Generating Gaussian Pseudo-Random Variates

Peter Kabal

Department of Electrical & Computer Engineering

McGill University

Version 2: 2019-07-05

http://WWW-MMSP.ECE.McGill.CA/MMSP

Revision History:

2000-02	v1	Initial release
2000-10-02	v1a	Revision
2019-07-05	v2	Creative Commons licence. Corrected Eq. (14); more on non-uniformly-spaced linear segments; error weighting for mixture probabilities

Table of Contents

Generating Gaussian Pseudo-Random Variates

1 Introduction

This report examines low-complexity methods to generate pseudo-random Gaussian (normal) variates. 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, yet does not require the calculation of transcendental functions.

The methods considered here map uniform distributions to create Gaussian variates. This report investigates the effect of the use of discrete variates, particularly in the tails of the Gaussian distribution. In addition, a new interpretation of the method of aliases leads its application to non-uniform quantization.

This report is an update of [1]. That report first introduced the piecewise-linear approximation approach for generating pseudo-random Gaussian variates. This version elaborates on the use of non-uniformly-spaced linear segments and introduces an error weighting for determination of the mixture probabilities.

2 Uniform Variates

2.1 Continuous Uniform Distribution

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

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

The mean and standard deviation for this distribution are,

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

2.2 Discrete Uniform Distribution

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

Consider the multiplicative congruential method for generating a uniform variate [2][3][4]. The basic procedure takes the form

$$x[k] = \text{mod}(ax[k-1], M). \tag{3}$$

where a is a carefully chosen multiplier, $x[k-1]$ is a previous (non-zero) variate and M is an appropriate modulus. All values are integers. The book *Numerical Recipes* [3], 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 [3, 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 [3]).

It is common for uniform random number generators to return uniform variates as floating- point numbers between 0 and 1. The routine given in [3] computes

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

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

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

The value $x_{ud}[k]$ takes on discrete values. Since each value of $x_{ud}[k]$ is equi-probable, the mean and variance of $x_{ud}[k]$ are

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

3 Gaussian Variates

Techniques for generating Gaussian variates from uniform variates are described in [2]. We consider two approaches for which computer programs are widely available

3.1 Central Limit Theorem

The Central Limit Theorem of probability states that a 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 [5]. 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 variates

Consider adding N independent (continuous) uniform variates,

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

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

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

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

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

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 probability density function (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 $X_c(s,N)/s$,

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

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,

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

The resultant pdf and cdf are

$$\begin{aligned} p_{nc}(x,N) &= \frac{1}{\sigma_c}p_c(\sigma_c x + m_c, N), \\ F_{nc}(n,N) &= F_c(\sigma_c x + m_c, N) \end{aligned} \tag{15}$$

The Berry-Esseen Theorem [5] bounds the rate of convergence to a Gaussian distribution. For the sum of N zero-mean uniform variates, the between the cdf of the sum and the Gaussian cdf (denoted as $\mathcal{N}(x)$), is bounded as,

$$|F_{nc}(x,N) - \mathcal{N}(x)| < \frac{C\rho_{uc}}{\sigma_{uc}^3\sqrt{N}}, \tag{16}$$

where ρ_{uc} is the third absolute central-moment.[1] Feller [5] gives $C = 3$ as the constant.[2] The Berry-Esseen Theorem shows that the error bound 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.

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.

---

[1] For the uniform distributions used here, $\sigma_{uc}^2 = 1/12$ and $\rho_{uc} = 1/32$.

[2] Recent work on refining the value of the bounding constant C has resulted $C < 0.4748$, see [6].

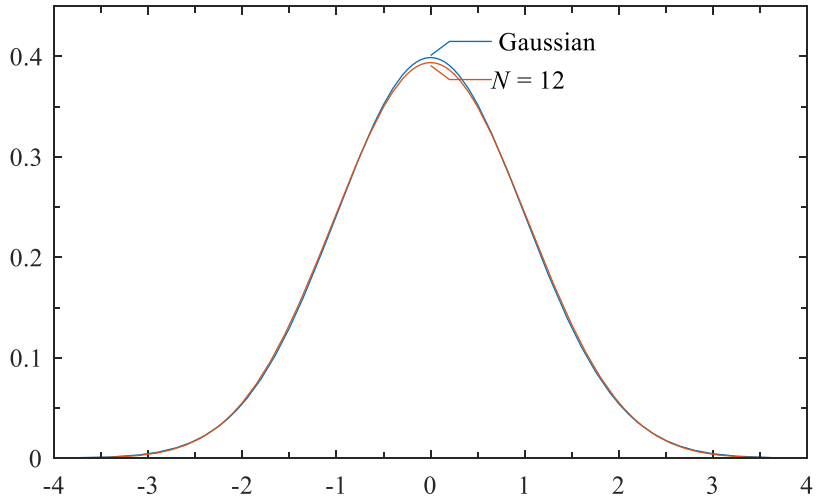

Fig. 1 Probability density function for a sum of $N = 12$ uniform variates.

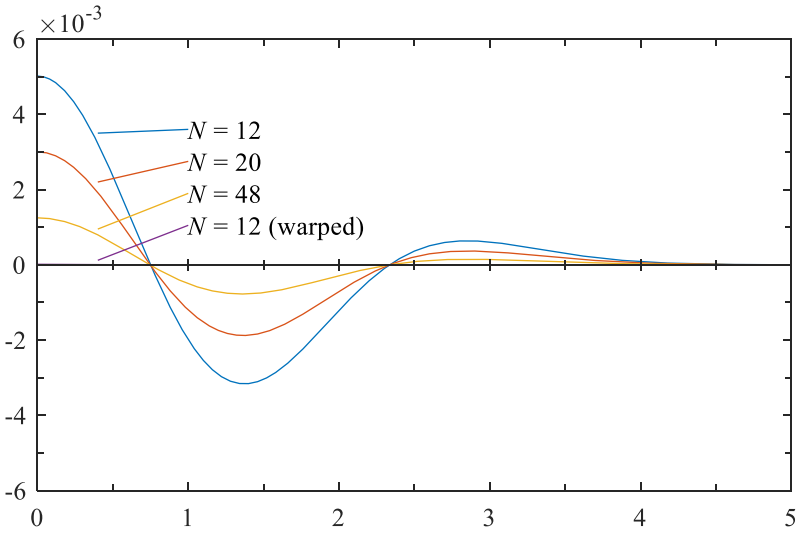

Fig. 2 Difference between the Gaussian pdf and the pdf of a sum uniform variates.

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,

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

For $N = 12$, an anti-symmetric warping polynomial (with only odd-numbered coefficients) is [7, Section 26.8],

$$\begin{aligned} a_1 &= 0.98746, & a_3 &= 3.9439 \times 10^{-3}, \\ a_5 &= 7.474 \times 10^{-5}, & a_7 &= -5.102 \times 10^{-7}, \\ a_9 &= 1.141 \times 10^{-7}. \end{aligned} \tag{18}$$

This polynomial function is a non-linear stretch of the deviates with the effect of increasing the reach of the pdf – the tail now goes out to 8.26 standard deviations. The warping function improves the match to the Gaussian pdf. The improved match is shown in Fig. 2, but at the scale presented, the error is indistinguishable from the zero line – the largest error is 1.4×10^{-5}.

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.

