# Bit-Wise Arithmetic Coding for Data Compression

A. B. Kiely
Communications Systems Research Section

*This article examines the problem of compressing a uniformly quantized independent and identically distributed (IID) source. We present a new compression technique, bit-wise arithmetic coding, that assigns fixed-length codewords to the quantizer output and uses arithmetic coding to compress the codewords, treating the codeword bits as independent. We examine the performance of this method and evaluate the overhead required when used block-adaptively. Simulation results are presented for Gaussian and Laplacian sources. This new technique could be used as the entropy coder in a transform or subband coding system.*

## I. Introduction

Many lossy data compression systems consist of separate decorrelation and entropy coding stages. In such schemes, the source data are transformed by some technique (e.g., discrete cosine transform or subband coding) with the goal of producing decorrelated components. Each component is independently scalar quantized and the quantizer output is losslessly compressed. Frequently each component is modeled as a sequence of IID random variables. This model motivates the topic of this article: block-adaptive compression of uniformly quantized samples from an IID source.

The traditional approach to the problem of efficiently encoding the quantizer output symbols is to use a variable-length code, assigning shorter codewords to the more probable symbols. The well-known Huffman coding technique gives the optimal such assignment. This method, however, has some performance limitations. Since each source sample must be mapped to a codeword of length at least 1, the rate of a Huffman code can never be less than 1 bit/sample, no matter how small the entropy. The redundancy of Huffman codes, the difference between rate and entropy, has been studied and bounded by many researchers, e.g., [4]. A common approach to reducing the redundancy at low entropy is to combine Huffman coding with zero-runlength encoding or to encode groups of symbols rather than individual symbols. In this article, we present an alternative solution.

Another problem with Huffman codes is that in block-adaptive situations there may be significant overhead costs (the extra symbols required to identify to the decoder the code being used) [1,5]. Finally, if the buffer overflows (say, if the source is much less compressible than anticipated), we may be forced to discard source samples.

With these problems in mind, we introduce a new technique, bit-wise arithmetic coding. The solution we propose is to assign a *fixed-length* binary codeword to each output symbol in such a way that a zero is more likely than a one in every codeword bit position. We then take the codewords corresponding to several adjacent quantizer output symbols and use a binary arithmetic encoder[1] to

[1] The term "binary encoder" is intended as shorthand for "binary-input binary-output encoder."

encode the first codeword bit for each of these symbols. We repeat this procedure for each group of codeword bits, treating each group independently from the others. This technique can be thought of as a simple progressive transmission scheme using an arithmetic coder to independently encode each level of detail. It turns out that this technique is often surprisingly efficient, despite the fact that interbit dependencies are ignored.

In Section II, we define the uniform quantizer parameters. In Section III, we analyze the block-adaptive binary arithmetic encoder that will be used as part of the bit-wise arithmetic encoding. We make use of this analysis in Section IV, where we examine in detail the the bit-wise arithmetic encoding procedure. We present performance results in Section IV.C.

## II. Uniform Quantizer

For several reasons we limit our investigation to the uniform quantizer, not the least of which is simplicity of implementation and analysis. The uniform quantizer often outperforms the Lloyd–Max quantizer in terms of rate-distortion performance (see, e.g., [2]); more important, if we do not know the source statistics a priori, it may be difficult to design a more suitable quantizer. The proposed new method could also be used with a nonuniform quantizer, but the analysis would be less tractable.

A continuous source with probability density function (pdf) $f(x)$ and variance $\sigma^2$ is quantized by a uniform quantizer having $b$ bits and bin width $\delta\sigma$, as in Fig. 1. The quantizer output is an index $i$ in the range $1 - 2^{b-1} \leq i \leq 2^{b-1}$, identifying which of the $2^b$ intervals contains the source sample. A source sample lying in $[T_{i-1}, T_i)$ has reconstruction point $i\delta\sigma$, where the quantizer thresholds are $T_i = (i + \frac{1}{2})\delta\sigma$, for $|i| \leq 2^{b-1} - 1$, and $T_{2^{b-1}} = \infty$, $T_{-2^{b-1}} = -\infty$. We could obtain a lower distortion by using reconstruction points that are equal to the center of mass [with respect to $f(x)$] of each interval, but since we wish to use this quantizer in adaptive situations, we cannot generally compute this quantity.

Note that the quantizer is asymmetric: There is a reconstruction point at the origin, and since there is an even number of points, there is an "extra" reconstruction point, which we arbitrarily choose to place on the positive side. The obvious alternative, a symmetric quantizer that has no reconstruction point at the origin, results in poor performance when $\delta$ is large relative to $\sigma^2$.

Let $p_i$ denote the probability that the quantizer output is index $i$, and let $d_i$ denote the contribution to the mean square error (MSE) from the interval $[T_{i-1}, T_i)$:

$$p_i = \int_{T_{i-1}}^{T_i} f(x)dx$$
$$d_i = \int_{T_{i-1}}^{T_i} (x - i\sigma\delta)^2 f(x)dx \tag{1}$$

The MSE is equal to $\sum_i d_i$. It will be convenient to let $\mathcal{P}$ denote the discrete distribution $\{p_i\}_{i=1-2^{b-1}}^{2^{b-1}}$.

In Figs. 2(a) and (b) we plot the resulting rate-distortion curves (computed analytically) for Gaussian and Laplacian sources over a wide range of $\delta$. Note that the curves show optimal performance when the uniform quantizer is used, rather than the theoretical rate-distortion limit, i.e., the rate shown is the entropy of $\mathcal{P}$. The large range of $\delta$ causes the peculiar loop in the curve: when $\delta 2^b$ becomes too small relative to $\sigma^2$, the performance is poor because the overload bin probabilities become large while their reconstruction points are too close to zero. This situation should be easily avoidable in practice, as we would expect the source range to be finite and known in advance because of hardware constraints.

In general we will assume

$$p_{-i} = p_i, \qquad 0 \leq i \leq 2^{b-1} - 2 \tag{2}$$

$$p_{1-2^{b-1}} = p_{2^{b-1}} + p_{2^{b-1}-1} \tag{3}$$

$$p_i \text{ is nonincreasing in } |i| \tag{4}$$

These conditions are true when $f(x)$ is symmetric about $x = 0$ and nonincreasing with $|x|$, and $\delta$ is not unreasonably small.

For sufficiently large $\delta$, the rate-distortion curve is virtually independent of $b$, as can be seen in Figs. 2(c) and (d), where we plot rate-distortion curves for $b = 4$ and $b = 8$ for Gaussian and Laplacian sources. Increasing $b$ has the effect of increasing the useful range of $\delta$ and lengthening the useful portion of the rate-distortion curve.

## III. Block-Adaptive Binary Arithmetic Encoding

In this section, we analyze the operation of the block-adaptive binary arithmetic encoder that will be used as part of the bit-wise arithmetic encoding procedure.

## A. Binary Arithmetic Encoder Operation

It is well known that a binary arithmetic encoder that is well-tuned to the source can achieve a rate quite close to the source entropy. Our goal in this section is to determine the performance we can expect from an encoder that may not be well-tuned. For more details on arithmetic coding see [6,7,8].

A binary arithmetic encoder with parameter $P$, the anticipated probability of a zero, maps an $N$-length sequence of bits $s$ into an interval $[l, r) \subset [0, 1]$ whose width is

$$r - l = P^{NF}(1 - P)^{N(1-F)} \tag{5}$$

where $F$ is the fraction of bits in $s$ that are zero. Ideally, we would like to have $P = F$, but this might not always be possible.

**Example 1.** Suppose $P = 5/13$, $s = 01$. Initially, $[l, r) = [0, 1)$. We divide this interval into $[0, 5/13)$, $[5/13, 1)$. Note that this first interval has width of $P = 5/13$ of the total interval. On receiving $s_1 = 0$, we assign $[l, r) = [0, 5/13)$ because the symbol with anticipated probability of $5/13$ was received. Again we divide $[0, 5/13)$ into $[0, 25/169)$, $[25/169, 5/13)$. After receiving $s_2 = 1$, we assign $[l, r) = [25/169, 5/13)$. Note that this assignment satisfies Eq. (5). ❐

A $K$-bit output sequence from the encoder maps to an interval $[i2^{-K}, (i + 1)2^{-K})$ for some $0 \leq i < 2^K - 1$. For our application, since $N$ is known to the decoder, the encoder must specify the largest interval of this form that is contained in $[l, r)$. That is, the encoder must use as few bits as possible to identify to the decoder a sequence beginning with $s$.

