JOURNAL OF COMPUTATIONAL PHYSICS 95, 29–39 (1991)

Functional Representation of Power-Law Random Fields and Time Series*

J. A. VIECELLI AND E. H. CANFIELD, JR.

Lawrence Livermore National Laboratory, Livermore, California 94550

Received August 10, 1989; revised March 19, 1990

Current methods of generating stationary random fields having power-law spectra are based on fast Fourier transforming an array of random numbers with zero mean and unit variance to wave space. Multiplication by the desired spectrum weight function, followed by inverse transformation to physical space then yields the sample field. We show that the desired spectral weightings can be generated directly in physical space, using the successive random addition methods previously employed for graphical display of random fractals, and derive expressions for the constants of the process in terms of the spectrum amplitude and exponent. A formula for the random number call sequence can be derived for the random addition process, eliminating the need for field array storage. This makes representation of random fields by a single computational function of the physical coordinates possible. Correspondingly, the scale range and dimension covered by the field function is limited only by machine word length. The same method can be used to represent time series with power-law frequency spectra in functional form, or to include time-dependence in random field problems.

INTRODUCTION

Monte Carlo computations have been employed to obtain statistical solutions for problems in which the equations governing the process are known, but the driving forces or fields are random. In the case of spatially-fluctuating stationary fields, the need to allocate storage for all of the field points severely limits the scale range that can be included in a computation. One example is the computation of the turbulent dispersion of Lagrangian particles in a Kolmogorovian velocity field [1]. In this case the three components of a random velocity vector potential are generated and stored for recall at each time step. In this example the rate of dispersion increases with time, and the limited storage for the velocity field severely limits the scale range that can be covered. Storage limitations currently preclude inclusion of power-law time fluctuations in these computations. For example, representing a three space dimensional time fluctuating field on a 64^4 grid requires over 1.67×10^7 words of memory, plus working space. Another example is the use of phase screens in optical calculations of propagation through turbulent media [2, 3]. In these

* Authors supported by U.S. Government Contract W-7405-ENG-48. The U.S. Government's right to retain a nonexclusive royalty-free license in and to the copyright covering this paper, for governmental purposes, is acknowledged.

0021-9991/91 $3.00

problems sets of two-dimensional fields of random numbers are generated at the beginning of a run. A fast Fourier transform is applied, and the resulting Hermitian fields are then weighted by an intertial range power-law for the turbulent thermal fluctuation in the medium. The fields are transformed back to the physical-space grid, and stored for recall at each time step of the propagation computation.

Many problems in forecasting and control can be described by a set of differential equations driven by stochastic forcing terms [4]. There is a large body of theory dealing with Markovian statistics, which can yield good approximate solutions in many instances; however, many, if not most, physical systems are characterized by stochastic forcing having a power-law frequency spectrum. Stochastic forcing with a power-law spectrum often leads to enhanced diffusion, where the variance of the driven variable is proportional to some power of the time greater than one [6]. The problems are similar to those encountered in turbulence, where many scales of the motion contribute to the forcing. This non-local dependence implies that very long time intervals, up to a physical cutoff, need to be included in generating the random forcing function. With the fast Fourier-transform method, or some of the multiple-correlation methods it is necessary to set aside memory for the entire time-series sample [6, 7]. Generation of fields that are fluctuating in both space and time, by these methods, is precluded by the enormous storage requirements.

The problem of how to generate fluctuating random fields or time series with power-law spectra is closely tied to that of generating random fractal curves and surfaces. The method of successive random addition has been successfully used to generate fractals that are quantitatively described by a structure function [5]. This method is very simple; however, in its present form it also supposes a large storage array on which successive operations are performed in repeated cycles through the mesh. Analytical relationships between the input constants for the method and the desired spectrum amplitude and exponent dependence also need to be derived in order to use the method. The process is one of successive grid refinement, with field values at new intermediate grid points obtained by interpolation from the previous grid. At each level of resolution, random increments are added to the existing field points, with variance rescaled to the given level.