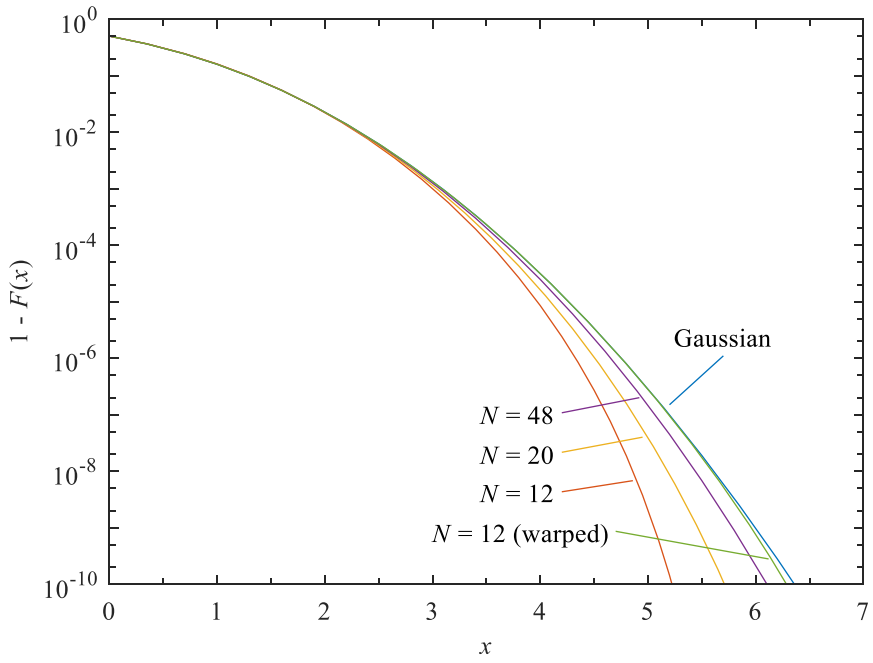

Fig. 3 Tail probability for the sum of uniform variates.

3.1.2 Sum of discrete uniform variates

For discrete valued uniform variates, the sum is

$$S_d = \sum_{k=0}^{N-1} x_{ud}[k]. \tag{19}$$

where $x_{ud}[k]$ is the integer-valued uniform variate that is used to calculate $x_d[k]$. For the following, assume that the uniform variate takes on equiprobable values in $[0, M-1]$. The sum takes on values between 0 and $N(M-1)$.

The uniform probability mass function can be written in terms of the difference between two discrete unit step functions,

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

where the discrete unit step function is defined as,

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

The generating function (z-transform) for $P_{ud}[n]$ is

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

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

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

The inverse transform gives the probability that $S_d = n$,

$$P_d[n] = P_r(S_d = n) = \frac{1}{M^N} \sum_{k=0}^{N} (-1)^k \binom{N}{k} \binom{n - kM + N - 1}{n - kM} u[n - kM]. \tag{24}$$

The probability mass function is symmetric ($P_d[n] = P_d[N(M-1) - n]$), finite length (zero outside of $[0, N(M-1)]$), and sums to one. The number of sum terms in Eq. (24) is at most $N + 1$. The sum (before normalization) calculates the number of combinations of N uniform variates that add to n.

The cumulative probability can be calculated by summing $P_d[n]$ for n running from $-\infty$ to n, or as the inverse transform of $X_d(z)/(1 - z^{-1})$,

$$F_d[n] = P_r(S_d \le n, N) = \frac{1}{M^N} \sum_{k=0}^{N} (-1)^k \binom{N}{k} \binom{n - kM + N}{n - kM} u[n - kM]. \tag{25}$$

The cdf, $P_r(S_d \le \lfloor x \rfloor, N)$, is a piecewise constant, non-decreasing function.

The total number of combinations (M^N) becomes enormous for practical values of M and N (say $M = 2^{31}$ and $N = 12$). Even for such cases, the end values of the probability mass function (before normalization by M^N) can be determined exactly with, say, 64-bit arithmetic.

The analysis above was done for the sum of integer-valued variates, with the variates taking on values $[0, M-1]$. The case of uniform variates in $[1, M-1]$ requires replacing M by $M-1$ and noting that the smallest sum with non-zero probability is now N.

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 variates, one uniform variate is transformed to a Rayleigh variate and a second uniform variate is transformed to a uniform angle. Two Gaussian variates, $y_1$and y_2 are 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 sinusoid values [3]. However, the number of iterations to attain an acceptance is unbounded, although small on average.

3.2.1 Polar transformation of discrete uniform variates

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 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 increases above 6 standard deviations.

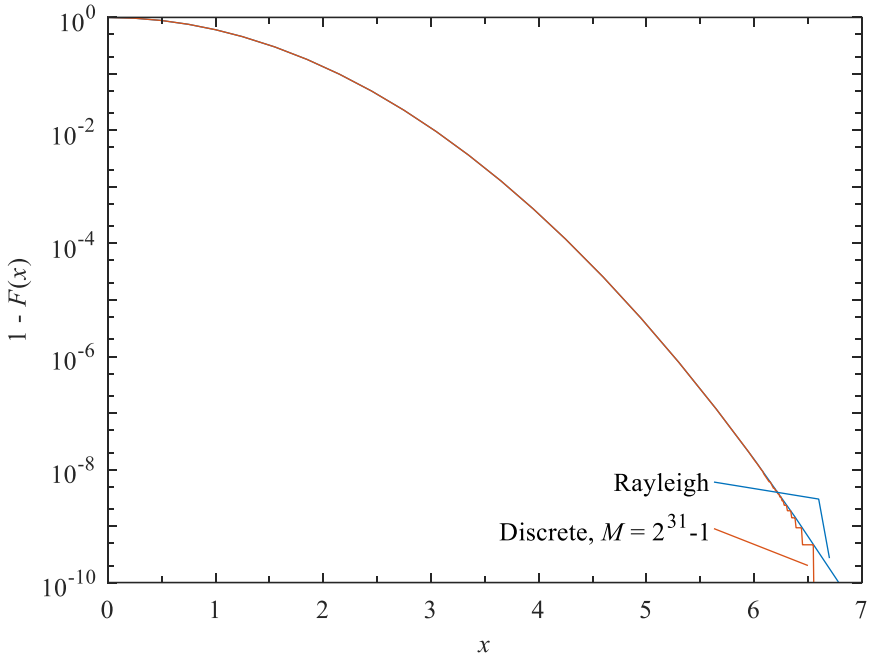

Fig. 4 Tail probability for a transformation of discrete uniform variates to a Rayleigh distribution.

3.3 CLT versus Polar Transformation

The Central Limit Theorem approach and the polar transformation method provide Gaussian variates 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 some 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

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

With this formulation, the overall pdf is expressed as the weighted sum of pdf's. The mixture weight q_i is 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, note that a triangular pdf can be generated as the sum of two uniform pdf's. By overlapping the triangular distributions, the overall pdf has piecewise linear segments. Fig. 5 shows a (low resolution) triangular approximation to the Gaussian density. The steps in generating the (approximate) Gaussian variate are as follows.

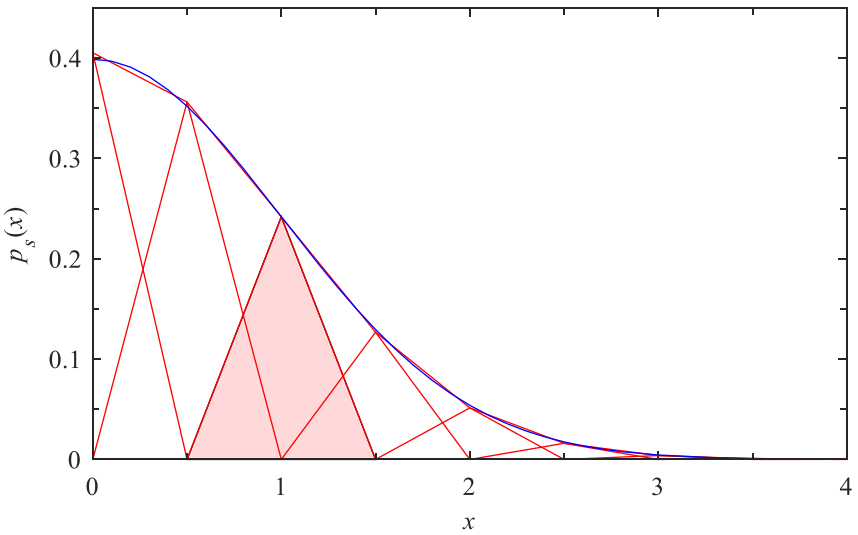

Fig. 5 Triangular pdfs used to approximate a Gaussian density.

