The Generation of Stationary Gaussian Time Series

Michael A. Hauser and Wolfgang Hörmann

University of Business Administration Vienna

Institut für Statistik

forthcoming

International Journal in Computer Simulation, 1997

Abstract: Three different algorithms for the generation of stationary Gaussian time series with given autocorrelation function are presented in this paper. The algorithms have already been suggested in the literature but are not well known and have never been compared before. Interrelations between the different methods, advantages and disadvantages with respect to speed and memory requirements and the range of autocorrelation functions for which the different methods are stable are discussed. The time-complexity of the algorithms and the comparisons of their implementations show that the method twice using the Fourier transform is by far the most efficient if time series of moderate or large length are generated. A tested C-code of the latter algorithm is included as this method is tricky to implement and very difficult to find in the literature. (We know only one reference, that gives a correct algorithm, but there the description is very short and no proof is included.)

Key Words:

random number generation, Cholesky decomposition, Durbin algorithm, fast Fourier transform, fractionally integrated processes

1. Introduction

The generation of stationary Gaussian time series is of interest by itself. Moreover, the potential applications are numerous, eg. in simulation estimation methods, stochastic simulation tests, calibration, apart from the common way to investigate the finite sample properties of estimators or tests. It is well known that for time series models which approximate the infinite moving average, MA, representation by rational functions (autoregressive moving average ARMA models) there exist rather quick generation procedures. However, for example for fractionally integrated models, cf. [1] and [2] p. 520ff, which are increasingly often applied (cf. eg. [3]), this approach cannot be used. Fractionally integrated processes are characterized by (in absolute value) hyperbolically decreasing autocorrelation functions, so that far distant observations are still correlated to a nonnegligible amount. Therefore we consider general (weakly) stationary processes. There are several papers describing the generation of stationary Gaussian time series (eg. [1], [4] and [5]). In most of them the proposed algorithm is hidden in a short section explaining how the simulation results were obtained. This seems to be the reason that the different ideas are not really well known. Therefore this paper gives a detailed explanation of the underlying ideas. It discusses the close relationships which were not pointed out in literature and compares the performance of the algorithms.

Section 2 presents the Cholesky decomposition method, which is algebraically equivalent to the conditional distribution method analyzed in Section 3. Section 4 discusses the Fourier transform method and gives its C-code. Section 5 compares the performance and applicability of the algorithms and variants thereof. Section 6 summarizes the results.

2. Cholesky decomposition

The best known standard method for generating samples of length n from the multidimensional normal distribution with zero expectation and given covariance matrix Σ is based on the decomposition of Σ into $\Sigma = LL'$ where L denotes a lower triangular matrix, which is called Cholesky factor. We get

Algorithm 1: (Cholesky decomposition method)

0: [setup] Compute the Cholesky factor L of the covariance matrix Σ.

1: Generate a vector $\varepsilon = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n)'$ of n independent standard normal variates.

2: Return the vector $x = L\varepsilon$.

It is easy to see that Algorithm 1 works correctly since $\mathrm{E}(xx') = L\mathrm{E}(\varepsilon\varepsilon')L' = LL' = \Sigma$, with E the expectation operator. It can be used to generate stationary Gaussian time series but it does not make use of the special structure of Σ, which is a symmetric Toeplitz matrix. The best available routines to compute the Cholesky factorization of arbitrary positive definite matrices need $n^3/3 + O(n^2)$ flops (floating point operations, ie. additions or multiplications) and there is no faster routine available for Toeplitz matrices (cf. [6]). Therefore the setup of Algorithm 1 becomes slow even for moderate n and a lot of memory is necessary as well, at least $n^2/2 + O(n)$ floating point numbers. The generation of one time series requires n Gaussian random numbers and $n^2 + O(n)$ flops. It is interesting that it is not possible to improve Algorithm 1 for stationary time series in a direct way, but in the next section we will present a much more efficient algorithm and we will prove that it is equivalent to Algorithm 1.

