Title	Simulation of fractional Brownian motion with conditionalized random midpoint displacement
Author(s)	Norros, Ilkka; Mannersalo, Petteri; Wang, Jonathan
Citation	Advances in Performance Analysis vol. 2(1999):1, pp. 77-101
Date	1999
Rights	This article may be downloaded for personal use only

Simulation of Fractional Brownian Motion with Conditionalized Random Midpoint Displacement

Ilkka Norros
VTT Information Technology
P.O.Box 1202
FIN-02044 VTT, Finland
ilkka.norros@vtt.fi

Petteri Mannersalo
VTT Information Technology
P.O.Box 1202
FIN-02044 VTT, Finland
petteri.mannersalo@vtt.fi

Jonathan L. Wang
Bellcore
331 Newman Springs Road
Red Bank, NJ 07701, USA
jwang@bellcore.com

Abstract

Recent measurement studies have shown that the burstiness of packet traffic is associated with long-range correlations that can be efficiently modeled by terms of fractal or self-similar processes, e.g., *fractional Brownian motion* (FBM). To gain a better understanding of queuing and network-related performance issues based on simulations as well as to determine network element performance and capacity characteristics based on load testing, it is essential to be able to *accurately* and *quickly* generate long traces from FBM processes. In this paper, we consider an approximate FBM generation method based on the concept of bisection and interpolation, which is an improvement of a much used but inaccurate method known as the *random midpoint displacement (RMD)* algorithm. We further extend our new algorithm (referred to as RMD_{mn}) to be able to generate FBM traces without a priori knowledge of the length of the simulation (i.e., on-the-fly generation), instead of being a pure top-down generation (that is, the entire trace has to be generated first before it can be used) like the original RMD algorithm. We present the mathematical and numerical aspects of the RMD_{mn} algorithm as well as compare it with two other widely favored FBM generation methods, i.e., the fast Fourier transform (FFT)

method and the method based on aggregating a large number of ON-OFF sources with infinite-variance sojourn times.

Keywords: fractional Brownian motion, self-similar traffic generation, random midpoint displacement.

1 Introduction

Recent measurement studies (see for example [7, 10, 18]) have shown that the burstiness of packet traffic is associated with long-range correlations than can be efficiently modeled by terms of fractal or self-similar processes, e.g., fractional Brownian motion (FBM). Fractional Brownian motions were included in the toolbox of statistical models by Mandelbrot and Van Ness [13], the first applications being in economy and hydrology. Later, these processes have been used in many other fields, including generation of artificial landscapes [12] and, more recently, teletraffic modeling [15]. However, the generation (or simulation) of FBMs has remained somewhat non-trivial since it is practically impossible to build large FBM samples precisely using a sequence of independent standard Gaussian random variables as the basic random elements. This paper presents a conceptually simple FBM generation algorithm which can reach practically arbitrary accuracy without sacrificing too much computation efficiency.

A normalized FBM with self-similarity parameter $H \in (0,1)$ is a stochastic process $(Z_t)_{t\geq 0}$ characterized by the following properties:

(i) Z_t has stationary increments;

(ii) $Z_0 = 0$, and $\mathrm{E}Z_t = 0$ for all t;

(iii) $\mathrm{E}Z_t^2 = t^{2H}$ for all t;

(iv) Z_t is Gaussian;

(v) Z_t has continuous sample paths.

Ordinary Brownian motion is obtained as the special case with $H = 1/2$. The existence of such a process was established by Kolmogorov [8] in the Hilbert space framework. In [13], the process was defined more constructively as an integral with respect to ordinary Brownian motion:

$$Z_t - Z_s = c_H \Big\{ \int_s^t (t-u)^{H-\frac{1}{2}} \, \mathrm{d}W_u + \int_{-\infty}^s \Big((t-u)^{H-\frac{1}{2}} - (s-u)^{H-\frac{1}{2}} \Big) \, \mathrm{d}W_u \Big\}. \tag{1.1}$$

The normalization $\mathrm{E}Z_1^2 = 1$ is achieved with

$$c_H = \sqrt{2H\Gamma(\frac{3}{2} - H)/(\Gamma(H + \frac{1}{2})\Gamma(2 - 2H))},$$

where Γ is Euler's Gamma function. The single most important property of a FBM is its statistical self-similarity: the processes $Z_{\alpha t}$ and $\alpha^H Z_t$ have the same path space distribution for any $\alpha > 0$. In particular, the correlation structure is the same in all timescales. For $H \neq 1/2$, this implies that the correlations do not die out in the same way as most processes used in statistical modeling, for example the finite order autoregressive processes. Thus, it is in general not a good idea to generate a long sequence of successive increments of a FBM using either an exact (too heavy) or truncated (eventually exponential decay of correlation) autoregressive scheme.

The exact generation of long FBM traces is infeasible in practice due to the amount of storage and CPU time required (consider, for example, the dimensionality of the covariance matrix of the sequence of FBM increments, and the difficulty of generating a sequence that exactly conforms to this correlation structure). For example, skillful use of matrix algebra allows the FBM simulation program of FRACLAB (a toolbox for signal processing with fractals developed in the FRACTALES team at INRIA Rocquencourt, France) to generate exact FBM sequences of length 10^4, but the algorithm slows down dramatically if longer sequences are required.

Therefore, well understood and efficient approximate algorithms become desirable, especially for generating long traces for the purpose of network performance testing, simulation and analysis. Several approximation algorithms have been proposed, which include:

- a fast but ad hoc method suggested by Mandelbrot that is based on short memory approximation [11];
- *queuing based methods* such as a method based on the $\mathrm{M/G/\infty}$ queue length with Poisson arrivals and heavy-tailed service time [4];
- *transform methods* based on inversely transforming known FBM coefficients in the transformed domain, these include methods based on the fast Fourier transform [17] and the wavelet transform [1, 5, 21, 20];
- *aggregation methods* based on aggregating a *large number* of single source models such as models based on chaotic maps [19], ON-OFF model with infinite-variance sojourn times [22], and AR(1)-processes with the AR(1) parameters chosen from a beta-distribution on [0,1] with shape parameters p and q [6].

- *bisection methods* based on generating points of the process path "top-down" by properly interpolating from the existing points.

The last group includes, as the simplest case, random midpoint displacement (RMD), which has traditionally been identified as a method appropriate for fast but somewhat inaccurate FBM generation. It gives realizations with roughly the right self-similarity parameter, but the correlations behave in a non-stationary manner. Lau *et al.* found it, however, quite suitable for the generation of simulated data traffic [9], at least to the extent that the generated traces are qualitatively satisfactory. In [14], a one step further conditionalized version of RMD was proposed, and it was found quite satisfactory for most needs.

The main idea behind the algorithm proposed in this paper is to apply the same principle of bisection, but to achieve arbitrary accuracy at minimal computation and storage costs. Moreover, the algorithm can be modified to produce long accurate FBM traces *on-the-fly* instead of being a pure top-down algorithm, with moderate memory requirements and high computational efficiency. Here, the term on-the-fly means that traces are generated without a priori knowledge of the length of the simulation; in pure top-down algorithms, the entire trace has to be generated first before it can be used. This on-the-fly capability greatly improves the usability of the proposed generation method in simulation and load testing that requires long traces such as the ATM environment with its stringent QoS objectives and high capacities.

This paper is organized as follows. The algorithm and some mathematical aspects are studied in Section 2. Numerical studies on the accuracy of the algorithm are presented in Section 3.1; an outline of our implementation and the computation complexity is given in Section 3.3; and comparison with two other widely favored algorithms, i.e., the FFT and the aggregation methods, in terms of their accuracy and time complexity is presented in Section 3.2. Finally, the paper is summarized in Section 4.

2 The Conditionalized Random Midpoint Displacement Algorithm

In this section, we first describe an exact construction for a FBM. Then we introduce our RMD_{mn} algorithm and describe an extension of the algorithm to enable generation of FBM traces on-the-fly. Finally, covariances of approximate traces are studied.