1. Determine which triangular pdf to use – 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 variate, 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 $\lceil \log_2(N) \rceil$.

4.1.1 Method of Aliases

To generate the index i, the method of aliases is used. In this procedure, the segments are rearranged so 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.

4.1.2 Generalization of the triangular distribution

The piecewise linear approximation shown above uses equal width triangular sub-distributions. The variates for the individual triangular sub-distributions can be generated a sum of two independent uniform variates.

A procedure to generate a variate with non-symmetric triangular pdf is described in Appendix C. Let u_1 and u_2 be two uniform variates. Form the sum,[3]

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

The sum has a triangular pdf in the interval $[0, 1]$ with the apex of the triangle at α. For $\alpha = 1/2$, the pdf of the sum is symmetric about α. Using non-symmetric triangles allows for the piecewise linear approximation to use variable length segments.

4.2 Choosing the Model Parameters

Consider a piecewise linear approximation, anchored at the N values $x_t(i)$, for $i = 1, \ldots, N$. Each value $x_t(i)$ value locates the apex of a triangle with base extending from $x(i-1)$ to $x(i+1)$. The $x_t(\cdot)$ vector is extended with $x_t(0)$ and $x_t(N+1)$ to specify the ends of the

[3] This method for generating a non-symmetric triangular pdf appears in [1], without attribution or proof, and has been independently described in [8].

first and last triangles. The $x_t(\cdot)$ values will be referred to as the anchor points of the piece-wise linear approximation.

Let the $x_t(\cdot)$ values be symmetric about zero ($x_t(i) = -x_t(N+1-i)$) and let N be odd so that $x_t((N+1)/2) = 0$. Further, let outermost triangles have apices at $\pm C_{\max}$.

The modelling of the Gaussian pdf with linear segments involves choosing parameters for the model ($x_t(\cdot)$ and $q(\cdot)$). For motivation, suppose the probabilities of the sub-distributions are chosen so that at the anchor points, 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.). Triangular distribution i has a base width of $w_i = x_t(i+1) - x_t(i-1)$ and apex at $x_t(i)$. In the overall approximation, the triangular pdf i is scaled by the probability q_i. Then to have the approximating pdf equal that of a Gaussian at c_i,

$$q_i = \frac{w_i}{2} p_g(x_t(i)). \tag{29}$$

where $p_g(x)$ is the Gaussian probability density.

4.3 Optimizing the Model Parameters

Consider approximating the Gaussian density with N 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

$$\mathbf{x} = [x_0, x_1, \ldots, x_{N_x-1}]. \tag{30}$$

The pdf evaluated at $\mathbf{x}$ 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 $p_j(x_i)$ (the pdf of sub-distribution j evaluated at x_i) and $q = [q_0, \ldots, q_{N-1}]^T$ is the vector of mixture probabilities. The approximating error can then be written as

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

where $\mathbf{p}_g(\mathbf{x})$ is the Gaussian pdf evaluated at $\mathbf{x}$. The sum of squared errors is $\mathbf{e}(\mathbf{x})^T\mathbf{e}(\mathbf{x})$. This will be minimized with respect to the choice of $\mathbf{q}$, with the constraint that the probabilities sum to unity. This constraint is added to the squared error with a Lagrange multiplier λ. Suppressing the dependence on $\mathbf{x}$, the function to be 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 is the constraint equation $\mathbf{1}_N^T\mathbf{q} = 1$. 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 constraint guarantees only that the sum of the probabilities is one, not that they are positive.

Because of the concavity/convexity of the Gaussian curve, the maximum error will occur near the middle of the linear segments. The sampling vector $\mathbf{x}$ is chosen to include the anchor points and the points mid-way between the anchor points.

4.3.1 Error Weighting

The procedure above minimizes the unweighted mean-square error. As such the error in the middle region dominates the error in the tails. For instance, the mixture probabilities in the outer reaches of the approximation can be set to zero without incurring a large change in the overall mean-square error. The relative importance of the mid-region and the tails can be changed with a weighting matrix. Let the weighting matrix be an $N_x \times N_x$ diagonal matrix $\mathbf{W}$. The error then is

$$\begin{aligned} \mathbf{e} &= \mathbf{W}\big(\mathbf{p}_g - \mathbf{A}\mathbf{q}\big), \\ &= \mathbf{p}_{gw} - \mathbf{A}_w\mathbf{q}. \end{aligned} \tag{36}$$

The rest of the algorithm proceeds as before using the weighted matrix and weighted Gaussian pdf vector.

One possible weighting has the diagonal elements of $\mathbf{W}$ set as follows,

$$W_{ii} = \left(\frac{1}{p_g(x_t(i))}\right)^{w_x}. \tag{37}$$

The weight values are constant when $w_x = 0$. The error criterion becomes a relative weighting when $w_x = 1$.

A Matlab implementation of the procedure to find the mixture probabilities **q** is shown in Appendix D.

4.4 Uniformly-Spaced Anchor Points

The case of uniformly-spaced symmetric triangles gives uniformly spaced anchor points $x_t(\cdot)$. The specification of the triangles is completed by specifying N and $C_{\max}$ (centre of the outermost triangle). The Gaussian density is concave downward for $x < 1$ and concave upward for $x > 1$. For our example implementation, the anchor points are chosen so that one lies at the mean and two lie at ± 1 (standard deviations).

For an example implementation we have chosen to go out to ± 6 standard deviations, with different numbers of approximating segments. It was found that for uniformly-spaced anchor points, the approximation error is not strongly dependent on the weighting. The weight parameter was set to $w_x = 0.5$.

The approximation error for $N = 61$ (anchor points separated by 0.2) is shown in Fig. 6. The peak error is smaller than for the central-limit theorem approach (see Fig. 2). The tail probabilities for the approximation are shown in Fig. 7. In this case, the approximation extends to ± 6. The tail probabilities are more accurate than the CLT approach (c.f. Fig. 3).

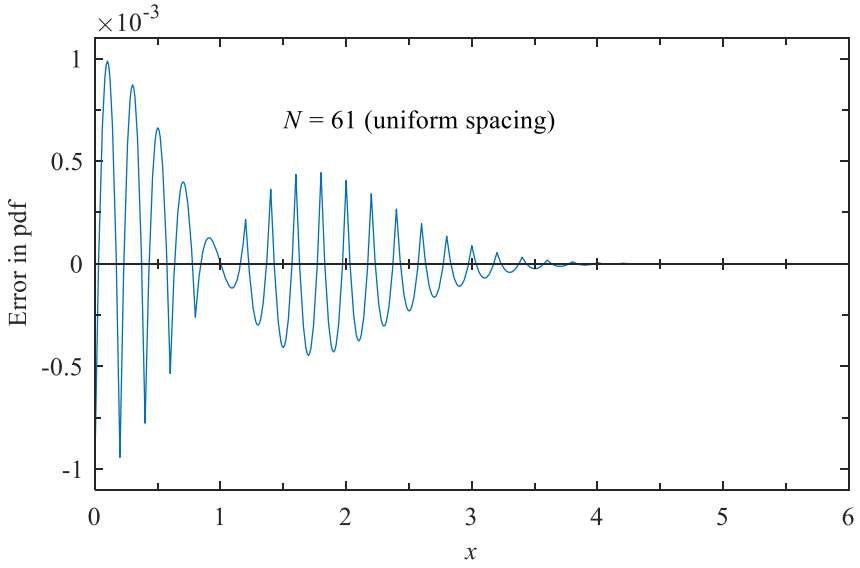

Fig. 6 Difference between the Gaussian density and the piecewise linear approximation for 61 uniformly-spaced anchor points.