3. Conditional distribution method

The conditional distribution method is a generally applicable method to generate an arbitrary random vector of length n. First generate x_1, a sample from the marginal distribution with respect to the first variable. Then proceed with $(x_2|x_1)$, $(x_3|x_1, x_2)$ and so on. For the joint normal distribution all conditional distributions are normal and it is not difficult to compute their means and variances. Using the notation

$$\Sigma_{t+1} = \begin{bmatrix} \Sigma_t & \Sigma'_{1t} \\ \Sigma_{1t} & \Sigma_{11} \end{bmatrix} \qquad \text{for } t = 1, \ldots, n-1$$

where Σ_t denotes the upper $t \times t$ submatrix of Σ we have: $(x_{t+1}|\mathcal{X}_t = (x_1, \ldots, x_t)')$ has expectation $\Sigma_{1t}\Sigma_t^{-1}\mathcal{X}_t$ and variance $\Sigma_{11} - \Sigma_{1t}\Sigma_t^{-1}\Sigma'_{1t}$. The vectors $\Sigma_{1t}\Sigma_t^{-1}$ and the standard deviations can be stored in a setup step. For a general multinormal distribution this results in an algorithm with a setup which is slower than that of Algorithm 1 whereas the time to generate one time series is the same (eg. [7] p. 99).

Although not evident, it is simple to prove that the two methods of Section 2 and 3 generate an identical time series as long as the same vector of iid Gaussian random variates is used. This holds also for general multidimensional Gaussian distributions, where Σ is only positive definite and symmetric. Astonishingly enough this simple fact was seemingly overlooked by the authors describing the generation of multi-dimensional normal vectors with the two standard methods.

For the Cholesky factor L of Σ and for the vector ε we use a partition similar to that of Σ:

$$L_{t+1} = \begin{bmatrix} L_t & 0 \\ L_{1t} & L_{11} \end{bmatrix} \qquad \mathcal{E}_{t+1} = \begin{bmatrix} \mathcal{E}_t \\ \varepsilon_{t+1} \end{bmatrix} \qquad \text{for } t = 1, \ldots, n-1$$

with $\mathcal{E}_t = (\varepsilon_1, \ldots, \varepsilon_t)'$ and $\mathcal{E}_{t+1} = (\varepsilon_1, \ldots, \varepsilon_{t+1})'$.

Proposition 1: Multiplying with the Cholesky factor and the conditional distribution method are algebraically equivalent. This means we have for all t:

$$L_{1t}\mathcal{E}_t + L_{11}\varepsilon_{t+1} = \Sigma_{1t}\Sigma_t^{-1}\mathcal{X}_t + \sqrt{\Sigma_{11} - \Sigma_{1t}\Sigma_t^{-1}\Sigma'_{1t}}\ \varepsilon_{t+1}$$

Proof: Using $\Sigma_{t+1} = L_{t+1}L'_{t+1}$ in the partitioned representation we get representations of Σ_{t+1} and Σ_{t+1}^{-1} in terms of L-matrices. Together with $\mathcal{X}_t = L_t\mathcal{E}_t$ this yields

$\Sigma_{1t}\Sigma_t^{-1}\mathcal{X}_t = L_{1t}\mathcal{E}_t$, and $\Sigma_{11} = L_{1t}L'_{1t} + L_{11}^2 = \Sigma_{1t}\Sigma_t^{-1}\Sigma'_{1t} + L_{11}^2$.$\square$

Now we restrict our attention to the stationary case with expectation zero and unit variance. In the following Σ will denote the correlation matrix of a stationary Gaussian time series with autocorrelations $r = (r_1, r_2, \ldots)'$ and $r_0 = 1$. The important advantage of the conditional distribution method lies in the fact that for a symmetric Toeplitz Σ the conditional expectation and variance of $\mathcal{X}_{t+1}$ can be expressed in terms of the solution y of the t-th order Yule-Walker equations $\Sigma_t y = -r$. And the Yule-Walker equations can be solved recursively taking only $2n^2 + O(n)$ flops using Durbin's Algorithm ([8], see also [6]). The idea to use Durbin's Algorithm was introduced in [1] and [4]. For further applications of the idea for time series analysis see [9].