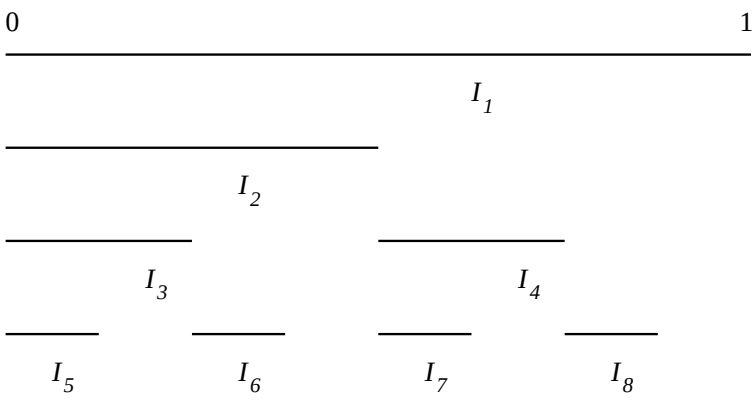

Figure 2.1: Ordering of the intervals in the binary construction.

2.1 The Full Binary Construction

Let $I_1, I_2, I_3, \ldots$ denote dyadic subintervals of $(0, 1]$, numbered as shown in Figure 2.1. For any i, denote by X_i the increment $X_i = Z_{b_i} - Z_{a_i}$, where $I_i = (a_i, b_i]$. Further, let $\mathbf{e}(i)$ and $v(i)$ be the i-vector and scalar, respectively, defined by

$$\begin{aligned} \mathbf{e}(i)[X_i, X_{i-1}, \ldots, X_1]^{\mathrm{T}} &= \mathrm{E}\left[X_{i+1} \mid X_i, X_{i-1}, \ldots, X_1\right], \\ v(i) &= \mathrm{Var}\left[X_{i+1} \mid X_i, X_{i-1}, \ldots, X_1\right]. \end{aligned}$$

Finally, denote $U_1 = X_1 = Z_1$ and

$$U_{i+1} = \left(X_{i+1} - \mathbf{e}(i)[X_i, X_{i-1}, \ldots, X_1]^{\mathrm{T}}\right) / \sqrt{v(i)}, \quad i = 1, 2, \ldots$$

The above relations establish a $1-1$ mapping between the sequence $X_1, X_2, \ldots$ of FBM increments and the sequence $U_1, U_2, \ldots$ of mutually independent standard Gaussian random variables. It can be considered as one of the many interesting orthogonalizations of a FBM. (For a very different kind of orthogonalization, see [16].) By the continuity of Z, the mapping $(U_1, U_2, \ldots) \mapsto (X_1, X_2, \ldots)$ extends to an almost everywhere defined map $\Psi : \mathbb{R}^{\mathbb{N}} \to C[0, 1]$ such that

$$\Psi(U_1, U_2, \ldots) = Z.$$

We note the following monotonicity properties:

Proposition 2.1 *The mapping*

$$(u_1, u_2) \mapsto \Psi(u_1, u_2, u_3, \ldots)$$

is increasing in the set of sequences $(u_1, u_2, u_3, \ldots)$ such that the right hand side is defined.

Proof It is enough to note that the coefficients of U_1 and U_2 are positive in the representation of any Z_t in the orthogonal basis $\{U_1, U_2, \ldots\}$. The coefficient of U_1 is simply $\mathrm{Cov}\,(Z_t, Z_1)$, which is positive for $t > 0$. Since $U_2 = (Z_{1/2} - Z_1/2)/\sqrt{v(1)}$, the coefficient of U_2 is

$$\frac{\mathrm{Cov}\,(Z_t, U_2)}{\sqrt{\mathrm{Var}\,(U_2)}} = \frac{\mathrm{Cov}\,\left(Z_t, Z_{1/2}\right) - \frac{1}{2}\mathrm{Cov}\,(Z_t, Z_1)}{\sqrt{v(1)\mathrm{Var}\,(U_2)}}.$$

It is easy to check that the nominator is positive for $t \in (0,1)$, with a maximum at $t = \frac{1}{2}$. □

On the other hand, the whole mapping Ψ is not monotone. For example, increasing U_3 while keeping the other U_i's fixed decreases $Z_{3/4}$.

2.2 Basic Algorithm

By truncating the exact construction described in the previous section, we now construct an approximate FBM realization $(Z(t) : t \in [0,1])$ with parameter $H \in (0,1)$. Note that using the property of self-similarity, $Z(t)$ can be scaled onto an interval of any desired length. First, we set $Z(0) = 0$ and draw $Z(1)$ from the standard Gaussian distribution. The conditional distribution of $Z(\frac{1}{2})$ given $Z(0) = 0$ and $Z(1)$ is then $N(\frac{1}{2}Z(1),\ 2^{-2H} - \frac{1}{4})$. Thus, we can next draw $Z(\frac{1}{2})$ from this distribution.

In principle, this *bisecting* (or *halving*) process could be continued using exact conditional distributions. However, the dimension of the multivariate Gaussian distribution grows by each new additional point, and it quickly becomes necessary to restrict the number of previously drawn points that are used in conditioning. In the traditional RMD algorithm, conditioning is made only on the increments of the interval to be halved. In the version proposed in [14], we additionally conditionalized on the increment of the left neighboring interval. This idea is further extended in the present version to conditionalize on a fixed finite number of already generated left and right neighboring intervals.

To be more precise, let us fix two integers $m \geq 0$ and $n \geq 1$. We shall conditionalize on (at most) m neighboring increments to the left of the interval to be halved and on (at most) n neighbor increments to the right of the interval to be halved, including the "mother" interval itself. Thus, we can call our algorithm RMD_{mn}, so that the conventional RMD is just $\mathrm{RMD}_{0,1}$ and the algorithm presented in [14] is $\mathrm{RMD}_{1,1}$.

Let $\left\{U_{ik} : \ i = 0, 1, \ldots;\ k = 0, \ldots, 2^{i-1} - 1\right\}$ be a set of independent standard Gaussian random variables. Denote $X_{i,j} = Z(j \cdot 2^{-i}) - Z((j-1) \cdot 2^{-i})$,

$i = 0, 1, 2, \ldots$, $j = 1, \ldots, 2^i$. (Note the different numbering of the intervals compared to the previous section!) We define the FBM generation algorithm recursively. Assume that we have obtained the values up to stage $i-1$ (resolution 2^{-i+1}). We then draw $X_{i,1}$, $X_{i,2}, \ldots$ as follows:

Since $X_{i,2j-1} + X_{i,2j} = X_{i-1,j}$, it is sufficient to generate $X_{i,j}$ for odd j. Let us proceed from left to right, and assume that $X_{i,1}, \ldots, X_{i,2k}$ have already been generated ($k \in \{0, 1, \ldots, 2^{i-1} - 1\}$). Now, we can choose

$$\begin{aligned} X_{i,2k+1} \;=\; & \mathbf{e}(i,k)[X_{i,(2k-m+1)\vee 1}, \ldots, X_{i,2k},\, X_{i-1,k+1}, \ldots, X_{i-1,(k+n)\wedge 2^{i-1}}]^{\mathrm{T}} \\ & + \sqrt{v(i,k)}\; U_{i,k}, \end{aligned} \tag{2.1}$$

where $\mathbf{e}(i,k)$ is a row vector such that

$$\begin{aligned} & \mathbf{e}(i,k)[X_{i,(2k-m+1)\vee 1}, \ldots, X_{i,2k},\, X_{i-1,k+1}, \ldots, X_{i-1,(k+n)\wedge 2^{i-1}}]^{\mathrm{T}} \\ = \;& \mathrm{E}\left[X_{i,2k+1} \mid X_{i,(2k-m+1)\vee 1}, \ldots, X_{i,2k},\, X_{i-1,k+1}, \ldots, X_{i-1,(k+n)\wedge 2^{i-1}}\right] \end{aligned} \tag{2.2}$$

and $v(i,k)$ a scalar defined by

$$v(i,k) = \mathrm{Var}\left[X_{i,2k+1} \mid X_{i,(2k-m+1)\vee 1}, \ldots, X_{i,2k},\, X_{i-1,k+1}, \ldots, X_{i-1,(k+n)\wedge 2^{i-1}}\right].$$

