Generating Gaussian Pseudo-Random Deviates

Peter Kabal

Department of Electrical & Computer Engineering

McGill University

February 2000 (revised Oct. 2000)

http://www.TSP.ECE.McGill.CA

Generating Gaussian Pseudo-Random Deviates

1 Introduction

This report examines low-complexity methods to generate pseudo-random Gaussian (normal) deviates. We introduce a new method based on modelling the Gaussian probability density function using piecewise linear segments. This approach is shown to be both efficient and accurate. It does not require the calculation of transcendental functions.

All of the methods considered map one or more uniform distributions to create the Gaussian deviates. This report investigates the effect of the use of discrete variates, particularly in the tails of the Gaussian distribution. In addition, we give a new interpretation of the method of aliases that suggests its application to non-uniform quantization.

2 Uniform Deviates

2.1 Continuous uniform distribution

Consider uniform continuous-valued deviates $x_{uc}[k]$ that lie in the range $[0,1]$. The probability density function (pdf) of $x_{uc}[k]$ is

$$p_{uc}(x) = \begin{cases} 1 & 0 \le x \le 1, \\ 0 & \text{elsewhere.} \end{cases} \tag{1}$$

The mean and standard deviation for this distribution are

$$m_{uc} = \frac{1}{2}, \qquad \sigma_{uc}^2 = \frac{1}{12}. \tag{2}$$

2.2 Discrete uniform distribution

A number of different schemes have been proposed to generate pseudo-random uniform deviates. We describe one here, but many others exhibit similar behaviour, specifically that the returned values lie on a discrete grid.

Consider the multiplicative congruential method for generating a uniform deviate [1,2,3]. The basic procedure takes the form

$$x[k] = \mathrm{mod}(a\,x[k-1], M), \tag{3}$$

where a is a carefully chosen multiplier, $x[k-1]$ is a previous (non-zero) deviate and M is an appropriate modulus. All values are integers. The book *Numerical Recipes* [2], suggests $a = 16807$ and $M = 2^{31} - 1$. The generation of each variate requires a multiplication and a modulo operation. An algorithm due to Schrage [2, p. 278] avoids overflow in the calculation and can be used to implement a portable random number generator. The period of the generator is $M - 1$ for a non-zero initial value. The output values are integers in the interval $[1, M-1]$. The value 0 does not appear in the output, since it would repeat for all future values. An additional shuffling step can be used to break up low order correlations (see [2]).

It is common for uniform random number generators to return uniform deviates as floating point numbers between 0 and 1. The routine given in [2] computes

$$x_{ud}[k] = \frac{x[k]}{M}. \tag{4}$$

The value $x_{ud}[k]$ satisfies

$$\frac{1}{M} \le x_{ud}[k] \le \frac{M-1}{M}. \tag{5}$$

The value $x_{ud}[k]$ takes on discrete values. Assuming that each value of $x_{ud}[k]$ is equiprobable, the mean and variance of $x_{ud}[k]$ are

$$m_{ud} = \frac{1}{2}, \qquad \sigma_{ud}^2 = \frac{1}{12} - \frac{1}{6M}. \tag{6}$$

3 Gaussian Deviates

There are a number of techniques for generating Gaussian deviates from uniform deviates [1]. We consider two approaches for which computer programs are widely available.

3.1 Central Limit Theorem

The Central Limit Theorem of probability says that an appropriately normalized sum of independent, identically-distributed random values has a cumulative distribution that approaches a Gaussian cumulative distribution in the limit of a large number of terms [4]. Here we are interested in a finite number of terms and wish to evaluate how close the distribution of the sum is to a Gaussian distribution.

3.1.1 Sum of continuous uniform deviates

Consider adding N independent (continuous) uniform deviates,

$$x_c = \sum_{k=0}^{N-1} x_{uc}[k]. \tag{7}$$

The probability density function of the sum can be obtained by convolving the N uniform densities,

$$p_c(x, N) = p_{uc}(x) * \cdots * p_{uc}(x). \tag{8}$$

We will use a generating function (here the Laplace transform) to express the result. The Laplace transform of the probability density of the sum can be expressed as the N-fold product of the Laplace transform of the uniform density.

The uniform pdf can be written as the difference between two unit step functions,

$$p_{uc}(x) = u(x) - u(x-1), \tag{9}$$

where the unit step function is defined as

$$u(x) = \begin{cases} 1 & x \geq 0, \\ 0 & \text{elsewhere.} \end{cases} \tag{10}$$

Then the Laplace transform of $p_c(x,N)$ is

$$\begin{aligned} X_c(s,N) &= \left(\frac{1-e^{-s}}{s} \right)^N \\ &= \frac{1}{s^N} \sum_{k=0}^{N} (-1)^k \binom{N}{k} e^{-ks}. \end{aligned} \tag{11}$$

The inverse transform of this expression gives the pdf of the sum,

$$p_c(x,N) = \frac{1}{(N-1)!} \sum_{k=0}^{N} (-1)^k \binom{N}{k} (x-k)^{N-1} u(x-k). \tag{12}$$

The pdf is formed from polynomial segments. The function value and $N-2$ derivatives are continuous between segments. From basic considerations, $p_c(x,N)$ is non-zero only for $0 \leq x \leq N$ and is symmetric about $N/2$ (i.e., $p_c(x,N) = p_c(N-x,N)$).

The cumulative distribution function (cdf) can be calculated by integrating $p_c(x,N)$ (or as the inverse transform of $X_c(s,N)/s$),

$$F_c(x,N) = \frac{1}{N!} \sum_{k=0}^{N} (-1)^k \binom{N}{k} (x-k)^N u(x-k). \tag{13}$$

The distribution of the sum has mean $m_c = N/2$ and variance $\sigma_c^2 = N/12$. A zero-mean, unit-variance variate can be created by scaling and shifting the sum,

$$x_{nc} = \sigma_c (x_c - m_c). \tag{14}$$

The resultant pdf and cdf are

$$p_{nc}(x,N) = \sigma_c p_c(\frac{x}{\sigma_c} + m_c, N),$$
$$F_{nc}(x,N) = F_c(\frac{x}{\sigma_c} + m_c, N). \tag{15}$$

The Berry-Esséen Theorem [4] gives us information about the rate of convergence as the number of terms in the sum, N, increases: the cdf of the normalized sum of uniform deviates can be bounded relative to the true Gaussian cdf (denoted as $\mathcal{N}(x)$),

$$|F_{nc}(x,N) - \mathcal{N}(x)| < \frac{9}{4\sqrt{N}}. \tag{16}$$

This shows that the error decreases as $1/\sqrt{N}$. However, for practical values of N, the actual deviation for the sum of uniform variates is much smaller than this bound.

