4. SCALAR LOSSY COMPRESSION METHODS

Uniform Quantization and Coding

General idea: Code highly probable symbols into short binary sequences without regard to their statistical, temporal, or spatial behavior. They are also known as scalar quantizer with dimension $k = 1$.

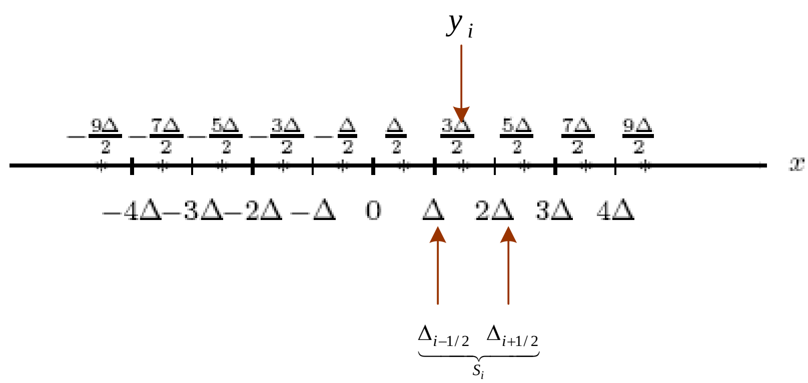
- Any real number x can be rounded off to the nearest integer say:

$$q(x) = round(x) \qquad (4.1)$$

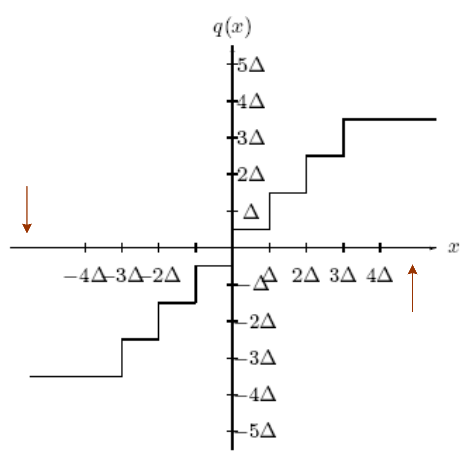
- We map the real-line $\Re$ (continuous space) into a discrete space in terms of "cells" S_i (bins, regions, code boundaries, etc.)

$$S_i = (i - \frac{1}{2}, i + \frac{1}{2}] \quad \text{and} \quad y_i = i = \hat{x}_i \quad \text{for all integers } i. \qquad (4.2)$$

- If the output levels are spaced equally then the quantizer is a uniform quantizer and it is the simplest one and realized by A/D converters. This is commonly used in all data acquisition tasks.



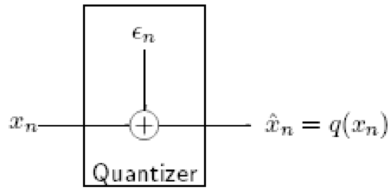
If input is	then output is
$[3\Delta, \infty)$	$7\frac{\Delta}{2}$
$[2\Delta, 3\Delta)$	$5\frac{\Delta}{2}$
$[\Delta, 2\Delta)$	$3\frac{\Delta}{2}$
$[0, \Delta)$	$\frac{\Delta}{2}$
$[-\Delta, 0)$	$-\frac{\Delta}{2}$
$(-2\Delta, -\Delta)$	$-3\frac{\Delta}{2}$
$(-3\Delta, -2\Delta)$	$-5\frac{\Delta}{2}$
$(-\infty, -3\Delta)$	$-7\frac{\Delta}{2}$



- Resulting quantizing error (noise, distortion):

$$\varepsilon = q(x) - x \quad or \quad q(x) = x + \varepsilon \qquad (4.3)$$

- This description implies the famous additive noise channel model:



- Quantizing distortion $\varepsilon \Rightarrow 0$ is desired.
- Unlike Sampling or A/D, quantization operation is lossy.

MSE Distortion and SNR Computation for a uniform quantizer:

Assume that our signal is uniformly distributed in the interval $[-A/2,+A/2]$ and there are $N = 2^R$ cells, resulting in cells with a width: $\Delta = A/N$ and the output values are in middle of cells. It has been shown in the literature that the **MSE** is given by

$$D(q) = \sum_{i=0}^{N-1} \int_{-\frac{A}{2}+i\Delta}^{-\frac{A}{2}+(i+1)\Delta} |\, x - q(x)\,|^2 \; dx = \frac{1}{A}\sum_{i=0}^{N-1} \int_{-\Delta/2}^{\Delta/2} |\, y\,|^2 \; dy = \frac{1}{N\Delta}.N.\frac{\Delta^3}{12} = \frac{\Delta^2}{12} \qquad (4.4)$$

which is known in the business as 1/12 law or Bennett's uniform quantizer distortion law. It has been also shown that (5.4) holds true for any signal distribution if the number of quantization levels is very large.

Entropy for this uniform distribution is simply:

$$H(q) = \log N \qquad (4.5)$$

For a uniform quantizer with N levels, we need an encoding rate of: $R = \log_2(N)$ bits/symbol. For our signal with range A and a step-size: $\Delta = A/N$ then (4.4) yields a signal-to-distortion (noise) ratio (in decibels):

$$SNR_{dB} = 10.\log_{10} \frac{\sigma_X^2}{E\{(q(X) - X)^2\}} \approx c + 20.R.\log_{10} 2 \approx c + 6.R \quad dB \qquad (4.6)$$

which is known as the "6 dB per bit rule", where SNR for uniform quantization increases 6 dB for each one bit increase on rate R. Here $E\{o\}$ corresponds to the averaging or expectation, σ_X^2 stands for the variance (average power) of the signal. c is a signal dependent loading factor, in telecom industry, there is a rule of 4-sigma loading, where $x_{\max} = 4.\sigma_x$ which results at:

$$SNR_{dB} = 10.\log_{10}\left(\frac{\sigma_X^2 .3.2^{2R}}{x_{\max}^2}\right) = 6.02R - 7.27\; dB \qquad (4.7)$$

It is very important to note that (4.7) is true for only 4-sigma loading case but (4.6) is true for all uniform quantizers:

PSNR Computation for Image Coder:

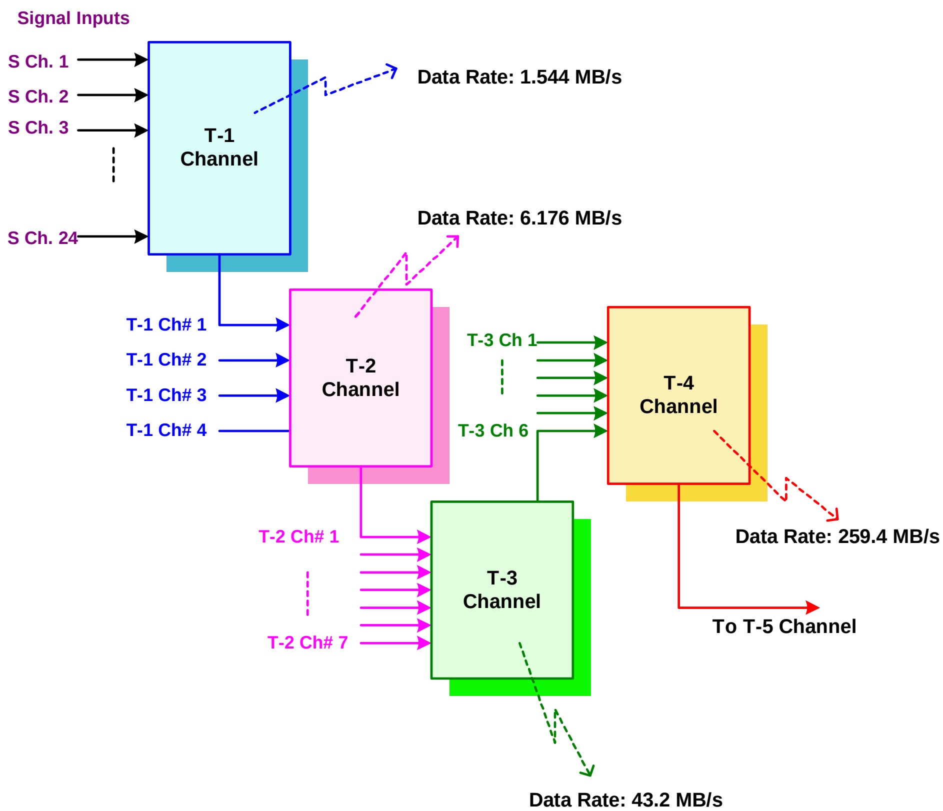
$$PSNR_{dB} = 10.\log_{10}\left(\frac{peak^2}{MSE}\right) \quad dB \qquad (4.8)$$

In many gray scale (monochrome) imaging applications $peak = 256$, 8-bit per pixel per color is used.

Example 4.1: Using VcDemo explore uniform quantization of imagery with and without channel errors.

Pulse Code Modulation (PCM) Infrastructure :

PCM systems constitute the backbone of the existing public telecommunication hierarchy throughout the world. There are basically two basic infrastructures: the North American and Japanese networks based on an aggregate transmission rates at integer multiples of 1.544 MB/s over T-1 lines and the CCITT networks based on integer multiples of 2.048 MB/s E-1 lines. This configuration is commonly known as the "*plain old telephone service (POTS*" in the telecommunication industry.



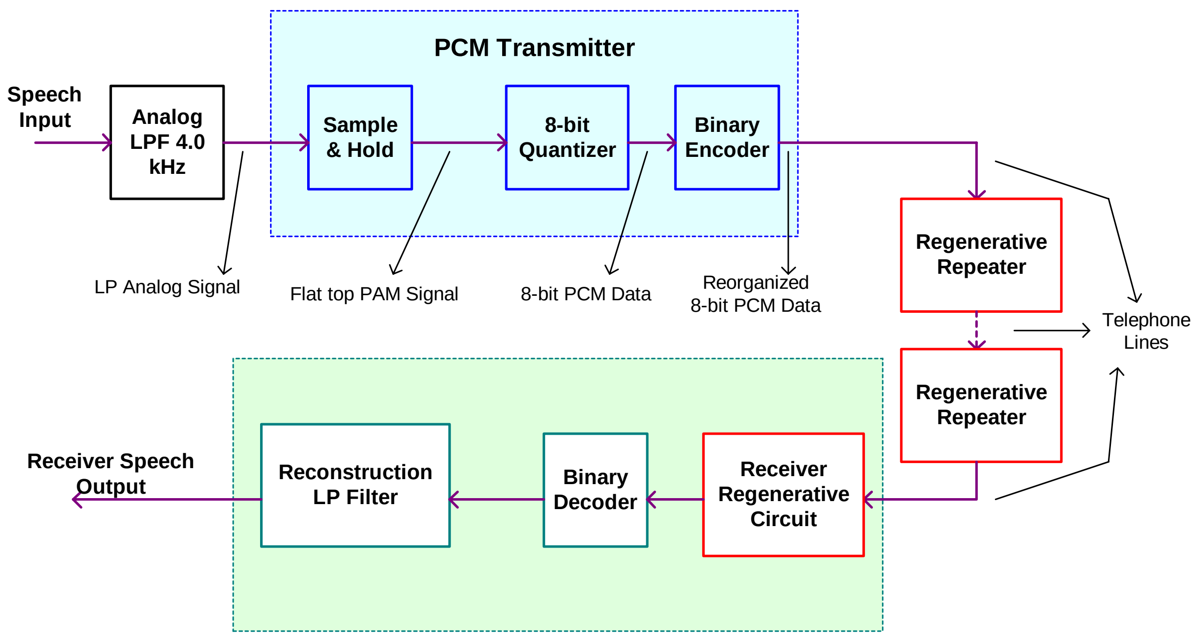
North American and Japanese TDM hierarchy for digital plain old telephone service (POTS).