The vector $\mathbf{e}(i,k)$ and the number $v(i,k)$ are computed by the standard formulae for multivariate Gaussian distributions as follows. Denote by Γ_{ik} the covariance matrix

$$\Gamma_{ik} = \mathbf{Cov}\left(\left[X_{i,2k+1},\, X_{i,(2k-m+1)\vee 1}, \ldots, X_{i,2k},\, X_{i-1,k+1}, \ldots, X_{i-1,(k+n)\wedge 2^{i-1}}\right]\right),$$

where

$$\mathbf{Cov}\left([x_1, \ldots, x_n]\right) = \begin{pmatrix} \mathrm{Cov}(x_1,x_1) & \mathrm{Cov}(x_1,x_2) & \cdots & \mathrm{Cov}(x_1,x_n) \\ \mathrm{Cov}(x_2,x_1) & \mathrm{Cov}(x_2,x_2) & \cdots & \mathrm{Cov}(x_2,x_n) \\ \vdots & \vdots & \vdots & \vdots \\ \mathrm{Cov}(x_n,x_1) & \mathrm{Cov}(x_n,x_2) & \cdots & \mathrm{Cov}(x_n,x_n) \end{pmatrix},$$

and split it as

$$\Gamma_{ik} = \begin{bmatrix} \mathrm{Var}\,(X_{i,2k+1}) & \Gamma_{ik}^{(1,2)} \\ \Gamma_{ik}^{(2,1)} & \Gamma_{ik}^{(2,2)} \end{bmatrix}.$$

Then

$$\mathbf{e}(i,k) = \Gamma_{ik}^{(1,2)} \left(\Gamma_{ik}^{(2,2)}\right)^{-1}, \quad v(i,k) = \frac{\det \Gamma_{ik}}{\det \Gamma_{ik}^{(2,2)}}.$$

By the stationarity of the increments of Z and by self-similarity, $\mathbf{e}(i,k)$ is independent of i and k when $2k \geq m$ and $k \leq 2^{i-1} - n$. Moreover, it depends on i only when $2^i < m + 2n$.

On the other hand, the number of the true covariance matrices, when running up to the stage $i = \lceil \log(m+2n) \rceil$, is $1 + 2 + 2^2 + \ldots + 2^{i-1} = 2^i - 1 < 2m + 4n$. Thus, less than $2m + 4n$ vectors $\mathbf{e}(i,k)$ need to be computed. The same holds for the scalars $v(i,k)$, except that "independence with respect to i" is replaced by scaling a constant factor 2^{-2Hi}.

In the case $m = n = 1$, the only special case is $k = 0$, that is, the first splitting for each i, which is then made according to the rule of the usual RMD. For all the other i and k, we need only one weight vector $\mathbf{e}$ and one conditional variance, which are computed as

$$\mathbf{e} = \Gamma^{(1,2)} \left(\Gamma^{(2,2)} \right)^{-1}, \quad \mathrm{Var}\left[X_{i,2k+1} \mid X_{i,2k}, X_{i-1,k+1} \right] = \frac{\det \Gamma}{\det \Gamma^{(2,2)}} \cdot 2^{-2Hi},$$

where

$$\Gamma = \begin{bmatrix} 1 & \Gamma^{(1,2)} \\ \Gamma^{(2,1)} & \Gamma^{(2,2)} \end{bmatrix} = \begin{bmatrix} 1 & 2^{2H-1} - 1 & 2^{2H-1} \\ 2^{2H-1} - 1 & 1 & \frac{1}{2}(3^{2H} - 2^{2H} - 1) \\ 2^{2H-1} & \frac{1}{2}(3^{2H} - 2^{2H} - 1) & 2^{2H} \end{bmatrix}$$

2.3 On-the-Fly RMD$_{mn}$ Generation

Without the on-the-fly generation capability, performance simulation of FBM queues with long synthetic input traces cannot be achieved. This is especially important in network environments where Quality-of-Service (QoS) objectives are stringent such as the case in ATM.

At the first sight, the RMD$_{mn}$ algorithm looks unsuitable for generating FBM "on-the-fly", which could be a requirement of a synthetic traffic generator for the purpose of simulation and load testing, e.g., to determine the queuing behavior and performance of ATM switches. However, the basic algorithm described in the previous section can be extended to have such a capability because of the following observation: *it is not necessary to generate the splits in exactly the same order as above.* That is, instead of completely generating each resolution before moving to the finer one, we can have several unfinished resolutions at the same time.

An "on-the-fly" FBM generation algorithm may be constructed so that after the interval $[0, 2^i\delta]$ is generated with resolution δ, the trace is "expanded" to the interval $[0, 2^{i+1}\delta]$. Naturally, the algorithm slows down after each "expansion",

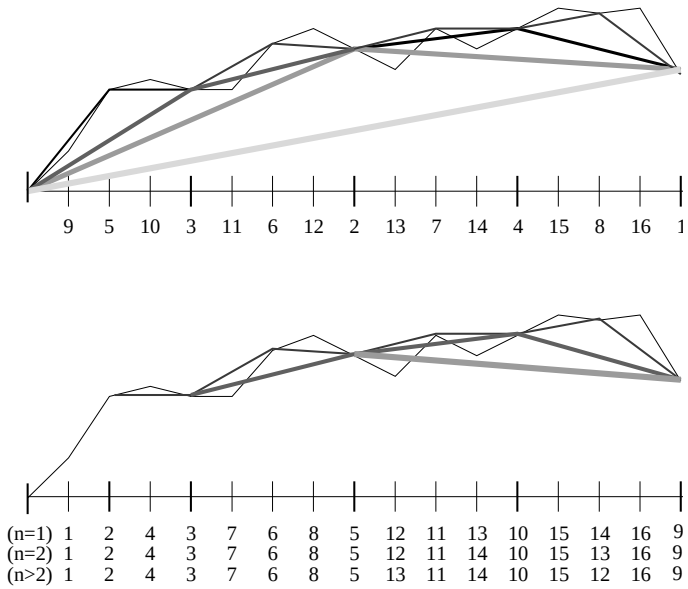

Figure 2.2: Running the RMD_{mn} algorithm over an interval and the on-the-fly version. The order of the splits is shown under the axis.

but if we generate the increments in an appropriate order, the speed of the slow-down is only logarithmic. The appropriate way is to choose the new increment to be as left as possible, i.e., the interval under consideration must have $n-1$ (or all) nearest intervals on the right hand side of the same size as the mother interval. In addition to these intervals, m (or all) nearest intervals on the left, whose length is same as the new interval, are taken into account in conditioning. (The algorithm could be still improved by counting, whenever it is possible, m neighboring intervals from the left; if there do not exist m intervals of the same size as the new interval, take m intervals as large as possible. For example, in the first step in figure 2.3, $Z(16\delta) - Z(8\delta)$ would be drawn conditioned on $Z(8\delta) - Z(4\delta)$ and $Z(4\delta) - Z(0)$.) Denoting

$$Y_{i,j} = Z(j \cdot 2^i \delta) - Z((j-1) \cdot 2^i \delta), \quad i = 0, 1, \ldots, \quad j = 1, 2, \ldots, \tag{2.3}$$

where δ is the desired resolution, then similar relations as in equation (2.1) can be written. The enlargement is defined by

$$Y_{i,1} = \mathrm{E}\left[Y_{i,1} \mid Y_{i-1,1}\right] + \sqrt{\mathrm{Var}\left[Y_{i,1} \mid Y_{i-1,1}\right]}\, U_{i,1} = \mathbf{e}(i,1) Y_{i-1,1} + \sqrt{v(i,1)}\, U_{i,1}$$

and increments on the inner intervals by

$$Y_{i,2k+1} \;=\; \mathbf{e}(i,k)[Y_{i,(2k+1-m)\vee 1}, \ldots, Y_{i,2k}; Y_{i+1,k+1}, \ldots, Y_{i+1,(k+m)\wedge N_{i+1}}]^{\mathrm{T}}$$

$$+\sqrt{v(i,k)}\; U_{i,k},$$

where N_{i+1} is the last generated increment on the resolution 2^{i+1}. One should notice that in the "on-the-fly" case the order of the splits depends on the parameter n, while in the basic algorithm it is always the same (see figure 2.2).