Fig. 1 shows a plot of the pdf $p_{nc}(x,N)$ for $N = 12$, along with a Gaussian pdf. The tails of $p_{nc}(x,N)$ extend from the mean out to $\pm\sqrt{3N}$ and are zero beyond that point. For instance for $N = 12$, the tails extend out to ± 6 standard deviations.

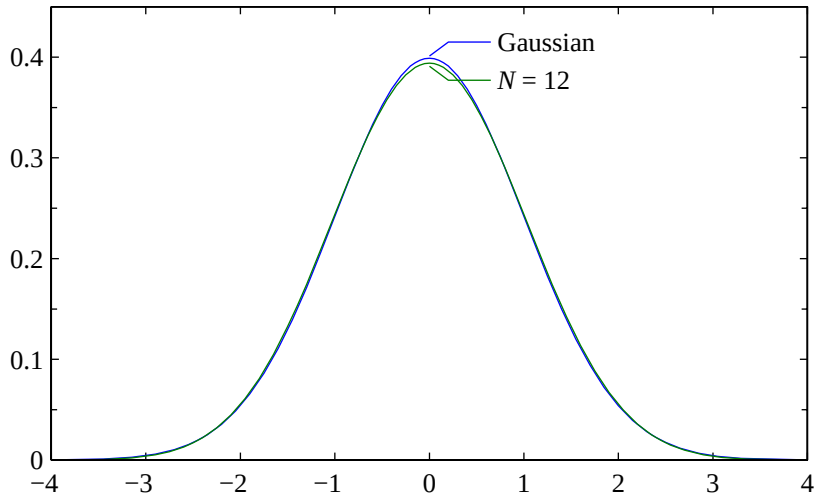

Fig. 1 Probability density function for a sum of $N = 12$ uniform deviates.

Since the area under any pdf is fixed at unity, the pdf of the sum must oscillate about the pdf of the true Gaussian density. The difference between the true Gaussian density and the pdf of the sum for different values of N is plotted in Fig. 2.

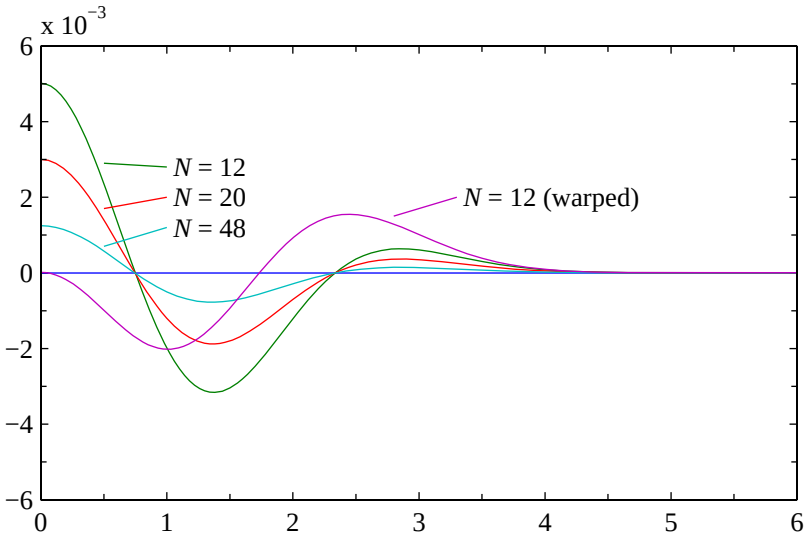

Fig. 2 Difference between the Gaussian density and the sum of uniform deviates.

Warping the output values

Warping the output value can reduce the error in the pdf. Consider a polynomial function applied to the sum variable x_c,

$$y_{nc} = \sum_{i=0}^{N_t} a_i x_{nc}^i. \tag{17}$$

For $N = 12$, an anti-symmetric warping polynomial (with only odd-numbered coefficients) is follows [5],

$$\begin{aligned} a_1 &= 0.98746, & a_3 &= 3.9439\times10^{-3}, \\ a_5 &= 7.474\times10^{-5}, & a_7 &= -5.102\times10^{-7}, \\ a_9 &= 1.141\times10^{-7}. \end{aligned} \tag{18}$$

This polynomial function deviates slightly from a straight line for small values and then stretches out the tail of the distribution. As shown in Fig. 2, warping reduces the maximum error.

Tail probabilities

Fig. 3 shows a plot of the tail probability $1-F_{nc}(x,N)$ for several values of N. The log scale shows the deviation of the tail probability from the true value. For $N=12$, the simple sum starts to deviate significantly from the true Gaussian probability above 4 standard deviations. The warped sum improves considerably on the simple sum.

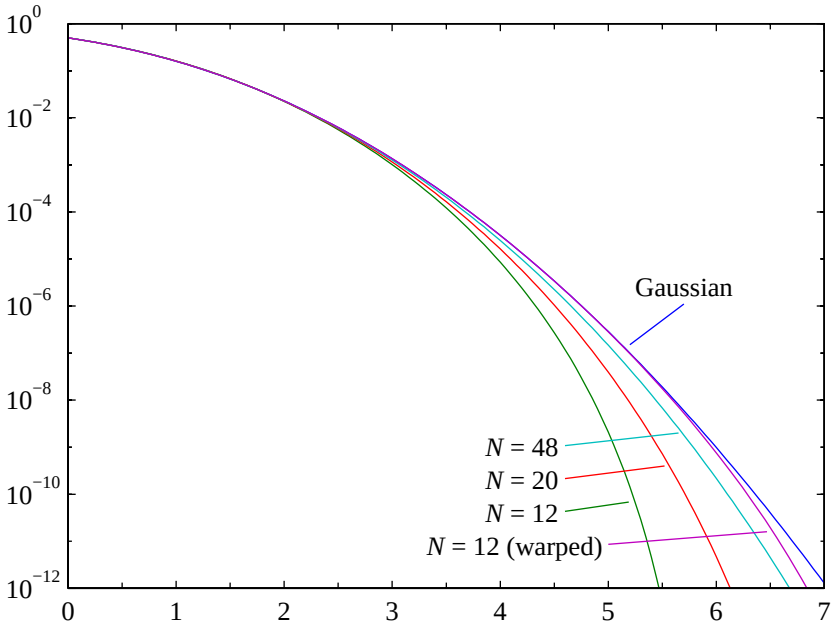

Fig. 3 Tail probability for a sum of uniform deviates.