**Example 2.** Continuing Example 1, after calculating $[l, r) = [25/169, 5/13)$, the encoder must find an interval of the form $[i2^{-K}, (i+1)2^{-K})$ such that $[i2^{-K}, (i+1)2^{-K}) \subseteq [l, r)$. One such interval is $[1/4, 3/8)$, which corresponds to output sequence 010 (because this interval is equal to the set of numbers having binary expansion beginning with 0.010). We will verify in Example 3 that this is in fact the encoder output sequence.

Knowing $P$, the decoder realizes that $[1/4, 3/8) \subseteq [525/2197, 5/13)$ and that this latter interval corresponds to input sequence 011. Since the decoder also knows that $N = 2$, it takes the first two bits, giving output 01, which is in fact the encoder input sequence. ❐

We continue with the derivation of the encoder rate. We can write $[l, r) = [j2^{-J} + L2^{-J}, j2^{-J} + R2^{-J})$ where $0 \leq L \leq 1/2 \leq R \leq 1$ for integers $j$ and $J$. That is,

$$[l, r) \subseteq [j2^{-J}, (j + 1)2^{-J}) \tag{6}$$

for maximum $J$ and some $j$. The first $J$ bits of the output sequence map to the interval $[j2^{-J}, (j + 1)2^{-J})$, and the remaining bits correspond to a subinterval of $[L, R)$. Comparing interval widths in Eq. (6), we find that $2^{-J} \geq r - l = P^{NF}(1 - P)^{N(1-F)}$, which implies $J \geq Nh(P, F)$ where

$$h(P, F) \triangleq -F \log_2 P - (1 - F) \log_2(1 - P)$$

which is a line tangent to the binary entropy function $\mathcal{H}(F)$ at the point $F = P$. If we are very lucky, then $[j2^{-J}, (j + 1)2^{-J}) = [l, r)$ exactly and we are done encoding, in which case the rate is $J/N$, so the encoder rate $R_{\text{arith}}$ (the number of output bits divided by the number of input bits) satisfies $R_{\text{arith}} \geq h(P, F)$.

Usually we will not be so lucky, and we must send additional bits. In the worst case, $L \neq 0$ and $R \neq 1$, in which case we can assign these final bits to be $10^n$, which corresponds to $[1/2, 1/2 + 2^{-n-1})$.

**Example 3.** Continuing Example 2, we can write $[l, r) = [25/169, 5/13) = [0 \times 2^{-1} + (50/169) \times 2^{-1}, 0 \times 2^{-1} + (10/13)2^{-1})$, i.e., $j = 0, J = 1, L = 50/169 < 1/2 < R = 10/13$. The first output bit is 0 (corresponding to $j = 0$). The remaining bits, $10^n$ (corresponding to $[1/2, 1/2 + 2^{-n-1})$ ) must map to a subinterval of $[L, R)$, which implies $n = 1$, so the output sequence is 010. ❐

The value $n$ must be sufficiently large that $1/2 + 2^{-n-1} \leq R$, which implies

$$2^{-n-1} \leq R - 1/2 \leq R - L = 2^J P^{NF}(1 - P)^{N(1-F)}$$

making use of Eq. (5) and the fact that $2^{-J}(R - L) = r - l$. Alternatively, we may use $01^n$ instead of $10^n$ if this gives smaller $n$. Consequently,

$$n \leq \left\lceil -\log_2 \left[ 2^J P^{NF}(1 - P)^{N(1-F)} \right] \right\rceil - 1$$

$$= \lceil Nh(P, F) \rceil - J - 1$$

The total number of encoder output bits is $J + 1 + n$, so we have

$$h(P,F) \leq R_{\text{arith}} \leq \frac{\lceil Nh(P,F) \rceil}{N} \tag{7}$$

which suggests the close approximation

$$R_{\text{arith}} \approx h(P,F) + \frac{1}{2N} \tag{8}$$

## B. The Effect of Finite Precision

The bound of Eq. (7) is valid when the encoder can perform arithmetic with arbitrary precision. In any practical system, however, we are limited in our ability to represent the interval endpoints $l$, $r$. Whenever $l \geq 1/2$ or $r \leq 1/2$, we can transmit an output bit and rescale the interval (reassign $l$ and $r$) in the obvious way so that we can make the most of the available resolution. Failing to do this would degrade performance and severely limit the length of input sequences that can be encoded. A consequence of this rescaling is that $l \in [0, 1/2)$ and $r \in (1/2, 1]$, which means that at 9 bits of resolution, which is the resolution used for all simulations and discussions that follow, the values of $l$ and $r$ used by the encoder are always multiples of $2^{-10}$.

The effect of the finite resolution is that $P$, the anticipated probability of a zero, is almost never represented exactly.[2] In fact, $P_{\text{eff}}$, the effective value of $P$, varies as the algorithm progresses. At each iteration, the interval between $l$ and $r$ is divided into units of length $2^{-10}$, and the number of these units is equal to the resolution available to represent $P$. For example, in the worst case, $l = 1/2 - 2^{-10}$ and $r = 1/2 + 2^{-10}$, producing $P_{\text{eff}} = 1/2$. Note that we can never allow $P_{\text{eff}}$ to be zero or one because this would result in an input symbol having no effect on the interval, making decoding impossible. Unfortunately, bounds on rate derived from bounds on $P_{\text{eff}}$ are uselessly weak.

Because of the finite resolution, the final interval width is not exactly $P^{NF}(1-P)^{N(1-F)}$ as it is for the arbitrary precision system, but rather $\prod_{i:s_i=0} P_{\text{eff}_i} \prod_{j:s_j=1} (1 - P_{\text{eff}_j})$. Thus, the rate depends not only on $P$, $F$, and the resolution available, but also on the particular input sequence since $P_{\text{eff}_i}$ depends on $s_1, s_2, \ldots, s_{i-1}$. The consequence is that Eq. (8) is a slightly optimistic estimate of the rate.

Let $k$ equal the number of extra encoded bits resulting from the limited resolution, when compared to the sequence length predicted by Eq. (8), so that

$$R_{\text{arith}} = h(P,F) + \frac{(k + 1/2)}{N}$$

At 9-bit resolution, simulations indicate that $k$ has an expected value of approximately 0.32 bits, and standard deviation of 1.01, nearly independent of $P$, $F$, and $N$.[3] Thus, as an approximation to the rate we use

$$R_{\text{arith}} \approx h(P,F) + 0.82/N \tag{9}$$

for a system with 9-bit resolution. Increasing the resolution should have the effect of reducing the variance and expected value of $k$. Figure 3 gives an example of encoder performance when $P$ is fixed.

## C. Overhead

We would like to use binary arithmetic encoding block-adaptively to transmit a sequence of $N$ bits, i.e., the encoder output sequence is preceded by overhead bits that identify to the decoder the value of $P$ being used. By using $\log_2 N$ bits of overhead, we could specify $P = F$ exactly, but by using fewer bits we can exchange accuracy for lower overhead. In this section, we will explore this tradeoff, showing how to find the optimal number of overhead bits.

Assume for now that we have a fixed number of overhead bits $m$. We select an ordered set of $M = 2^m$ probabilities $\{\rho_1, \rho_2, \ldots, \rho_M\}$, known to the encoder and decoder in advance, that can be used as values for $P$. We first show how to find the optimal assignment of these probability points.