The peak error depends on the choice of N. The peak error decreases with increasing N as shown in the upper curve in Fig. 8.

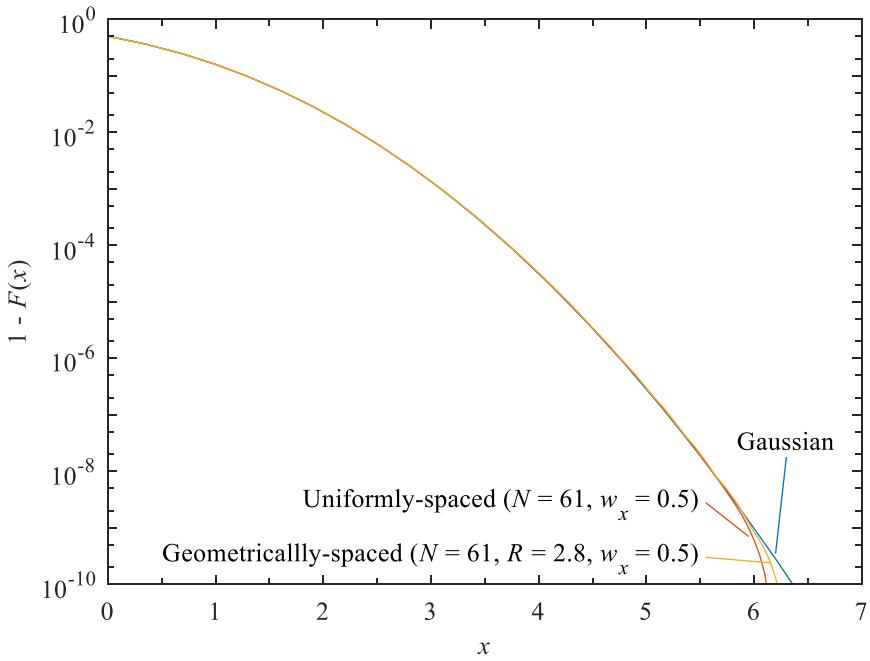

Fig. 7 Tail Probability for a piecewise linear uniformly-spaced approximation.

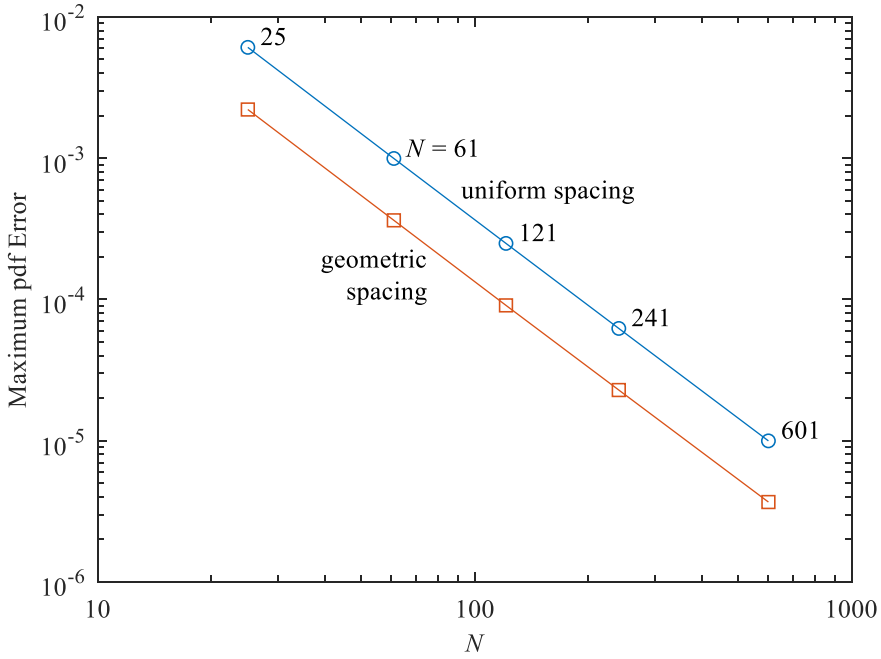

Fig. 8 Peak error in the pdf as a function of N.

Geometrically-Spaced Anchor Points

With uniformly-spaced anchor points, the error in pdf is largest near the origin. Using anchor points which are smaller near the origin can be a benefit. Let the spacing between anchor points increase geometrically on either side of the origin. The geometric spacing will be specified by R, the ratio of the largest spacing to smallest spacing.

With a constant weight ($w_x = 0$), experimentation using $N = 61$, shows that a geometrically-increasing spacing with $R = 2.8$ brings down the error near the origin, but at the expense of more error in the tail. Solving for the mixture probabilities using the method described in §4.2 gives negative values for the probabilities of the outermost sub-distributions. Instead, an alternate solution method as shown in Appendix D was used with the mixture probabilities are constrained to be non-negative.

Using a weighting with $w_x = 0.5$, the procedure given in §4.2 gives mixture probabilities are positive and the tail is well represented. Fig. 9 shows the error in pdf. The geometric spacing has brought down the maximum error by about a factor of 2.5. The tail probabilities are shown in Fig. 7. The maximum error in the pdf approximation as a function of N is shown in the lower curve in Fig. 8 ($R = 2.8$ and $w_x = 0.5$).

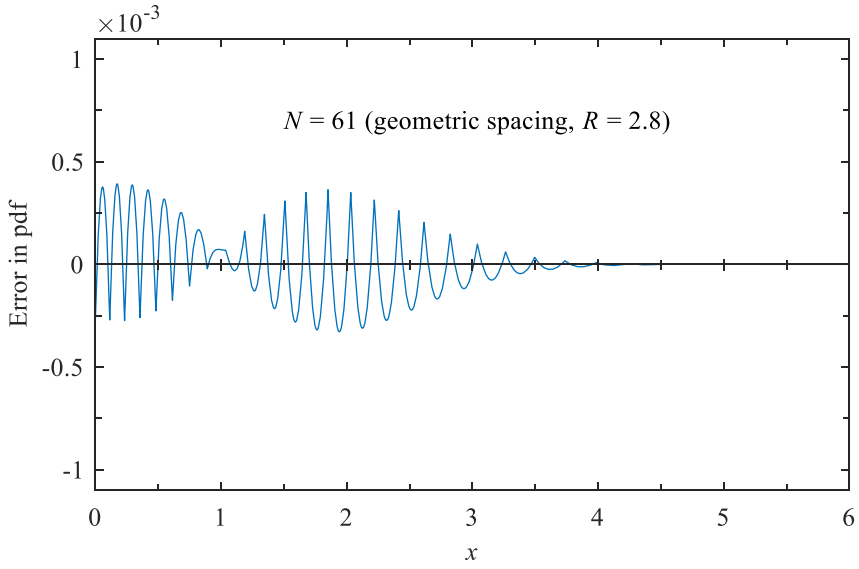

Fig. 9 Difference between the Gaussian density and the piecewise linear approximation for 61 geometrically spaced anchor points.