There are well-known relationships between the structure function, the autocorrelation function, and the power spectrum of a random field or time series [1, 4]. Specification of any one of the three functions determines the other two. Hence the successive random additions method also generates random fields with power-law spectra, and can be used in place of the fast Fourier-transform method, given the appropriate relationship between the constants in the structure function description and those of the power spectrum characterization. Computing time and memory requirements are of the same order, since in both methods nearly all of the time is spent in generating random numbers. However, its simplicity gives successive random addition the potential for generating the field without the use of a storage mesh.

Successive random addition can be performed without actually storing the field

or time series, because only the portion of each successively finer grid containing the desired point in space or time need be retained. Only the mesh values associated with a single zone or time increment are needed at any step in the computation. However, it is necessary to keep track of the ordering of the calls to the random number generator as if every grid point were being computed, and to be able to regenerate only the random numbers needed for those zones containing the desired field point. This can be done, without resorting to repeating the entire random number sequence each time a specific number is needed, if every step in the successive random addition process is associated with a sequence number of the call to the random number generator used in that step, and if the sequence number is expressed by an algebraic formula involving the grid indices and level in the successive random addition process. We have found that it is comparatively easy to do this because of the simplicity of the successive random addition construction. Given the sequence number, the corresponding random number is easily regenerated by well-known methods [8, 9].

The method is ideal in high-dimensional situations where values at a relatively small number of points within the field and/or time series are required, but very many decades of scale range are desired in defining the spatial extent of the field or the time interval covered. Since the method generates values from a function of the spatial or time coordinates, instead of a table look-up and interpolation from a large array of stored field or time series values, the memory storage problems associated with attaining a large scale range are eliminated. In the optical example, the field of turbulent thermal fluctuations may be moving across the beam, requiring that a small number of new field values, statistically consistent with the existing field, be supplied at periodic intervals at the upstream edge of the propagation mesh. In the Lagrangian particle dispersion example it may be desirable to track the diffusion of a relatively small number of particles over a great distance in three space dimensions and time. In the case of differential equations driven by stochastic forcing it may be desirable to follow the evolution over many decades of time resolution. Functional representation makes this possible by eliminating the need to store the field or time series.

Random Addition with Sequence Function

Random fields can be defined by a power spectrum, the autocorrelation function, or a structure function. Under isotropy of the field, all three are equivalent. The structure function D and the autocorrelation function Γ are related by

$$D(r) = 2[\Gamma(0) - \Gamma(r)]. \tag{1}$$

The structure function is defined by

$$D(r) \equiv \langle [A(r_0 + r) - A(r_0)]^2 \rangle, \tag{2}$$

where A and r are the random-field variable and physical separation, respectively.

In the case of a fractal, the structure function has the form

$$D(r) = \sigma_0^2 r^{2H}, \tag{3}$$

where σ_0^2 is the structure constant, and H is the Hurst exponent [5, 6]. Substituting Eq. (3) into (1), and making using of the autocorrelation theorem for the power spectrum $P(k)$,

$$P(k) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \Gamma(r) e^{-ikr}\, dr \tag{4}$$

yields

$$P(k) = a_0 k^{-(2H+1)} \tag{5}$$

for the one-dimensional case. More generally, if d is the Euclidean dimension of the field, then,

$$P(k) = a_0 k^{-(2H+d)}. \tag{6}$$

If we can generate a random field in physical space satisfying Eq. (3), then by Eq. (6), it has a power-law spectrum with exponent $-(2H+d)$.