3.1.2 Sum of discrete uniform deviates

Given that the underlying $x_{ud}[k]$ is discrete, the sum has a multinomial distribution. To simplify the notation, consider the sum,

$$s=\sum_{k=0}^{N-1}x[k], \tag{19}$$

where $x[k]$ is the integer-valued uniform deviate that is used to calculate $x_{ud}[k]$. The uniform probability function can be written in terms of the difference between two discrete unit step functions,

$$p_{ud}[n] = \frac{1}{M}(u[n-1] - u[n-M]), \tag{20}$$

where the discrete unit step function is defined as,

$$u[n] = \begin{cases} 1 & n \geq 0, \\ 0 & \text{elsewhere.} \end{cases} \tag{21}$$

The generating function (z-transform) for this density is

$$X_{ud}(z) = \frac{z^{-1}}{M}\frac{1 - z^{-(M-1)}}{1 - z^{-1}}. \tag{22}$$

The probability density of the sum corresponds to the following z-transform,

$$\begin{aligned} X_d(z,N) &= \frac{z^{-N}}{M^N(1-z^{-1})^N}\sum_{k=0}^{N}(-1)^k\binom{N}{k}z^{-k(M-1)} \\ &= \frac{z^{-N}}{M^N}\sum_{l=0}^{\infty}\binom{N+l-1}{l}z^{-l}\sum_{k=0}^{N}(-1)^k\binom{N}{k}z^{-k(M-1)} \end{aligned} \tag{23}$$

The inverse transform then gives the probability distribution for the sum,

$$P(s=n,N) = \frac{1}{M^N}\sum_{k=0}^{N}(-1)^k\binom{N}{k}\binom{n-kM-1}{n-kM-N}u[n-kM-N]. \tag{24}$$

The cumulative distribution function can be calculated by summing $P(s=k,N)$ for k running from $-\infty$ to n, or as the inverse transform of $X_d(z)/(1-z^{-1})$,

$$P(s \leq n,N) = \frac{1}{M^N}\sum_{k=0}^{N}(-1)^k\binom{N}{k}\binom{n-kM}{n-kM-N}u[n-kM-N]. \tag{25}$$

The cdf is a piecewise constant, non-decreasing function.

The analysis above was done for the sum of integer-valued variates. For the scaled variates (see Eq. (4)), the sum values are scaled and lie on a discrete grid (lattice). A plot of the pdf or cdf is indistinguishable from that of the sum of continuous-valued uniform variates.

3.2 Transformation of variables

Consider a two dimensional Gaussian variable with independent identically-distributed components. When plotted in two dimensions, the radial distance to the value has a Rayleigh distribution, and the angle is uniformly distributed between 0 and 2π. In the polar transformation method for generating Gaussian deviates, one uniform deviate is transformed to a Rayleigh variate and a second uniform deviate is transformed to a uniform angle. The final Gaussian deviates, y_1 and y_2 are then formed as

$$\begin{aligned} y_1 &= \sqrt{-2\log(x_1)}\cos(2\pi x_2), \\ y_2 &= \sqrt{-2\log(x_1)}\sin(2\pi x_2). \end{aligned} \tag{26}$$

An accept-reject approach can be used to obviate the need for calculating the sinusoids [2].

3.2.1 Polar transformation of discrete uniform deviates

Consider the discrete uniform variates with values between $1/M$ and $(M-1)/M$, see Eq.(5). The number of distinct values for, say y_1, is $(M-1)^2$ — the product of the number of different cosine values and the number of different Rayleigh values. The cosine and sine terms in the transformation are always bounded by unity. The Rayleigh term determines the range of the output variates. The largest possible value for the Rayleigh term is $\sqrt{-2\log(1/M)}$. This is also bounds the largest Gaussian variate. For $M = 2^{31}-1$, the largest value corresponds to 6.56 standard deviations.

A plot of the tail probability for the Rayleigh term (discrete values), Fig. 4, shows the deviation from the true distribution above 6 standard deviations.

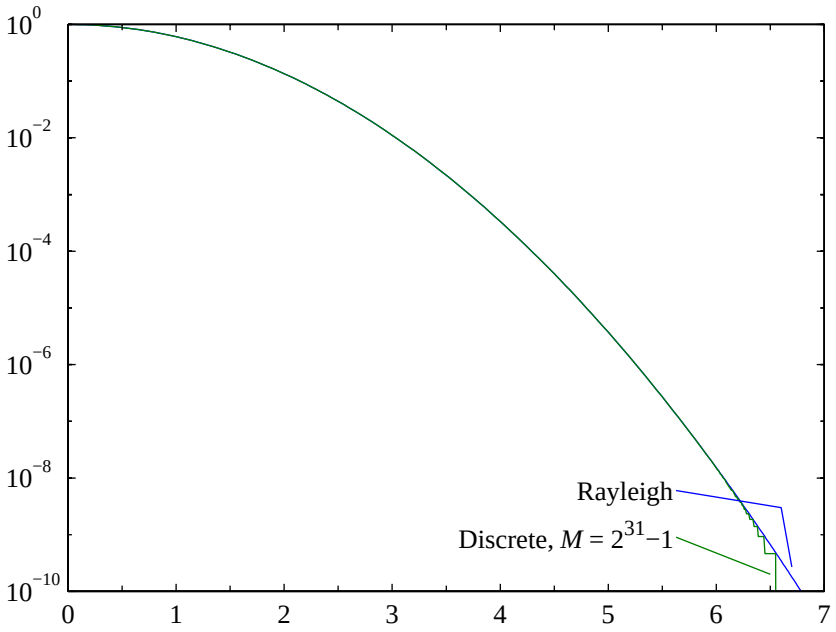

Fig. 4 Tail probability for a transformation of discrete values (Rayleigh).

3.3 CLT versus polar transformation

The Central Limit Theorem approach and the polar transformation method provide Gaussian deviates in quite different ways. In the basic CLT approach, the discrete output values are uniformly spaced, but the probability masses for the output points differ. The cdf consists of steps, uniformly spaced in x, but with heights proportional to the probability masses. In the transformation method, the discrete output values for the Rayleigh component are non-uniformly spaced. The pdf is discrete with equal masses $(1/(M-1))$ for the non-uniformly spaced values. The cdf consists of steps, non-uniformly spaced in x, but all of the same height.

The CLT approach is simple to program, but is approximate. The most significant drawback for many applications is the poor approximation of the tails of the Gaussian distribution. The question of how well the tails have to be modelled is discussed in Appendix A. The polar transformation method matches the Gaussian distribution better in the tails, though

the maximum value is still limited. It also requires the calculation of transcendental functions.

4 Gaussian Probability Density: Piecewise Linear Approximation

Another approach to generating an arbitrary probability density function is based on the observation that any pdf can be written in the following form

$$p_x(x) = \sum_{i=0}^{N-1} q_i \, p_i(x). \tag{27}$$

With this formulation, the overall pdf is expressed as the weighted sum of pdf's. The weight q_i represents the probability of choosing the pdf $p_i(x)$

This approach can be used to approximate the Gaussian density. The goal is to produce an algorithm that can be coded in a program that is regular and simple (like the basic CLT approach), that does not use transcendental functions, but that has a smaller approximation error than the CLT approach.

