.1 decoder has provision to fill in missing frames of data. The basic strategy is to generate an output based on past pitch lags and excitation energies. The regenerated excitation is used as input to an LP synthesis filter based on past LP coefficients. The packet loss concealment (PLC) strategy has tentacles in many parts of the decoder.

In the reference code and Matlab implementation, lost frames are signalled in a frame loss file. Each word of that file is a 0 (no frame loss) or a 1 (frame loss). The presence of certain illegal code values in the bitstream file are also used to signal a lost frame.

6.1 Preparations for PLC

During active speech frames, certain quantities are calculated in anticipation of subsequent lost frames.

Pitch Lag

The unclipped excitation is used to choose a pitch value that will be used for packet loss concealment in subsequent frames. That pitch lag is chosen over the last half frame (two subframes). The lag is chosen based on the maximizing the correlation $R[0,L]$,

$$L_p = \max_L R[0,L], \tag{104}$$

The correlation is computed from

$$R[i,j] = \sum_{n}^{N-1} e[n-i]e[n-j], \tag{105}$$

where N is the length of a two subframes. The search for lags is around (–3 to +3 relative to) the pitch lag for subframe 2. Only positive correlations are considered in the search for the lag. The lag is set to zero to disable a pitch contribution to the packet concealment excitation. There is also a test on the prediction gain. If the prediction gain is less than $P_{G\min} = 8/7$, the pitch contribution is turned off (compare with the analysis in Section 3.1.7),

$$\begin{aligned} P_G &> P_{G\min} \\ R^2[0,L_p] &> \frac{P_{G\min}-1}{P_{G\min}} R[0,0]R[L_p,L_p]. \end{aligned} \tag{106}$$

The value L_p is saved lest the next frame is lost. In the Matlab code, value of `NaN` (not-a-number) signals that the pitch contribution should be turned off.

Excitation Gain

A gain value for an "unvoiced" contribution for future lost frames is also saved. This value is taken from the pulse amplitudes (multipulse or ACELP) for the last two subframes in a frame. The gain value is determined by averaging the code indices of the pulse amplitudes for those two frames. Let the table of gains be $A_g(i)$ (24 values). The gain values are ordered in amplitude. Given the two indices i_2 and i_3, the gain is calculated as,

$$A_{\text{PLC}} = A_g\left(\left\lfloor \frac{i_2 + i_3}{2} \right\rfloor\right). \tag{107}$$

In the Matlab code, the gain calculation uses a table indexed by i_2 and i_3.

SID Gain

As the earlier mode switching diagram showed, an SID frame should precede null frames. An SID gain is pre-calculated during an active speech frame as protection against the loss of an SID frame. The average excitation energy in the last half frame of the last active frame is used to calculate the SID gain,

$$g_{\text{SID}} = \sqrt{\mathbf{e}^T \mathbf{e}}, \tag{108}$$

where $\mathbf{e}$ is the vector of excitation samples for the last half of the frame.

The gain is quantized using the same quantizer used to generate the gain indices for SID frames. The quantizer uses a segmented compression function. The quantizer has 4 segments; each segment has 16 uniformly spaced levels; the step sizes are in the ratios 1:2:4:4, i.e., the last two segments have the same step size. In the Matlab code, the quantizer is specified by a table of $N_q - 1$ decision levels. Let the decision levels be $q_D(i)$, for $i = 1, \ldots N_q - 1$. The decision levels are in increasing order. For convenience, let $q_D(0) = -\infty$ and $q_D(N_q) = \infty$. The quantizer cells are the intervals

$$S_i = \{q_D(i) \le g < q_D(i+1)\}. \tag{109}$$

The quantizer index is determined by which region g lies in,

$$i_Q(g) = i \quad \text{if } g \in S_i. \tag{110}$$

The inverse quantization step is a table lookup,

$$\hat{g} = q_V(i_Q(g)). \tag{111}$$

The quantizer input/output characteristic is shown in Fig. 14. The thick lines indicate regions in which both end points (in the reference code) are included in the region. Thin lines indicate regions in which neither end point is in the region.

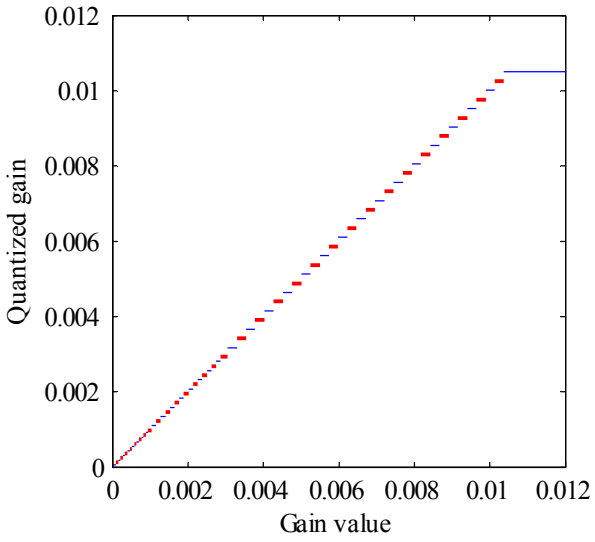

Fig. 14 SID gain quantizer. Thick lines indicate quantizer regions in which both end points are included in the region. Thin lines indicate quantizer regions in which neither end point is included in the region.