4.5 Execution Time

The computer code for generating a piecewise linear approximation of any pdf is 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.[4] This affects only the table sizes and not the speed of execution. C-language routines were implemented to assess the speed of execution. Appendix E.1 shows the code for the piecewise linear algorithm with uniformly-spaced anchor points.

The use of non-uniformly-spaced intervals requires an additional table (the $x_t(\cdot)$ values) and additional logic to implement the non-symmetric triangular pdf's. The C-language implementation is shown in Appendix E.2

Tests were run on a 3.4 GHz 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 `ran1` (multiplicative congruential, with shuffle) from [3]. 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 [3]. The next is the sum of 12 uniform deviates (CLT method). The last two rows are for piecewise linear approximations.

Table 1 Execution times for random number generators

Type	**Routine**	**Execution Time** ns
Uniform	`ran1`	5.3
Gaussian	`gasdev`	21.1
Gaussian	CLT ($N = 12$)	60.9
Gaussian	Piecewise Linear Uniformly-Spaced	21.2
Gaussian	Piecewise Linear Non-Uniformly-Spaced	28.9

[4] The mixture probabilities decrease as the number of sub-divisions increases. As noted in Appendix B.3, the resolution of the uniform deviates may have to be increased when individual mixture probabilities become small.

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 4 times slower than the uniform generator. This is perhaps a tribute to the efficient implementation of the transcendental functions in the C-language library.

The CLT method calls the uniform random number generator 12 times and runs about 12 times slower than the uniform generator.

The piecewise linear approximations call the uniform generator 3 times. For uniformly-spaced anchor points, the code is about 4 times slower than the uniform random number generator. For non-uniformly-spaced anchor points, the additional computations result in code that runs 5.5 times slower than the uniform generator.

4.6 Portability and Fixed-Point Considerations

An implementation in high-level language is portable if it assumes minimal constraints on the underlying computer architecture. The underlying discrete uniform random number generator can easily be made portable [3]. The piecewise linear approximation step is table-driven, memoryless, and very portable.

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 set to be 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 [2]. In this approach, the area under the Gaussian pdf is partitioned into rectangular regions, wedge-shaped regions, and the tail. In a suggested implementation, the method of aliases is used to determine which one of 64 sub-distributions to use.

The rectangular regions are generated by a scaled and shifted uniform variate. The wedge-shaped regions are generated by an accept-reject approach using two uniform variates. 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 [2]). The number of iterations in the accept-reject approach is unbounded. 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, has a constant workload, 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.

References

1. P. Kabal, *Generating Gaussian Pseudo-Random Deviates*, Technical Report, Department of Electrical & Computer Engineering, McGill University, October 2000, on-line document http://www-mmsp.ece.mcgill.ca/MMSP/Documents/Reports).
2. D. E. Knuth, *Seminumerical Algorithms*, Third Edition, Vol. 2 of *The Art of Computer Programming*, Addison-Wesley, 1997.
3. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C*, Second Edition, Cambridge University Press, 1992.
4. M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communication Systems*, Plenum Press, 1992.
5. W. Feller, *Introduction to Probability Theory and Its Applications*, Vol. II, Second Edition, John Wiley & Sons, 1971.
6. Wikipedia, *Berry-Esseen theorem*, on-line document, https://en.wikipedia.org/wiki/Berry%E2%80%93Esseen_theorem (retrieved2019-05-01).
7. M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions*, Dover Publications, 1965.
8. W. E. Stein and M. F. Keblis, "A New Method to Simulate the Triangular Distribution", *Mathematical and Computer Modelling*, vol. 49, pp. 1143–1147, March 2009.
9. ITU-T, *Recommendation P.810, Modulated Noise Reference Unit (MNRU)*, ITU, Geneva, February 1996.
10. ITU-T, *Recommendation P.191, Software tools for speech and audio coding standardization*, ITU, Geneva, November 1996 (includes: *Software Tool Library Manual*).
11. A. J. Walker, "An efficient method for generating discrete random variables with general distributions", *ACM Trans. Math. Software*, vol. 3, pp. 253–256, Sept. 1977.
12. L. Devroye, *Non-Uniform Random Variate Generation*, Springer-Verlag, 1986.
13. H. A. David and H. N. Nagaraja, *Order Statistics*, Third Edition, Wiley-InterScience. 2003.

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) [9] 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 [10] 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 a 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 [4],

$$P(k) = \binom{n}{k} p^k (1-p)^{n-k} . \tag{38}$$

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

$$\sigma^2 = \frac{p(1-p)}{n}. \tag{39}$$

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

$$\frac{\sigma}{p} = \sqrt{\frac{1-p}{np}} \approx \frac{1}{\sqrt{np}}. \tag{40}$$

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 smallest probability of error that can be simulated.

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

$$P_e = Q(\sqrt{\rho}). \tag{41}$$

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

B.1 Partitioning a Square

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 Walker [11], trades off the non-uniform quantization problem for a uniform quantization problem and additional comparison. Devroye [12] has an interpretation of the problem in terms of partitioning a unit square.

A unit square is shown in Fig. 10. The square is partitioned into vertical strips, each of area $1/N$. 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. The index associated with he upper part of strip j is I_j.

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,

$$j = \lfloor Nu \rfloor. \tag{42}$$

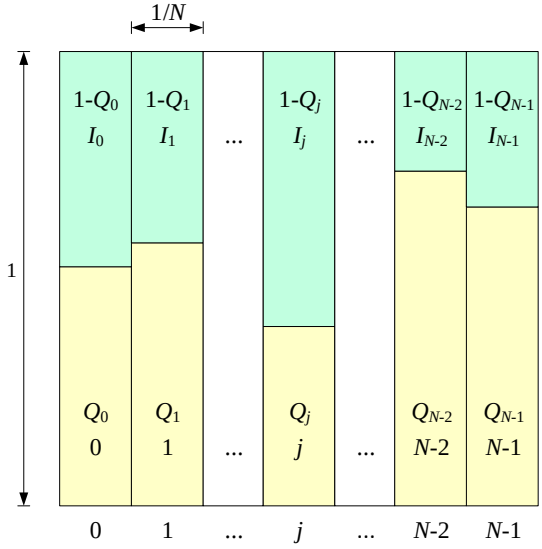

Fig. 10 Unit square partitioned into vertical strips of area $1/N$..

In strip j, we need to determine whether v is below or above the dividing line,

$$l = \begin{cases} j, & v < Q_j, \\ I_j, & v \geq Q_j. \end{cases} \tag{43}$$

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, those probabilities that are less than $1/N$, and the remainder. 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., set $I_j = m$. One strip is filled. Now reduce q_m by the length of the upper part of strip j,

$$q_m \leftarrow q_m - (1 - q_j). \tag{44}$$

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

B.2 Dividing up a Line

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 concatenated. The uniform deviate chooses a point on the line. The integer part of the uniform deviate selects a unit-length segment. The threshold value for the unit segment starting at j is $j + Q_j$.

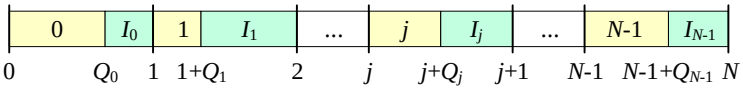

Fig. 11 Line segment divided into unit length segments.

The random variate generation algorithm is implemented as part of the code shown in Appendix E. The input is u, a uniform random variate. Two tables of size N are necessary. The first contains the threshold values $i + NQ_i$. The second is the index array containing the indices for the second parts of the unit segments.