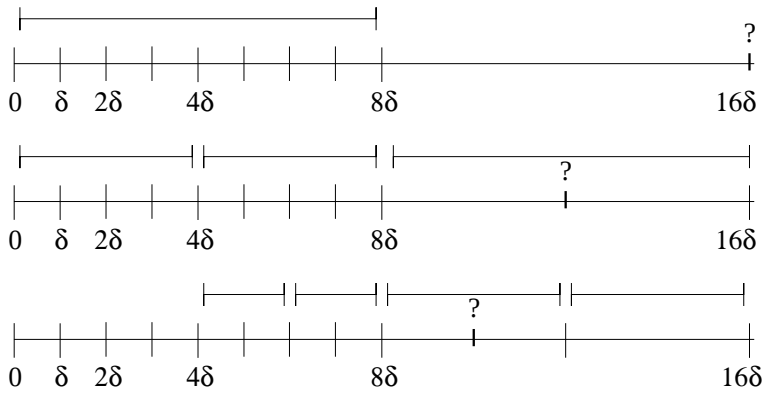

Figure 2.3: Conditionalizing the "on-the-fly" generation algorithm with $m = n = 2$. The line segments above the axis correspond to the intervals (i.e., increments) to be counted in the conditionalizing.

The critical aspect in the "on-the-fly" generator, related to the speed of the slowdown, is the memory usage. Assume that we have generated the interval $[0, T]$ with the resolution δ and the trace is enlarged to $[0, 2T]$. In order to generate $Z(T+\delta)$ we need either m or all the nearest values from each resolution on the left, i.e, all the values from the resolutions T, $T/2$, $T/4$, ..., $\lceil T/m \rceil$ and m nearest values from the resolutions δ, 2δ, 4δ, ..., $\lfloor T/m \rfloor$ need to be stored. Summing up the number of the needed values from the left, we get

$$\#_{\text{left}} = 1 + 1 + 2 + 4 + \ldots + 2^{\lfloor \log_2 m \rfloor} + \left(\log_2(\frac{T}{\delta}) - \lceil \log_2 m \rceil \right) \frac{m}{2} \leq m \log_2(\frac{T}{\delta}).$$

From the right we must keep all the generated values in the memory. The rule of choosing an interval to be split guarantees that there can never exist more than $n+1$ intervals of the size 2δ and n intervals of the coarser resolutions (if n is even, then the worst possible case is n intervals of the size 2δ and $n-1$ intervals of the coarser resolutions), i.e.,

$$\#_{\text{right}} \leq 1 + n \log_2 \left(\frac{T}{2\delta} \right) \leq n \log_2(\frac{T}{\delta}), \quad n \geq 1.$$

Moving towards the point $2T$ decreases $\#_{\text{right}}$ and increases $\#_{\text{left}}$, nevertheless, the latter is bounded by $m \log_2(2T/\delta)$. Thus, by a clever memory management one can generate simulation traces on the interval $[0, 2T]$ with less than $m \log_2(2T/\delta) + n \log_2(T/\delta)$ values of the trace stored at a time. For example, to generate a one-day simulated trace with the resolution of one millisecond and $m = n = 2$ we need only about one hundred memory locations.

2.4 Exact Covariances of $\text{RMD}_{0,1}$ and $\text{RMD}_{1,1}$

For the usual $\text{RMD}_{0,1}$, the true covariances of the simulated process can be easily determined. Indeed, given the increments X_i and X_j on two disjoint intervals I_i and I_j, the further evolution of the simulation inside these intervals proceeds independently. If $I_{i'}$ is either half of I_j and $I_{j'}$ is either half of I_j, then we have

$$\begin{aligned}\text{Cov}\left(X_{i'}, X_{j'}\right) &= \text{Cov}\left(\frac{1}{2}X_i + \sqrt{v(i'-1)}U_{i'}, \frac{1}{2}X_j + \sqrt{v(j'-1)}U_{j'}\right) \\ &= \frac{1}{4}\text{Cov}\left(X_i, X_j\right).\end{aligned}$$

Iterating this, we see that the correlation coefficient of the increments X_i, X_j on two intervals of equal size is

$$\text{Corr}\left(X_i, X_j\right) = 2^{(k(i,j)-1)(2H-2)}\text{Corr}\left(Z_1, Z_2 - Z_1\right) = 2^{k(i,j)(2H-2)}(2^{2H-1} - 1),$$

where $k(i, j)$ is the number of generation steps from I_i and I_j up to their closest common ancestor. The covariance matrix

$$\Gamma^{(i)} = \mathbf{Cov}[X_{i,1}, X_{i,2}, \ldots, X_{i,2^i}]$$

can be computed by a simple rule: start with $\Gamma^{(0)} = [1]$ and proceed from $\Gamma^{(i)}$ to $\Gamma^{(i+1)}$ by replacing each diagonal element of $\gamma_{jj}^{(i)}$ of $\Gamma^{(i)}$ by the 2×2 matrix

$$\gamma_{jj}^{(i)} \begin{bmatrix} 2^{-2H} & \frac{1}{2} - 2^{-2H} \\ \frac{1}{2} - 2^{-2H} & 2^{-2H} \end{bmatrix}$$

and each non-diagonal element $\gamma_{jk}^{(i)}$ $(j \neq k)$ by the 2×2 matrix

$$\gamma_{jk}^{(i)} \begin{bmatrix} \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} \end{bmatrix}.$$

The matrix $\Gamma^{(5)}$ is shown in Figure 2.4. The true covariance matrix, shown in the next figure, is thus approximated by "terrace architecture".

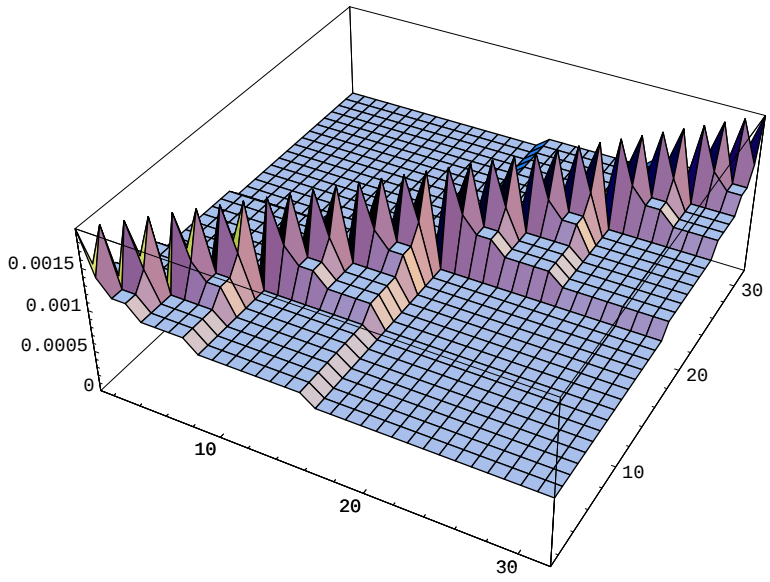

Figure 2.4: The 32×32 covariance matrix of $\mathrm{RMD}_{0,1}$. $H = 0.9$.