4.1 Piecewise linear approximation using triangular distributions

First we note that a triangular pdf can be easily generated as the sum of two uniform pdf's. By overlapping the triangular distributions, we can generate an overall pdf with piecewise linear segments. Fig. 5 shows (a low resolution) triangular approximation to the Gaussian density. The steps in generating the (approximate) Gaussian deviate are as follows.

1. Determine which triangular pdf to use. We have to select $p_i(x)$ with probability q_i.

2. Generate a sample from $p_i(x)$. This pdf is a shifted and scaled triangular pdf.

For the first task, we want to randomly generate a discrete index, say i, where the index occurs with probability, q_i. Starting from a uniform deviate, the straightforward approach is to set up thresholds that divide the unit interval into N segments, each of length equal to one of

the given probabilities. A binary search can be used to limit the number of comparisons to at most $\lceil \log_2(N) \rceil$. An alternate approach is the alias method. In this procedure, the segments are rearranged in such a manner as to allow a correctly distributed index value to be determined with a few simple operations. This method is reviewed and interpreted in Appendix B.

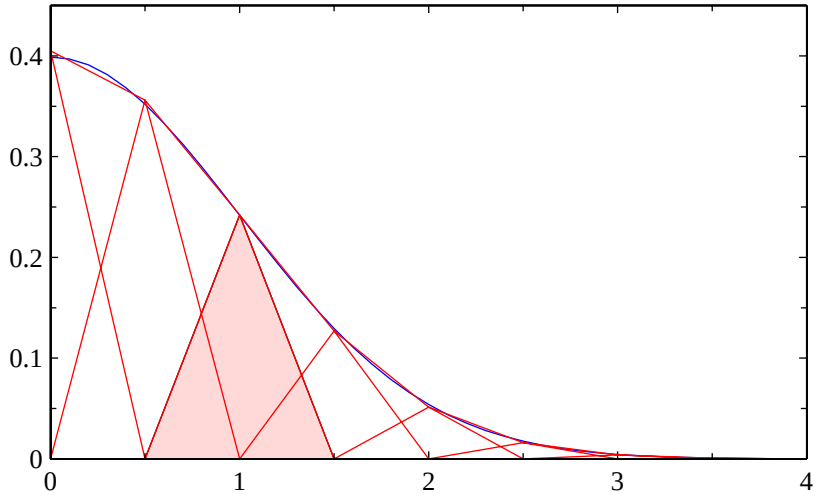

Fig. 5 Triangular pdf's used to approximate a Gaussian density.

Generalization of the triangular distribution

The linear approximation described above uses equal width triangular sub-distributions. The deviates for the individual triangular sub-distributions can be generated a sum of two independent uniform deviates. In our case, the triangular distributions are symmetric about their mean. Consider a generalization of this procedure. Let u_1 and u_2 be two uniform deviates. Form the sum,

$$v = \alpha \min(u_1, u_2) + (1-\alpha)\max(u_1, u_2). \tag{28}$$

For $\alpha = 1/2$, this reverts to the scaled sum of u_1 and u_2 and gives a symmetric triangular distribution. For other values of α, we can form non-symmetric triangular sub-distributions that could then be stitched together to form the overall distribution. For instance, the gener-

alized procedure could be used to stretch the last triangular distribution to go further into the tail.

Use of this more generalized formulation would require additional tables to describe the parameters (location and skew) of the sub-distributions. In the sequel we consider just the simpler case of symmetric triangular distributions.

4.2 Choosing the model parameters

The modelling of the Gaussian pdf with linear segments involves choosing parameters for the model. Consider only a piecewise linear approximation made from triangular sub-distributions. The sub-distributions are uniformly spaced. For ease of argument, suppose the probabilities of the sub-distributions are chosen so that at the centre of each sub-distribution, the approximation equals the true Gaussian distribution. (This cannot occur exactly, since we have to respect the constraint that the area under the approximating function must be unity.). Each triangular distribution has a base width of w and a centre at $c_i = iw/2$. A unit area triangular pdf with width w has a height $2/w$. In the overall approximation, this is scaled by the probability q_i. Then to have the approximating pdf equal that of a Gaussian at c_i,

$$q_i = \frac{w}{2} p(c_i), \tag{29}$$

where $p(x)$ is the Gaussian density.

As we have seen, a uniform variate is used to choose the index i such that it occurs with probability q_i. The uniform deviate is actually discrete, with each value occurring with probability in the order of 10^{-10}. for $M = 2^{31} - 1$. Any sub-distributions with probability less than this value will never be chosen. For large w, say equal to 1, this limits c_i to about 6.3 standard deviations before q_i falls below the threshold value. For small w, say equal to 0.01, c_i is limited to about 5.5.

For our example implementation we have chosen to go out to ±6 standard deviations, with different numbers of approximating segments. The Gaussian density is concave downward for $|x|<1$ and concave upward for $|x|>1$. For our implementation, the centres are chosen to be symmetrical about the mean of the distributions and have one of the centres fall at 1 standard deviation. This means that that w is of the form $2/K$, where K is an integer.

4.3 Optimizing the model parameters

Consider approximating the Gaussian density with mixture probabilities. We will minimize the sum of the squared deviations at a set of points. Let the points be written in vector form as

$$\mathbf{x} = [x_0 \ldots x_{N_x-1}]^T. \tag{30}$$

The overall pdf can be written as

$$\mathbf{p}(\mathbf{x}) = \mathbf{A}(\mathbf{x})\mathbf{q}, \tag{31}$$

where $\mathbf{A}(\mathbf{x})$ is an $N_x \times N$ matrix with elements $p_j(x_i)$ and $\mathbf{q} = [q_0 \ldots q_{N-1}]^T$ is the vector of mixture probabilities. The approximating error can then be written as

$$\mathbf{e}(\mathbf{x}) = \mathbf{p}_g(\mathbf{x}) - \mathbf{A}(\mathbf{x})\mathbf{q}. \tag{32}$$

We can formulate the sum of squared errors as $\mathbf{e}(\mathbf{x})^T\mathbf{e}(\mathbf{x})$ and minimize this with respect to the choice of $\mathbf{q}$. However, we also want to add the constraint that the probabilities sum to unity. We add this to the squared error with a Lagrange multiplier λ. Suppressing the dependence on $\mathbf{x}$, the function to minimized is

$$\varepsilon = \mathbf{p}_g^T\mathbf{p}_g - 2\mathbf{p}_g^T\mathbf{A}\mathbf{q} + \mathbf{q}^T\mathbf{A}^T\mathbf{A}\mathbf{q} + \lambda(1 - \mathbf{1}_N^T\mathbf{q}), \tag{33}$$

where $\mathbf{1}_N$ is a vector of N ones. Taking a derivative with respect to $\mathbf{q}$ and setting this to zero gives us a set of equations with $N+1$ unknowns,

$$\mathbf{A}^T\mathbf{A}\mathbf{q} = \mathbf{A}^T\mathbf{p}_g - \frac{\lambda}{2}\mathbf{1}_N. \tag{34}$$

The additional equation needed is the constraint equation $\mathbf{1}^T\mathbf{q} = 1$. Now writing the combined equations,

$$\begin{bmatrix} \mathbf{A}^T\mathbf{A} & \mathbf{1}_N/2 \\ \mathbf{1}_N^T & 0 \end{bmatrix}\begin{bmatrix} \mathbf{q} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{A}^T\mathbf{p}_g \\ 1 \end{bmatrix}. \tag{35}$$

The constraints guarantee only that the sum of the probabilities be one, not that they all be positive. However the form of the problem will assure that they are indeed positive.