The method of successive random additions generates a field with structure function given by Eq. (3), for a specified input value of H. The basic idea is to generate a set of nested grids, each grid a factor of 2 smaller in size, and to generate the field by a process of interpolation from one grid to the next finer grid, with the addition at each step of a random perturbation of appropriate amplitude. An example in two-space dimensions illustrates the process. On the first pass, the values of the field at the four corners of a square covering the largest region of interest are initialized by drawing values from a normal distribution with zero mean, and variance σ_0^2. The mesh lines of the grid are defined by the edges of the square. On the second pass the grid consists of the mesh lines formed by the division of the original square into four equal squares. Values of the field at the five additional points on this grid are linearly interpolated from the four original points. Then an additional random perturbation is added to the field at each of the nine points in the new grid. The variance of the additional random perturbation is

$$\sigma_1^2 = (\tfrac{1}{2})^{2H} \sigma_0^2. \tag{7}$$

On the third pass the grid consists of the mesh lines formed by the division of the four second-pass squares into 16 equal squares. Values of the field are linearly interpolated from the existing nine intersection points of the mesh lines in the second-pass grid to the 16 additional points on the third-pass grid. Additional random perturbations with variance $(\frac{1}{2})^{4H}$ are then added to each of the existing mesh-line intersection values. The cycle can be continued until the perturbation variance for the nth grid, given by

$$\sigma_n^2 = (\tfrac{1}{2})^{2Hn} \sigma_0^2 \tag{8}$$

is reduced to the level of roundoff error. For the purpose of displaying fractal surfaces generated by this process, enough storage for the finest grid corresponding to the raster-display resolution would be required. Normally, this would stop the iteration at a level of resolution many decades from that available in single-precision floating-point computation.

If the value of the field is needed at only one point (x_k, y_l), where (k, l) are the indices of the square containing (x, y) on the finest resolution grid, then only the part of the nested grids containing the point need be computed, provided the random numbers corresponding to those points can be regenerated without going through the entire random-number sequence for the given seed. To do this it is necessary to find the relation between the sequence number of a given call to the random number generator, and the level number and grid location in the successive random addition iteration. If (k, l) is confined to a mesh of side 2^N, where N corresponds to the finest mesh, then the mesh interval δ_n of the nth grid is

$$\delta_n = 2^{N-n}. \tag{9}$$

We compute the nth level indices of the square containing (x, y) on the nth grid from

$$\begin{aligned} k_n &= 1 + \left[\frac{k-1}{\delta_n}\right] \\ l_n &= 1 + \left[\frac{l-1}{\delta_n}\right], \end{aligned} \tag{10}$$

where the brackets signify the integer part. For $N = 3$, corresponding to division of the original square region into 64 smaller squares, the index $k = 1, 2, 3, 4, 5, 6, 7, 8$ for the finest resolution mesh lines. At the first level $n = 1$, the index k_n takes on values 1, 2. At the second level $n = 2$, the index k_n takes on values 1, 2, 3, 4. At the third level $n = 3$, the index k_n covers the range 1, 2, 3, 4, 5, 6, 7, 8. We have indexed the grid squares at each level starting at 1 increasing by increments of 1, because this may be more helpful for computer programming purposes.

The calling sequence index i is a function of the relative coordinate indices (k_n, l_n) and the level of grid-refinement n,

$$i = 2\left[k_n + (l_n - 1)2^n + \sum_{j=2}^{n} 2^{2(j-1)}\right]. \tag{11}$$

Each time the grid interval is reduced by 2, the nth-level square containing the field point is split into four addiational squares, and it is necessary to determine which of the four new sub-regions contains the point (x_k, y_l). Once this is known, field values at the corners of the $(n+1)$th square can be interpolated from the corners of the nth square. Storage for only one square is needed, since old values

can be overwritten as soon as the interpolation is completed. The relative indices (k', l') on the $n+1$ level grid square containing (k, l) are

$$k'_{n+1} = 1 + \left\lfloor \frac{(k-1) \bmod \delta_n}{\delta_{n+1}} \right\rfloor$$
$$l'_{n+1} = 1 + \left\lfloor \frac{(l-1) \bmod \delta_n}{\delta_{n+1}} \right\rfloor, \tag{12}$$

where the brackets signify the integer part. With this choice, k'_{n+1} and l'_{n+1} take on values either 1 or 2. These relative indices tell which of the four new squares within the nth-level square bounded by mesh intersection points (k_n, l_n), (k_n+1, l_n+1), (k_n, l_n+1), and (k_n+1, l_n), contains the field point (k, l), on sub-dividing the nth-level mesh.