Derivation of Algorithm 2:

We define the matrix W as the square matrix with ones in the secondary diagonal and zeros else. Thus $W = W' = W^{-1}$ and $W\Sigma^{-1} = \Sigma^{-1}W$ for any symmetric Toeplitz matrix Σ as Σ^{-1} is symmetric and persymmetric [6]. Now the first two conditional moments of x_{t+1} as given above can be expressed via the solution y of the Yule-Walker equations, $\Sigma_t y = -r$, $y = -\Sigma_t^{-1} r$. We have:

$$\mathrm{E}(x_{t+1}|\mathcal{X}_t) = (Wr)'\Sigma_t^{-1}\mathcal{X}_t = r'\Sigma_t^{-1}W\mathcal{X}_t = -(Wy)'\mathcal{X}_t$$

$$\mathrm{V}(x_{t+1}|\mathcal{X}_t) = 1 - (Wr)'\Sigma_t^{-1}Wr = 1 - r'\Sigma_t^{-1}r = 1 + r'y. \quad \Box$$

One advantage of Algorithm 2 over Algorithm 1 is that the number of flops for the setup and the generation of one time series is $4n^2 + O(n)$ instead of $n^3/3 + O(n^2)$, the second advantage are the linear storage requirements. We present a variant needing $4n + O(1)$ floating point numbers of storage, $4n^2 + O(n)$ flops for the first time series and $3n^2 + O(n)$ for additional time series. The formal description of the algorithm follows the presentation in [6] [$(1:n)$ denotes $1, 2, \ldots, n$; $(n:-1:1)$ denotes $n, n-1, \ldots, 1$].

Algorithm 2: (Marchenko and Ogorodnikov [4], and Hosking [1])

(Requires as input: n, the autocorrelations $r(1:n-1)$ and the Gaussian noise vector $\varepsilon(1:n)$. Puts the generated time series $x = L\varepsilon$ into the vector $x(1:n)$.)

Set $x(1) = \varepsilon(1);\ y(1) = -r(1);\ \beta = 1;\ \alpha = -r(1)$

$x(2) = -x(1) * y(1) + \varepsilon(2) * \sqrt{1 + r(1) * y(1)}$

for $k = 1 : n-2$

Durbin Algorithm computes the solution y of the Yule-Walker equations

Set $\beta = (1 - \alpha * \alpha) * \beta$

$\alpha = -(r(k+1) + r(k:-1:1)' y(1:k))/\beta$

for $i = 1 : (k+1)/2$

set $h(1) = y(i); h(2) = y(k+1-i)$

$y(i) = h(1) + \alpha * h(2)$

$y(k+1-i) = h(2) + \alpha * h(1)$

end

$y(k+1) = \alpha$

The conditional standard deviations $\sigma_k = \sqrt{V(x_k|\mathcal{X}_{k-1})}$ are computed only if the first time series for this r is generated

Set $\sigma(k+2) = \sqrt{1 + r(1:k+1)' y(1:k+1)}$

The result is put together

Set $x(k+2) = -x(1:k+1)' y(k+1:-1:1) + \varepsilon(k+2) * \sigma(k+2)$

end

Remark 1: It follows from Proposition 1 that Algorithm 2 can be used to multiply the Cholesky factor L of a Toeplitz Matrix with an arbitrary vector without computing L with linear storage requirements using only $4n^2$ flops. This is a remarkable fact as the computation of L itself requires at least $n^3/3$ flops.

Remark 2: If n is small the generation of additional time series with Algorithm 2 becomes considerably faster if the solutions y of the Yule-Walker equations are stored in an array of size $n^2/2 + O(n)$. The number of flops is reduced to $n^2 + O(n)$. This variant is called Algorithm 2a.

4. Fourier transform method

In many papers Fourier transform, FT, methods were suggested to generate stationary Gaussian time series. But the only paper we know that clearly states the correct algorithm is [5]. We need the definition of the discrete $\mathcal{F}(h)$ of N (possibly complex) points h_j, $j = 0, 1, \ldots, N-1$:

$$(\mathcal{F}(h))_k = \sum_{j=0}^{N-1} h_j e^{2\pi ikj/N} \quad \text{for} \quad k = 0, \ldots, N-1$$

and the inverse FT $\mathcal{F}^{-1}$:

$$(\mathcal{F}^{-1}(h))_k = \frac{1}{N} \sum_{j=0}^{N-1} h_j e^{-2\pi ikj/N} \quad \text{for} \quad k = 0, \ldots, N-1$$

Algorithm 3: (Davies and Harte [5])

(In our notation n is the length of the real valued time series to be generated, contrary to $n+1$ in [5]!)