Because of the concavity of the Gaussian curve, the maximum error will occur near the middle of the segments. The sampling vector $\mathbf{x}$ was chosen to include the centres of the triangles and points mid-way between them. This leads to a solution that has nearly the minimum peak error. Adding more intermediate points actually increases the peak error. Using a general-purpose (and computationally intensive) minimization routine to minimize the peak deviation does not result in much of a decrease in the peak distortion.

4.4 Approximation error

The approximation error for $w = 0.4$ (61 sub-distributions) is shown in Fig. 6. The peak error is smaller than for the central-limit theorem approach (even with warping, see Fig. 2). The peak error depends on the choice of w. The peak error decreases rapidly with decreasing w as shown in Fig. 7.

The tail probabilities for the approximation are shown in Fig. 8). In this case, the approximation extends to ± 6 with $w = 0.4$. Below this value, the tail probabilities are much more accurate than the simple CLT approach (c.f. Fig. 3).

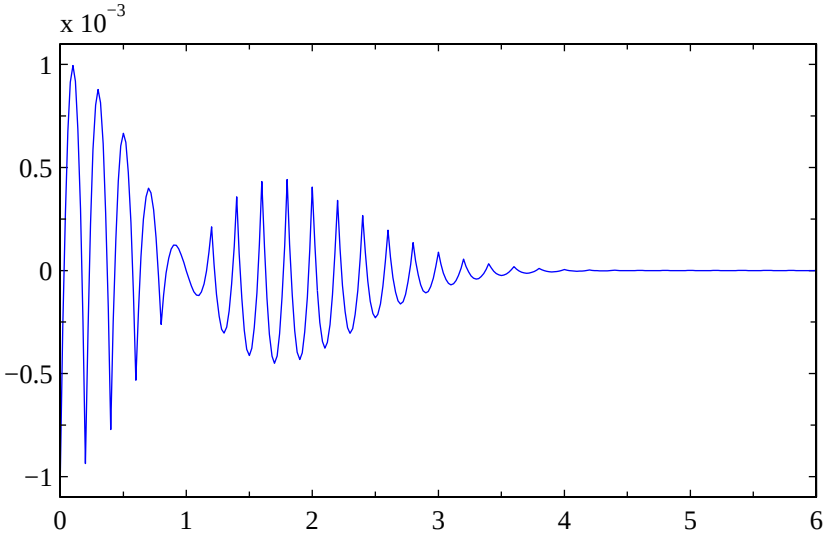

Fig. 6 Difference between the Gaussian density and the piecewise linear approximation for $w = 0.4$.

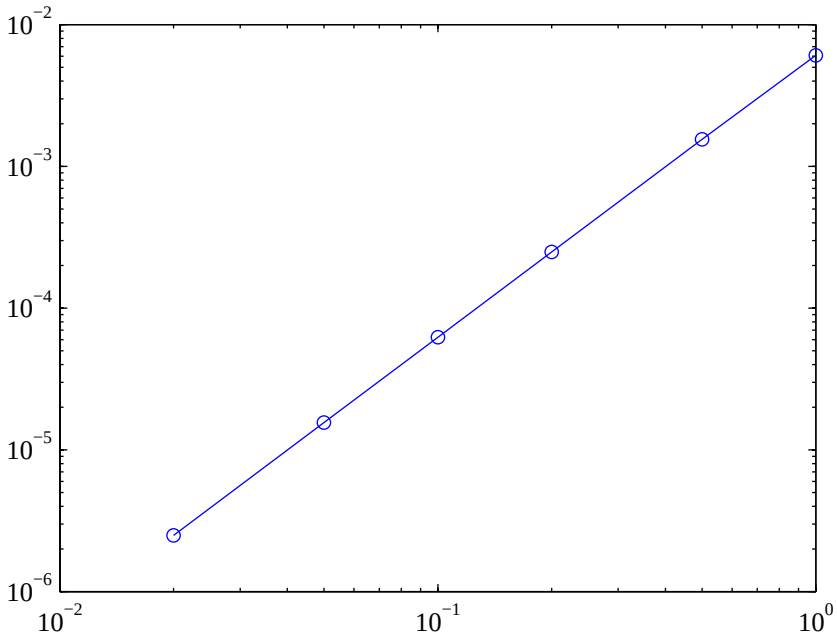

Fig. 7 Peak error in the pdf as a function of w.

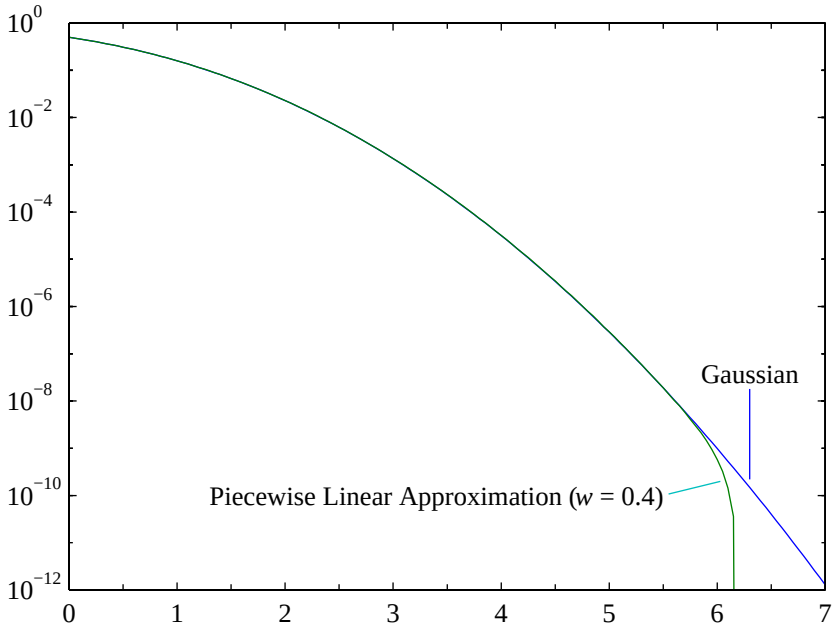

Fig. 8 Tail probability for the piecewise linear approximation.