If (k'_{n+1}, l'_{n+1}) is (1, 1) then the corner value (1, 1) remains the same, (1, 2) is replaced by the average of (1, 1) and (1, 2), and (2, 1) is replaced by the average of (1, 1) and (2, 1), while (2, 2) is replaced by the average of (1, 1), (1, 2), (2, 1), and (2, 2). Permutations of the interpolation correspond with each of the three other possible values for the relative indices, $(k'_{n+1}, l'_{n+1}) = (1, 2)$, (2, 1), and (2, 2). Figure 1 shows a schematic of the initial steps in the sequence of repeated opera-

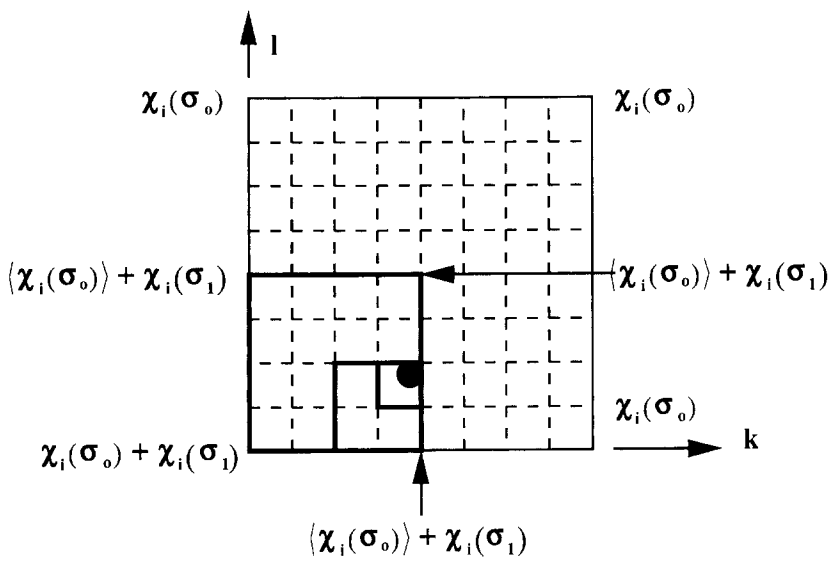

FIG. 1. Schematic showing initial steps in repeated sequence of operations used to generate the field value at the point (x_k, y_l) indicated by the solid dot. $\chi_i(\sigma)$ is the ith random number in a sequence of random numbers distributed normally with variance σ, starting from fixed seed x_0. The sequence number i to use in regenerating the random number is determined from the virtual mesh coordinates and level of the construction. The only storage required for the computation, aside from the program instructions, is working memory space for the four current values of the field at the corners of the square containing the field point.

tions performed in generating the field at the point indicated by the solid dot. Except for values at the corners of the single square containing the point at each level of the construction, none of the other mesh values need be generated or stored.

On the first pass, values for the field at each of the four corners of the initial mesh of side W are obtained from the random number generator with variance σ_0^2. Then, using the general formulas Eqs. (11)–(12), the index (k_1', l_1') of the square of side $W/2$ containing point (x_k, y_l) is computed, and field values are interpolated to the corners of the smaller square containing (x_k, y_l). Random perturbations are then added to the field values on the smaller mesh containing the point (x_k, y_l). The variance of the perturbations σ_n^2 for the nth level is the fraction of the initial variance specified by Eq. (8). If the entire field were being generated there would be no need to keep track of the number of times the random number generator had been called at the given stage and operation in constructing the field. However, if only the square in each stage of grid-refinement containing (x_k, y_l) is to be computed, it becomes necessary to know the number of calls to get to those particular squares in the full construction in order to be able to regenerate the particular random number without repeating the entire random number sequence.