1. (Setup) Compute the discrete Fourier transform g of length $N = 2(n-1)$ of a vector ρ of autocorrelations, $\rho = (\rho_0, \rho_1, \ldots, \rho_{N-1}) := (1, r_1, \ldots, r_{n-2}, r_{n-1}, r_{n-2}, \ldots, r_1)$:

$$g_k = \sum_{j=0}^{N-1} \rho_j e^{2\pi ikj/N} \quad \text{for} \quad k = 0, \ldots, N-1$$

Store the vector ϕ of length N with $\phi_k = \sqrt{g_k}$.

2. Compute a sequence z of length N out of N iid standard normal variates, so that z is conjugate symmetric (ie. $z_k = \bar{z}_{N-k}$, $n-1 < k < N$) with variances $N/2$ for the real and imaginary part, and z_0, z_{n-1} real with variances N.

3. The time series x of length n, $x = (x_1, \dots, x_n)'$ with

$$x_j = \frac{1}{N} \sum_{k=0}^{N-1} \phi_k z_k e^{-2\pi ijk/N} \quad \text{for} \quad 0 \le j < n$$

has the desired distribution. (x are the first n elements of the FT of $\phi_k z_k$ multiplied with $\frac{1}{N}$.)

In the following we propose to make use of efficient procedures and appropriate transformations in order to reduce the computation time:

Remark 1: As it is well known that the discrete Fourier transform can be performed in $O(N \log_2 N)$ instead of $O(n^2)$ steps using an algorithm called the fast Fourier transform (FFT). However the standard FFT procedure works only for N's of the form 2^k with k is the smallest integer so that $N \le N_0 = 2^k$. The corresponding n is given by $n \le n_0 = 2^{k+1} + 1$. Thus we generate a series of length n_0 and throw away the x_j with $j \ge n$. To do this it is also necessary to elongate the autocorrelation vector up to the element $n_1 = n_0 - 1$.

Remark 2: We use the algorithm `drealft()` of [10], which computes $N/2$ times the inverse FFT of a conjugate symmetric vector. Therefore multiplications can be saved if we compute

$$x_j = \frac{1}{2} \sum_{k=0}^{N-1} \tilde{\phi}_k \tilde{z}_k e^{-2\pi ijk/N} \quad \text{for} \quad 0 \le j < n$$

with $\tilde{\phi}_k = \phi_k / \sqrt{N/2}$ for $0 < k < N/2$ and $\tilde{\phi}_0 = \phi_0 / \sqrt{N/4}$, $\tilde{\phi}_{N/2} = \phi_{N/2} / \sqrt{N/4}$ and the $\tilde{z}_k$ defined as the z_k but with variance 1.

Proof of correctness of Algorithm 3:

It is clear that the x_k are real since we have $\phi_k z_k = \bar{\phi}_{N-k} \bar{z}_{N-k}$ and normal as the x_k are the sum of independent normal variables. To compute the autocorrelations of the x_k it is enough to see that we have $E[z_k\bar{z}_k] = N$ and $E[z_k\bar{z}_l] = 0$ for $k \neq l$. Thus

$$E[x_p\bar{x}_q] = \frac{1}{N^2}\sum_{k=0}^{N-1}\sum_{m=0}^{N-1}\phi_k\phi_m e^{-2\pi i(pk-qm)/N}E[z_k\bar{z}_m] = \frac{1}{N}\sum_{k=0}^{N-1}\phi_k^2 e^{-2\pi i(p-q)k/N} = r_{p-q}$$

due to the Fourier inversion theorem and the symmetry of the autocorrelation. □

C-Implementation of Algorithm 3:

`fftrand` requires the routine `drealft()` the double precision version of `realft()` described in [10] p. 512 ff., which calculates the FFT of a sequence of real-valued data points, and `nacr()` a generator for standard Gaussian variates.

It implements Algorithm 3 using Remark 2 and generates a stationary Gaussian time series of length n in the first half of vector `data[0..2(n-1)-1]`. `n-1` must be a power of 2. The autocorrelations $(1, r_1, \ldots, r_{n-1})$ must be contained in the vector `r[0..n-1]`. In order to perform the initialization of the algorithm for a new autocorrelation vector set `*setup` equal to 1. The ϕ_k calculated in the setup are stored in `r[]` as well.