Given the set $\{\rho_1, \rho_2, \ldots, \rho_M\}$, the encoder would choose to use the $\rho_i$ that minimizes the rate, i.e., the $\rho_i$ that minimizes $h(\rho_i, F)$. As illustrated in Fig. 4, this amounts to using line segments to approximate the binary entropy function. Let $t_i$ denote the solution to $h(\rho_i, t_i) = h(\rho_{i+1}, t_i)$ and define $\Delta_i \triangleq h(\rho_i, t_i) - \mathcal{H}(t_i)$, which is the redundancy (additional rate) at this "corner" point $t_i$. Clearly if we choose the $\rho_i$'s to minimize $\max_i \Delta_i$, then we minimize the maximum redundancy over all possible values of $F$.

Suppose that for some $i$, we have $\Delta_{i-1} > \Delta_i$, as in Fig. 4. Then by decreasing $\rho_i$, we can decrease $\Delta_{i-1}$ and increase $\Delta_i$. From this argument we can see that the assignment of the $\rho_i$ is optimal when $\Delta_i$ is the same for all

[2] There are some trivial exceptions to this, such as when $P = 1/2$.

[3] When $P$ is very close to 0 or 1, the encoder rate is less well behaved. For example, if $P < 2^{-10}$, then clearly $P_{\text{eff}}$ will always exceed $P$. We ignore these extreme cases.

$i$. We call this optimal quantity $\Delta^*(M)$. Thus, we want to find $M$, the minimum number of line segments required to approximate the binary entropy function to within $\Delta^*$ everywhere.

Given $\Delta^*(M)$, we find that $\rho_1 = 1 - 2^{-\Delta^*(M)}$. Given $\rho_i$, we find $t_i$ by solving

$$h(\rho_i, t_i) - \mathcal{H}(t_i) = \Delta^*(M)$$

Given $t_i$, we find $\rho_{i+1}$ by solving

$$h(\rho_{i+1}, t_i) - \mathcal{H}(t_i) = \Delta^*(M)$$

i.e., we solve the same equation in the other direction. This procedure can be used to find the optimal set of $\rho_i$ and $t_i$ given $\Delta^*(M)$, or in an iterative procedure to compute $\Delta^*(M)$. We can also take advantage of the fact that the optimal $\rho_i$ must be symmetric about 1/2. In Table 1, $\Delta^*(M)$ is given for several values of $M$.

To find the relationship between $\Delta^*$ and $M$ for large $M$, let $w(\rho)$ denote the spacing between adjacent probability points in the neighborhood of $\rho$ so that the "corner" point is $t \approx \rho + w/2$. At this point,

$$\Delta^* \approx h(\rho, \rho + w/2) - \mathcal{H}(\rho + w/2) \approx \frac{w^2}{8\rho(1-\rho)\ln 2}$$

using the first three terms of the Taylor series expansion of $\mathcal{H}(\rho)$. Now $1/w(\rho)$ is the density of probability points, so

$$M \approx \int_0^1 \frac{1}{w(\rho)} d\rho \approx \frac{1}{\sqrt{8\Delta^* \ln 2}} \int_0^1 \frac{d\rho}{\sqrt{\rho(1-\rho)}}$$

$$= \frac{\pi}{\sqrt{8\Delta^* \ln 2}}$$

thus for large $M$,

$$\Delta^*(M) \approx \left(\frac{\pi^2}{8 \ln 2}\right) M^{-2} \tag{10}$$

In Fig. 5 we show the exact and approximate relationship between $M$ and $\Delta^*$.

The encoding operation is straightforward. The encoder computes $F$ by counting the number of zeros in the input sequence. The largest integer $I$ satisfying $t_{I-1} < F$ gives the optimal $\rho_I$, which is the parameter used in the encoding procedure. The encoder uses $m$ bits to identify $I$, followed by the arithmetic encoder output sequence. This procedure guarantees that

$$h(\rho_I, F) \le \mathcal{H}(F) + \Delta^*(2^m)$$

The rate (including overhead) required to block-adaptively transmit a sequence of $N$ bits is approximately

$$\mathcal{H}(F) + m/N + \Delta^*(2^m) + 0.82/N \tag{11}$$

where $\mathcal{H}(F)$ comes from the source uncertainty; $m/N$ comes from the $m$ overhead bits used to identify $\rho_I$; $\Delta^*(2^m)$ is a result of using $m$ bits to specify $F$ approximately instead of using $\log_2 N$ bits to specify $F$ exactly (this amounts to a worst-case assumption); and $0.82/N$ comes from the finite resolution of the encoder [see Eq. (9)].

Given $N$, the optimal number of overhead bits is

$$m^*(N) = \min_m{}^{-1}\{\mathcal{H}(F) + 0.82/N + \Delta^*(2^m) + m/N\}$$

$$= \min_m{}^{-1}\{\Delta^*(2^m) + m/N\} \tag{12}$$

which is tabulated in Table 2.

Using Eqs. (10) and (12), we find that for large $m$, the optimal value of $m$ satisfies

$$0 = \frac{\partial}{\partial m}[\Delta^*(2^m) + m/N] \approx \frac{1}{N} - \pi^2 2^{-2-2m}$$

which gives

$$m^*(N) \approx \frac{1}{2}\log_2 N + \log_2 \pi - 1 \tag{13}$$

and

$$\Delta^*\left(2^{m^*(N)}\right) \approx \frac{1}{2N \ln 2} \tag{14}$$

Substituting into Eq. (11), we find that the rate of a block-adaptive binary arithmetic encoder, including overhead, is approximately

$$\mathcal{H}(F) + \frac{1}{N}\left[\frac{1}{2}\log_2 N + 0.18 + \log_2 \pi + \frac{1}{2\ln 2}\right] \qquad (15)$$

## IV. Bit-Wise Arithmetic Coding

### A. Operation of the Bit-Wise Arithmetic Encoder

We now use the results of Section III to analyze the performance of bit-wise arithmetic encoding. First we outline the encoding procedure.

Each quantizer output symbol is mapped to a $b$-bit codeword. The first bit indicates the sign of the quantizer reconstruction point.[4] The remaining bits are assigned to quantizer levels in increasing lexicographic order as we move away from the origin. Figure 6 illustrates this mapping for $b = 4$. Because of Eqs. (2)–(4), a zero will be more likely than a one in every bit position.

Codewords corresponding to $N$ adjacent source samples are grouped together. The $N$ sign bits of the codeword sequence are encoded using the block-adaptive binary arithmetic encoder analyzed in Section III. Then the $N$ next-most-significant bits are encoded, and so on. This can be viewed as a simple progressive transmission system—each subsequent codeword bit gives a further level of detail about the source. Each bit sequence is encoded independently—at the $i$th stage the arithmetic coder calculates (approximately) the *unconditional* probability that the $i$th codeword bit is a zero.

The obvious loss is that we lose the benefit of interbit dependency. For example, the probability that the second bit is a zero is not generally independent of the value of the first bit, though the encoding procedure acts as if it were. Huffman coding does not suffer from this loss, which we examine in Section IV.B.

The advantage is that for many practical sources, this technique has lower redundancy than Huffman coding, because the arithmetic coder is not required to produce an output symbol for every input symbol. Also this scheme is relatively simple[5] and has some advantages in terms of overhead: because the number of codewords is $2^b$, the overhead of block-adaptive Huffman coding increases exponentially in $b$ unless we are able to cleverly exploit additional information about the source [5]. By contrast, the overhead required for bit-wise arithmetic encoding increases linearly in $b$ because the codeword bits are treated independently.

Another advantage is that this technique gives us a simple means of handling situations where we are rate constrained (or equivalently, buffer-constrained): We simply encode the blocks of $N$ bits until the allocated rate is exhausted (or the buffer is full). The distortion is automatically reduced for "more compressible" sources—when the most significant bits can be efficiently encoded, we are able to send additional (less-significant) bits, so the encoder resolution increases automatically. This would mean, for example, that a block having 6-bit resolution might be followed by a block having only 8-bit resolution. The advantage is that we hope to prevent any sample from having zero-bit resolution.