1. First stage in this architecture is the T-1 Carrier System, where a set of 24 voice-grade signals sampled at 8,000 samples per second.
2. Sampling resolution of 8-bits per sample.
3. This corresponds to a data rate of 64,000 b/s is time-multiplexed.
4. Overall bandwidth and the associated bit rate is 1.544 MB/s, where 1.536 MB/s is for data from 24 voice channels and the remaining 8.0 kb/s is for framing and synchronization.
5. T-1 information is transmitted over nominally 22 - 25 gauge copper wire pair and it is mostly used in the terrestrial networks.
6. The international networks, however, has a similar infrastructure where the building block is the 32-Channel E-1 Carrier system.
7. Here 30 voice grade channels are TDM multiplexed together with two control, protocol, and synchronization channels.
8. Each with a bit rate of 64 kb/s results in an overall bit rate of 2.048 MB/s and the bandwidth requirement is appropriately increased to 2.048 MHz.

At present, the TDM systems have been generally replaced by Time-Division Multiple Access (TDMA), especially, in communication satellites and cellular communication systems. Since not every channel is always “on-line” the channel capabilities are fully utilized. In order to keep the overall system almost always “FULL,” neat procedures are developed to accept data from higher than 24 (or 30 in CCITT systems) according to some statistics-based switching operations. This is done at the expense of adding a buffer to the system. Buffer size plays a key role since a large buffer could be unacceptably costly, whereas, a very small one can easily overflow. Independent of the size, when a demand is not met due to overflow, it is not a good business practice in these days of stiff competition and dropping service charges.

End-to-End Single Channel PCM System: If we take only one of these voice grade input signals and go through the complete process of communicating over a PCM system, this is called an end-to-end single channel PCM configuration in the engineering jargon. A general block diagram for a voice grade speech communication over telephone lines is shown below.

1. First step is to bandlimit the analog signal with a low-pass filter of B=4,000 Hz.
2. This filter is an analog anti-aliasing filter with a cut-off frequency of 3300 Hz and the speech is suppressed to a minimum of 35 dB at 4.0 kHz.
3. Next step is to obtain samples of this baseband signal at 8,000 samples/s.



End-to-end single channel PCM block diagram.

Non-Uniform Quantizers: The uniform quantizer used in the previous section has exactly the same step-size for each level and it is used in applications where data compression is not the primary motive. In most public and secure communication systems, we deal with signals, which exhibit non-uniform amplitude distribution. Particularly, smaller amplitudes dominate the transmitted sample sequence in traditional communication, whereas large amplitudes are very rare. In other words, quantizer steps at the center of the staircase are heavily used and the tail levels are very infrequent. To save bandwidth one idea is to truncate these large values since they are very infrequent. However, we pay the price of loosing details in video or richness in sound. Instead, we attempt to exploit this infrequency for the purposes of improved SNR or reduced bit rate by assigning non-uniform step sizes. This is due to the fact that there will be more steps in the center and the range of the error signal will be smaller. This is demonstrated in the figure below.

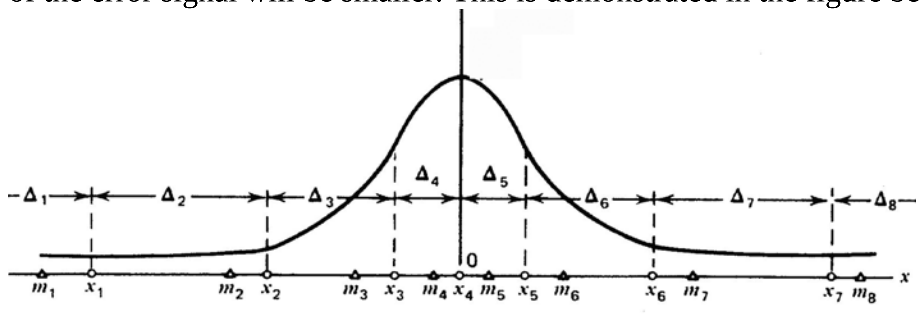


Illustration of a 3-bit non-uniform quantization.

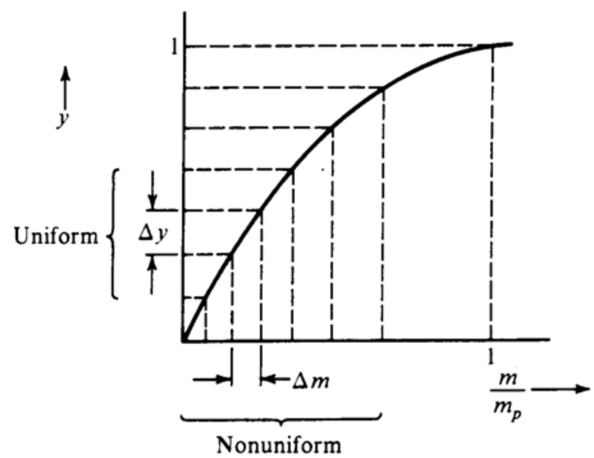
- Step sizes are variable.
- They are more closely spaced for samples which are frequent.
- Very infrequent samples are bundled together or even some samples neglected totally.

There are three well known classes of quantizers use this approach:

(1) Lloyd (II)-Max non-linear quantizers, where each step-size optimized according to the underlying statistical distribution. For instance, an exponential distribution is assumed for the intensity levels of pixels in an image frame.

(2) Generalized Lloyd I type quantizers, where the levels are matched to the statistics of large training databases. These quantizers are usually multi-dimensional and they form the basis for Vector Quantization (VQ) in modern communication systems.

(3) Obtain optimal quantizer levels is to pass the signal through a companding network, whose output is a uniformly distributed signal as shown below.



A Compandor for mapping non-uniform input levels to uniform ones. (Reprint from Lathi's text.)

As it is clear from above, the input signal falling into regions with non-uniform lengths Δm, which are increasing as the amplitude increases, are mapped into uniform regions with range Δy. The system, which does this type of transformation, is called a *compandor*. It should be expected that the precisely the opposite procedure will need to be performed in the receiver by an *expandor*.

This approach has been the norm in the existing telecommunication infrastructure. According to long-term studies on telephone speech samples, it observed that speech samples exhibit roughly an exponential distribution, where samples with small amplitude occur exponentially more often than the larger ones. To exploit this exponential character, input speech is processed by a logarithmic network and its outputs are expected to be significantly more uniform. Next, we pass these logarithmically compressed signals by a uniform quantizer of the previous section. In the receiver, however, samples are expanded by an exponential network to cancel out the compression. This process is called *Companding*. There are two logarithmic laws used for companding voice/audio grade signals, namely, the *A-Law* for the international circuits and the $\mu-Law$ for the North American and Japanese systems. Input-output characteristics of these two laws are shown below.

Case 1: CCITT Law with A=87.6: If the input signal is $x(t)$ with a peak amplitude level m_P the output signal is given by:

$$y(t)=\begin{cases} \left(\dfrac{A}{1+\log_e A}\right)\dfrac{x(t)}{m_P} & if \quad |x(t)/m_P| \le 1/A \\ Sgn(x(t)).\left\{\dfrac{1+\log_e(A|x(t)/m_P|}{1+\log_e A}\right\} & if \quad 1/A \le |x(t)/m_P| \le 1 \end{cases} \tag{4.9A}$$

Case 2: $\mu-Law$ **with** $\mu=255$:

$$y(t)=Sgn(x(t)).\frac{log_e[1+\mu.|x(t)/m_P|]}{log_e(1+\mu)} \qquad if \quad |x(t)/m_P| \le 1 \tag{4.9B}$$

In either case, the signal-to-quantizing distortion ratio: S_0/N_q is nearly constant over most voice-grade input signal with a power range of 40 dB. For instance, the output SDR for the $\mu-Law$ can be approximated by:

$$\frac{S_0}{N_q} \approx \frac{3L^2}{[\log_e(1+\mu)]^2} \qquad if \quad \mu^2 > \frac{m_P^2}{x^2(t)} \tag{4.10}$$

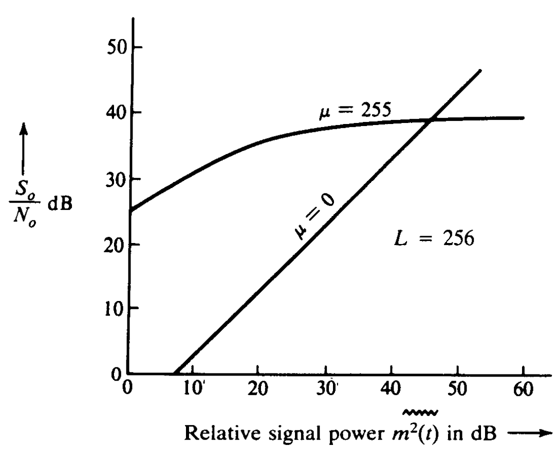
where L is the number of quantizer levels and $\mu=255$ is uniformly used in practice. Similar to non-compressed case, this result can be rewritten by:

$$\frac{S_0}{N_q} \approx 3k.2^{2n} \qquad where \quad k=\frac{1}{[\log_e(1+\mu)]^2}$$

or equivalently,

$$SNR_{dB}=(S_0/N_q)_{|dB} \approx 6n+\alpha \qquad where \quad \alpha=10\log_{10}(3k) \tag{4.11}$$

The plot of this curve for $\mu=255$ together with that of a PCM system without companding, i.e., $\mu=0$ is shown below.