The description above gives an explicit method to generate the tables. Knuth [2] describes 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. 12. 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
  [~, Is] = sort(Pn);   % ascending order
  Is = Is - 1;          % [0, N-1]
  j = Is(i+1);          % j, smallest Pn >= 0
  k = Is(N-1+1);        % k, largest Pn

% 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;         % Finished with Pn(j+1)
end

end
```

Fig. 12 Matlab code for generating alias table values.

B.3 Using Discrete Uniform Variates

Consider generating a discrete variate i with a probability q_i using a discrete uniform variate. Let uniform variate occur in steps of $1/M$. The variate $v = Nu - [Nu]$ has steps of size N/M. If q_i is less than N/M, the index i will occur with probability 0 or N/M. As an example, if $N = 64$ and $M = 2^{31}$, N/M is about 3×10^{-8}. For q_i values smaller than some multiple of

N/M, the probability of index i may not be accurate. These considerations suggest the use of a discrete uniform random number generator with a large M.

B.4 Other Considerations

The alias method requires a multiplication by the 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 can be accommodated by introducing additional sub-distributions with zero probability.

B.5 Application to Quantization

The alias method generates indices with given probabilities. It is an alternate to a 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, the goal is to find the index corresponding to an 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). Or barring that, using a binary search. The one-dimensional view of the alias method gives us an alternate approach.

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. Let the smallest interval between quantizer decision boundaries be $\Delta_{\min}$. Choose N such that $N\Delta_{\min} \geq 1$ and scale the interval $[0, 1]$ by N. No segment of unit length of the scaled interval will contain more than one decision boundary. We can now use the processing of the alias method to set up two tables. The first table has thresholds for each unit interval. The second table maps the interval below each threshold to a quantization index. Any value above the threshold will be mapped to next quantization index. Non-uniform quantization is implemented by first identifying the unit segment and then comparing the value with the threshold for that segment to determine the quantization index.

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 General Triangular Probability Density

C.1 Minimum and Maximum Order Statistics

A variate with a symmetric triangular pdf can be generated by adding two uniform variates. A non-symmetric can be formed with a scaled addition of the maximum and minimum of the two uniform variates. This can be shown as follows.

The properties of order statistics are described in [13]. For N random variables, the order statistics are the sorted values,

$$x_{(1)} \le x_{(2)} \le \cdots \le x_{(N)}. \tag{45}$$

Consider $N = 2$. Then the order statistics are the minimum and maximum. For the special case of independent uniform variates (u_1 and u_2), designate the order statistics as v_1 and v_2. These have joint pdf [13],

$$p_{12}(v_1, v_2) = \begin{cases} 2, & 0 \le v_1 \le v_2 \le 1, \\ 0, & \text{otherwise.} \end{cases} \tag{46}$$

The joint probability is uniform in the triangular region. Alternately, the limits can be expressed with unit step functions

$$p_{v_1 v_2}(v_1, v_2\,) = 2\,\mathrm{I}(v_1, v_2),. \tag{47}$$

where the indicator function is

$$\mathrm{I}(x, y) = [u(x) - u(x-1)]\,[u(y) - u(y-1)]\,u(y-x). \tag{48}$$

C.2 Sum of the Scaled Minimum and Maximum

The goal is to find the pdf of the sum of scaled v_1 and v_2. This will be carried out in three steps: (i) scaling, (ii) transformation of variables to create the sum, and (iii) integrating out the unused variable.

The scaled variables ($\alpha > 0$ and $\beta > 0$) are

$$\begin{aligned} w_1 &= \beta v_1, \\ w_2 &= \alpha v_2. \end{aligned} \tag{49}$$

The joint pdf of w_1 and w_2 is

$$p_{w_1 w_2}(w_1, w_2) = \frac{2}{\alpha\beta}\,\mathrm{I}\left(\frac{w_1}{\beta}, \frac{w_2}{\alpha}\right). \tag{50}$$

The sum of the scaled variables is $s = w_1 + w_2$, with $0 \le s \le \alpha + \beta$,

$$p_{sz}(s, w_2) = \frac{2}{\alpha\beta} \mathrm{I}\left(\frac{s - w_2}{\beta}, \frac{w_2}{\alpha}\right). \tag{51}$$

Integrating out w_2 in this expression will give the pdf of the scaled sum. The three terms of the indicator function determine the integration limits for w_2,

$$\begin{aligned} 0 \le \frac{s - w_2}{\beta} \le 1 \;&\rightarrow\; s - \beta \le w_2 \le s, \\ 0 \le \frac{w_2}{\alpha} \le 1 \;&\rightarrow\; 0 \le w_2 \le \alpha, \\ \frac{w_2}{\alpha} \ge \frac{s - w_2}{\beta} \;&\rightarrow\; \frac{\alpha s}{\alpha + \beta} \le w_2. \end{aligned} \tag{52}$$

The inequalities are plotted in Fig. 13 with the filled-in region representing

$$\frac{\alpha s}{\alpha + \beta} \le w_2 \le \min(s, \alpha). \tag{53}$$

This figure is shown for $\alpha < \beta$, but the bounds on w_2 are unchanged if $\beta \ge \alpha$.

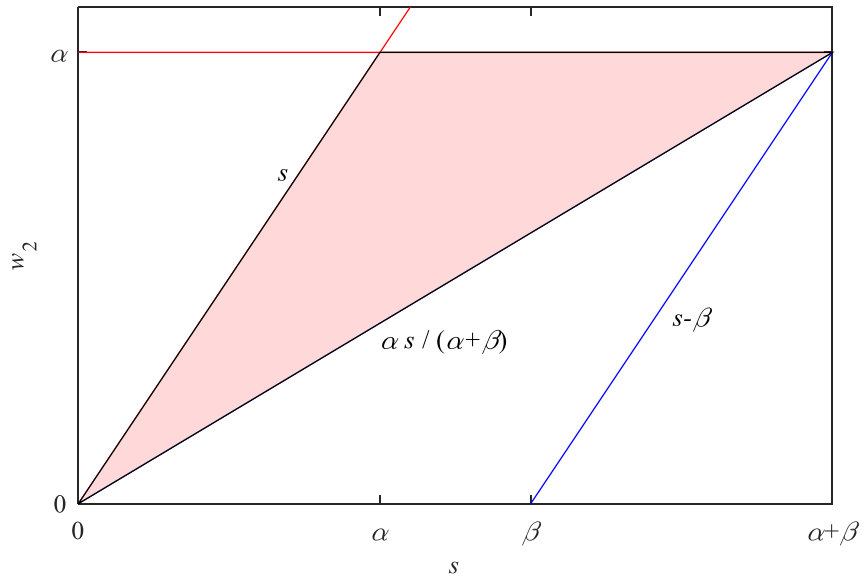

Fig. 13 Inequality regions.

The integration the gives

$$p_s(s) = \begin{cases} \dfrac{2}{\alpha}\dfrac{s}{\alpha + \beta}, & 0 \le s \le \alpha, \\ \dfrac{2}{\beta}\left(1 - \dfrac{s}{\alpha + \beta}\right), & \alpha \le s \le \alpha + \beta, \\ 0, & \text{otherwise.} \end{cases} \tag{54}$$

The pdf $p_s(s)$ integrates to one. The probability density is triangular with a base width of $\alpha + \beta$ and peak at $s = \alpha$.

Setting the base width to be constant at one and with $\beta = 1 - \alpha$, the sum can be written as

$$\begin{aligned} s &= (1-\alpha)v_1 + \alpha v_2, \\ &= (1-\alpha)\min(u_1, u_2) + \alpha \max(u_1, u_2)\,. \end{aligned} \tag{55}$$

The pdf of the sum is

$$p_s(s) = \begin{cases} \dfrac{2}{\alpha}\, s, & 0 \le s \le \alpha, \\ \dfrac{2}{1-\alpha}(1-s), & \alpha \le s \le 1, \\ 0, & \text{otherwise.} \end{cases} \tag{56}$$

If $\alpha = 0$ or $\alpha = 1$, the multivariate distribution $p_{sz}(s, w_2)$ is degenerate. For $\alpha = 0$, the distribution of s is that of the maximum order statistic; for $\alpha = 1$, the pdf is that of the minimum order statistic. In Eq. (56) taking the limit of $p_s(\alpha)$ as $\alpha \to 0$ from above and the limit of $p_s(\alpha)$ as $\alpha \to 1$, the results are consistent with the densities of the minimum and maximum order statistics.

The sum of the scaled minimum and maximum values of Eq. (55), for independent uniform densities in $[0, 1]$ has a triangular pdf. For $\alpha = 0.5$, $s = 0.5(u_1 + u_2)$. The triangular densities for different values of α are plotted in Fig. 14.

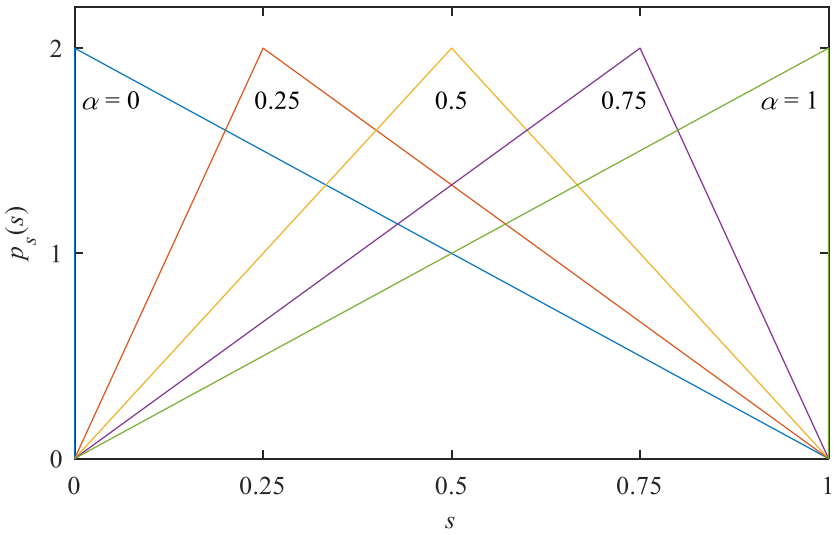

Fig. 14 Triangular pdf with changing α.

Appendix D Mixture Probabilities 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. The routine `qms` sets up symmetric anchor points. It then calls the routine `qopt` to find the mixture probabilities. It also calculates the peak difference between the piecewise linear approximation and the true Gaussian pdf.

The routine `qopt` finds mixture probabilities that minimize the mean-square difference between the piece-wise linear approximation and a target density. It first solves the linear equations that result with no constraint on the positiveness of the mixture probabilities. If any of the mixture probabilities is negative, it resorts to calling the Matlab routine `lsqlin` with the added constraint of positiveness.