The obvious question is whether the gains offset the losses. Given $\delta, b$, and $f(x)$, we can use Eq. (1) to compute $\mathcal{P}$, the distribution on the quantizer output symbols, and use this result to compute $\pi_i$, the probability that the $i$th bit is a zero. For example, for $b = 4$,

$$\begin{bmatrix}\pi_1\\ \pi_2\\ \pi_3\\ \pi_4\end{bmatrix} =$$

$$\begin{bmatrix}1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&1&1&1&1&1&1&1&0&0&0&0\\ 0&0&1&1&0&0&1&1&1&1&0&0&1&1&0&0\\ 0&1&0&1&0&1&0&1&1&0&1&0&1&0&1&0\end{bmatrix}$$

$$\times \begin{bmatrix}p_{-7}\\ p_{-6}\\ \vdots\\ p_8\end{bmatrix} = \begin{bmatrix}1&1&1&1&1&1&1&1&1\\ 1&2&2&2&1&0&0&0&0\\ 1&2&1&0&1&2&1&0&0\\ 1&1&1&1&1&1&1&1&0\end{bmatrix}\begin{bmatrix}p_0\\ p_1\\ \vdots\\ p_8\end{bmatrix}$$

using the symmetries of the $p_i$ described in Section II. It turns out that $\pi_1 = (1 + p_0)/2$, $\pi_b = \pi_1 - p_{2^{b-1}}$. The rate obtained is (assuming for the moment that we have an idealized system where we may neglect overhead)

$$\sum_{i=1}^{b} \mathcal{H}(\pi_i)$$

[4] Or, to be precise, the first bit indicates whether the reconstruction point is positive.

[5] To the extent that a binary arithmetic coder is simple.

For the Gaussian and Laplacian sources, this result can be compared to Huffman coding in Figs. 7 and 8. Over the useful range of the quantizer, the rate of the bit-wise arithmetic coding scheme is quite close to the entropy.

### B. The Redundancy of Bit-Wise Transmission

We have already examined some of the sources that contribute to the rate of bit-wise arithmetic encoding: overhead bits, finite precision arithmetic, and the use of approximate rather than exact representation of $F$. We now examine the most obvious redundancy component: the added rate that results from treating each bit independently.

Let $\beta_i$ be the random variable that is equal to the value of the $i$th bit of the codeword corresponding to the quantizer output. The source entropy is then the entropy of the $\beta_i$, $H(\beta_1\beta_2\cdots\beta_b)$, but the independent encoding of the bits results in a rate of $H(\beta_1) + H(\beta_2) + \cdots H(\beta_b)$. Thus, the redundancy is

$$\begin{aligned}\mathcal{R} &= \sum_{i=1}^{b} H(\beta_i) - H(\beta_1\beta_2\cdots\beta_b) \\ &= \sum_{i=1}^{b} H(\beta_i) - [H(\beta_1) + H(\beta_2|\beta_1) \\ &\quad + \cdots H(\beta_n|\beta_{n-1}\beta_{n-2}\cdots\beta_1)] \\ &= \sum_{i=2}^{b} I(\beta_i, \beta_{i-1}\beta_{i-2}\cdots\beta_1) \qquad (16)\end{aligned}$$

where $I$ is the mutual information function. So, for example, if $b = 2$, the redundancy is equal to the mutual information between $\beta_1$ and $\beta_2$.

Of course, $\mathcal{R} \geq 0$, with equality if and only if the $\beta_i$ are independent. This bound is tight—zero redundancy occurs, for example, if the $p_i$ are distributed according to the two-sided exponential distribution $p_i = \alpha\theta^{|i|}$, which is a distribution suggested in [3] as a model for certain real-world sources. The exponential distribution is close to the distribution obtained for a Laplacian source, which explains why bit-wise arithmetic coding works well for this source.

A greedy assignment of bits results in almost exactly the same mapping from quantizer output symbol to codeword. In the first bit position, assign a 0 to the $2^{b-1}$ indices having the largest $p_i$, and a 1 to the others. In the second bit position, among the quantizer output indices having the same value in the first bit position, assign a 0 to the $2^{b-2}$ indices having the largest $p_i$, and so on. In this manner, the sign bit is the last codeword bit.

It should be noted that other codeword assignments, for example, assigning codewords so that Hamming weight is strictly nonincreasing in $|i|$, can sometimes give lower redundancy. Unless the distribution of the $p_i$ is known a priori, we cannot in general determine the optimal assignment. The codeword assignment proposed here has the advantages of symmetry and usefulness from a progressive transmission or buffer-constrained standpoint.

If we relax the assumptions of Eqs. (2)–(4), then the pathological case $p_0 = p_{2^{b-1}} = 1/2$ gives the maximum possible redundancy of $\mathcal{R}_{\max} = b - 1$. We would like to have a tight upper bound on redundancy when we maintain Eqs. (2)–(4). For $b = 2$, it is simple to verify that the maximum redundancy occurs when $\mathcal{P} = \{p_{-1}, p_0, p_1, p_2\} = \{1/3, 1/3, 1/3, 0\}$, which gives redundancy of $\log_2 3 - 4/3 \approx 0.252$. For larger $b$, it becomes more difficult to determine analytically what distribution gives maximum redundancy.

**Example 4.** If $b = 3$, the restrictions of Eqs. (2)–(4) imply that any valid distribution $\mathcal{P}$ can be written as a convex combination of the (mostly) uniform distributions

$$\mathcal{P}_1 = \{0, 0, 0, 1, 0, 0, 0, 0\}$$

$$\mathcal{P}_2 = \{0, 0, 1/3, 1/3, 1/3, 0, 0, 0\}$$

$$\mathcal{P}_3 = \{0, 1/5, 1/5, 1/5, 1/5, 1/5, 0, 0\}$$

$$\mathcal{P}_4 = \{1/7, 1/7, 1/7, 1/7, 1/7, 1/7, 1/7, 0\}$$

$$\mathcal{P}_5 = \{1/7, 1/7, 1/7, 1/7, 1/7, 1/7, 1/14, 1/14\}$$

Thus, we wish to maximize the redundancy function of Eq. (16) over the convex hull of $\{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_4, \mathcal{P}_5\}$. Simulations suggest that the maximum occurs at $\mathcal{P}_3$, which gives redundancy of $2/5[5\log_2 5 - 3\log_2 3 - 6] \approx 0.342$. Unfortunately, $\mathcal{R}$ is not convex $\cup$ in $\mathcal{P}$, so we are not certain that for arbitrary $b$ maximum redundancy occurs for some uniform distribution.

We conjecture that for any $b$, the maximum redundancy subject to Eqs. (2)–(4) occurs for some uniform $f(x)$. If

this conjecture is correct, then to find a bound on redundancy we examine redundancy in the limit $b \to \infty$. Consider what happens when the quantizer range is $[-1/2, 1/2]$ and $f(x)$ is uniform over $[-W/2, W/2]$, for some $0 < W < 1$, as in Fig. 9. The lines and spaces in the figure replace the codeword assignment matrix: the lines denote a one in the corresponding bit position and the gaps denote a zero (compare to Fig. 6). Note that in the limit the codeword assignment is symmetric about $x = 0$.

For a large fixed $b$, $\mathcal{R}_u^b$, the redundancy for a uniform distribution is

$$\mathcal{R}_u^b(W) = \sum_{i=1}^{b} \mathcal{H}(\pi_i) - H(\mathcal{P}) = \sum_{i=1}^{b} \{\mathcal{H}(\pi_i) - 1\} - \log_2 W$$

using the fact that $H(\mathcal{P}) = \log_2(W2^b) = b + \log_2 W$, which is a consequence of the uniform distribution. Given $i$, $1 - \pi_i$ is equal to the sum of the lengths of the darkened line segment portions divided by $W$. We ignore the sign bit because $\pi_{\text{sign}} = 1/2$, so this bit makes no contribution to the redundancy in the limit. Examining Fig. 9, we can see that the interval $[-W/2, W/2]$ will always contain either an integer number of line segments or an integer number of gaps. Thus, either $\pi_i$ or $1 - \pi_i$ will be equal to $\lfloor 1/2 + W2^{i-1} \rfloor / (W2^i)$, so $\mathcal{H}(\pi_i) = \mathcal{H}\left(\lfloor 1/2 + W2^{i-1} \rfloor / W2^i\right)$ and in the limit the redundancy is

$$\mathcal{R}_u^\infty(W) = -\log_2 W - \sum_{i=1}^{\infty} f_i(W) \tag{17}$$

where

$$f_i(W) \triangleq 1 - \mathcal{H}\left(\frac{\lfloor 1/2 + W2^{i-1} \rfloor}{W2^i}\right)$$

Note that we can limit our analysis to the case where $W > 1/2$ without loss of generality, because $\mathcal{R}_u^\infty(W/2^n) = \mathcal{R}_u^\infty(W)$ for any integer $n$. Figure 10 shows $\mathcal{R}_u^\infty(W)$, and several of the $f_i$ are shown in Fig. 11. If $W = j/2^n$ for integer $j$ and $n$, then only the first $n$ terms in the summation of Eq. (17) are nonzero.