Performance of a companded and uniform quantizers. (Reprint from Lathi's text, courtesy of Oxford Press)

SNR for Logarithmic Quantizers

A-Law:

Case 1: $|x| \to 0$; small input

$$SNR_A = SNR_{uniform} + 20\log_{10} G_c$$

where $G_c = \dfrac{A}{1+\ln A} = 16$

Case 2: $|x|/x_{\max} \to 1$;large input

$$SNR_A = 6.02R + 4.77 - 20\log_{10}(1+\ln A)$$
$$= 6.02R - 10 \quad dB$$

μ -Law:

Case 1: $\mu|x_{\max}| >> x_{\max}$

$$SNR_\mu = 6.02R + 4.77 - 20\log_{10}(\ln(1)+\mu) = 6.02R - 10.1 \quad \text{for } \mu = 255$$

Case 2: $|x| \to 0$; small input

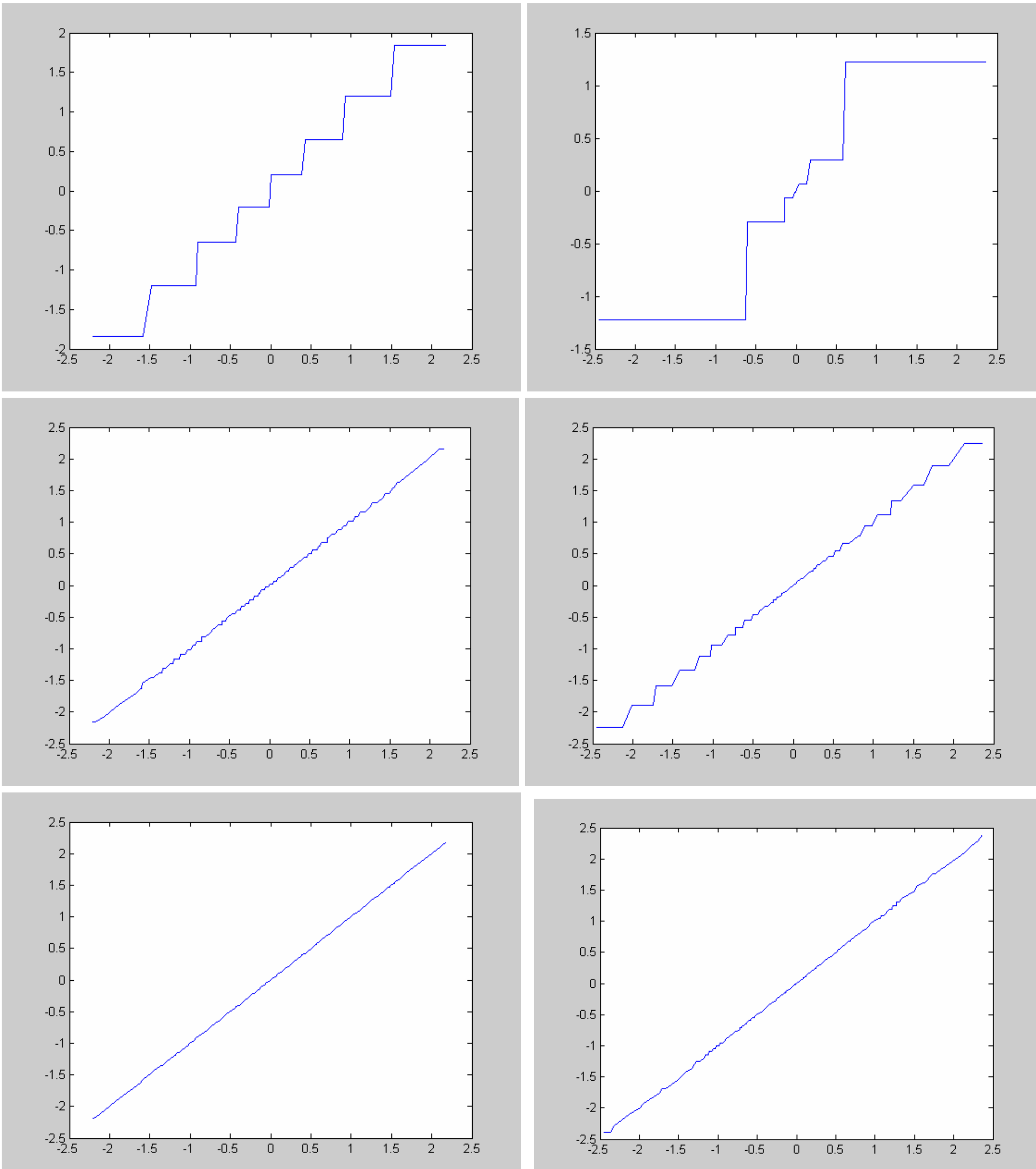
$$SNR_\mu = SNR_{uniform} + 20\log_{10} G_c$$

where $G_c = \dfrac{\mu}{\ln(1+\mu)}$

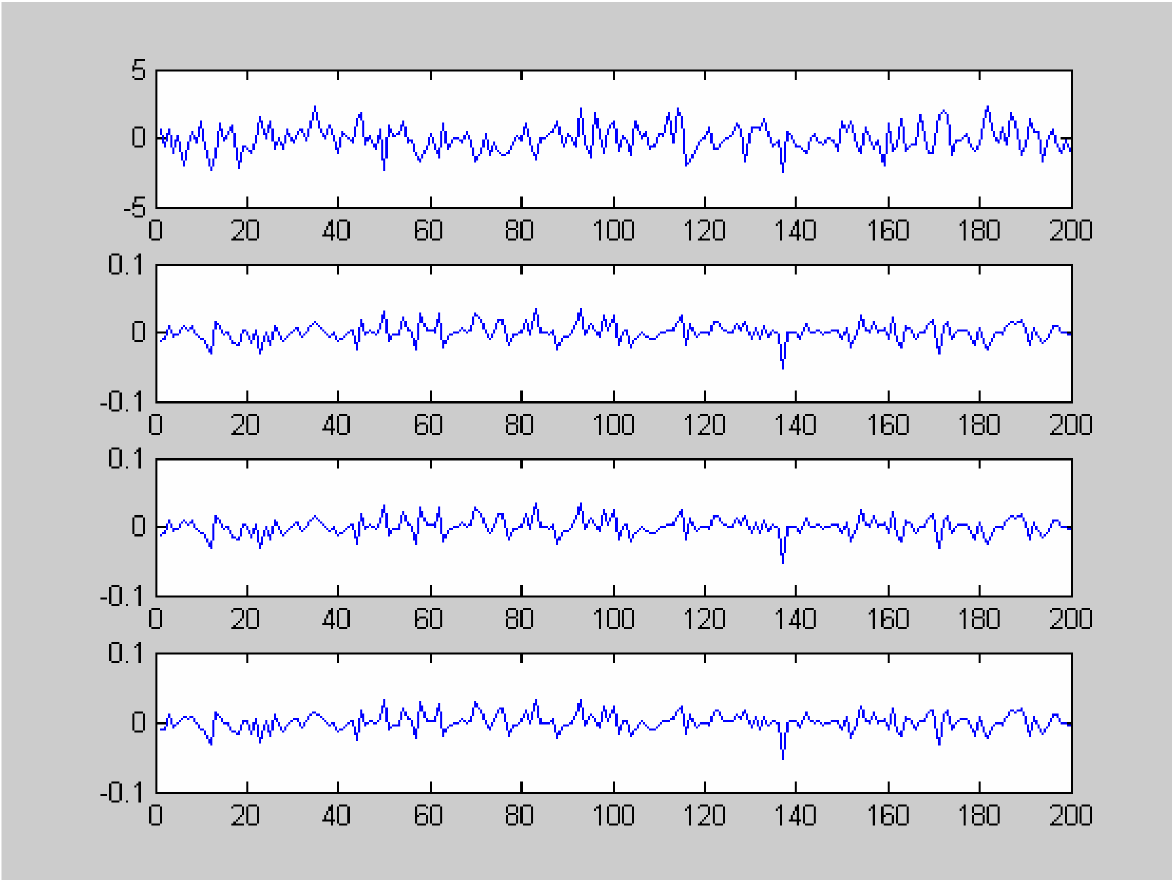
Example 4.2: Let us examine the characteristics of μ -Law quantization.

```
% Case 1: mhu=1 (almost 0=uniform PCM); N=8, 64, 256
t=1:200; a=randn(1,200); [sqnr81,a_quan,code]=mula_pcm(a,8,1);
[Y,I]=sort(a); figure;plot(Y,a_quan(I));
[sqnr641,a_quan,code]=mula_pcm(a,64,1);
[Y,I]=sort(a); figure;plot(Y,a_quan(I));
[sqnr256,a_quan,code]=mula_pcm(a,256,1);
[Y,I]=sort(a); figure;plot(Y,a_quan(I));

disp ('sqnr8_1='), disp(sqnr81); disp ('sqnr64_1='), disp(sqnr641);
disp ('sqnr256_1='), disp(sqnr256)

% Plots for quantizer levels, input & error signal
figure;
error81=a-a_quan; subplot(4,1,1); plot(t,a); subplot(4,1,2); plot(t,error81);
error641=a-a_quan; subplot(4,1,3); plot(t,error641);
error2561=a-a_quan; subplot(4,1,4); plot(t,error2561);

% Case 2: mhu=255 (Industry Standard); N=8,64,256
t=1:200; a=randn(1,200); [sqnr8255,a_quan,code]=mula_pcm(a,8,255);
[Y,I]=sort(a);  figure;plot(Y,a_quan(I));
[sqnr64255,a_quan,code]=mula_pcm(a,64,255);
[Y,I]=sort(a); figure;plot(Y,a_quan(I));
[sqnr256,a_quan,code]=mula_pcm(a,256,255);
[Y,I]=sort(a); figure;plot(Y,a_quan(I));

disp ('sqnr8_255='), disp(sqnr81); disp ('sqnr64_255='), disp(sqnr641)
disp ('sqnr256_255='), disp(sqnr256)
```

```
% Plots for quantizer levels, input & error signal
figure;
error8255=a-a_quan; subplot(4,1,1); plot(t,a); subplot(4,1,2); plot(t,error8255);
error64255=a-a_quan; subplot(4,1,3); plot(t,error64255);
error256255=a-a_quan; subplot(4,1,4); plot(t,error256255);
```



sqnr8_1= 15.5507; sqnr64_1=33.7532; sqnr256_1= 45.9716
sqnr8_255=15.5507; sqnr64_255=33.7532; sqnr256_255=37.7977



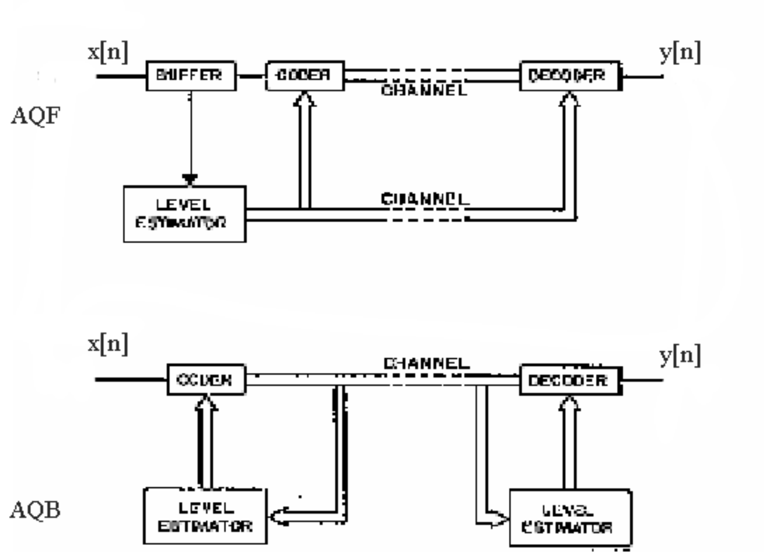
Example 4.3: Using a well-known website for http://www.its.bldrdoc.gov/audio/examples.php let us hear 16-bit Uniform PCM and 8-bit μ-Law 64,000 bits per second speech samples.