```
function [q, xt] = qms(N, Cmax, R, wx)
% Solve for the mixture probabilities for a piecewise linear
% approximation to a Gaussian pdf.
% q    <-  Probabilities of the sub-distributions (N values)
% xt   <-  Edges of the overlapping triangles (for N triangles, xt has
%          Nt+2 values)
% N     -> Number of triangles (odd number, at least 3)
% Cmax  -> Largest peak location
% R     -> Ratio of the largest to smallest distance between triangle
%          peak locations
% wx    -> Weight exponent
% The triangle peaks are placed at [-Cmax, ..., 0, ..., Cmax]. The peak
% locations are geometrically spaced.

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

% Generate geometrically spaced triangle points
xt = GeoInc(N, R, Cmax);

% mse evaluation points - column vector
% - pdf at xt(1) and xt(end) is zero, and unaffected by q
% - sample at xt and midway between xt values
Nxt = length(xt);
x = interp1(xt, 1.5:0.5:(Nxt-0.5))';

% Optimum q (minimizes weighted mse)
A = Amat(x, xt);
pg = Gpdf(x);
W = diag(1 ./ (pg.^wx), 0);
q = qopt(W * A, W * pg);

% Check symmetry
if (any(abs(q - flipud(q)) > 2e-16))
  error('q vector not sufficiently close to symmetric');
end

Wmin = min(diff(xt(2:end-1)));
```

```
Wmax = max(diff(xt(2:end-1)));
fprintf('N = %d, Spacing: [%.3f, ..., %.3f], Cmax = %g\n', ...
        N, Wmin, Wmax, Cmax);
fprintf('  R = %g, wx = %g\n', R, wx);

% Find the peak error on a dense grid
xe = interp1(xt, 1:0.05:Nxt);
perr = Gpdf(xe) - PLApdf(xe, q, xt);
fprintf('Max pdf error: %.3g\n', max(abs(perr)));

end

% ----- -----
function xt = GeoInc(N, R, Cmax)
% Generate N+2 symmetric xt values
% The ratio of the largest interval between triangle peaks to smallest
% interval between peaks is R
% triangle i (i = 1:N)
%   xt(i), xt(i+1), xt(i+2), peak at xt(i+1)
assert(N >= 3 && mod(N, 2) == 1);

% Triangles for xt >= 0
% Peak values   x  x   x    x  ...   x       x         x
%               1  2   3    4  ...  M-2     M-1       M
% xt            0                         Cmax
% spacing        1   r  r^2    ...   r^(M-3)    r^(M-2)
% Generate values with ratio of spacing r^n
% M-1 spacings [1, r, r^2, ... , r^(M-3), r^(M-2)]
Nxt = N + 2;
M = (Nxt + 1) / 2;
if (M <= 3)
  r = 3;
else
  r = nthroot(R, M-3);
end

x = [0, cumsum(r.^(0:M-2))];
x = x * (Cmax / x(end-1));   % Normalize so x(end-1) = Cmax
xt = [-fliplr(x(2:end)), x];

end

% ----- -----
function p = PLApdf(x, q, xt)
% Piecewise linear pdf formed from triangular sub-distributions.
% p  <- Probability density function evaluated at x
% x   -> Evaluation points
% q   -> Probability of choosing sub-distribution j (column vector)
% xt  -> Triangular pdf edges, sub-distribution j is a triangle defined by
%        the points
%          [xt(j), 0], [xt(j+1), q(i) 2 / W(j)], [xt(j+2), 0],
%        where W(j) = xt(j+2) - xt(j).

W = xt(3:end) - xt(1:end-2);
ht = [0, 2 * q(:)' ./ W, 0];
```

```
% Linear interpolation within xt, zero outside
p = interp1(xt, ht, x, 'linear', 0);

end

% ----- -----
function A = Amat(x, xt)
% Form the pdf mixture matrix A, where A(i,j) is the contribution
% of sub-distribution ps(j) to the overall pdf at x(i). The
% sub-distributions are triangular, defined by the three points
%     [xt(j), 0], [xt(j+1), 2/W(j)], [xt(j+2), 0],
% with W(j) = xt(j+2)-xt(j).
% If the number of triangles is N, xt has length N+2.

% 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), xt(j:j+2))
N = length(xt) - 2;
Nx = length(x);

% Fill in matrix A
A = zeros(Nx, N);
for j = 1:N
  % Define triangle values at xt(j), xt(j+1), xt(j+2)
  hj = [0, 2/(xt(j+2) - xt(j)), 0];

  % Linear interpolation inside the triangle, zero outside
  A(:,j) = interp1(xt(j:j+2), hj, x, 'linear', 0);
end

end
```

Fig. 15 Matlab routine `qms` to calculate the mixture probabilities.