The function $\mathcal{R}_u^\infty(W)$ attains a maximum of approximately 0.34544 near $W^* \approx 0.610711$. It is difficult to determine an analytic expression for $W^*$ or $\mathcal{R}_u^\infty(W^*)$, in part because the first derivative of $\mathcal{R}_u^\infty(W)$ is discontinuous at infinitely many points in $[1/2, 1]$. Given any $\mathcal{R}' < \mathcal{R}_u^\infty(W^*)$, we can find a uniform $f(x)$ and finite $b$ producing redundancy $\mathcal{R}'$ or higher. We conjecture that in general

$$\mathcal{R} < \mathcal{R}_u^\infty(W^*) \approx 0.34544$$

is a tight upper bound for any pdf producing $\mathcal{P}$ satisfying Eqs. (2)-(4). It is interesting that this bound is independent of $b$, while without any restrictions on $\mathcal{P}$ the redundancy can be as large as $b - 1$.

## C. Performance

Including all of the overhead effects, using Eqs. (11), (13), and (14), the rate of the bit-wise arithmetic coder is approximately

$$\begin{aligned} R_{\text{bit-arith}} &\approx H(\beta_1\beta_2\cdots\beta_b) + \mathcal{R} \\ &\quad + b\left[\Delta^*\left(2^{m^*(N)}\right) + \frac{m^*(N) + 0.82}{N}\right] \\ &\approx H(\beta_1\beta_2\cdots\beta_b) + \mathcal{R} \\ &\quad + \frac{b}{N}\left[\frac{1}{2\ln 2} + 0.18 + \frac{1}{2}\log_2 N + \log_2 \pi\right] \\ &\approx H(\beta_1\beta_2\cdots\beta_b) + \mathcal{R} + \frac{b}{N}\left[2.55 + \frac{1}{2}\log_2 N\right] \end{aligned} \tag{18}$$

In Fig. 12, we plot theoretical and simulated rate-distortion curves for the bit-wise arithmetic coder applied to Gaussian and Laplacian sources. Bit-wise arithmetic coding performs particularly well on the Laplacian source. Note that the increase in rate when we decrease $N$ from 512 to 256 is rather small. This is not surprising considering Eq. (18). The use of smaller $N$ implies faster adaptability to a nonstationary source, and also gives advantages when using a small buffer.

## D. Possible Refinements

There are a few tricks that we might use to further reduce the rate in a practical system:

(1) The arithmetic encoder could adaptively estimate the probability rather than simply count the number of zeros in a sequence. This might improve adaptivity to nonstationary sources, in addition to reducing overhead.

(2) The relative frequency of zeros in a sequence, $F$, must be a multiple of $1/N$, so not all "corners" (i.e., the $t_i$ in Fig. 4) can be reached. Keeping this fact in mind, we could adjust the $\rho_i$ values to obtain a slight improvement in performance: lower $\Delta^*(m)$ and perhaps lower $m^*(N)$. The rate reduction would probably be minuscule except at very small $N$.

(3) Additional savings may be obtained by considering the variations in rate due to the finite precision of the encoder. When $F$ is near a corner, we could (time and complexity permitting) encode the sequence using the two nearest $\rho_i$ to see which produces a shorter sequence.

(4) We could make the $\rho_i$ more dense in the regions that are more probable. For example, if Eqs. (2)–(4) are satisfied, then the probability of a zero will always be higher than the probability of a one, so we could require most of the $\rho_i$ to be less than $1/2$.

(5) Since the decoder knows $N$, $t_{I-1}$, and $t_I$ before it decodes, it knows that the sequence must contain between $\lceil Nt_{I-1} \rceil$ and $\lfloor Nt_I \rfloor$ zeros. In the current implementation, the decoder does not explicitly exploit this information. We could update $P$ as we encode/decode, taking into account the number of zeros that must remain in the sequence, "like counting cards in Vegas," comments Sam Dolinar.

(6) We might also get a slight improvement by combining the overhead of blocks, thus not requiring that $M$, the number of $\rho_i$, be a power of two.

(7) We could require that $\rho_1 = 0$, so that if a block consisted of the all-zeros sequence, we could encode the entire sequence simply by using the $m$ overhead bits to identify $P = \rho_1$.

(8) It might be convenient to keep track of the encoder output sequence length during the encoder operation, and send the data unencoded if the length exceeds $N$. This corresponds to forcing one of the $\rho_i$ to be equal to $1/2$. Once this happens, we might assume that all remaining bit positions are sufficiently random so that we are better off sending them unencoded and saving the overhead. This reflects the conventional wisdom that for many real-world sources, quantized samples are often compressible only in the most-significant bits.

## V. Conclusion

The bit-wise arithmetic encoding technique provides a simple method for data compression. The independent treatment of the codeword bits provides its main assets: The technique is simple, it can be used in progressive transmission or as a means of alleviating buffer overflow problems, and it has low overhead that increases linearly in the number of quantizer bits rather than exponentially. For the Gaussian and Laplacian sources, the rate is quite close to the entropy. The independent treatment of bits can also be its greatest liability—for sources where the codeword bits are highly correlated, the redundancy can be substantial. As with any data compression method, the usefulness of this technique ultimately depends on the source to be compressed.

## Acknowledgments

The author is grateful to Sam Dolinar, Kar-Ming Cheung, Fabrizio Pollara, and Laura Ekroot for their helpful comments.

# References


[1] Y. S. Abu-Mostafa and R. J. McEliece, "Maximal Codeword Lengths in Huffman Codes," *The Telecommunications and Data Acquisition Progress Report 42-110, vol. April–June 1992*, Jet Propulsion Laboratory, Pasadena, California, pp. 188–193, August 15, 1992.

[2] T. Berger, *Rate-Distortion Theory: A Mathematical Basis for Data Compression*, Englewood Cliffs, New Jersey: Prentice Hall, 1971.

[3] K. Cheung, P. Smyth, and H. Wang, "A High-Speed Distortionless Predictive Image Compression Scheme," *The Telecommunications and Data Acquisition Progress Report 42-102, vol. April-June 1990*, Jet Propulsion Laboratory, Pasadena, California, pp. 73–90, August 15, 1990.

[4] R. G. Gallager, "Variations on a Theme by Huffman," *IEEE Transactions on Information Theory*, vol. IT-24, no. 6, pp. 668–674, November 1978.

[5] R. J. McEliece and T. H. Palmatier, "Estimating the Size of Huffman Code Preambles," *The Telecommunications and Data Acquisition Progress Report 42-114, vol. April-June 1993*, Jet Propulsion Laboratory, Pasadena, California, pp. 90–95, August 15, 1993.

[6] J. J. Rissanen, "Generalized Kraft Inequality and Arithmetic Coding," *IBM Journal of Research and Development*, vol. 20, no. 3, pp. 198–203, May 1976.

[7] J. J. Rissanen and G. G. Langdon, Jr., "Universal Modeling and Coding," *IEEE Transaction on Information Theory*, vol. IT-27, no. 1, pp. 12–23, January 1981.