Adaptive Quantizers

Uniform and non-uniform quantizers with fixed quantization levels can overflow and underflow easily if the input signal levels change over time. In other words, a large number of higher index levels are not used at all for low amplitude signals. In the case of signals with large amplitude ranges, the lower indices are never transmitted.

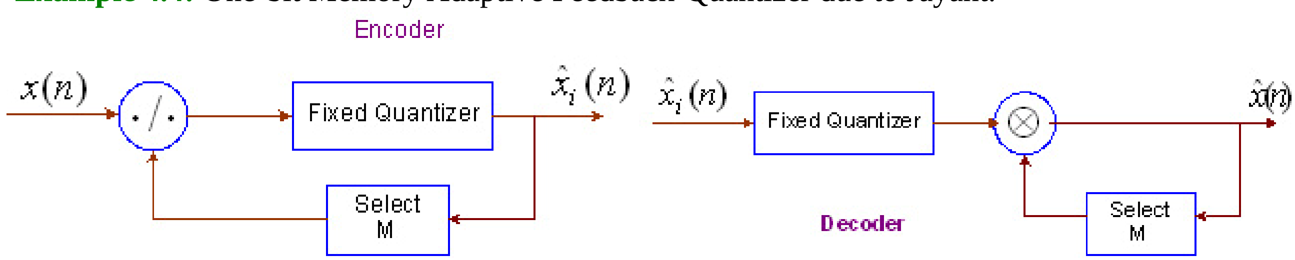
The remedy to this situation is to adapt the quantizer levels to the dynamics of the input signal. Adaptation could be in two ways:

1. **Backward adaptation:** Quantize the current signal; estimate the levels for the next iteration; quantize the new signal with the estimate from the previous iteration.
2. **Forward adaptation:** Buffer the signal; estimate the levels and quantize with a small delay.



Forward- and Backward-Adaptive Quantizers

Example 4.4: One-bit Memory Adaptive Feedback-Quantizer due to Jayant.



Step Size Multiplier for Jayant's one-bit Memory Feedback Adaptive Quantizer			
Adaptation (multiply or divide) Values			
Previous Output Levels	2.Bit	3.Bit	4.Bit
L1	0.60	0.85	0.8
L2	2.20	1.00	0.08
L3		1.00	0.80
L4		1.50	0.80
L5			1.20
L6			1.60
L7			2.00
L8			2.40

Performance of AQF by Jayant:

SNR (dB) Values for 3-bit speech quantization with Jayant adaptation		
Non-Uniform Quantizers	**Non-Adaptive**	**Adaptive**
μ-Law	9.5	X
Gaussian Optimized	7.3	15.0
Laplacian Optimized	9.9	13.3
Uniform Quantizers		
Gaussian Optimized	6.7	14.7
Laplacian Optimized	7.4	13.4

Differential Quantizers (DPCM and DM)

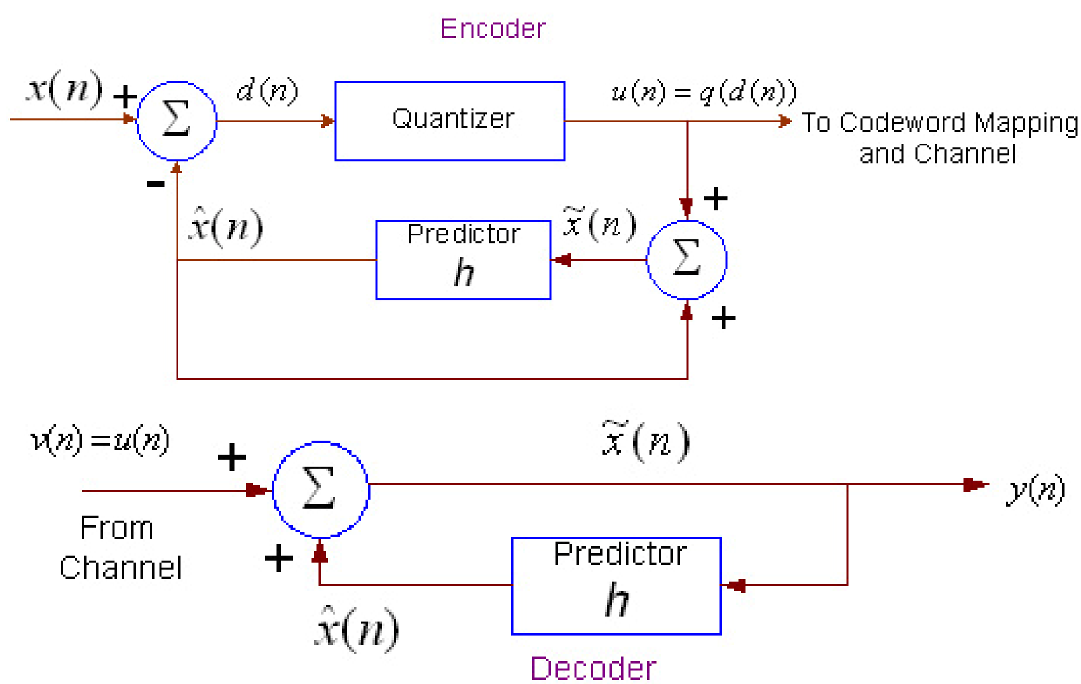
Digitized samples of signals occurring in nature, i.e, speech, imagery, radar, sonar, telemetry, and others, usually have strong correlation. In other words, subsequent samples of speech are highly correlated, so are the adjacent pixels of an image frame. This correlation implies redundancy, which can be reduced by encoding difference between subsequent samples or adjacent pixels. Quantizers of this class are called differential coders and they perform typically 6 dB better than their non-differential counterparts.

There are two fundamental system groups in this class: Differential Pulse Code Modulation (DPCM) due to Cutler and Delta Modulation (DM) developed deJager, van de Weg, Zetterberg, O'Neal and Abate. As in other coders, there are non-adaptive and adaptive versions of each.

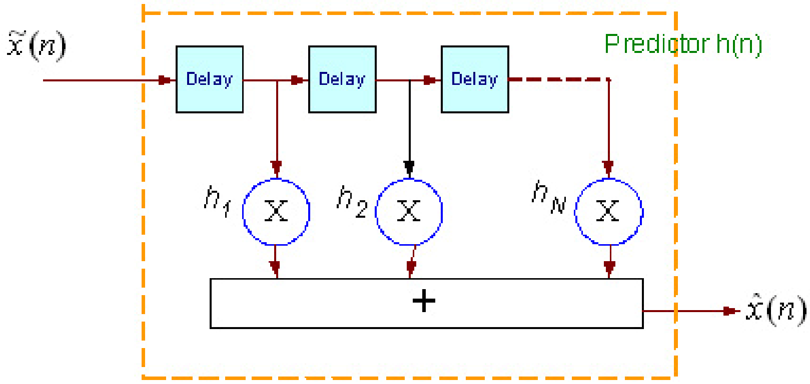
Key features of a DPCM system are:

- Quantizer. It is an n-bit encoder as in PCM.
- Linear Predictor predicts $\hat{x}(n)$ an estimate of the current sample x(n)

$$\hat{x}(n) = \sum_{j=1}^{N} h_j \cdot \tilde{x}(n-j) \tag{4.12}$$



- The predictor can be interpreted as a linear filter (finite impulse response, FIR) and represented by its impulse response in the discrete-time or digital frequency-domains, where N is the order of prediction and hB_i are weights of individual taps.



Note: Delay operation is simply delaying the current value of the signal for one clock cycle T:

$$\tilde{x}(n-1) = D(\tilde{x}(n)) \tag{4.13}$$

- Differentiator to find the difference between the input at time n and its estimate:
  $$d(n) = x(n) - \hat{x}(n) \tag{4.14}$$
- The decoder has a replica of the predictor and its output is simply
  $$\tilde{x}(n) = u(n) + \hat{x}(n) \tag{4.15}$$
- General N[th]–order predictor output (filter response) can be modeled as:
  $$\hat{x} = \sum_{i=1}^{N} a_i \tilde{x}_{n-i} = \sum_{i=1}^{N} h_i \tilde{x}_{n-i} \tag{4.16}$$

Optimum Predictors in DPCM:

Let us define the variance (power) of the difference sequence in (4.14) by:

$$\sigma_d^2 = E\{[x(n) - \hat{x}(n)]^2\} = E\{[x_n - \sum_{i=1}^{N} a_i \tilde{x}_{n-i}]^2\} \tag{4.17}$$

Optimum predictor values or the multipliers in the transversal filter model $P_{opt} = H_{opt} = \{h_1, h_2, \cdots, h_N\}$ can be found by minimizing this prediction variance σ_d^2 using the extremum point analysis on real data not including the quantizer.

$$\begin{aligned}
\frac{\delta\sigma_d^2}{\delta a_1} &= \frac{\delta}{\delta a_1} E\{[x_n - \sum_{i=1}^{N} a_i x_{n-i}]^2\} = -2E\{[x_n - \sum_{i=1}^{N} a_i x_{n-i}]x_{n-1}\} = 0 \\
\frac{\delta\sigma_d^2}{\delta a_2} &= \frac{\delta}{\delta a_2} E\{[x_n - \sum_{i=1}^{N} a_i x_{n-i}]^2\} = -2E\{[x_n - \sum_{i=1}^{N} a_i x_{n-i}]x_{n-2}\} = 0 \\
\vdots &= \vdots \\
\frac{\delta\sigma_d^2}{\delta a_N} &= \frac{\delta}{\delta a_N} E\{[x_n - \sum_{i=1}^{N} a_i x_{n-i}]^2\} = -2E\{[x_n - \sum_{i=1}^{N} a_i x_{n-i}]x_{n-N}\} = 0
\end{aligned} \tag{4.18}$$

After taking the expectations we can write these equations in terms of the autocorrelation function of x_n:

$$R_{xx}(k) = E\{x_n x_{n-k}\} \tag{4.20}$$

$$\begin{aligned}
\sum_{i=1}^{N} a_i R_{xx}(i-1) &= R_{xx}(1) \\
\sum_{i=1}^{N} a_i R_{xx}(i-2) &= R_{xx}(2) \\
\vdots \quad &= \quad \vdots \\
\sum_{i=1}^{N} a_i R_{xx}(i-N) &= R_{xx}(N)
\end{aligned} \tag{4.21}$$

or equivalently in matrix form:

$$RA = P \tag{4.22}$$

where

$$R = \begin{bmatrix} R_{xx}(0) & R_{xx}(1) & R_{xx}(2) & \cdots & R_{xx}(N-1) \\ R_{xx}(1) & R_{xx}(0) & R_{xx}(1) & \cdots & R_{xx}(N-2) \\ R_{xx}(2) & R_{xx}(1) & R_{xx}(0) & \cdots & R_{xx}(N-3) \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ R_{xx}(N-1) & R_{xx}(N-2) & R_{xx}(N-3) & \cdots & R_{xx}(0) \end{bmatrix} \quad A = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_N \end{bmatrix} \quad P = \begin{bmatrix} R_{xx}(1) \\ R_{xx}(2) \\ R_{xx}(3) \\ \vdots \\ R_{xx}(N) \end{bmatrix}$$

- These N-equations for optimum predictor or filter coefficients are called "Normal Equations," which are also known as Wiener-Hopf or Yule-Walker equations. They are solved by either a matrix inversion process or Levinson-Durbin type recursive algorithms or their special forms including LaRoux-Geugen integer algorithm are used in speech coding, especially in the framework of LPC based codecs of the mobile phone technology. (Solution will be discussed in Chapter 9.) Let us now compute the predictors for the scalar DPCM system at hand.