The setup builds up the vector ρ and computes its real FFT g which is – due to its symmetry – contained in `data[2*i]` (only g_{n_1} is stored in `data[1]`). For the case that the g_k are non negative $\tilde{\phi}_k$ ($= \sqrt{g_k/n_1}$ for $0 < k < n_1$ and $= \sqrt{g_k 2/n_1}$ for $k = 0,\, n_1$) is computed and stored in `r[]`. If a g_k is negative a warning is printed and the $\tilde{\phi}_k$ is set to 0, which implies that the method is not exact for the given autocorrelation.

For the generation of the time series the $\tilde{z}_k$ are generated and multiplied with $\tilde{\phi}_k$. As $\tilde{z}_0$ and $\tilde{z}_{n_1}$ are real it is possible (and necessary for the call of the real FFT) to store $\tilde{\phi}_{n_1}\tilde{z}_{n_1}$ in `data[1]`. Then the inverse real FFT is called.

```
void fftrand(double data[],long n,double r[],int *setup)
{  int i,n1;
   n1=n-1;
   if(*setup)
/* start of the setup*/
```

```
   {  *setup=0;
      for (i=0;i<n;i++) data[i]=r[i];
      for (i=n;i<2*n1;i++) data[i]=data[2*n1-i];
      drealft(data-1,2*n1,1);
      for(i=0;i<n1;i++) {
         if (data[2*i]<0.) {
            printf("gk negativ i=%d: %e !!!!\n",i,data[2*i]);
            r[i]=0.;
         } else r[i]=sqrt(data[2*i]/n1);
      }
      r[0]=r[0]*sqrt(2.);
      if (data[1]<0.) {
         printf("gk negativ i=%d:%e !!!!\n",n1,data[1]);
         r[n1]=0.;
      } else r[n1]=sqrt(data[1]*2./n1);
   }
/* start of the generation of the time series*/
   data[0]=r[0]*nacr();
   data[1]=r[n1]*nacr();
   for(i=2;i<2*n1;i++) data[i]=r[i/2]*nacr();
   drealft(data-1,2*n1,-1);
}
```

In the following we give the idea behind Algorithm 3 and compare it to the Cholesky decomposition algorithm.

To make use of the Fourier transform we define a vector u so that the time series x can be generated as the convolution

$$x = u * \varepsilon \qquad \text{ie.} \qquad x_j = \sum_{k=0}^{N-1} u_{j+k}\varepsilon_k.$$

It is easy to see that x has the required autocorrelation if

$$r = u * u \qquad \text{ie.} \qquad r_j = \sum_{k=0}^{N-1} u_{j+k}u_k = \sum_{k=0}^{N-1} u_{j-k}u_k.$$

To construct u out of r we use the convolution theorem and get $\mathcal{F}(r) = \mathcal{F}(u) \cdot \mathcal{F}(u)$ or equivalently

$$u = \mathcal{F}^{-1}(\sqrt{\mathcal{F}(r)}).$$

In order to write the above convolutions in matrix notation we need a $n \times N$ matrix U. As u is the FT of a real vector only the first n components of u are necessary to construct the matrix.

$$U = \begin{pmatrix} u_0, & \dots, & u_{n-3}, & u_{n-2}, & u_{n-1}, & \dots, & u_1 \\ u_1, & \dots, & u_{n-2}, & u_{n-1}, & u_{n-2}, & \dots, & u_0 \\ u_2, & \dots, & u_{n-1}, & u_{n-2}, & u_{n-3}, & \dots, & u_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_{n-1}, & \dots, & u_2, & u_1, & u_0, & \dots, & u_{n-2} \end{pmatrix}$$

Then we have:

$$x = U\varepsilon \quad \text{and} \quad \Sigma = UU'$$

which reveals the relation to Algorithm 1. The only difference is that a new decomposition of the correlation matrix Σ into the non quadratic matrices U and U' is used. The advantage of that decomposition is the fact that it is a convolution and can be computed very fast. However a vector ε of length $N = 2(n-1)$ instead of n is required.

Algorithm 3 uses the convolution theorem to compute x as

$$x = u * \varepsilon = \mathcal{F}^{-1}(\sqrt{\mathcal{F}(r)} \cdot \mathcal{F}(\varepsilon)).$$

To gain speed $\phi = \sqrt{g} = \sqrt{\mathcal{F}(r)}$ is computed and stored in the setup. In addition the vector $z = \mathcal{F}(\varepsilon)$ is generated directly without computing a Fourier transform using the well known properties of Fourier transform of a vector of iid. Gaussian variates (cf. step 2 of Algorithm 3).