The sequence number has to be expressed as a formula, since storing the sequence numbers in a pointer array corresponding to the spatial grid and level would require a large storage array. The formula given by Eq. (11) was derived for a virtual array with l entries separated by $k_{\max}$, and the inner loops on k. The sequence number formula also needs to take into account the number of calls required to generate a normal distribution with zero mean and unit variance. For the standard square root log cosine method, two successive calls to the random number generator are required.

The relationship between the structure constant σ_0^2, the Hurst exponent H, and the power spectrum amplitude C_0^2, and power-law exponent α, can be obtained from substitution of the Fourier expansion of the autocorrelation function Γ into Eq. (1) and carrying out the inverse transform. For the two-dimensional example case with power spectrum

$$\Psi = C_0^2 k^{-\alpha} = \frac{C_0^2}{(k_x^2 + k_y^2)^{\alpha/2}}, \tag{13}$$

Eq. (1) yields

$$D(r) = \frac{C_0^2}{2\pi^2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \frac{[1 - e^{i\mathbf{r}\cdot\mathbf{k}}]}{[k_x^2 + k_y^2]^{\alpha/2}} \, dk_x \, dk_y, \tag{14}$$

where r is the separation in physical space. The imaginary part of the integrand cancels by symmetry yielding

$$D(r) = \frac{2C_0^2}{\pi^2} \int_0^{\infty} \int_0^{\pi/2} \frac{[1 - \cos(rk \cos\theta)]}{k^{\alpha}} \, k \, dk \, d\theta. \tag{15}$$

By change of variable Eq. (15) becomes

$$D(r) = \frac{2C_0^2}{\pi^2} \int_0^{\pi/2} (\cos\theta)^{\alpha-2}\, d\theta \int_0^{\infty} \frac{1-\cos(rx)}{x^{\alpha-1}}\, dx. \tag{16}$$

Evaluation of the integrals yields

$$D(r) = \frac{2C_0^2}{\pi^2}\left[\frac{\pi\Gamma(\alpha-1)}{2^{\alpha-1}\Gamma^2(\alpha/2)}\right]\left\{[1-\alpha]\,\Gamma(1-\alpha)\sin\left[\frac{\pi}{2}(1-\alpha)\right]\right\} r^{\alpha-2}. \tag{17}$$

For the case $\alpha = \frac{11}{3}$, Eq. (17) yields

$$D(r) = \sigma_0^2 r^{2H} \cong 0.3565 C_0^2 r^{5/3}. \tag{18}$$

Similar relationships are derivable for the three-dimensional case with spectrum

$$\Psi = \frac{C_0^2}{(k_x^2 + k_y^2 + k_z^2)^{\alpha/2}}, \tag{19}$$

yielding the structure function

$$D(r) = -\frac{C_0^2}{\pi^2}\Gamma(2-\alpha)\sin\left[\frac{\pi}{2}(2-\alpha)\right] r^{\alpha-3}. \tag{20}$$

For the case $\alpha = 11/3$, corresponding to inertial range turbulence, we obtain

$$D(r) \cong 0.1221 C_0^2 r^{2/3}. \tag{21}$$

Note that we use Bracewell's [10] system 2 Fourier transform convention, in which the factors of $1/2\pi$ are lumped with the inverse transform, whereas most applications to turbulence lump the $1/2\pi$ factors with the forward transform. Multipication of Eq. (21) by $8\pi^3$ yields the well-known form

$$D(r) \cong \frac{1}{0.033} C_0^2 r^{2/3}. \tag{22}$$

The sequence index for the three-dimensional case is

$$i = 2\left[k_n + (l_n - 1)2^n + (m_n - 1)2^{2n} + \sum_{j=2}^{n} 2^{3(j-1)}\right]. \tag{23}$$

The steps in generating the field on a three-dimensional grid, or in any number of dimensions, are identical to those for the two-dimensional case.