Special case for *N=1* (Single tap predictor):

$$h_{1,opt} = R_{xx}(1)/R_{xx}(0) \tag{4.23}$$

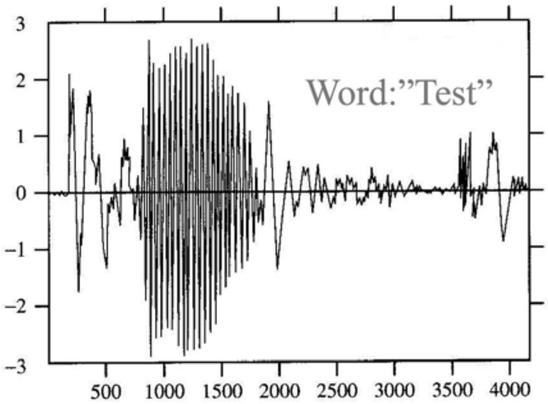
$$R_{xx}(0) = E\{x(n).x(n)\} \qquad \text{and} \qquad R_{xx}(1) = E\{x(n).x(n-1)\} \tag{4.24}$$

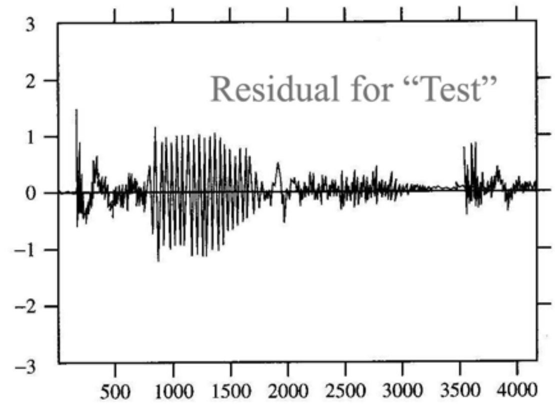
For speech and image compression single-tap prediction coefficient is in the range: (0.75 - .90)

The impact of the prediction on speech and imagery is also interpreted as “whitening” and de-correlating the signal as it can be seen from the figure below where the word “test” and its residual after being passed through a third order (N=3) predictor filter as implemented as a transversal filter of order 3.

Example 4.5:

Consider a segment of the word “Test” for about 0.5 seconds long shown below. The residual output from a third order predictor is also shown resulting in a significant reduction in dynamic range.





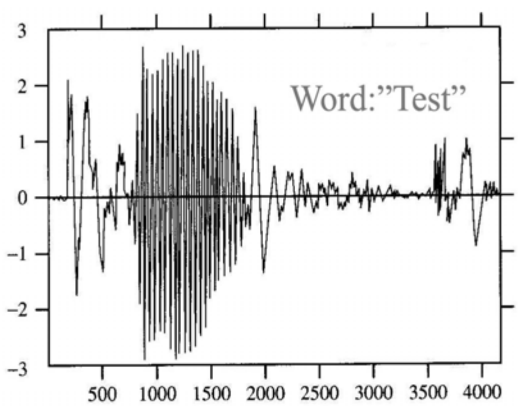
Quantitatively we measure the performance of prediction from the Singal-to-prediction-error ratio as defined by:

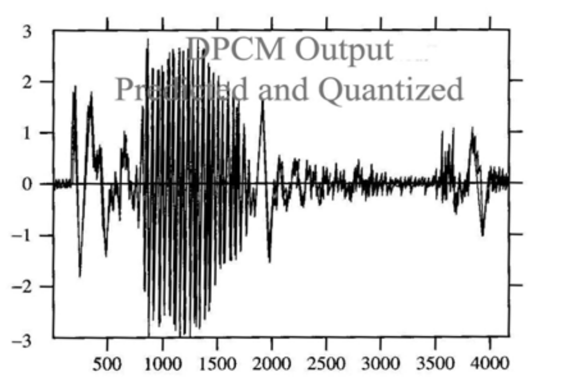
$$SPER_{dB} = \sum_{i=1}^{M} x_i^2 / \sum_{i=1}^{M} (x_i - \hat{x}_i)^2 \qquad (4.25)$$

Even simplest first order DPCM system is superior to PCM by a minimum of 6.0 dB. Higher order prediction yields better SNR. In the table below, we list the performance of DPCM system with different predictors and quantizers. Also we show the original signal and the reconstructed signal from a third order predictor and an 8-level (3-bits/symbol) quantizer.

Performance of DPCM system with different predictors and quantizers

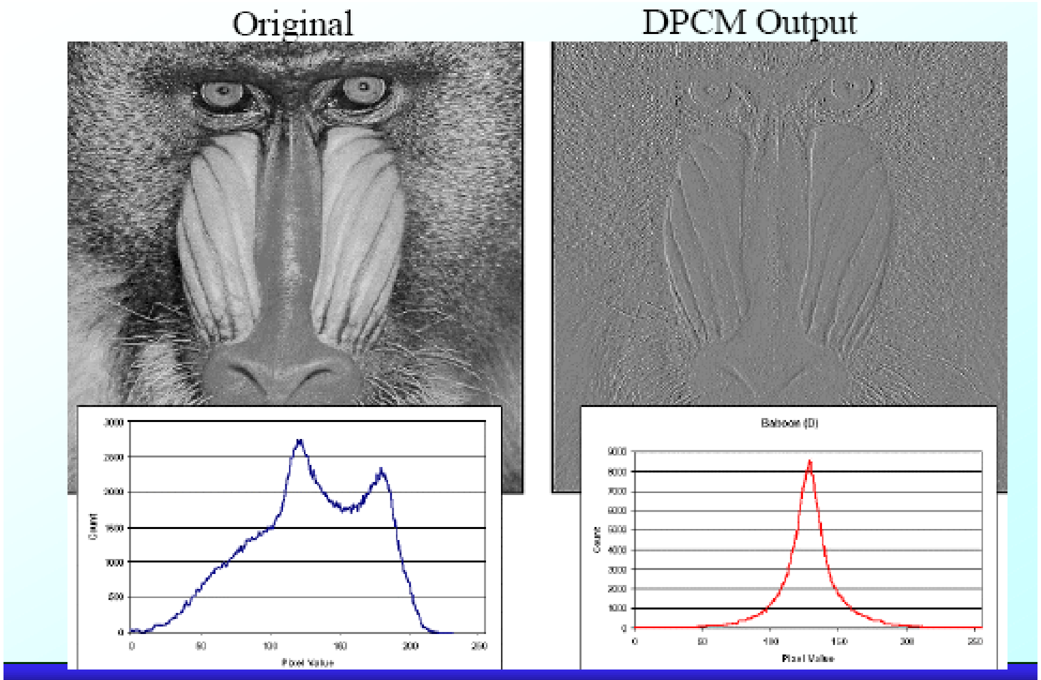
Quantizer	Predictor Order	SNR (dB)	SPER (dB)
Four-level	None	2.43	0
	1	3.37	2.65
	2	8.35	5.9
	3	8.74	6.1
Eight-level	None	3.65	0
	1	3.87	2.74
	2	9.81	6.37
	3	10.16	6.71

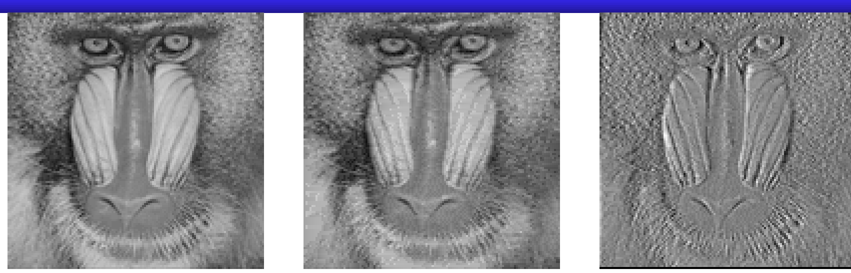




Performance of DPCM is even more pronounced in the case of image compression and more than 17.5 dB improvement has been reported in the literature.

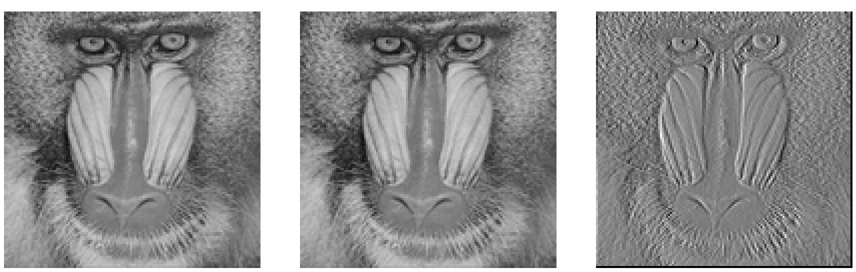
Example 4.6: Let us explore the performance of DPCM for a few different cases using VcDemo. The original image called "Mandrill" and the out of an 1-tap DPCM system is shown below, which clearly exhibits double-sided exponential distribution (Laplacian).





DPCM CODING RESULTS:

Variance of prediction error	: 351.7	Prediction gain	: 3.6 db
Coded bitrate	: 2.0 (bpp)	Est. entropy-coded bitrate	: 1.7 (bpp)
Mean square error	: 60.8		
Signal-to-noise ratio	: 13.1 (dB);	PSNR	: 30.3 (dB)



DPCM CODING RESULTS:

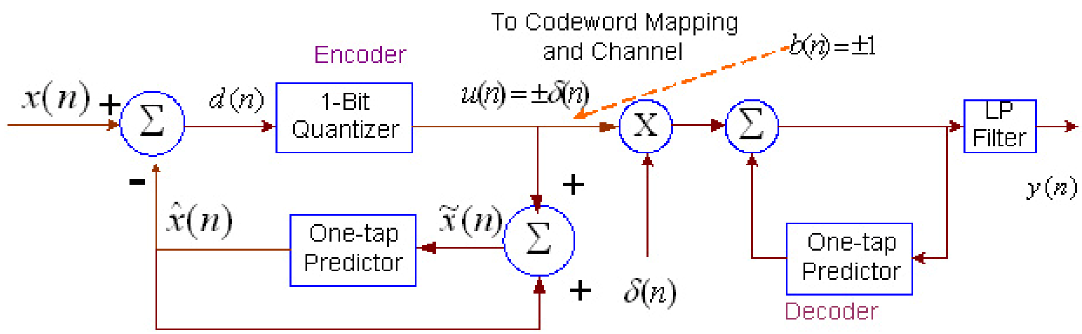
Variance of prediction error	: 351.7	Prediction gain	: 3.6 db
Coded bitrate	: 5.0 (bpp)	Est. entropy-coded bitrate	: 4.1 (bpp)
Mean square error	: 1.7		
Signal-to-noise ratio	: 28.8 (dB);	PSNR	: 45.9 (dB)

There are two issues with DPCM systems:

1. **Delay of N-units** since the sample at time n needs to wait an estimation process of order N to finish.
2. **Error propagation**. If there is a single bit error in the channel, this error will result in error for all subsequent signals due to the feedback loop. There are ways to control this propagation either by means of a resetting procedure or via exponentially decaying memory content. Below is an example for effects of bit errors in image compression.