The quantized SID gain value will be used for null frames if the preceding (non-PLC) frame was active speech. This indicates that an SID frame was lost. In that case, the CNG gain is reset to the SID gain.

Differences: Matlab / Reference Code

- In the reference code, the SID gain quantizer is implemented as a search for the appropriate segment and then a binary search within a segment. In the Matlab code, a binary search based on a table of quantizer break-points is used.
- The quantizer as defined above (and used in the Matlab implementation) always has the lower decision level included in the region. In the reference code the end-point behaviour is not consistent from quantizer cell to quantizer cell. To be compatible, the Matlab code adds a small value (the machine epsilon) ε to the decision boundaries as needed to be compatible with the reference code.

6.1.1 PLC Interaction with Modes

A PLC frame can occur at any point, knocking out active speech or comfort noise frames. These cases are handled differently.

- If a PLC frame follows an active speech frame, a packet loss concealment strategy (PLC Mode) is employed using the information saved from the last active frame.
- If a PLC frame follows an SID frame or a null frame, it is assumed that the lost frame can be well-represented with comfort noise. In that case, the null frame mode is employed.
- If a PLC frame knocks out an SID frame, a following null frame picks up the saved value of the SID gain (as outlined above).

6.2 PLC Mode Excitation

The PLC mode keeps a count of the number of successive frame losses. The excitation gain is scaled down on each PLC frame (including the first one in a sequence),

$$A_{\mathrm{PLC}} \leftarrow \beta A_{\mathrm{PLC}}, \tag{112}$$

where β is 3/4. On the third error frame, the output is muted. This is accomplished by setting the excitation, including the past excitation memory, to zero.

Voiced Frames

If the PLC pitch lag (generated during the last active speech frame) indicates that the frame should be considered to be voiced, the excitation is calculated as

$$e[n] = \beta \tilde{e}[n], \tag{113}$$

where β is 3/4 and $\tilde{e}[n]$ is the pitch repeated past excitation,

$$\tilde{e}[n] = \begin{cases} e[n], & n < 0, \\ e[\mathrm{mod}(n, L_p) - L_p], & n \geq 0. \end{cases} \tag{114}$$

This contribution is calculated for the whole frame. There is no other component to the excitation during voiced speech.

Unvoiced Frames

For unvoiced speech, the excitation consists of scaled uniform random values,

$$e[n] = A_{\mathrm{PLC}} \eta[n], \tag{115}$$

where $\eta[n]$ is a random value in the range [–1,1). The random number generator is basically the same as used for comfort noise, except that the resulting values are interpreted as signed values. A separate seed (never reset) is kept for the PLC mode. The random number generated by the random number generator is x_k, taking on values between $0,\ldots,65535$ (see Eq. (86)). The random excitation sample is calculated as

$$\eta[n] = \begin{cases} \dfrac{x_k}{32768}, & x_k < 32768, \\ \dfrac{x_k - 65536}{32768}, & x_k \geq 32768. \end{cases} \tag{116}$$

The past excitation is set to zero for subsequent frames.

6.3 PLC Mode LP Synthesis

There are no LP parameters available during a PLC mode. Instead, a default LP code vector is used. This code vector gives an all-zero difference LSF vector. The LSF vector is then (see Eq. (18)),

$$\hat{\boldsymbol{\omega}} = b(\hat{\boldsymbol{\omega}}_P - \bar{\boldsymbol{\omega}}) + \bar{\boldsymbol{\omega}}, \tag{117}$$

where $\hat{\boldsymbol{\omega}}_P$ is the quantized LSF vector from the last "good" frame and $\bar{\boldsymbol{\omega}}$ is the fixed average LSF value. In addition, the prediction coefficient b used for the differential coding is increased from the normal 12/32 to 23/32. The LSF vector moves from the previous quantized LSF vector towards the mean LSF vector.

Before conversion from LSFs to LP coefficients, an increased minimum separation of the LSF values is imposed.

7 Comments

Through careful attention to detail, the Matlab implementation duplicates the results for the decoder. The coder is only a partial implementation – only the multipulse mode has been programmed. In the process of implementing G.723.1 in Matlab, much has been learned. Some comments on the implementation are given below.