Computational Results and Conclusions

The two-dimensional stationary-field case was solved by the three independent methods: the fast Fourier transform, successive random addition, and successive

random addition in functional form with sequence key. We examined the case $\alpha = \frac{11}{3}$ corresponding to the spectral-density function for hydrodynamic-inertial-range turbulent fluctuations [1]. Successive random addition in either mesh-based form or functional form gives the same field values to machine roundoff, since the numerical operations are identical. Figure 2 shows the spectral density function computed by successive random addition and by the fast Fourier-transform method for a 256×256 grid of field values, using Eq. (18) to determine σ_0^2 and H in terms of α and C_0^2. The two methods are in good agreement, except at the tail end of the spectrum, where successive random addition has a slight increase in the amplitude, resulting from the linear interpolation. This can be improved by using a higher-order scheme, such as a fourth-order Lagrangian formula, in the interpolation between meshes. The additional cost is negligible, but the logic of the functional form is more complicated because additional points have to be carried from one level to the next.

Although there is some overhead involved in computing relative indices, and interpolating between meshes, nearly all of the computing time used in successive random additions, in either mesh-based or functional form, is spent in generating random numbers satisfying a normal distribution. For large n, the number of

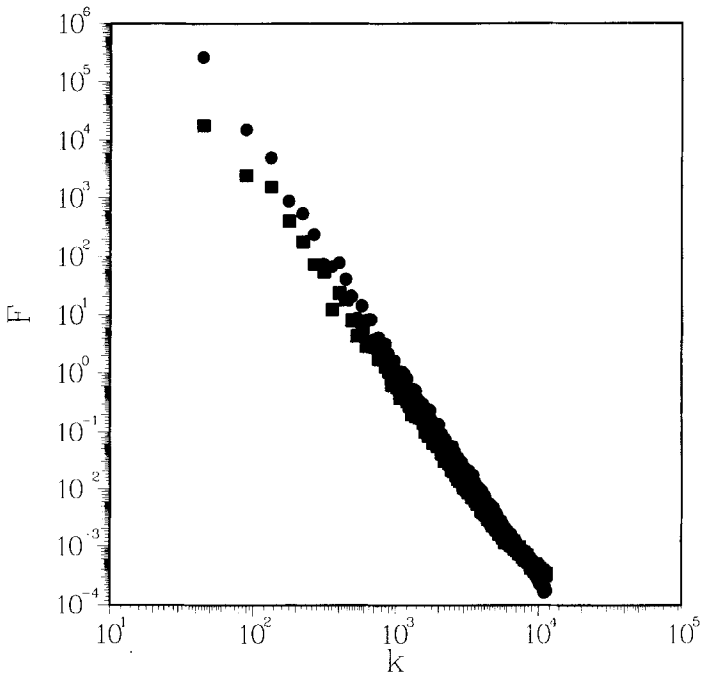

FIG. 2. Inertial range $k^{-11/3}$ spectral density function generated for 256×256 mesh by the fast Fourier transform method (dots), and by successive random addition (squares) using linear interpolation between meshes.

Gaussian random numbers required to generate a mesh with side 2^n in d-dimensional space is

$$N_0 = \frac{2^{nd}}{1 - 1/2^d}, \tag{24}$$

for successive random addition without sequence function, and

$$N_1 = n2^{d(n+1)}, \tag{25}$$

using successive random addition in functional form, with the sequence key, to generate every point on the 2^n grid. We would, of course, not normally be using the functional form to generate a large stored array, but doing so provides one way of evaluating the additional computing cost of the functional form. The extra time required is

$$\frac{N_1}{N_0} = n(2^d - 1). \tag{26}$$

For example, to generate the field at every point in a 64×64 mesh, using the functional form to load the mesh, requires $6(2^2 - 1) = 18$ times as many Gaussian random number evaluations. The actual cost is greater, because computing the sequence key and the additional bit extraction, tests, and multiplications associated with extracting the nth random number in a sequence, require more arithmetic operations. Another indirect factor increasing the cost of extracting particular random numbers is that the advantage of generating and storing short tables of random numbers on vector machines is eliminated. The latter factor is a significant one, and can speed up sequential random-number generation by perhaps a factor of between 5 to 10. However, the time required to do the log, square root, and cosine evaluations is the same, so the overall difference is not as great. Typically, without the benefit of vectorization, we have observed about a 3.5-fold increase in computing time for recalling a specific Gaussian random number by the method described in [9], compared with generating the next number in the sequence, so that the total difference in computing time on the 64×64 mesh is about a factor of 63. In practice the amount of computing time to load every value in a given mesh can be estimated quite well by multiplying N_0 or N_1 in Eqs. (24) or (25) by the time required to generate one Gaussian random number.