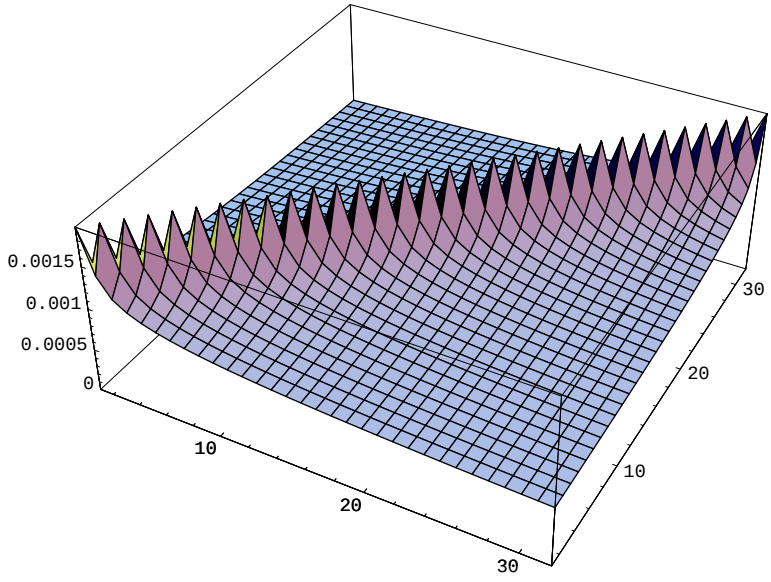

Figure 2.5: The true 32×32 covariance matrix of FGN. $H = 0.9$.

When $m > 0$ and/or $n > 1$, computation of the exact covariance matrix of the RMD_{mn} process is more complicated, since the dependence of increments is not restricted to the common ancestor type, and each split corresponds to a less trivial extension of the covariance matrix than by $\mathrm{RMD}_{0,1}$. We have derived this in the next simplest case $\mathrm{RMD}_{1,1}$. The 32×32 matrix is shown in Figure 2.6. Note that the asymmetricity of the algorithm causes a strange, robust look of the surface.

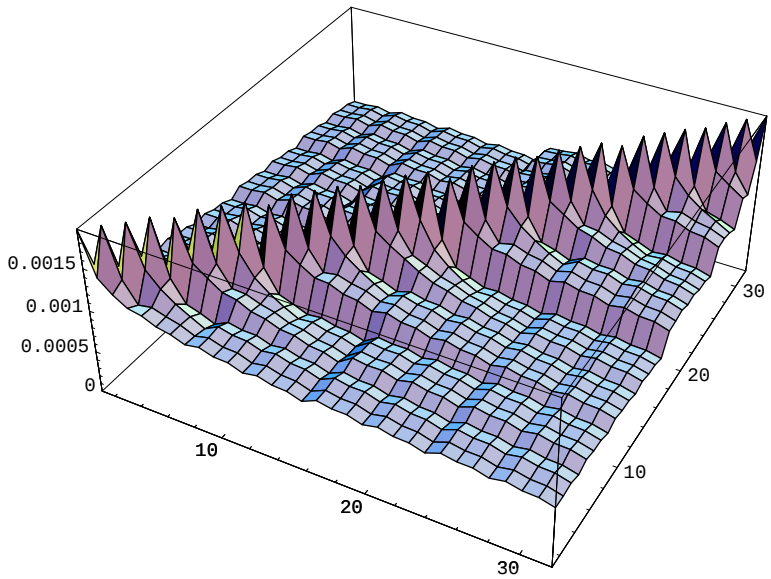

Figure 2.6: The 32×32 covariance matrix of $\mathrm{RMD}_{1,1}$. $H = 0.9$.

Let us have a closer look at some correlations. First, note that $\mathrm{RMD}_{0,1}$ gives the variances of the dyadic increments X_i exactly right, whereas $\mathrm{RMD}_{1,1}$ doesn't — see Figure 2.7. We see an interesting fractal shape. The behavior of these curves remains to be further investigated and is beyond the scope of this paper.

3 Evaluation of RMD_{mn}

We have coded a prototype program (`http://www.vtt.fi/tte/tte23/cost257/`) for generating approximate FBM traces for a given interval with RMD_{mn}. The program is written in C and, in addition to the standard C–libraries, it uses the random number library Ranlib and the matrix algebra library Meschach, both available via Netlib (`http://www.netlib.org/`).

As a starting point, we show in Figure 3.1 synthetic traces generated by

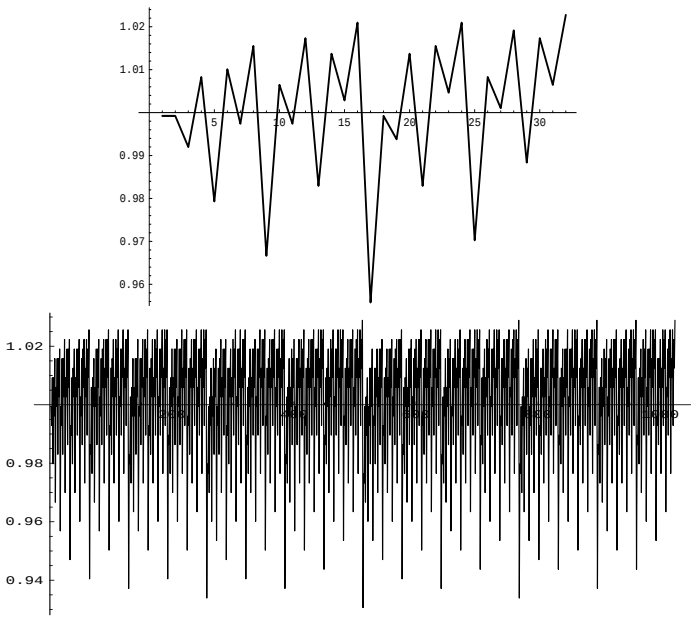

Figure 2.7: The diagonals of the 32×32 and 1024×1024 covariance matrices of $\mathrm{RMD}_{1,1}$, each normed by the true FGN value. $H = 0.75$.

$\mathrm{RMD}_{1,2}$ with input $H = 0.5, 0.6, 0.7, 0.8, 0.9,$ and 0.95. We can see that, as the H value increases, the traces indeed become more and more (long-term) correlated (shown as a low frequency fluctuation in the figure). In contrast, the trace for $H = 0.5$, corresponding to independent increments, does indeed resemble white noise. Thus, the RMD_{mn} algorithm generates traces that qualitatively resemble FBM. We next provide a quantitative assessment of the RMD_{mn} algorithm and compare it with other widely favored approaches such as the method based on the fast Fourier transform (referred to as the FFT method) [17] and the method based on aggregating a large number of ON-OFF sources with infinite-variance sojourn times [22] (referred to as the aggregation method).

3.1 Accuracy Analyses

3.1.1 Variations in the Sample Path

As an internal check of the accuracy of the algorithm with different values of m and n, we rely on the algorithm being very accurate when m and n are large, and choose the trace $\tilde{Z}(t)$ calculated with parameters $H = 0.8$, $m = 50$ and $n = 25$ as a reference trace. Differences between this and simulations with parameters

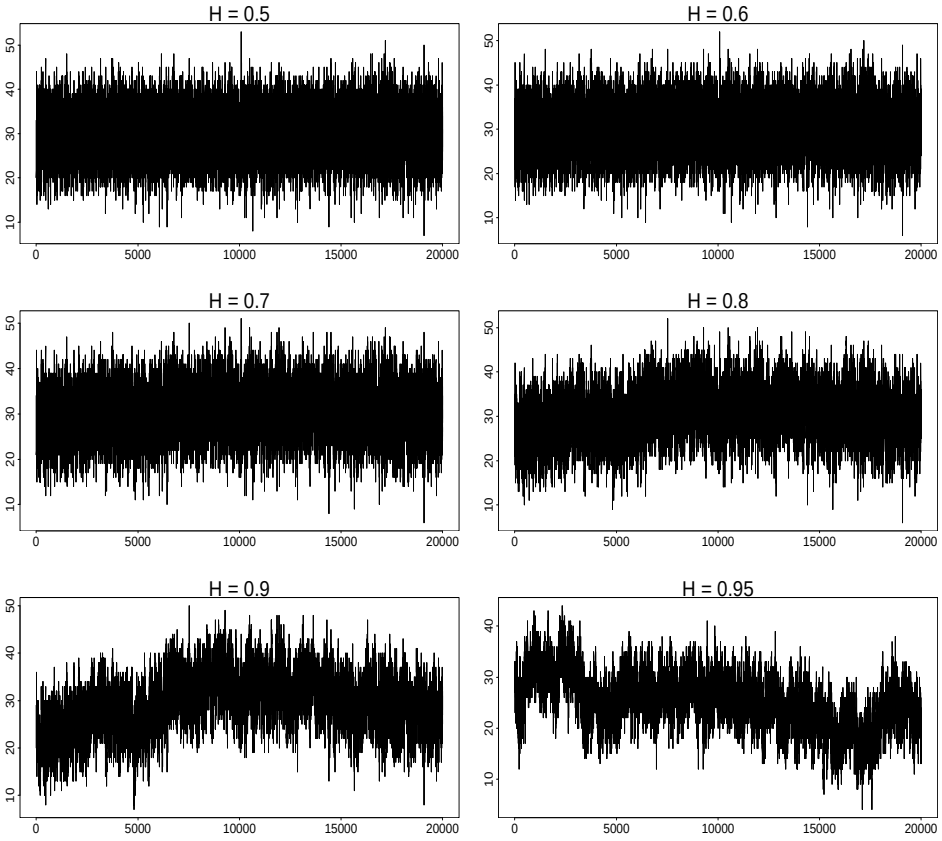

Figure 3.1: Synthetic traffic traces generated by $\mathrm{RMD}_{1,2}$.