4.5 Execution time

The computer code for generating a piecewise linear approximation of any pdf is very simple. The modelling of a particular pdf changes only the tabulated values. The accuracy of the approximation depends on the number of sub-distributions used. This affects only the table sizes and not the speed of execution. C-language routines were implemented to assess the speed of execution. Fig. 9 shows the code for the piecewise linear algorithm.

```
#define Ns  61
#define Wh  0.2F

static double Qp[Ns] = {
0.0000000701,  1.0000002218,  2.0000006971,  3.0000021050,
4.0000061097,  5.0000170365,  6.0000456337,  7.0001174142,
8.0002901888,  9.0006889145, 10.0015709960, 11.0034412091,
12.0072405984, 13.0146341180, 14.0284111359, 15.0529836447,
16.0949133810, 17.1633227139, 18.2699605826, 19.4286371286,
20.6537562461, 21.9578101899, 22.9986541831, 23.9172244647,
24.9819145361, 25.9964118982, 26.8108406334, 28.0000000000,
28.9344349895, 29.9470410790, 30.9329056057, 31.9461591492,
32.9382339036, 33.9910814509, 34.8089704203, 35.9807579384,
36.9852901274, 37.9172244647, 38.9995628437, 39.9578101899,
40.6537562461, 41.4286371286, 42.2699605826, 43.1633227139,
```

```
44.0949133810, 45.0529836447, 46.0284111359, 47.0146341180,
48.0072405984, 49.0034412091, 50.0015709960, 51.0006889145,
52.0002901888, 53.0001174142, 54.0000456337, 55.0000170365,
56.0000061097, 57.0000021050, 58.0000006971, 59.0000002218,
60.0000000701 };

static int It[Ns] = {
30, 32, 33, 30, 31, 34, 32, 27,
35, 30, 29, 26, 36, 27, 35, 25,
37, 31, 26, 28, 24, 33, 27, 22,
22, 22, 38, 27, 27, 24, 33, 22,
38, 25, 22, 27, 38, 38, 27, 36,
36, 32, 34, 29, 23, 30, 33, 24,
32, 28, 34, 31, 25, 33, 28, 26,
29, 27, 28, 31, 29 };


float
gTriang (long int *idum)

{
  int j;
  double uN;

/* Alias method to get mixture index */
  uN = Ns * ran1(idum);
  j = (int) uN;
  if (uN > Qp[j])
    j = It[j];

/* Generate a triangular density */
  return (Wh * (ran1(idum) + ran1(idum) + (j - (Ns+1)/2)));
}
```

Fig. 9 C-language code for the piecewise linear approximation method.

Experiments were run on a 600 MHz PC to measure the execution times. The average execution times for generating one random deviate are shown in Table 1. The first row is for the uniform random number generator `rand1` (multiplicative congruential, with shuffle) from [2]. This is the basic uniform random generator used by all of the Gaussian generators. The first Gaussian random number generator is `gasdev`, the implementation of the polar transformation method from [2]. The next is the new piecewise linear approximation, and the last is the sum of 12 uniform deviates (CLT method).

Table 1 Execution times for random number generators

Type	Routine	Execution Time μs
Uniform	`rand1`	0.07
Gaussian	`Gasdev`	0.38
Gaussian	Piecewise Linear	0.37
Gaussian	CLT ($N = 12$)	0.98

The CLT method calls the uniform generator 12 times, and runs about 13 times slower than the uniform generator. The piecewise linear approximation calls the uniform generator 3 times and is about 5 times slower than the uniform generator. The polar transformation method (`gasdev`) calls the uniform generator only once per output value on average. Somewhat surprisingly in spite of having to invoke a square root and a logarithm, it runs only about 5 times slower than the uniform generator. This is perhaps a tribute to the efficient implementation of the transcendental functions in the C-language library.

4.6 Portability and fixed-point considerations

A portable implementation in high-level language is portable if it assumes only minimal constraints on the underlying computer architecture. The underlying discrete uniform random number generator can easily be made portable [2]. The piecewise linear approximation step is table-driven, memoryless, and very portable.

Even a portable routine will not necessarily be bit-exact between different compilers even on the same architecture. For bit-exact implementations, we consider a fixed-point implementation. The core of the uniform generator is already implemented in fixed-point arithmetic. The piecewise linear approach can also be implemented in fixed-point arithmetic, giving a scaled fixed-point output value. Furthermore as noted in Appendix B, the table sizes can be inflated to become a power of 2, further simplifying the fixed-point implementation on binary computers.

4.7 Rectangle-wedge-tail method

A related approach for generating Gaussian variates is the rectangle-wedge-tail method; see for instance [1]. In this approach, the area under the Gaussian pdf is partitioned into rectangular regions, wedge-shaped regions and the tail. The rectangular regions are generated by a scaled and shifted uniform variate. The wedge-shaped regions are generated by an accept-reject approach. However, since most of the area is covered with rectangular regions, the more complicated wedge shaped regions are needed only a small fraction of the time (about 8% of the time in the example given by Knuth [1]). The rectangle-wedge-tail method is computationally efficient *on the average*. The overall program is much more complicated than the other methods considered here.

5 Summary and Conclusions

The piecewise linear approximation method for generating Gaussian variates is simple in structure and does not need transcendental functions (problematic in fixed-point implementations). The results show that it is a viable option for implementation: it is both efficient and accurate. There is a straightforward trade-off between memory (table sizes) and accuracy with no effect of execution time. This method is an excellent candidate for a portable (and possibly fixed-point, bit-exact) implementation of a Gaussian pseudo-random number generator.

6 References

1. D. E. Knuth, *Seminumerical Algorithms*, Third Edition, Vol. 2 of *The Art of Computer Programming*, Addison-Wesley, 1997.
2. W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes in C*, Second Edition, Cambridge University Press, 1992.
3. M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communication Systems*, Plenum Press, 1992.
4. W. Feller, *Introduction to Probability Theory and Its Applications*, Vol. II, Second Edition, John Wiley & Sons, 1971.
5. M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions*, Dover Publications, 1965.
6. ITU-T, *Recommendation P.810, Modulated Noise Reference Unit (MNRU)*, ITU, Geneva, February 1996.

7. ITU-T, *Recommendation P.191, Software tools for speech and audio coding standardization*, ITU, Geneva, November 1996 (includes: Users' Group on Software Tools, *Software Tool Library Manual*).
8. A. J. Walker, "An efficient method for generation discrete random variables with general distribution", *ACM Trans. Math. Software*, vol. 3, pp. 253–256, Sept. 1977.
9. L. Devroye, *Non-Uniform Random Variate Generation*, Springer-Verlag, 1986.

Appendix A. How Far Should the Tails Reach?