[8] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic Coding for Data Compression," *Communications of the ACM*, vol. 30, no. 6, pp. 520–540, June 1987.


**Table 1. The optimal relationship between $M$ and $\Delta$.**

| $M$ | $\Delta^*(M)$ |
|---|---|
| $2^0$ | 1.0 |
| $2^1$ | 0.32193 |
| $2^2$ | 0.093506 |
| $2^3$ | 0.025407 |
| $2^4$ | 0.0066389 |
| $2^5$ | 0.0016980 |
| $2^6$ | 0.00045745 |
| $2^7$ | 0.00010800 |
| $2^8$ | 0.000027077 |

**Table 2. The optimal relationship between block length $N$ and overhead bits $m$.**

| $N$ | $m^*(N)$ |
|---|---|
| 1 | 0 |
| [2, 4] | 1 |
| [5, 14] | 2 |
| [15, 53] | 3 |
| [54, 202] | 4 |
| [203, 806] | 5 |
| [807, 2861] | 6 |
| [2862, 12358] | 7 |

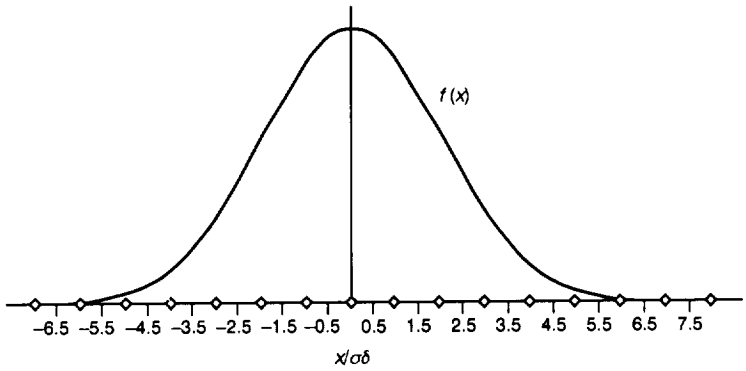


**Fig. 1. Example of a pfd and reconstruction points for a four-bit uniform quantizer.**

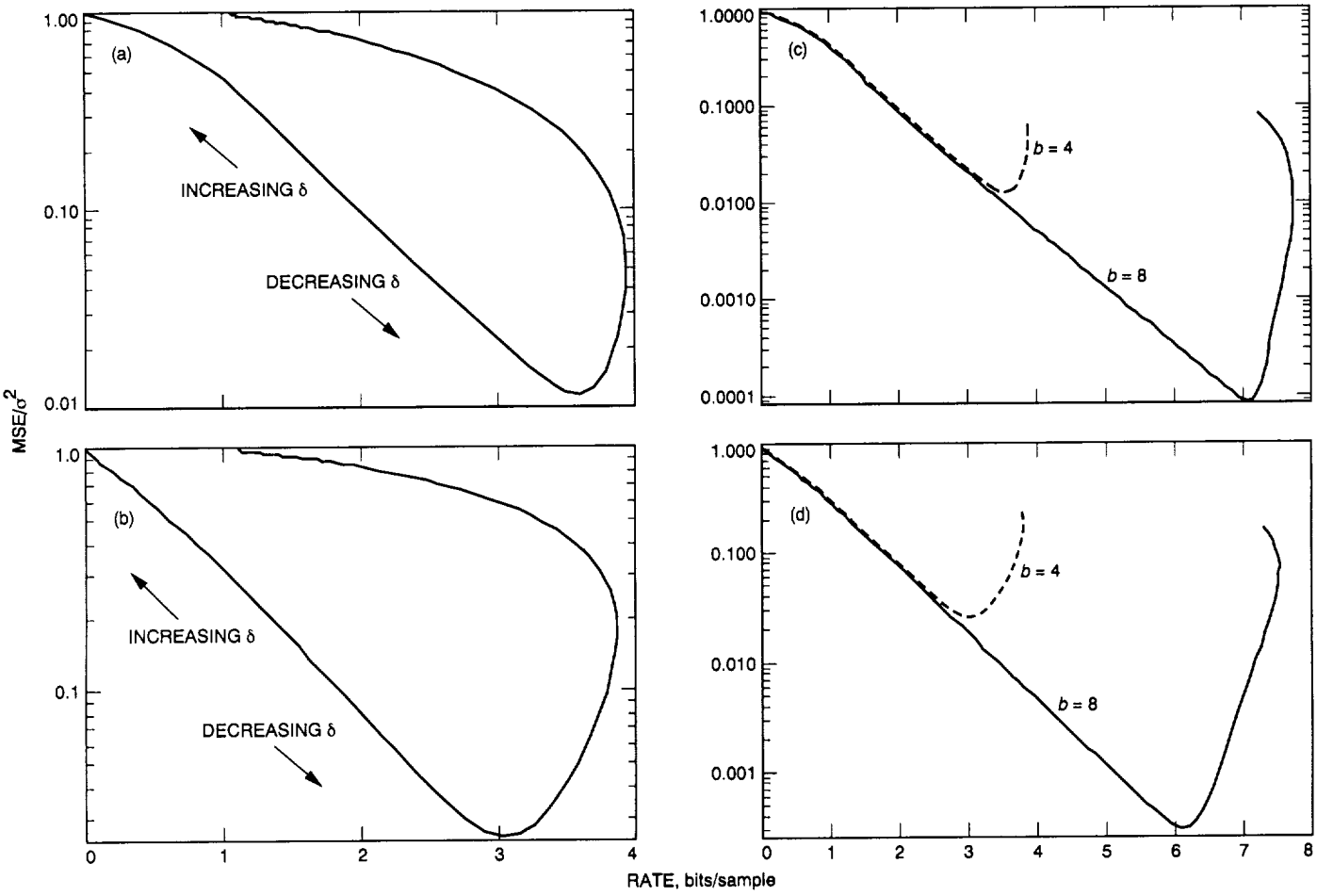


**Fig. 2. Rate distortion for the uniform quantizer over a wide range of δ: (a) Gaussian source, $b = 4$; (b) Laplacian source, $b = 4$; (c) Gaussian source, $b = 4$ and $b = 8$; and (d) Laplacian source, $b = 4$ and $b = 8$.**

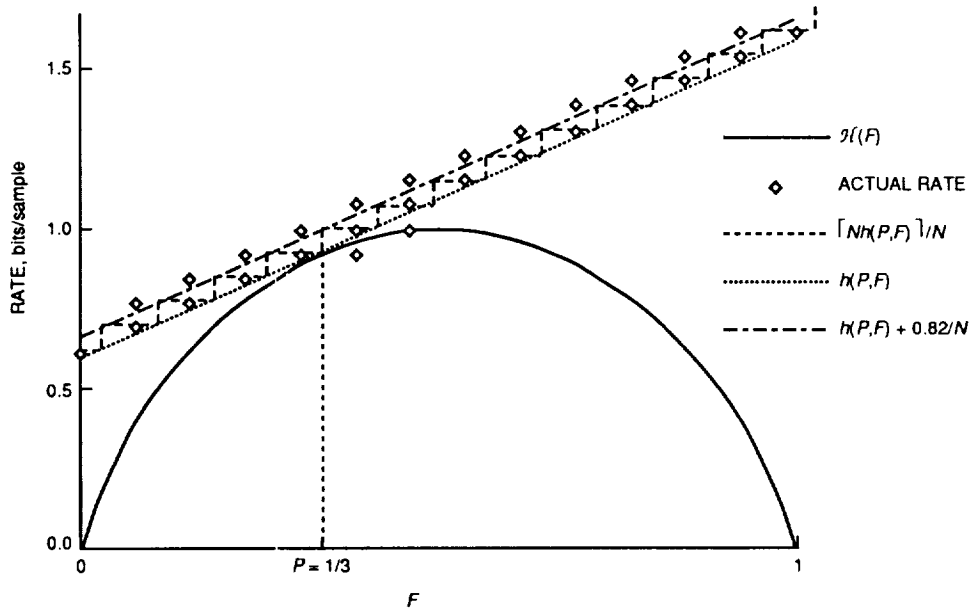


**Fig. 3. Binary entropy, predicted and actual performance of the 9-bit binary arithmetic encoder as a function of $F$ for all possible sequences of length $N = 13$ when $P = 1/3$.**