$(m = 0, n = 1)$, $(m = 2, n = 2)$, and $(m = 6, n = 4)$ are shown in figure 3.2. Note that all the simulations are performed with same values of the random number generator. One way to measure the error is to consider absolute values of the differences of the increments. An error function may be defined as

$$\text{error} = \frac{\sum_{k=1}^{N} |\tilde{X}_k - X_k|}{\sum_{k=1}^{N} |\tilde{X}_k|},$$

i.e., relative error in the ℓ_1–norm, where $X_k = Z(t_k) - Z(t_{k-1})$ and $\tilde{X}_k = \tilde{Z}(t_k) - \tilde{Z}(t_{k-1})$. The observed values of the error function with different trace lengths and parameters m and n are shown in table 3.1. It seems to be quite obvious that there is no need to require more than a few, in this case four, nearest neighbors in order to get a good simulation trace. The other remarkable fact observed is that the length of a trace does not play any significant role in the error. The reason for that is, of course, the self-similarity of fractional Brownian motion.

		Length of trace			
		2^{12}	2^{14}	2^{16}	2^{18}
$m = 0$	$n = 1$	0.2138	0.2203	0.2256	0.2286
$m = 2$	$n = 2$	0.0126	0.0128	0.0131	0.0132
$m = 4$	$n = 3$	0.0055	0.0055	0.0056	0.0056
$m = 6$	$n = 4$	0.0028	0.0028	0.0028	0.0028
$m = 8$	$n = 5$	0.0020	0.0019	0.0019	0.0018
$m = 10$	$n = 6$	0.0014	0.0015	0.0014	0.0013

Table 3.1: Relative ℓ_1-error as a function of trace length

3.1.2 Hurst Parameter

We next provide an assessment of the method by evaluating the difference between the target (input to the method) and the output Hurst parameters. The evaluation is performed by randomly generating 100 traces (for each target Hurst parameter) and estimate the output Hurst parameter based on the wavelet estimation method [3, 2] which has been proven to be unbiased and robust (against deterministic trends, for example). These 100 estimates for each target Hurst parameter value are then averaged and plotted on Figure 3.3 against 16 choices of (m, n) pairs ranging from (0, 1) (ordinary RMD) to (20, 10). The estimation is set up so that the 95% confidence intervals are the same for all the cases we simulated.

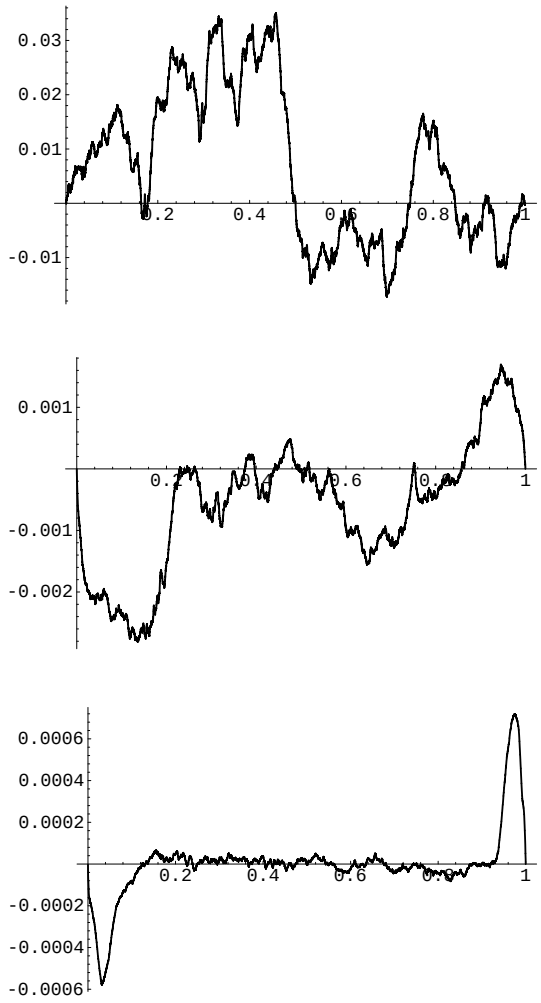

Figure 3.2: Differences between the reference trace ($m = 50, n = 25$) and traces ($m = 0, n = 1$), ($m = 2, n = 2$) and ($m = 6, n = 4$).

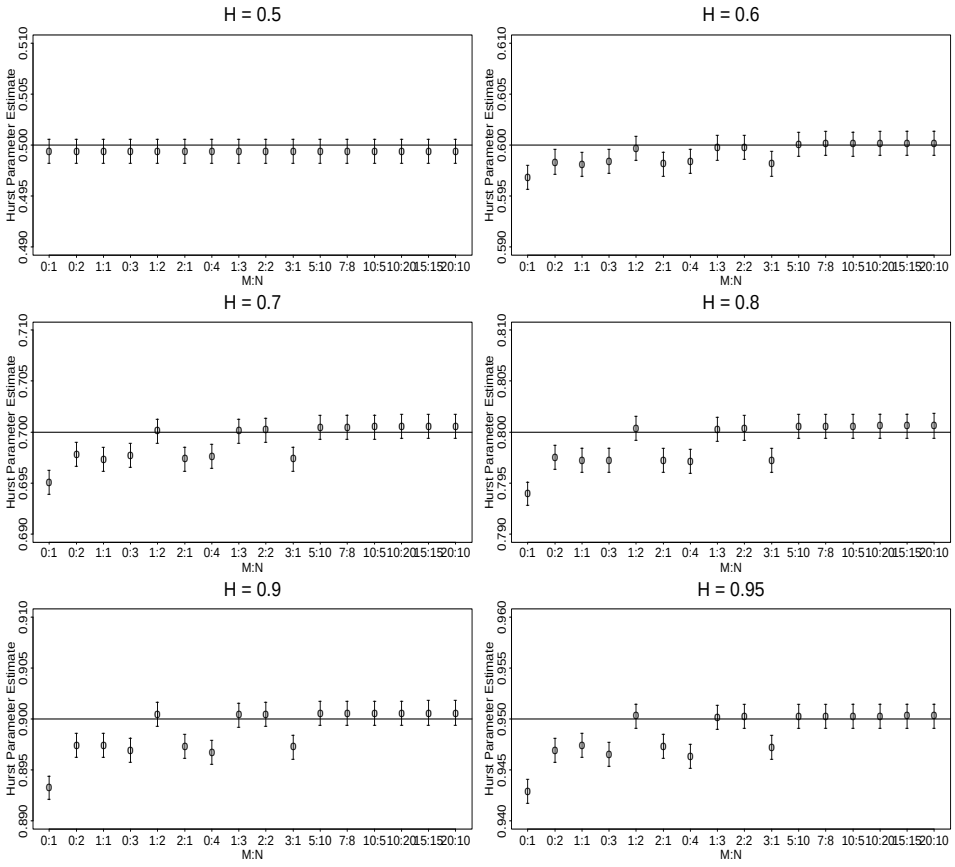

Figure 3.3: Comparing the estimated Hurst parameters of the traces generated by the RMD_{mn} algorithm and the target Hurst parameter values.