The methods for generating Gaussian random variates necessarily generate distributions that are thin in the tails. This by itself does not necessarily hinder their usefulness. We will consider two scenarios.

Audio Noise

Consider generating white noise to add to an audio signal, for instance for testing noise reduction schemes or assessing the performance of speech or audio coding systems. For such purposes, the absence of large (but small probability) noise samples is not a deficiency.

As a concrete example, consider the Gaussian random number generated used in the Modulated Noise Reference Unit (MNRU) [6] to add multiplicative noise for speech quality assessments. Major requirements for a reference implementation are that the random number generator be accurate and portable. The Gaussian noise generator suggested in [7] is table driven. For each output noise sample, eight randomly chosen values from a fixed table of 8192 Gaussian values are combined to generate each output noise sample. This leads to a huge number of different possible output values, but the range of values is limited by the initial values used to populate the table. This is an example of an application where tail accuracy is not of prime concern.

Communications System Simulation

In communication system simulation, the tail probabilities of the noise determine the error rates. Consider a simulation system in which errors occur with the (true) probability p. Further consider evaluating n symbols passing through the system, with the probability of error being independent from symbol to symbol. The probability of k errors in n trials follows a binomial distribution [3],

$$P(k) = \binom{n}{k} p^k (1-p)^{n-k}. \tag{36}$$

The mean number of errors for n trials is pn and the variance is

$$\sigma^2 = \frac{p(1-p)}{n}. \tag{37}$$

The ratio of the standard deviation relative to the mean value is

$$\frac{\sigma}{p} = \sqrt{\frac{1-p}{np}} \simeq \frac{1}{\sqrt{np}}. \tag{38}$$

The latter approximation is for small probability of error. To get an error estimate that has standard deviation that is 10% of the expected number of errors, the expected number of errors (np) should be 100. This means that to simulate a system with an error probability of 10^{-6}, the number of trials should be on the order of 10^8. For a simulation of a complicated system, this number of trials may be unreasonably large. This then limits the minimum probability of error that can be simulated.

For binary transmission with additive Gaussian noise, the error rate is

$$P_e = Q\left(\sqrt{\rho}\right), \tag{39}$$

where $Q(x)$ is the tail probability for a Gaussian density and ρ is the signal-to-noise ratio. In simulating this (admittedly simple) system operating at an error rate of 10^{-6}, errors occur when the noise exceeds 4.7 standard deviations. Simulation of this system operating at this error rate would require generation of Gaussian deviates that extend well beyond this value. This then sets the accuracy requirements for the tails. The total probability of the tails is 10^{-6}. To bring the neglected probabilities below 1% of this value requires that the tails be accurate to about 5.6 standard deviations.

Appendix B. Method of Aliases for Generating Discrete Distributions

Given a uniform random number generator (0 to 1), consider the generation of N random values with given probabilities, $q_0,\ldots,q_{N-1}$. The alias method of A. J. Walker [8], trades off the non-uniform quantization problem for a uniform quantization problem and additional comparison. L. Devroye [9] has an interpretation of the problem in terms of partitioning a unit square.

Consider the unit square shown in Fig. 10. The square is partitioned into vertical strips, each of area $1/N$. Furthermore, each strip is divided into two parts, with the lower part of strip j having area Q_j/N. The index associated with the lower part is j itself. The upper part of strip j has an index I_j associated with it. The generation of the discrete variable can then be done as follows. Generate two uniform random deviates, u and v.. These define a point (u,v) in the unit square. To locate the strip, uniformly quantize u,

$$j = \lfloor Nu \rfloor. \tag{40}$$

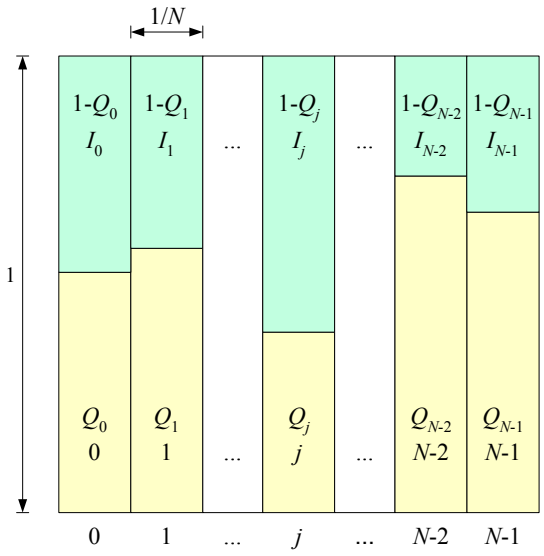

Fig. 10 Unit square partitioned into vertical strips of area 1/*N*.

In strip k, we need to determine whether v is below or above the dividing line. This means that v is compared to Q_k,

$$l = \begin{cases} j & v < Q_j \\ I_j & v \geq Q_j \end{cases}. \tag{41}$$

When properly set-up, the index l will take on the value i with probability q_i.

The task is to construct the partitions of the table. First note that some of the probabilities q_i will be less than $1/N$, while others will be greater than or equal to $1/N$. Group the probabilities into two groups, one with those probabilities that are less than $1/N$, the remainder in the other group. Choose one from the group of smaller probabilities, say q_j. In strip j, set $Q_j = q_j$. Since q_j is smaller than $1/N$, it will take up only part of strip j. The index of the lower part of strip j is set to j itself. We are now finished with q_j.

Now select one of the probabilities that is larger than $1/N$, say q_m. The length of the upper part of strip j, is smaller than this value. Nonetheless, we label the upper part of strip j with index m, i.e., we set $I_j = m$. One strip is filled We must now reduce q_m by the length of the upper part of strip j,

$$q_m \leftarrow q_m - (1 - q_j). \tag{42}$$

Having done this, we place the new value of q_m into one of the two groups of probabilities: those smaller than $1/N$ and those larger than $1/N$.

The process can now be repeated for the remaining strips. When finished, each part of the unit square will be identified with an index. A given index i may occur in several different parts of the square, but the fraction of the square labelled with index i will be exactly q_i.

The procedure above was described in terms of generating two uniform random variables. One can note, however, that $k = \lfloor Nu \rfloor$ is a discrete equiprobable value and that

$v = Nu - k$ is a uniform random value in $[0,1)$. Then we can operate with just a single uniform random deviate.

This single uniform deviate approach can be viewed in terms of a line from 0 to N as shown in Fig. 11. In this figure, the strips from the previous figure are lined up onto a line of length N. The uniform deviate chooses a point on the line. The integer part of the uniform deviate determines which unit segment the value lands in. This segment number lets us choose the appropriate threshold value. The threshold value for the unit segment starting at j is $j + Q_j$.