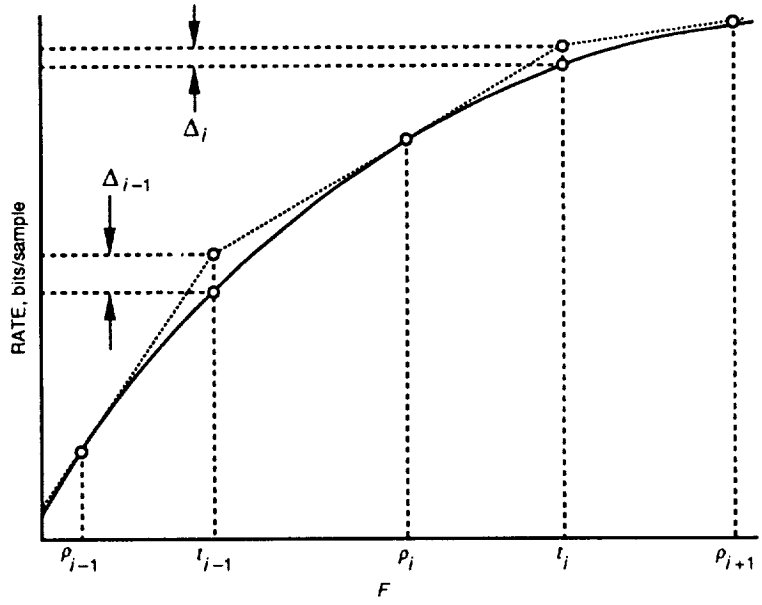


**Fig. 4. Binary entropy and line-segment approximation.**

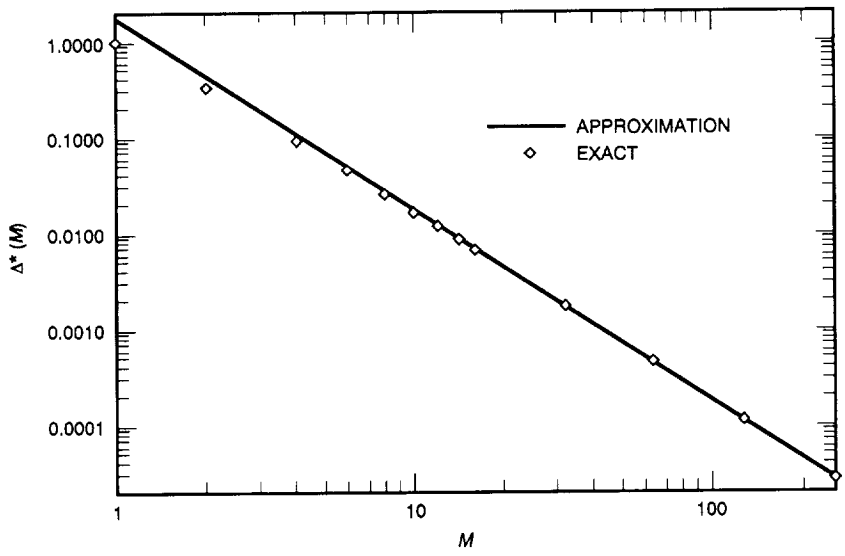


Fig. 5. $\Delta^*(M)$ and approximation $(\pi^2/8 \ln 2)\, M^{-2}$.

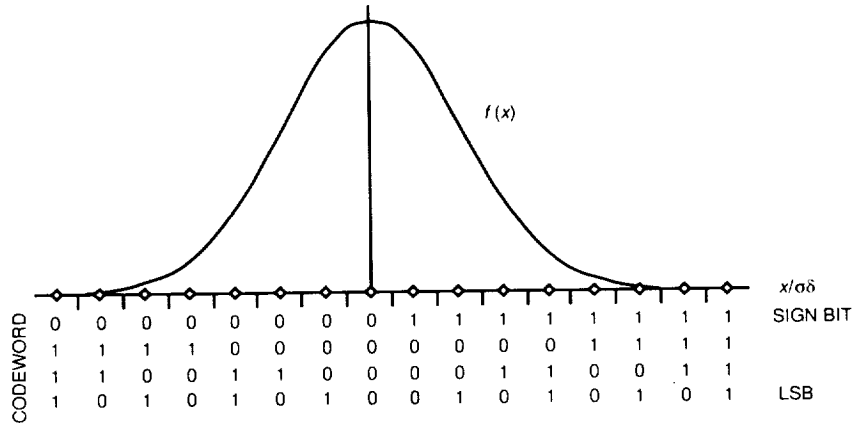


Fig. 6. Codeword assignment for the four-bit quantizer.