Delta Modulation (DM)

- Very simple
- ONE-BIT Encoder
- ONE-TAP Feedback (predictor) differential coding system with an important difference:
- Sampling rate of DM is a several times higher than the Nyquist rate to compensate it is simplicity.
- The oversampling factor is usually 4-6 times the Nyquist rate.
- Only one bit differences are transmitted as shown in the following block diagram.
- The channel symbols are received simply as $b(n) = \pm 1$.
- Feedback predictor is identical to the encoder side.



- Since there is only a one-bit quantizer the coded signal may not be able to follow the input if it is rapidly changing. This is called "Slope Overload Noise" in the DM jargon. It is very critical since the coder may not be able to follow the input at all.
- If the signal is very slowly varying from sample-to-sample, then quantizer makes the same amount of error as if it were changing rapidly. This is called "Granular Noise" and it is equally detrimental.

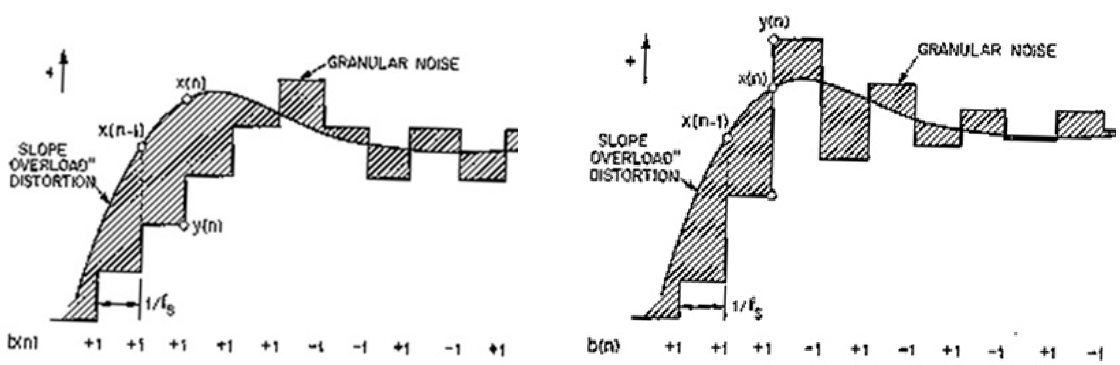
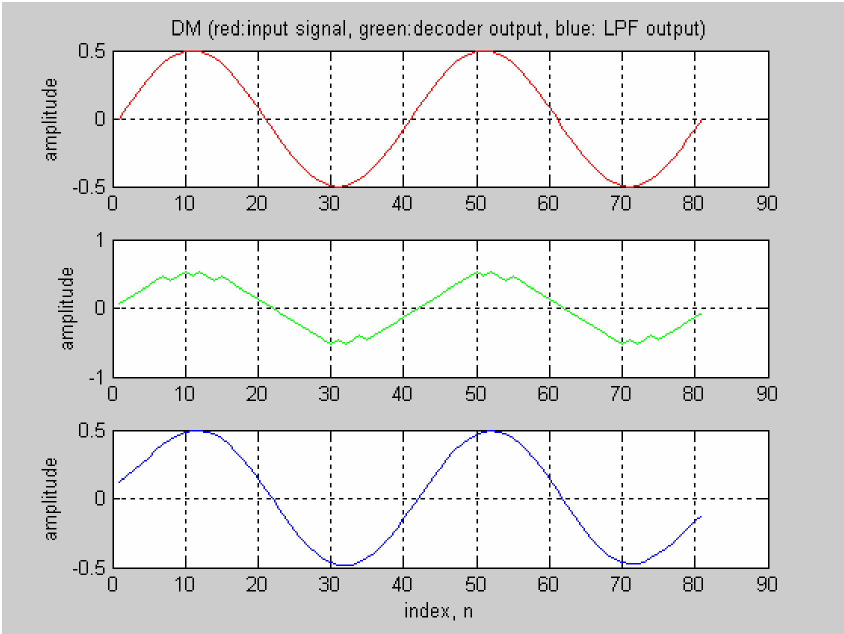
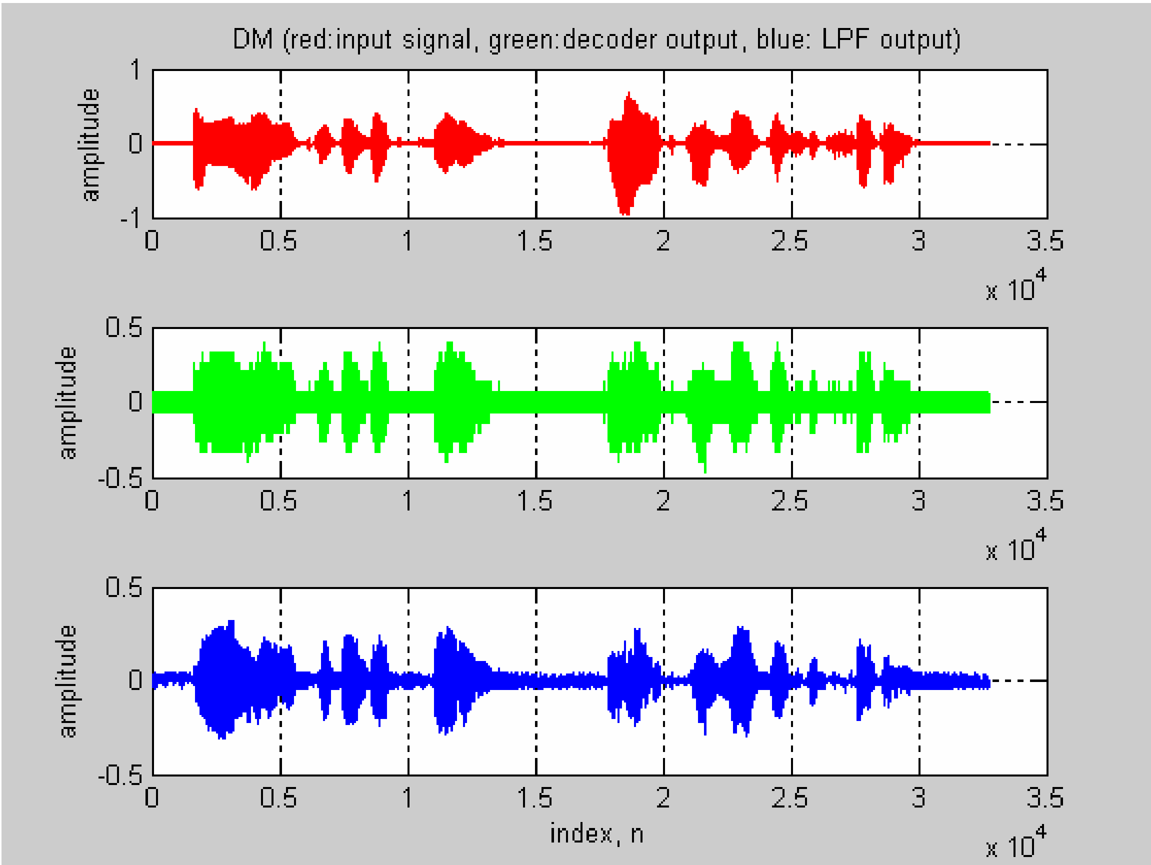


Illustration of Granular and Slope overload Noise in Linear DM and Adaptive DM.

Remedy: Variable step-size or adaptation of the quantizer according to some pre-defined adaptation logic. There are many examples to that. Performance of adaptive, linear or non-linear DM systems is very much dependent on the sampling rate factor.

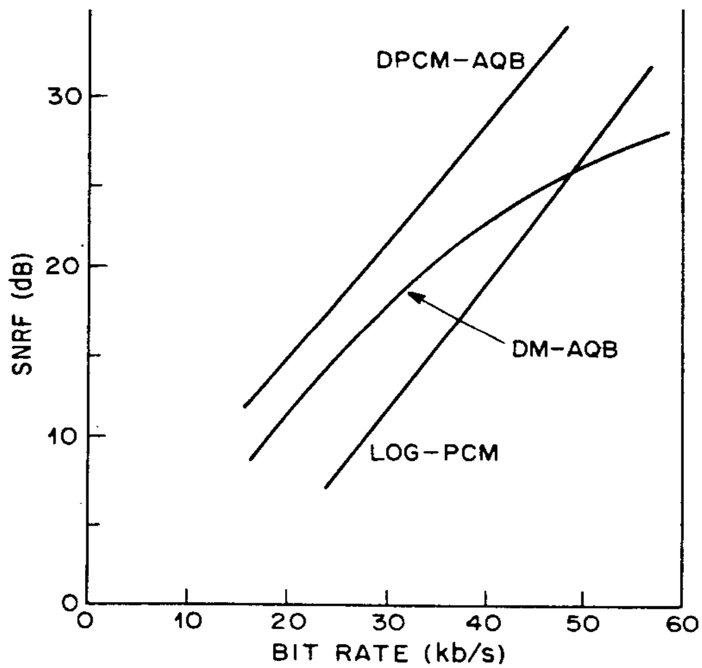
Example 4.7: Let us explore the performance of DM for sinusoidal signal and speech sample using a simple Delta Modulation Demo package, which can be found in the web.