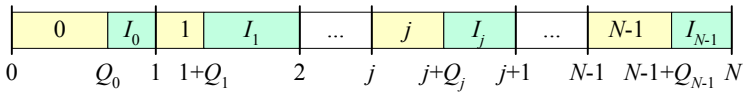

Fig. 11 Line segment divided into unit segments.

The random variate generation algorithm can be expressed shown in Fig. 12. The input is u, a uniform random variate. Two tables of size N are necessary. The first contains the values $i + NQ_i$. The second is the index array containing the indices for the second parts of the unit segments.

```
n = floor(N*u);
if (u < Qp(n+1)
  m = n;
else
  m = I(n+1);
end
```

Fig. 12 Code fragment for calculating a discrete random index.

The description above suggested an explicit method to generate the tables. Knuth [1] gives a modified procedure for setting up the tables. This method sorts the probabilities such that the indices of the smallest probability and largest probabilities are used to populate a strip at any step. In this way, it attempts to maximize the probability that $v < Q_j$ (no table lookup for the index aliases). A procedure written in Matlab for generating the table values

is shown below in Fig. 13. The input is a vector of probabilities. The output is a table of thresholds ($i + NQ_i$) and a table of index aliases.

```
function [Qp,It] = AliasTable(q)

Pn = q;
N = length(q);

Qp = zeros(1,N);          % pre-allocate space
It = zeros(1,N);
for(i = 0:N-1)
   [Ps, Is] = sort(Pn); % ascending order
   Is = Is - 1;          % [0,N-1]
   j = Is(i+1);          % index of smallest
   k = Is(N-1+1);        % index of largest

% Set table values
   Qp(j+1) = j + N * Pn(j+1);
   It(j+1) = k;          % [0,N-1]

% Update probabilities
   Pn(k+1) = Pn(k+1) - (1/N - Pn(j+1));
   Pn(j+1) = -1;
end
```

Fig. 13 Matlab code for generating alias table values.

Other considerations

The alias method requires a multiplication by table size and the evaluation of a floor function (integer part of a positive number). For computer architectures based on binary arithmetic, these operations can be simplified if the table size is a power of 2. This is easily accommodated by introducing additional sub-distributions with zero probability.

Application to quantization

The alias method correctly generates indices with given probabilities. It is an alternate to binary search. The latter algorithm can be viewed as implementing a non-uniform quantizer. In generating random indices, it matters not which index some particular range of the uniform variate is associated with, only that the indices occur with the correct probability.

In the non-uniform quantization problem, we have to find the index corresponding to a particular input value. Non-uniform quantizers can be implemented with a transformation to a domain in which a uniform quantizer can be used (a companding function — named for compression and expanding). Or barring that, using a binary search process. The one-dimensional view of the alias method gives us an alternate viewpoint.

Consider for simplicity, the problem of quantizing a value x taking on values in the interval $[0,1]$. This interval is then partitioned into segments with labelled indices. Suppose we choose N such that the smallest segment corresponding to a given index is smaller than $1/N$. Now scale the input value by N. No segment of unit length of the scaled variable will contain more than one decision boundary. We can now use the processing of the alias method to set up tables. Non-uniform quantization can then proceed by first identifying the unit segment and then comparing the value with the threshold for that segment.

For non-uniform quantizers with a large spread in interval sizes, a non-linear function can be used to decrease the spread. The function need not be exactly the companding function associated with the non-uniform quantizer. It serves only to reduce the number of intervals (table size).

Appendix C. Parameters for the Piecewise Linear Approximation

The figure below shows Matlab code that can be used to calculate the mixture probabilities for a piecewise linear approximation to a Gaussian pdf.

```
function [q,pPar] = qms(Wh, Cmax)
% [q,pPar] = qms(Wh, Cmax)
% Wh - distance between centres
% Cmax - largest centre
% Solve for the mixture probabilities for a piecewise linear
% approximation to a Gaussian pdf.  The sub-distributions are
% uniformly spaced.

Gpdf = inline('1/sqrt(2*pi) * exp(-x.^2 / 2)');

% Generate linearly spaced values
pPar.C = -Cmax:Wh:Cmax;
pPar.Wh = Wh;

N = length (pPar.C);
Nx = 2*N - 1;
x = linspace(-Cmax, Cmax, Nx)';

% Optimum q
q = qopt(Amat(x, pPar), Gpdf(x));

%=====
function q = qopt (A, p)
% Solve for the mixture probabilities that minimize the sum of
% the squared errors at given points.
%  A, Nx by N pdf mixture matrix; contribution to the overall
%     pdf at point i from sub-distribution j
%  p, Nx column vector of target pdf values

% Solve for the q which minimizes the sum of squared errors.
% The error is
%   e(x) = p(x) - A(x)*q.
% The sum of the squared errors is
%   E = e'*e
%     = p'*p - 2*p'*A*q + q'*A'*A*q.
% Setting the derivative with respect to q to zero, gives
% the minimum squared error solution,
%   A'*A*qopt = A'*p.
% However, the value of q must be normalized such that the
% total probability is unity.  We impose this constraint with
% a Lagrange multiplier,
%   E = p'*p - 2*A'*p*q + q'*A'*A*q + u*(1 - S'*q),
% where S is a vector of ones.  Setting the derivative with
% respect to q to zero,
%   A'*A*q + u*S*q/2 = A'*p.
```

```
% Also setting O'*q = 1, these can be combined into a single
% set of equations,
%   [ A'*A | S/2 ] [ q ]   [ A'*p ]
%   [ -------   ] [ - ] = [ ---- ] .
%   [   S' |  0  ] [ u ]   [   1  ]
N = size(A,2);
S = ones(N,1);
qu = [[(A'*A); S'], [0.5*S; 0]] \ [A'*p; 1];

q = qu(1:N);
if (any(q < 0))
   error ('Invalid (negative) probability');
end
if (abs(sum(q)-1) > 1e-10)
   error ('Invalid probability sum');
end

%=====
function A = Amat(x, pPar)
% A = Amat(x, pPar)
% Form the pdf mixture matrix A, where A(i,j) is the
% contribution of sub-distribution j to the overall pdf
% at frequency x(i)

% The overall pdf is
%   pa(x) = A(x)*q,
% where q is a vector of mixture probabilities, with q(j)
% representing the probability of using sub-distribution ps(j).
%   A(i,j) = ps(x(i),pPar)

N = length(pPar.C);
Nx = length(x);

A = zeros(Nx, N);
for (j = 1:N)
   A(:,j) = ps(x, pPar.C(j), pPar.Wh);
end

%=====
function px = ps(x, C, Wh)
% x, vector of values
% C, Wh are scalars (center & half-width) of the triangular
%       pdf

px = zeros(size(x));
Ind = (abs(x - C) < Wh);
px(Ind) = (1 - abs(x(Ind) - C) / Wh) / Wh;
```

Fig. 14 Matlab code to calculate the parameters of an approximation to a Gaussian pdf.