- There are differences in the bitstream files produced by the Matlab implementation and the reference code. In most cases, the differences are due to quantizing the LP parameters to a different value. A difference in quantized LSF values for one frame will result in the next few frames differing due to the memory implicit in the coder. In fact, the Matlab implementation probably computes more accurate LP values. The conversion from quantized values back again is not the culprit since the decoder uses the same procedure successfully. In the end, the differences are insignificant in terms of the quality of the reproduced speech. A future endeavour would be to reproduce exactly the LP analysis and the LP coefficient to LSF conversion as done in the reference code. This should reduce, if not eliminate, differences between the Matlab and reference code for the coder results.
- The Matlab code has been written for enhanced modularity and not necessarily for fast computation. Nonetheless, the coder runs relatively fast. The elapsed time for coding a speech file of length t seconds is about $3.5t$ on a PC workstation (3 GHz). The elapsed time decoding the t seconds of speech is about $0.8t$, faster than real time.
- During the investigation of the ACB tables, it was noticed that some of the gain vectors do not have the largest coefficients near the middle of the vector. A multi-element pitch gain vector can serve to interpolate between integer lags, but also can supply a delay or advance relative to the pitch lag. Some of the gain vectors have the largest element at one end or the other. To wit, the 119th and 120th elements of the 170 element ACB gain vector table are

$$\begin{aligned} \mathbf{b}_{117}^T &= [1.0342 \quad 0.0999 \quad -0.0634 \quad 0.0139 \quad -0.0170], \\ \mathbf{b}_{120}^T &= [-0.0065 \quad -0.0368 \quad 0.0350 \quad -0.0887 \quad 0.9630]. \end{aligned} \tag{118}$$

These cases could be represented with a change in lag (by two samples) and a vector with the largest value in the middle. It is my suspicion, that the ACB gain codebook could be more effective if such changes were made.

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. ITU-T Recommendation G.723.1 Annex B, *Dual Rate Speech Coder for Multimedia Communications Transmitting at 5.3 and 6.3 kbit/s: Silence compression scheme*, Nov. 1996
3. ITU-T Recommendation G.723.1 Annex A, *Dual Rate Speech Coder for Multimedia Communications Transmitting at 5.3 and 6.3 kbit/s: Alternate specification based on floating point arithmetic*, Nov. 1996.
4. 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-mmsp.ECE.McGill.CA/Documents.
5. J. P. M. Schalkwijk, "An Algorithm for Source Coding", *IEEE Trans. Inform. Theory,* vol. 18, pp. 395–399, May 1972.
6. D. E. Knuth, *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms*, 3rd edition, Addison-Wesley, 1997.

Appendix A Sine Detector

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

$$S(z) = \frac{1}{1 - 2\cos\omega_o z^{-1} + z^{-2}}. \tag{119}$$

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

$$H(z) = 1 - 2\cos\omega_o z^{-1} + z^{-2}. \tag{120}$$

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

$$s[n] = 2\cos\omega_o s[n-1] - s[n-2], \tag{121}$$

where $s[n] = \cos(\omega_o n + \phi)$. This relation can also 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 values near unity 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 [5].

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

$$N_{\min} = \log_2 \binom{N}{K} \tag{122}$$

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. 15. 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.

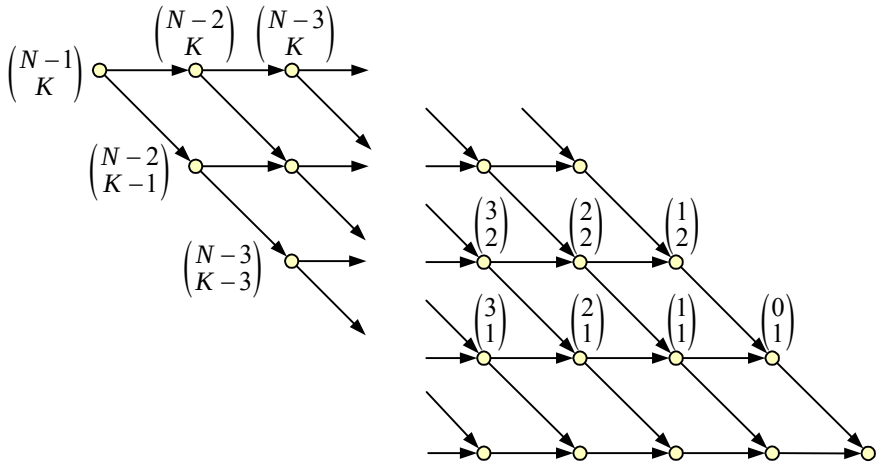

Fig. 15 Combinatorial coding trellis.

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

$$\binom{n}{m} > \binom{n-k}{m} \quad \text{for } k \geq 1. \tag{123}$$

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,

$$\begin{aligned} i_{\max} &= \sum_{k=1}^{K} \binom{N-k}{K-k+1} \\ &= \binom{N}{K} - 1. \end{aligned} \tag{124}$$

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

$$i_{\min} = 0. \tag{125}$$

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. (124).

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.