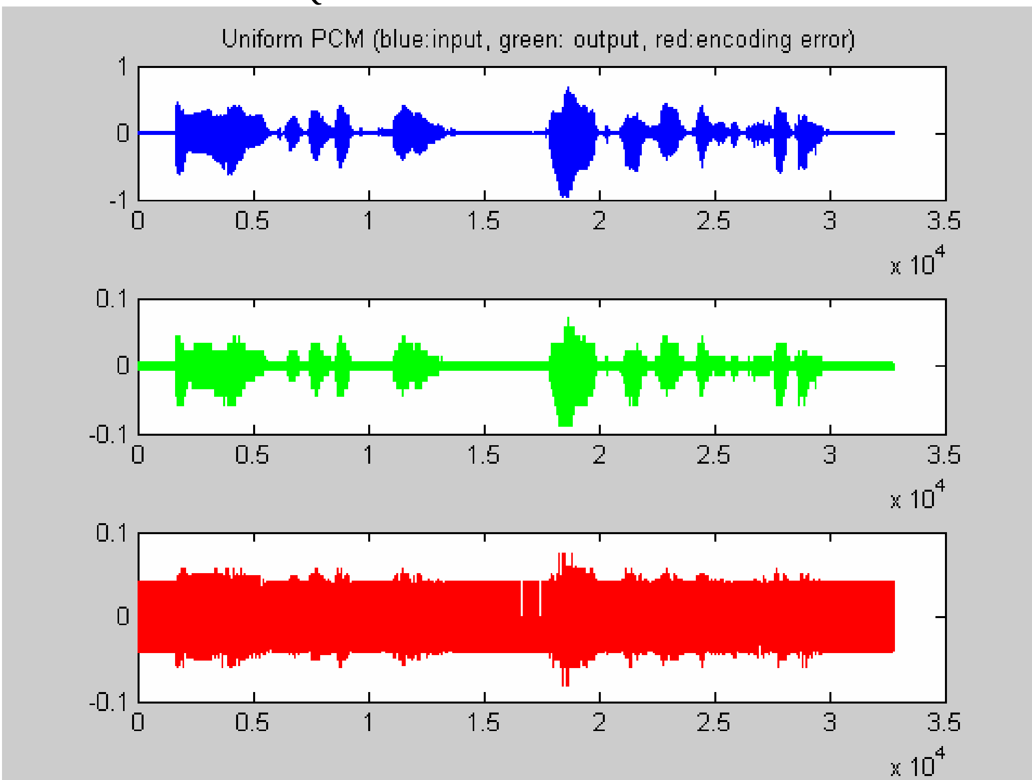
Performance Comparison of Advanced Pulse Modulation Systems

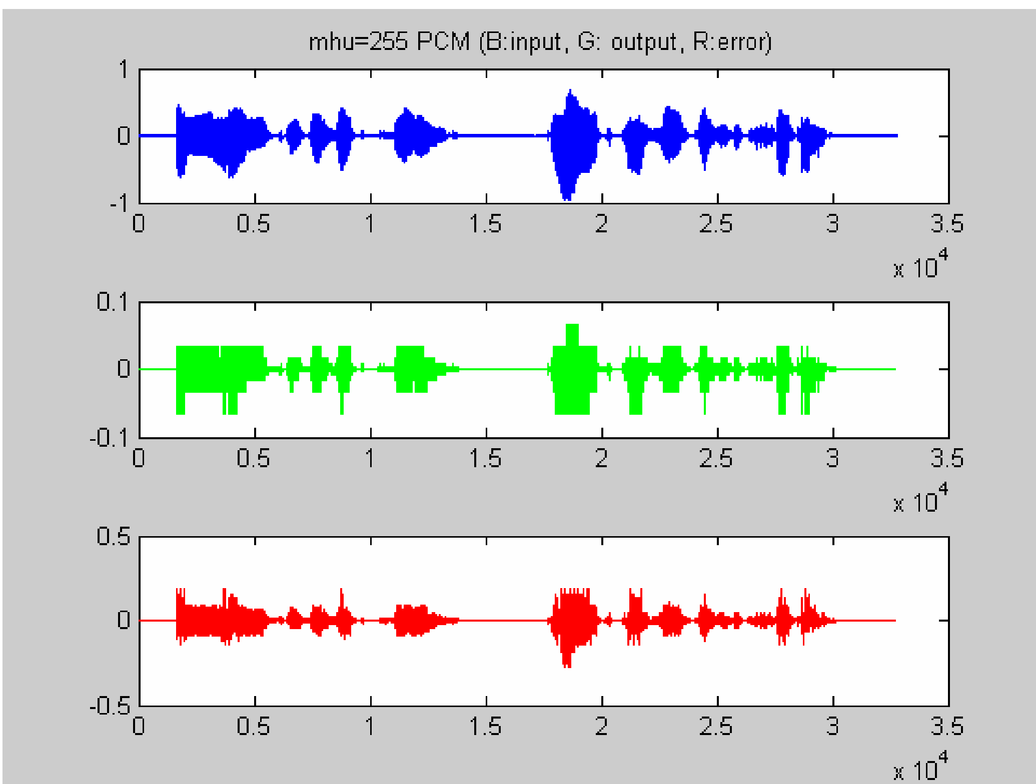
Finally, we illustrate below a comparative plot of the three systems used in speech waveforms and in pixel-domain image compression. Highest performance for comparable rate is obtained in the DPCM system with a feedback adaptation for its quantizer step-size. The next one is an Adaptive Delta modulation with a one-bit memory, and the final system is a logarithmically companded PCM. It is worth noting that the DM and LOG-PCM curves have a crossover point, which is around 48 kbits/s. This implies that a simple logarithmic PCM will do a better job than the equivalent DM and no need to worry about the feedback mechanism and the propagation of bit errors in the latter one.



Comparative performance curves for PCM, DPCM and DM systems. (Reprint with permission from *Modern Analog and Digital Communication Systems*, Third Edition, B. Lathi, courtesy of Oxford Press)

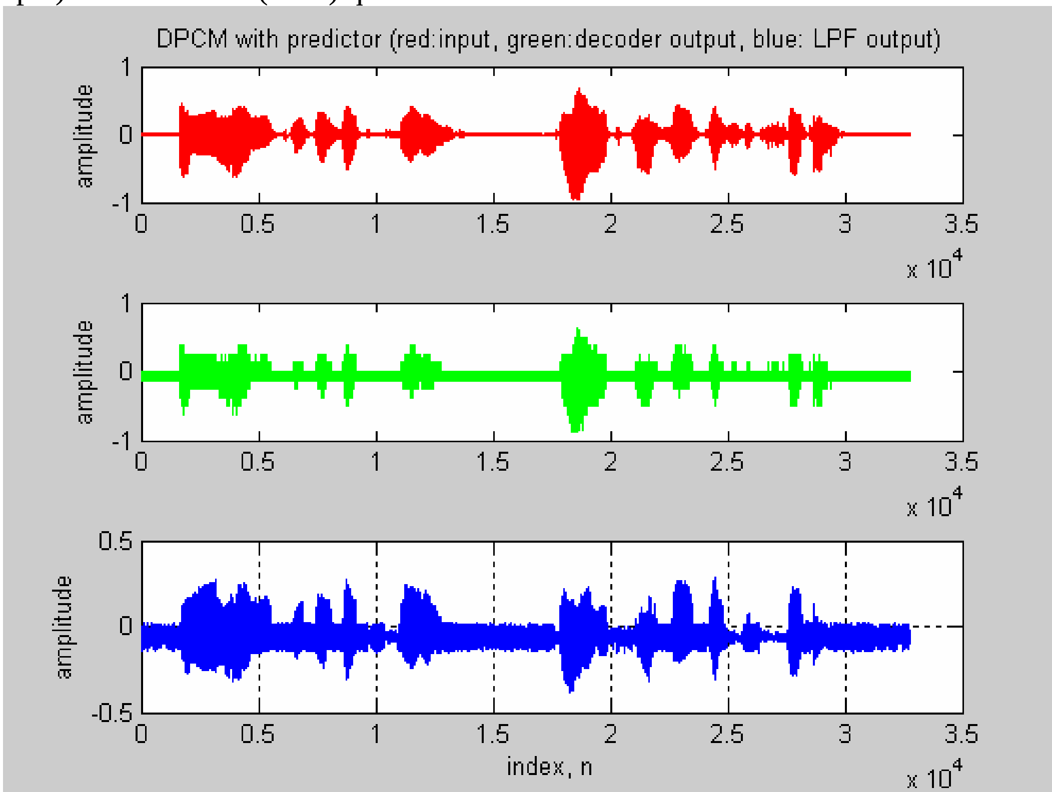
APPENDIX: SNR performance of uniform and μ-Law (255) PCM for speech. The figures below are for 16-level (bits/sample) and 256-level (8-bit) quantizers. Speech Compression based on Uniform and mhu-law PCM Quantizers

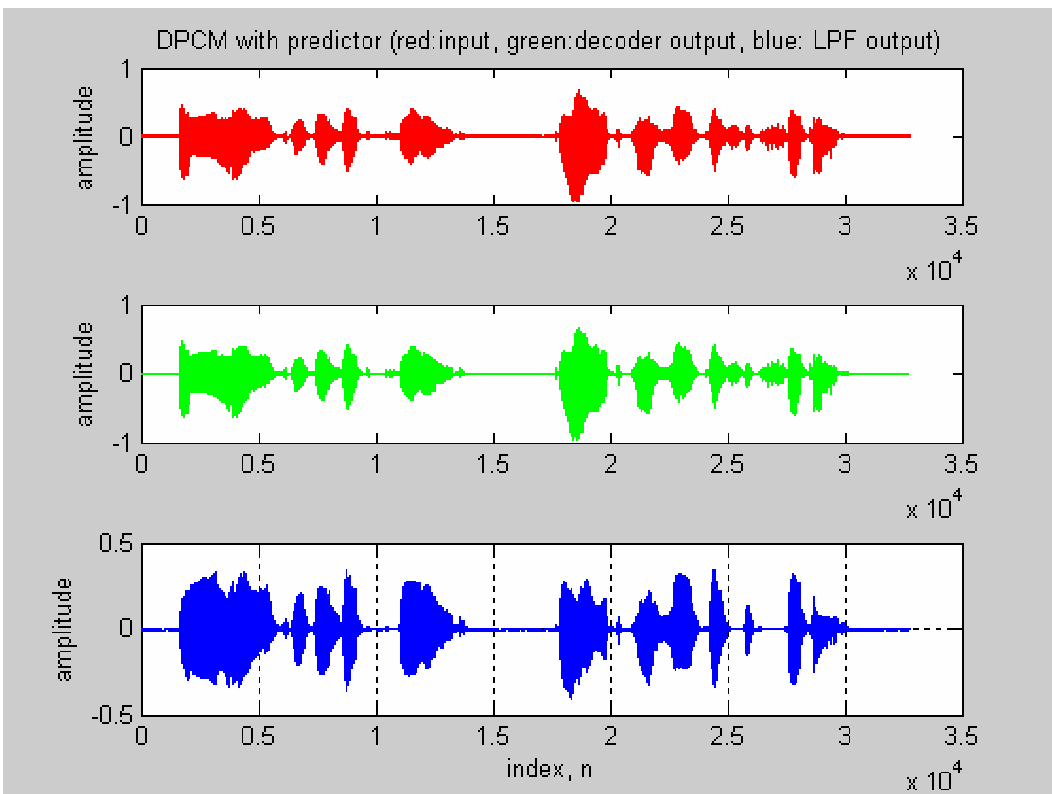




```
% Uniform and mhu-law PCM compression of speech
% Written by  H. Abut: March 2006
% Case 1: mhu=1 (almost 0=uniform PCM); N=8,...,256

% Input speech file
[s,fs]=auread('bye441');                    %read .au file
% [s,fs]=wavread('bye440');                 %read .wav file

t=1:size(s);
N=input('Enter Quantizer size in levels, \n');
pause; sound(s,fs); pause;
[sqnr,a_quan,code]=mula_pcm(s,N,1);
disp ('Quantizer Size='), disp(N), disp ('sqnr_1='), disp(sqnr)

% Plots for quantizer, input, output & error signal
error=s-a_quan;
% Lowpass filtering to smooth the output
Sa=lpf(100, .1, a_quan);

% Plots
subplot(3,1,1), plot(t,s,'b'), title('Uniform PCM (blue:input, green: output, red:encoding error)');
subplot(3,1,2), plot(t,Sa,'g'); subplot(3,1,3), plot(t,error,'r');
pause; Sa= 20 .* Sa; Sound(Sa,fs);

% Case 2: mhu=255 (Industry Standard); N=8,...,256
[sqnr,a_quan,code]=mula_pcm(s,N,255);
disp ('Quantizer Size='), disp(N), disp ('sqnr_255='), disp(sqnr)
pause;

% Plots for quantizer, input, output & error signal
error=s-a_quan;
% Lowpass filtering to smooth the output
Sa=lpf(100, .1, a_quan);
figure,
subplot(3,1,1), plot(t,s,'b'), title('mhu=255 PCM (B:input, G: output, R:error)');
subplot(3,1,2), plot(t,Sa,'g'); subplot(3,1,3), plot(t,error,'r');

% Speech output is scaled up (8 times) due to low-level output from PC audio cards.
Sa= 8 .* Sa; Sound(Sa,fs);
```

APPENDIX: SNR performance of a DPCM system for speech. The figures below are for 16-level (bits/sample) and 256-level (8-bit) quantizers.