The additional time to regenerate a random number is spent in testing the bits of the sequence number and, for non-zero bits, performing a multiply mod 2^N, where N is the number of bits in the machine word length. If the word length is $N = 48$ bits, the number of multiplications is at most 48, whereas only one multiply is needed to generate the next number in the sequence. The number of multiplications can be reduced if the function storage space is increased to include precomputed tables of factored powers of the multiplier [11].

The method described in [9] assumes that the power of the multiplier is

expressed in base 2, but a higher base can be used. For example, if we take $N = 48$ and express the sequence number i in base 8, and if we precompute and store an 8×16 table of values of powers of the multiplier, then after extracting the 16 digits of i, we need at most to do only 16 multiplications of entries from the table to compute the multiplier raised to the ith power. Correspondingly, in base 16 the number of multiplications is reduced to at most 12, at the cost of storage for a 16×12 table of powers of the multiplier. The size of the table increases rapidly, so one may wish to stop at base 256, using a 256×6 table, requiring at most six multiplications per random number. The additive term in linear-congruent generators, although not often used, can also be included and is treated at length in [11].

The scale range that can be covered is determined by the machine word length in bits. For a 48-bit word length the random number generator repeats at interval 2^{46}. From Eqs. (24) and (25) we need $nd \ll 46$ in order to have a wide margin. For example, using a 48-bit word for a three-dimensional field $d = 3$, the maximum scale range is approximately $2^{15} = 3 \times 10^4$. To include a larger scale range it is necessary to use double-precision arithmetic in computing the sequence function and regenerating the random numbers.

The advantages of functional generation are in supplying moderate numbers of values at specific points, with no storage limits on the scale range or dimension. In the Lagrangian particle dispersion example, unless the number of particles is very large, functional generation offers significant advantages in both speed and storage reductions, as well as the ability to generate a field covering a very large scale range and to include a fourth dimension for statistical fluctuations in time. In the optical example, functional generation greatly reduces the storage requirements associated with supplying new boundary values. For high-dimensional problems, functional generation has an overwhelming advantage over storage-based methods.

REFERENCES

1. A. S. MONIN AND A. M. YAGLOM, *Statistical Fluid Mechanics*, Vol. 2, edited by J. L. Lumley (MIT Press, Cambridge, MA, 1987).
2. M. G. RUSBRIDGE, *J. Comput. Phys.* **2**, 288 (1968).
3. P. B. ULRICH AND J. WALLACE, *J. Opt. Soc. Amer.* **63**, 8 (1973).
4. G. E. P. BOX AND G. M. JENKINS, *Time Series Analysis* (Holden–Day, San Francisco, 1976).
5. R. F. VOSS, "Random Fractal Forgeries," in *Fundamental Algorithms in Computer Graphics*, edited by R. A. Earnshaw (Springer-Verlag, Berlin, 1985), p. 805.
6. J. FEDER, *Fractals* (Plenum, New York, 1988), p. 180.
7. B. B. MANDELBROT, *Water Resour. Res.* **7**, 543 (1972).
8. D. E. KNUTH, *The Art of Computer Programming* (Addison–Wesley, Reading, MA, 1969).
9. A. E. KONIGES AND C. E. LEITH, *J. Comput. Phys.* **81**, 230 (1989).
10. R. BRACEWELL, *The Fourier Transform and Its Applications* (McGraw–Hill, New York, 1965), p. 6.
11. E. H. CANFIELD, JR. AND J. A. VIECELLI, "Random Access to a Random Number Sequence," *J. Comput. Phys.*, in press.