We can see that in general the quality of the traces is quite good, i.e., the Hurst parameters of the simulated traces are close to the target values, and as expected as the m and/or n increase, the quality of the generated traces improves. Note that traditional RMD produces traces with too low Hurst parameter, and the same holds, to lesser extent, for RMD_{mn} when m and n are very small, with one surprising exception: with $m = 1$ and $n = 2$, the Hurst parameters of the synthetic traces are extremely close to the target values.

3.2 Comparison with Other Methods

In this section, we compare the RMD_{mn} algorithm with two other popular generation methods: the fast Fourier transform (FFT) method (note that a version of the FFT method was proposed in the ATM Forum as a method to generate synthetic self-similar traffic traces) and the method based on aggregation of ON-OFF sources with heavy-tailed sojourn times. The comparison is based on the time complexity and the quality of the generated traces in terms of the matching between the Hurst parameter values of the synthetic traces and their targets.

The left panel of Figure 3.4 demonstrates the quality of the FFT method by depicting the difference of the average Hurst parameter estimates of 100 generated traces and their target values against the target Hurst parameter value. The right panel shows the same figure-of-merit for the $\text{RMD}_{1,2}$ algorithm. We can see that in terms of the Hurst parameter estimates of the generated traces, these two algorithms are quite comparable.

For the aggregation method, we found that the Hurst parameter value of the generated trace is close to its target value *asymptotically*, i.e., when the time scale is *large*. However, the convergence to the asymptotic result (i.e., the target Hurst value) may be slow. This is demonstrated in Figure 3.5 which shows the variance-time plot [10] of a trace generated based on the aggregation method with a target Hurst parameter value of 0.5. We see that the Hurst parameter estimate is only close to 0.5 (i.e., parallel to the dotted line) when the time scale is sufficiently large. This issue could potentially hinder the use of the aggregation method in practice. (Note that the theory states that aggregation in time and space (i.e. number of sources) with proper normalization is required for the convergence to FBM to take place, thus the discussion here relating to the quality and complexity of the aggregation-based method needs to be taken with care. However, one needs to be aware of the trade-off between accuracy and time-complexity with the need for large aggregations (both in space and time) when using the aggregation-based method.)

The comparison of the time complexity of these three algorithms (for generating a synthetic trace with 2^{17} points) are depicted in Figure 3.6, with the

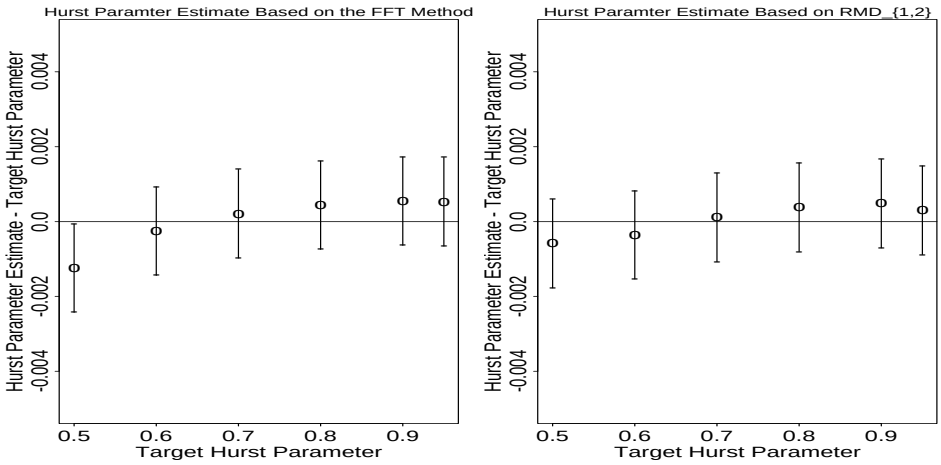

Figure 3.4: Comparing the quality of traces generated by the FFT method and the $\mathrm{RMD}_{1,2}$ algorithm.

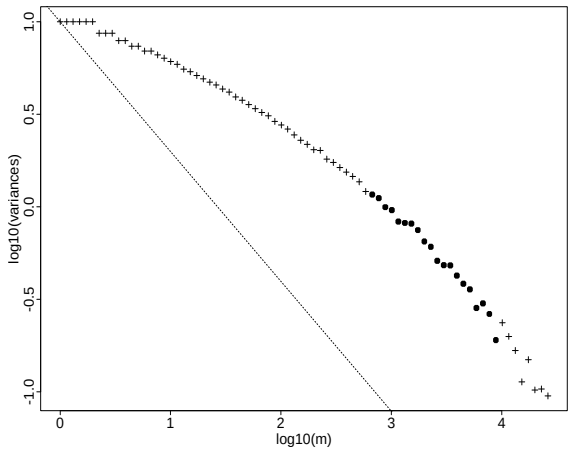

Figure 3.5: Variance-time plot of a trace generated by the aggregation method with target Hurst parameter value of 0.5.

left panel showing the computation time in seconds against the length of the generated trace and the right panel showing time against the length of the generated trace in *logarithmic 2 base*. We see that the time of generating a synthetic trace for all three algorithms increases linearly with the length of the trace, with $\text{RMD}_{1,2}$ slightly faster than the FFT method, and both are *much* faster than the aggregation method. (All three algorithms were coded in C and no efforts have been attempted to optimize the codes; therefore, the difference in the *absolute* computational time may be debatable.) Note that the time complexity of the aggregation method depends on the number of sources and the target Hurst parameter value. The curve shown here is for a target Hurst parameter value of 0.7 and 30 aggregated ON-OFF sources. Furthermore, the aggregation method was originally suggested in the context of a parallel computing architecture [22], where the method can be easily implemented simply by multiplexing. When using the FFT or RMD_{mn} algorithms, it is not so clear whether the benefit of parallelizing is as large as in the aggregation-based methods.

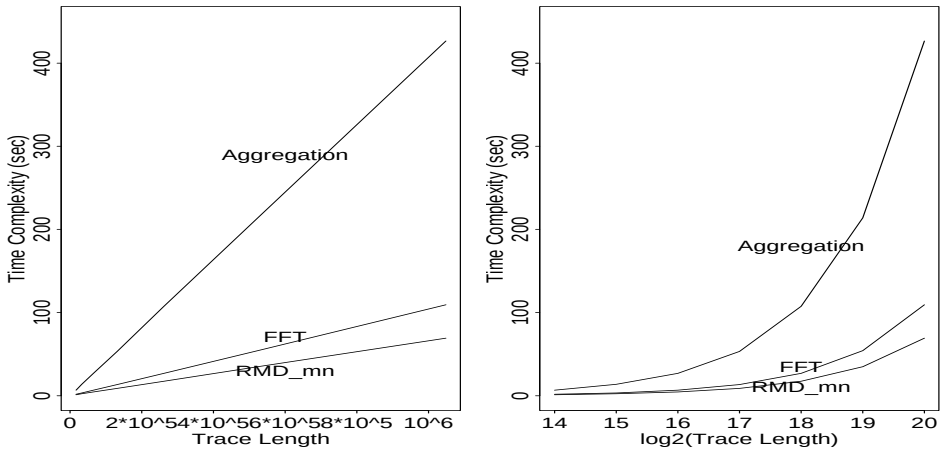

Figure 3.6: Comparing the time complexity of the FFT method and the $\text{RMD}_{1,2}$ algorithm.

3.3 Implementation Issues and Computational Complexity

The main structure of our implementation and some statistics about runtime are presented in this section.