MATLAB CODE for Speech Compression Using DPCM:

```
% Speech compression using DPCM
close all; clear
% signal sampling
fs=1/8000; tn=0:fs:1/25;
%       SELECT A SIGNAL TYPE ****************
% Use next lines for sine wave, for an .au file or an .wav file
% s=.5*sin(2*pi*50*tn);
[s,fs]=auread('bye441');                %read .wav file
%[s,fs]=wavread('bye440');               %read .wav file
sound(s,fs)

%Predictor and encoder-decoder parameters
lpclen=20;
bitsize=input('bitsize=');
fprintf('\nPlease wait... data length is %i\n',length(s))
%LPF parameters
tap=100;
cf=.15;

% DPCM with predictor
[Q,b, ai] = dpcm_enco_lpc(s, lpclen, bitsize);
[st]=dpcm_deco_lpc(b, ai, bitsize);
Sa=LPF(tap,cf,st);

figure; subplot(3,1,1):plot(s,'r'); ylabel('amplitude');
title('DPCM with predictor (red:input, green:decoder output, blue: LPF output)');
subplot(3,1,2):plot(st,'g'); ylabel('amplitude');
subplot(3,1,3):plot(Sa,'b'); ylabel('amplitude'); xlabel('index, n'); grid

pause; sound(s,fs);
pause; Sa=10 .* Sa;
% write speech out to a file called "dpcm_out"
auwrite(Sa,fs,'dpcm_output')
sound(Sa,fs);

function [Q,b,ai] = dpcm_enco_lpc(s, lpclen, bitsize)
% s : input signal;           bitsize : encoder bit size
% Q : qunatizer output;       b : encoder output
% e = s(i+1) - s(i)
%s = 2^(-1)*s/max(abs(s));
slen = length(s); e(1) = s(1);
[Q(1),b(1,:)] = pcm_quan_enco(e(1), bitsize);
st(1) = Q(1);
```

```
for i=2:slen
   if i<=lpclen
      e(i) = s(i)-st(i-1); [Q(i),b(i,:)] = pcm_quan_enco(e(i), bitsize); st(i) = st(i-1) + Q(i);
   else
      m=0; [a,G]=lpc(s(i-lpclen:i-1),lpclen); a = a*G;
      m = 1:lpclen;
      j=2:lpclen + 1;
      sth = sum(a(j).*st(i-m));      ai(i,:)=a;        e(i) = s(i)-sth;
      [Q(i),b(i,:)] = pcm_quan_enco(e(i), bitsize); st(i) = sth + Q(i);
   end
end
b=b'; b=b(:)';

function [Q,B] = pcm_quan_enco(e,bitsize)
% e : input to quantizer; bitsize : encoder bit size
% Q : qunatazer output; B : encoder output

if bitsize<4
  slen = length(e); D = 2^(-bitsize);
  switch bitsize
  case 3,
      for i=1:slen
         if e(i) < -3*D
                     Q(i) = -(7/2)*D;  b(i,:) = [ 0 0 0 ];
            elseif e(i) >= -3*D & e(i) < -2*D
                        Q(i) = -(5/2)*D; b(i,:) = [ 0 0 1 ];
            elseif e(i) >= -2*D & e(i) < -D
                        Q(i) = -(3/2)*D;  b(i,:) = [ 0 1 0 ];
            elseif e(i) >= -D & e(i) < 0
                        Q(i) = -(1/2)*D; b(i,:) = [ 0 1 1 ];
            elseif e(i) >= 0 & e(i) < D
                        Q(i) = (1/2)*D;  b(i,:) = [ 1 0 0 ];
            elseif e(i) >= D & e(i) < 2*D
                  Q(i) = (3/2)*D;  b(i,:) = [ 1 0 1 ];
            elseif e(i) >= 2*D & e(i) < 3*D
                  Q(i) = (5/2)*D;  b(i,:) = [ 1 1 0 ];
            elseif e(i) >= 3*D
                  Q(i) = (7/2)*D; b(i,:) = [ 1 1 1 ];
         end
      end

   case 2,
      for i=1:slen
         if e(i) < -D
                        Q(i) = -(3/2)*D; b(i,:) = [ 0 0 ];
            elseif e(i) >= -D & e(i) < 0
```

```
            Q(i) = -(1/2)*D;  b(i,:) = [ 0 1 ];
          elseif e(i) >= 0 & e(i) < D
                  Q(i) = (1/2)*D; b(i,:) = [ 1 0 ];
          elseif e(i) >= D
                  Q(i) = (3/2)*D; b(i,:) = [ 1 1 ];
        end
      end

    case 1,
      for i=1:slen
        if e(i) < 0
                  Q(i) = -(1/2)*D; b(i,:) = [ 0 ];
          elseif e(i) >= 0
                  Q(i) = (1/2)*D; b(i,:) = [ 1 ];
        end
      end
    otherwise
      fprintf('choose a bit size 1,2 or 3.\n');
    end
    b=b'; B=b(:)';
else
  [b0, b, bb] = dbc(e, bitsize);   [Q] = bdc(b0,b);   B = [b0 b];
end

function [Q] = pcm_deco_quan(B,bitsize)
% pcm decoder and quantizer;                bitsize : encoder bit size
% B : input to decoder from encoder ouput;  Q : qunatazer output
if bitsize<4
  b=B; slen = length(b); D = 2^(-bitsize);
  i = 0;
  for j=1:bitsize:slen
          mask=j:j+bitsize-1;
   i = i+1;
   switch bitsize
      case 3,
        if b(mask) == [ 0 0 0 ]
                  Q(i) = -(7/2)*D;
          elseif b(mask) == [ 0 0 1 ]
                  Q(i) = -(5/2)*D;
          elseif b(mask) == [ 0 1 0 ]
                  Q(i) = -(3/2)*D;
          elseif b(mask) == [ 0 1 1 ]
                  Q(i) = -(1/2)*D;
          elseif b(mask) == [ 1 0 0 ]
                  Q(i) = (1/2)*D;
          elseif b(mask) == [ 1 0 1 ]
```

```
                Q(i) = (3/2)*D;
        elseif b(mask) == [ 1 1 0 ]
                Q(i) = (5/2)*D;
        elseif b(mask) == [ 1 1 1 ]
                Q(i) = (7/2)*D;
       end

    case 2,
       if b(mask) == [ 0 0 ]
                Q(i) = -(3/2)*D;
        elseif b(mask) == [ 0 1 ]
                Q(i) = -(1/2)*D;
        elseif b(mask) == [ 1 0 ]
                Q(i) = (1/2)*D;
        elseif b(mask) == [ 1 1 ]
                Q(i) = (3/2)*D;
        end
    case 1,
       if b(mask) == [ 0 ]
                Q(i) = -(1/2)*D;
        elseif b(mask) == [ 1 ]
                Q(i) = (1/2)*D;
       end
    otherwise
       fprintf('choose a bit size 1,2 or 3.\n');
    end
  end
else
  slen=length(B)
  i = 0;
  for j=1:bitsize:slen
        i = i + 1;  mask=j:j+bitsize-1;  bb = B(mask);
        b0 = bb(1);   b = bb(2:bitsize);   Q(i)=bdc(b0,b);
  end
end

function [st]=dpcm_deco_lpc(b, ai, bitsize)
% b : input to decoder from communication channel;          bitsize : encoder bit size
% st : s_tilda (decoder output to lpf)
[jj,size_ai] = size(ai);  [Q] = pcm_deco_quan(b, bitsize);
st=cumsum(Q(1:size_ai-1));  slen=length(Q);
m=1:size_ai-1;
j=2:size_ai;
for i=size_ai:slen
   sth = sum(ai(i,j).*st(i-m));  st(i) = Q(i) + sth ;
end
```

```
function Sa=lpf(tap, cf, Sn)
%LPF lowpass filter
%tap: filter order. cf: cut-off frequency.
%Sa: decoder output.
b=fir1(tap,cf); Sa = conv2(Sn,b,'same');

function [x]=bdc(b0,b);
%  Binary-to-Decimal  conversion
% b0 : sign bit ( 0 represent + sign, and 1 represents - sign)
% y  : binary bits after the binary point;        x  : a constant in decimal
%
% Example: x=-0.778; b=4;
%          [x]=bdc(b0,b);    % returns decimal result.
%
N=length(b);             % finds the bit precision, B
y=0;

for i=1:N,
          y=y+b(i)*2^(-i);        % for  x >0, converts from binary to decimal
end
x=-b0+y;
if x < 0
          [b0,b,bb]=dbc(x,N+1);          %+1 bit is for sign
          y=0;
          for i=1:N,
                    y=y+b(i)*2^(-i);        % for x < 0, converts from binary to decimal
          end
end
x=-b0+y;

function [b0,b,bb]=dbc(x,B);
% Decimal-to-Binary conversion using B+1 bit precision
% x  : a constant in decimal;   B  : number of the precision bit
% b0 : sign bit ( 0 represent + sign, and 1 represents - sign); b  : binary bits after the binary point
% Example: x=-0.778; b=4;
%          [b0,b,bb]=dbc(x,B);             % returns binary result.
%          [b]=qround(b,bb);               % rounds off the binary input, returns binary.
%          [Y]=bdc(b0,b);                  % returns decimal result.
B = B-1;
if x>1, error(' x is not normilized.'); end
if x>=0
          b0=0;             % + sign is assigned.
          z=x;
else
          b0=1;             % - sign is assigned.
          z=-x;
```

```
end
if z >= 0
    for i=1:B,
            a=2*z;
     if a>=1
            b(i)=1; z=a-1;
     else
        b(i)=0;  z=a;
     end
    end
        a=2*z;                  % finds B+1 th bit in the binary point part
     if a>=1
            bb=1;
      else
        bb=0;
     end
end
```