```
function q = qopt(A, p)
% Solve for the mixture probabilities that minimize the sum of the
% squared errors at given points.
%  q <-  mixture probabilities
%  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. This constraint is implemented 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.
% Setting S'*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))
  fprintf('*** Using lsqlin to avoid negative probabilities ***');
  % Find min(|Aq - p|^2) subject to sum(q(i))=1, q(i) >= 0
  options = optimoptions('lsqlin', 'OptimalityTolerance', 1e-12);
  q = lsqlin(A, p, [], [], S', 1, zeros(1, N), Inf(1, N), [], ...
             options);
end

if (any(q < 0))
  error('Invalid (negative) probability');
end
if (any(q == 0))
  warning('Some mixture probabilities are zero');
end
if (abs(sum(q) - 1) > 1e-10)
  error('Invalid probability sum');
end

end
```

Fig. 16 Matlab routine `qopt` to calculate the mixture probabilities

Appendix E Gaussian Variates from a Piecewise Linear Approximation

E.1 Uniformly-Spaced Anchor Points

Fig. 17 shows the C-language code for generating Gaussian variates using a piecewise linear approximation to the Gaussian pdf. The anchor points for the approximation are uniformly spaced (spacing 0.2). The smallest mixture probability is 7.4×10^{-8} at the outer sub-distributions.

E.2 Geometrically-Spaced Anchor Points

Fig. 18 shows the C-language code when the anchor points are geometrically-spaced. The smallest spacing between triangle apices is 0.114 on either side of the origin, increasing to 0.319 at the ends. The ratio of the largest spacing to smallest spacing is $R = 2.8$.

The code for non-uniformly-spaced anchor points uses a table of the anchor points in addition to the tables for the alias method of selecting the mixture.

```
float ran1(long int *idum);

/* Tables for N = 61, Cmax = 6, R = 1, wx = 0.5 */
#define N   61
#define Wh  0.2F

static const float Qp[N] = {
  0.000000073831554,  1.000000220755458,  2.000000699949626,
  3.000002112556964,  4.000006129940982,  5.000017085084540,
  6.000045743273636,  7.000117647500467,  8.000290658799035,
  9.000689809453677, 10.001572603900692, 11.003443925497274,
 12.007244893543472, 13.014640429389804, 14.028419659188476,
 15.052994021471081, 16.094924338968948, 17.163331791564396,
 18.269964136950911, 19.428630833920096, 20.653736189335955,
 21.957774441609310, 22.998747226828737, 23.917177941698039,
 24.985184046814965, 25.980728281751325, 26.810836243741747,
 27.991183498160449, 28.934490407170735, 29.946253240075073,
 30.933010235454027, 31.947135887598186, 32.938290896022764,
 34.000000000000000, 34.808964922145165, 35.996486142194378,
 36.981941028650169, 37.917177941698043, 38.999413080343423,
 39.957774441609310, 40.653736189335959, 41.428630833920096,
 42.269964136950911, 43.163331791564396, 44.094924338968951,
 45.052994021471079, 46.028419659188479, 47.014640429389800,
 48.007244893543472, 49.003443925497272, 50.001572603900691,
 51.000689809453675, 52.000290658799038, 53.000117647500467,
 54.000045743273638, 55.000017085084544, 56.000006129940985,
 57.000002112556963, 58.000000699949624, 59.000000220755460,
 60.000000073831551 };

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

float
gTriangU61 (long int *idum)
{
    int j;
    float uN;

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

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

Fig. 17 C-language code for the piecewise linear approximation method with uniformly spaced anchor points.

```
float ran1(long int *idum);

/* Tables for N = 61, Cmax = 6, R = 2.8, wx = 0.5 */
#define N   61

static const float Qp[N] = {
  0.000000100357494,  1.000000617706903,  2.000003391363740,
  3.000015907028457,  4.000064724087186,  5.000230975513477,
  6.000730434095417,  7.002066615536646,  8.005277624097332,
  9.012264910286158, 10.026134636740759, 11.051419239875893,
 12.094014163970524, 13.160697217032322, 14.258204237164311,
 15.391984669013334, 16.564895535430360, 17.776153536370334,
 18.982630204058232, 19.997409598799560, 20.963594488490518,
 21.864069493423752, 22.984359824505184, 23.999999999999996,
 24.982453245564511, 25.958977324785231, 26.895123415238423,
 27.902258506759772, 28.864374862383681, 29.860893355842435,
 30.938316679757598, 31.903488279937068, 32.857387576194853,
 33.876973903624638, 34.961806468300225, 35.958811073358945,
 36.994628975729754, 37.999825679533501, 38.996642313597249,
 39.877939219878350, 40.963594488490521, 41.996115767052586,
 42.979801060543245, 43.776153536370337, 44.564895535430360,
 45.391984669013333, 46.258204237164307, 47.160697217032322,
 48.094014163970527, 49.051419239875891, 50.026134636740757,
 51.012264910286156, 52.005277624097332, 53.002066615536648,
 54.000730434095416, 55.000230975513475, 56.000064724087188,
 57.000015907028455, 58.000003391363741, 59.000000617706903,
 60.000000100357497 };

static const int It[N] = {
 28, 33, 29, 34, 35, 25, 24, 37, 32, 28, 39, 29,
 26, 30, 25, 40, 24, 23, 36, 37, 22, 19, 41, 23,
 19, 42, 38, 22, 41, 19, 36, 24, 41, 37, 18, 36,
 37, 23, 23, 23, 24, 23, 38, 37, 36, 20, 35, 34,
 31, 27, 33, 21, 38, 22, 23, 36, 30, 26, 31, 27,
 32 };

static const float xt[N+2] = {
 -6.330911971340154, -6.000000000000000, -5.680630648543379,
 -5.372401296076172, -5.074923365620288, -4.787821834244457,
 -4.510734760282270, -4.243312827041446, -3.985218902429073,
 -3.736127613937675, -3.495724938456292, -3.263707806389455,
 -3.039783719584986, -2.823670382588944, -2.615095346762861,
 -2.413795666814612, -2.219517569309903, -2.032016132746505,
 -1.851054978787882, -1.676405974266980, -1.507848943584492,
 -1.345171391139028, -1.188168233439263, -1.036641540560349,
 -0.890400286618645, -0.749260108950200, -0.613043075689385,
 -0.481577461454677, -0.354697530858796, -0.232243329570277,
 -0.114060482663079,  0.000000000000000,  0.114060482663079,
  0.232243329570277,  0.354697530858796,  0.481577461454677,
  0.613043075689385,  0.749260108950200,  0.890400286618645,
  1.036641540560349,  1.188168233439263,  1.345171391139028,
  1.507848943584492,  1.676405974266980,  1.851054978787882,
  2.032016132746505,  2.219517569309903,  2.413795666814612,
  2.615095346762861,  2.823670382588944,  3.039783719584986,
  3.263707806389455,  3.495724938456292,  3.736127613937675,
  3.985218902429073,  4.243312827041446,  4.510734760282270,
  4.787821834244457,  5.074923365620288,  5.372401296076172,
  5.680630648543379,  6.000000000000000,  6.330911971340154 };

float
gTriangG61 (long int *idum)
{
```

```
    int j;
    float u1, u2, uN;

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

/* Generate a general triangular density */
  u1 = ran1(idum);
  u2 = ran1(idum);
  if (u1 >= u2)
    return (((xt[j+1] - xt[j]) * u1 + (xt[j+2] - xt[j+1]) * u2) +
            xt[j]);
  else
    return (((xt[j+1] - xt[j]) * u2 + (xt[j+2] - xt[j+1]) * u1) +
            xt[j]);
}
```

Fig. 18 C-language code for the piecewise linear approximation method with geometrically spaced anchor points.