Because the covariance matrices Γ_{ik} are needed at each step of the algorithm and, furthermore, they are independent of i if multiplied by the scaling term

2^{-2Hi}, it is natural to calculate them in the beginning. Equivalently to (2.2) we have

$$\mathbf{e}(i,k) = \Gamma_{ik}^{(1,2)} \left(\Gamma_{ik}^{(2,2)}\right)^{-1}$$

and

$$v(i,k) = \frac{\det \Gamma_{ik}}{\det \Gamma_{ik}^{(2,2)}} = \mathrm{Var}(X_{i,2k+1}) - \Gamma_{ik}^{(1,2)} \left(\Gamma_{ik}^{(2,2)}\right)^{-1} \Gamma_{ik}^{(2,1)},$$

where the last equivalence is due to simple manipulations of matrices. Because $\Gamma_{ik}^{(2,2)}$ is a symmetric matrix, we can solve $\mathbf{e}(i,k)$ and $v(i,k)$ using the Cholesky method. (If m and n are small enough, $\mathbf{e}(i,k)$ and $v(i,k)$ can be solved exactly, e.g., with Mathematica; thus, if one knows beforehand that neither large m nor large n are needed then it is maybe better to use the exact covariance matrices.) After initialization of $\mathbf{e}(i,k)$ and $v(i,k)$, carrying through the FBM simulation is only a matter of generating random numbers and performing simple matrix algebra.

Runtime as a function of trace length and conditionalizing parameters is shown in table 3.2. The real runtime varied from one second to one minute demonstrating that the algorithm is really a fast one. The difference in runtime of the traces with different m and n is mainly due the time spent in the initialization – the bigger the covariance matrix the slower the Cholesky method. After solving $\mathbf{e}(i,k)$ and $v(i,k)$, the simulation runs about same speed for all n and m not too big, say, smaller than 30.

Conditionalizing parameters		**Length of trace**			
		2^{12}	2^{15}	2^{18}	2^{21}
$m = 0$	$n = 1$	0.02	0.04	0.26	3.39
$m = 4$	$n = 3$	0.02	0.04	0.26	3.40
$m = 10$	$n = 6$	0.02	0.05	0.27	3.42
$m = 20$	$n = 12$	0.07	0.10	0.32	3.49
$m = 30$	$n = 20$	0.28	0.32	0.53	3.70

Table 3.2: CPU time in seconds elapsed when simulating FBM in a HP 9000/J210XC workstation.

4 Conclusions

In this paper, we considered an approximate FBM generation method, RMD_{mn}, based on the concept of bisection and interpolation, which is a generalization

of the ordinary random midpoint displacement (RMD) algorithm. We further modified RMD_{mn} to be able to generate FBM traces on-the-fly. We discussed both mathematical and numerical aspects of RMD_{mn} as well as compared it with two other methods widely favoured in generating FBM traces for teletraffic modeling purposes: the fast Fourier transform (FFT) method and the method based on aggregating a large number of ON-OFF sources with infinite-variance sojourn times.

One should notice that the fair comparison between different methods is a very difficult task while all approximate generation methods have their pro's and con's. By choice of the aspects to compare, almost any method can be oversold. Our comparison is strictly based on the criteria mentioned in this paper, and other criteria, such as the need to demultiplex the trace into different individual traffic streams, may need to be considered in the choice of the appropriate traffic generation method in practice.

We have made the following observations:

- the time complexity of all three algorithms grows linearly with the desired trace length;
- the aggregation method requires a couple of orders of magnitude of time periods to converge to the asymptotic results, and is therefore much slower;
- the FFT method is comparable to the RMD_{mn} algorithm in terms of quality and time complexity, however, the FFT method is strictly a top-down algorithm, that is, the whole trace has to be generated before it can be used in any application;
- the RMD_{mn} algorithm gives quite satisfactory results both in terms of the trace quality as well as the computational efficiency, in addition to the ability to generate synthetic traces on-the-fly, and thus, in our opinions, is a better choice for the FBM traffic generation among the three.
- it is possible to design algorithms based on some other techniques (e.g. wavelets), which are superior to RMD_{mn} in some aspects, but outperforming the simplicity of the RMD_{mn} algorithm may be difficult.

As a final remark, we note that it is straightforward to generalize RMD_{mn} to produce good traces of any centered Gaussian process Y with stationary increments and a variance function $v(t) = \text{E}Y_t^2$. The only difference to our case is that without self-similarity, one has to generate the needed multiplier vectors and constants for each resolution separately, which increases memory requirements and the duration of the initialization phase. When this has been done, the generation algorithm runs as fast as in our case.

References

[1] P. Abry and D. Sellan. The Wavelet-based Synthesis for the Fractional Brownian Motion Proposed by F. Sellan and Y. Meyer: Remarks and Fast Implementation. *Appl. Comp. Harmonic Anal.*, 3:377-383, 1996.

[2] P. Abry and D. Veitch. Long-range dependence: Revisiting aggregation with wavelets. *Journal of Time Series Analysis*, 19(3):253–266, 1998.

[3] P. Abry and D. Veitch. Wavelet analysis of long-range dependent traffic. *IEEE Trans. Info. Theory*, 44(1):2–15, 1998.

[4] D.R. Cox. Long-range dependence: A review. In H.A. David and H.T. David, editors, *Statistics: An Appraisal*, pages 55–74. The Iowa State University Press, Ames, Iowa, 1984.

[5] P. Flandrin. Wavelet analysis and synthesis of fractional Brownian motion. *IEEE Transactions on Information Theory*, 38(2), 1992.

[6] C.W.J. Granger. Long memory relationships and the aggregation of dynamic models. *J. Econometr.*, 14:227–238, 1980.

[7] J.L. Jerkins and J.L. Wang. A measurement analysis of ATM cell-level aggrate traffic. In *Proc. IEEE Globecom*, pages 1589–1595, Phoenix, AZ, USA, November, 1997.

[8] A.N. Kolmogorov. Wienersche Spiralen und einige andere interessante Kurven im Hilbertschen Raum. *C.R. (Doklady) Acad. Sci. USSR (N.S.)*, 26:115–118, 1940.

[9] W.-C. Lau, A. Erramilli, J.L. Wang, and W. Willinger. Self-similar traffic generation: The random midpoint displacement algorithm and its properties. In *1995 IEEE International Conference on Communications (ICC'95)*, pages 466–472, Seattle, USA, 1995.

[10] W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson. On the self-similar nature of Ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, 2(1):1–15, February 1994.

[11] B.B. Mandelbrot. A fast fractional Gaussian noise generator. *Water Resources Research*, 7:543–553, 1971.

[12] B.B. Mandelbrot. *Fractals. Form, Chance, and Dimension.* W.H. Freeman and Company, San Francisco, 1977.

[13] B.B. Mandelbrot and J.W. Van Ness. Fractional Brownian motions, fractional noises and applications. *SIAM Review*, 10:422–437, 1968.

[14] I. Norros. Studies on a model for connectionless traffic, based on fractional Brownian motion. Technical Report 242TD(92)041, COST, 1992. (Presented also in June 1993 at the Conference on Applied Probability in Engineering, Computer and Communication Sciences, Paris).

[15] I. Norros. On the use of fractional Brownian motion in the theory of connectionless networks. *IEEE Journal on Selected Areas in Communications*, 13(6), August 1995.

[16] I. Norros, E. Valkeila, and J. Virtamo. An elementary approach to a Girsanov formula and other analytical results on fractional Brownian motions. To appear in Bernoulli in 1999.

[17] V. Paxson. Fast, approximate synthesis of fractional Gaussian noise for generating self-similar network traffic. *Computer Communication Review*, 27(5):5–18, 1997.

[18] V. Paxson and S. Floyd Wide-area traffic: The failure of Poisson modeling. In *Proc. ACM Sigcomm*, pages 257–268, London, UK, 1994.

[19] P. Pruthi. *An Application of Chaotic Maps to Packet Traffic Modeling*. PhD thesis, Royal Institute of Technology, Dept of Teleinformatics, 1995. ISSN 1103-534X.

[20] F. Sellan. Synthèse de mouvements browniens fractionnaires à l'aide de la transformation par ondelettes. *C.R. Acad. Sci. Paris, Série I*, 321:351–358, 1995.

[21] M.A. Stoksik, R.G. Lane, and D.T. Nguyen. Accurate synthesis of fractional Brownian motion using wavelets. *Electronics Letters*, 30(5):383–384, 1994.

[22] W. Willinger, M.S. Taqqu, R. Sherman, and D.V. Wilson. Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level. In *Proc. ACM/Sigcomm'95*, pages 100–113, Cambridge, MA, USA, 1995.