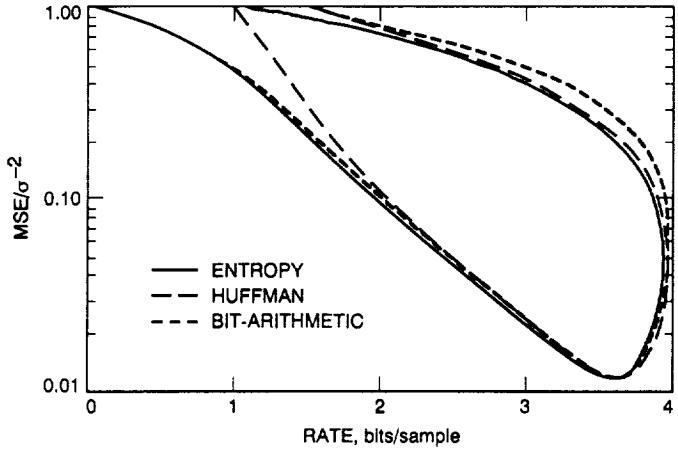


**Fig. 7. Bit-wise arithmetic coding, Huffman coding, and entropy for Gaussian source, $b = 4$, without overhead.**

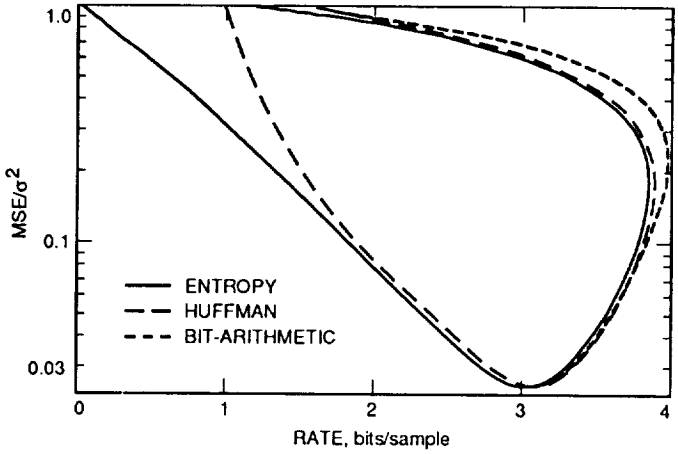


**Fig. 8. Bit-wise arithmetic coding, Huffman coding, and entropy for Laplacian source, $b = 4$, without overhead.**

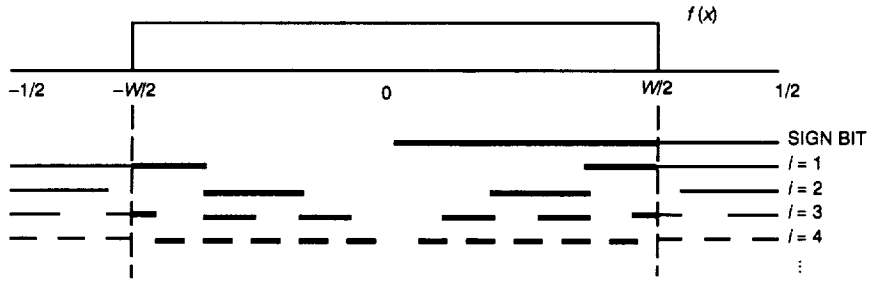


**Fig. 9. Uniform distribution and codeword assignment in the limit $b \to \infty$.**

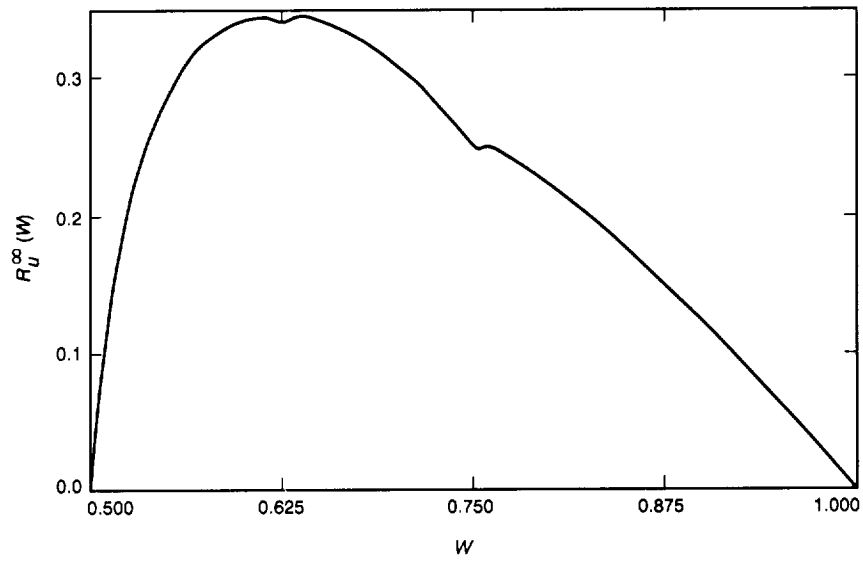


Fig. 10. The redundancy function for a uniform distribution, $R_u^\infty(W)$.

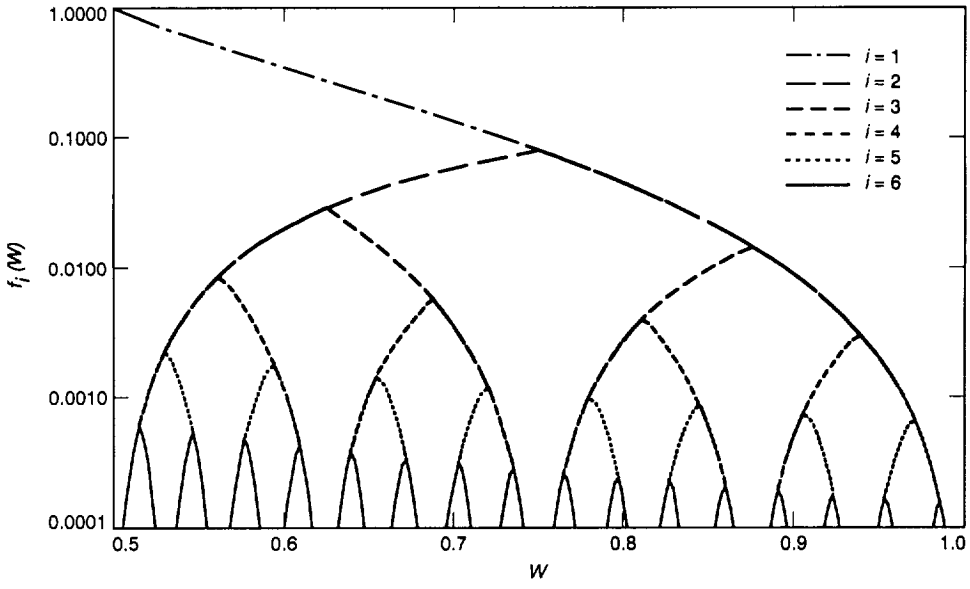


Fig. 11. Components $f_i(W)$ of the redundancy function for a uniform distribution.

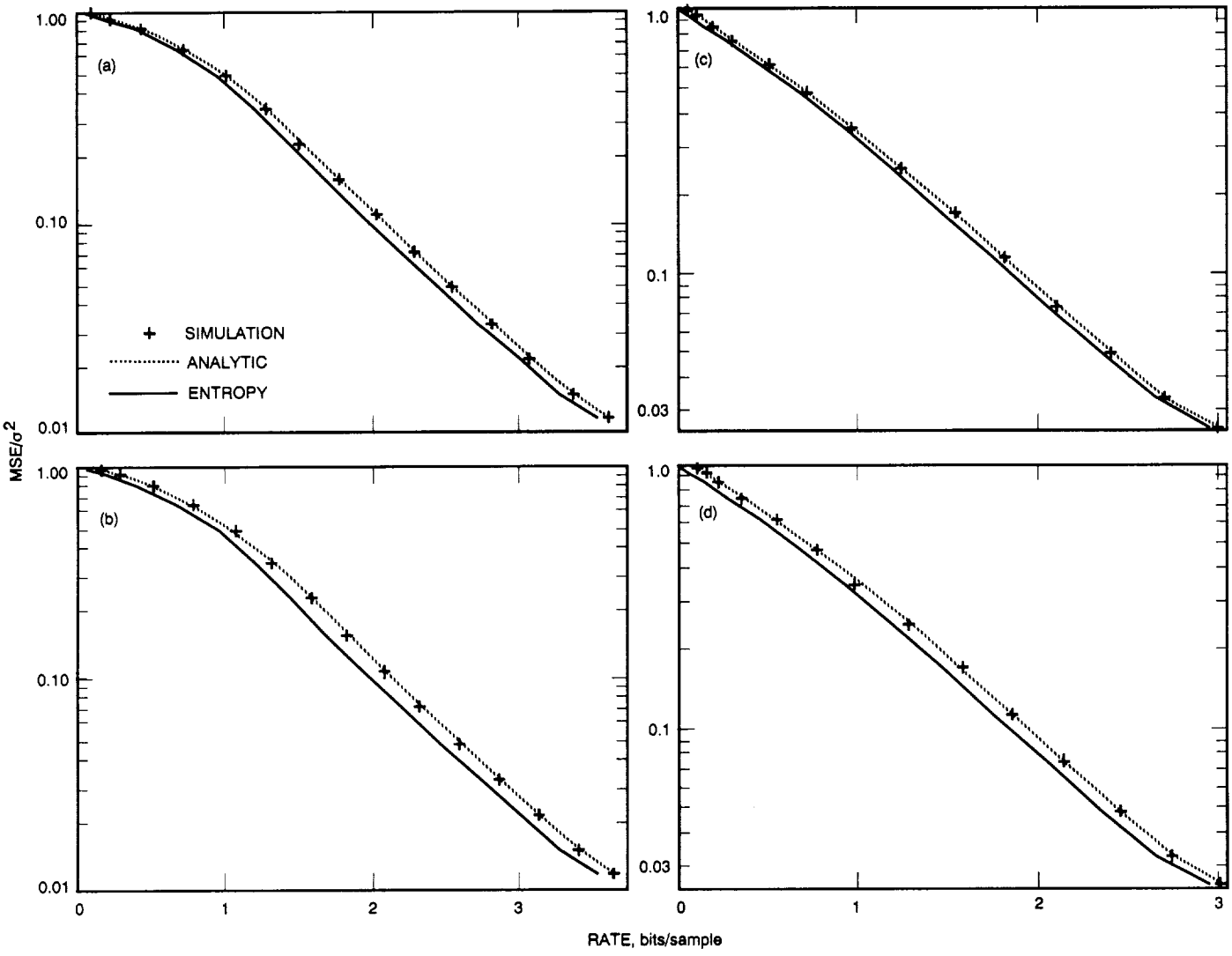


**Fig. 12. Bit-wise arithmetic coding performance, including overhead: (a) Gaussian source, $b = 4$, $N = 512$; (b) Gaussian source, $b = 4$, $N = 256$; (c) Laplacian source, $b = 4$, $N = 512$; and (d) Laplacian source, $b = 4$, $N = 256$.**