One problem of Algorithm 3 is that it is only applicable as long as all g_k are nonnegative. Otherwise it is impossible to find a vector ϕ with the desired properties. The question for which autocorrelations Algorithm 3 works has been entirely neglected in the literature. From theory it is known that as $n \to \infty$ the finite Fourier transform of the autocovariance function converges towards the spectral density of the process which must be nonnegative. This implies that the method works for n large enough if the spectral density of the process is bounded away from zero (which is the case for pure fractionally integrated processes with a non negative fractional integration parameter cf. [1]). Even if the spectral density has zero points our experience shows that it is not easy to find examples where the method fails for large n (see Section 5).
The main advantage of Algorithm 3 is that it requires $O(n \log(n))$ flops for generation and setup compared with $O(n^2)$ for its competitors which makes it by far the fastest alternative if n is moderate or large.

5. Comparison of Algorithms

In this section we compare Algorithm 1 (using the Cholesky factorization of [6] p. 142 Algorithm 4.2.1), Algorithm 2, Algorithm 2a (a variant of Algorithm 2 that stores the vectors y in the setup step) and Algorithm 3. Normal random variates were generated with algorithm NACR described in [11] using a linear congruential generator with module 2^{32} for generating the uniform random numbers. All of the algorithms were coded in C and run on a DEC-station 5000/240.
First it is necessary to investigate for which autocorrelations r the different methods are applicable. In exact arithmetic Algorithms 1, 2 and 2a work for any vector r that leads

to a positive definite Σ. Due to rounding errors there can occur numerical instabilities for certain r's. Considerations summarized in [6] imply that the stability of the Cholesky decomposition and of the Durbin algorithm are approximately the same and our experiences summarized below are in accordance with this result. For Algorithm 3 the situation is different as there are cases where the algorithm does not work in precise arithmetic due to negative g_k. To get an impression how these problems can influence the applicability of the algorithms in practice we tested all four of them for the following classes of stationary processes for a variety of parameter values including the cases very close to the border of the stationary region and for n between 9 and 2049.

- MA(1) process: no problems occurred for the root of the polynomial larger than $1 + 10^{-10}$.
- AR(1) process: no problems occurred for the root of the polynomial larger than $1 + 10^{-10}$.
- pure fractionally integrated process: no problems occurred for the fractional integration parameter d between -0.5 and $0.5 - 10^{-10}$.
- AR(2) process: no problems occurred for Algorithms 1, 2 and 2a for a lot of polynomials $(1 - \alpha_1 z - \alpha_2 z^2)$ with roots $|z| > 1 + 10^{-10}$; for Algorithm 3 we found parameter values (for example $\alpha_1 = 0.5$ and $\alpha_2 = -0.995$) where negative g_k's occurred. However, for larger n (eg. $n = 4097$) Algorithm 3 worked.
- processes with autocorrelation functions associated with polynomial spectral densities with zero points: no problems occurred for Algorithms 1, 2 and 2a. For Algorithm 3 the zero point can cause problems especially if it coincides with a Fourier frequency and the spectral density is close to zero in a neighborhood.
- $r_k = e^{-\alpha k^2}$: Processes with this kind of correlation function are discussed in [4] as they cause problems when α is small. In our implementation (using 64-bit floating

point numbers) all algorithms worked for $\alpha \geq 0.1$ and all of them did not work for $\alpha \leq 0.03$. For α in between some worked and some not. Algorithm 2 and 2a had of course the same behaviour and were more stable than Algorithm 1 which was more stable than Algorithm 3. The latter fails if the spectral density of the process is very close to zero on a large interval. This is the case for small values of α.

In [4] a method for regularizing Algorithm 2 is suggested: Find an ϵ as small as possible so that Algorithm 2 using $\tilde{\Sigma} = (1-\epsilon)\Sigma + \epsilon I$ instead of Σ becomes stable. For our floating point arithmetic a value of $\epsilon = 10^{-10}$ is enough so that Algorithm 2 becomes stable for the last example with $\alpha \geq 10^{-10}$ in the latter example The change of Σ is so small that it seems impossible that it could influence the results of a simulation.

For Algorithm 3 the regularization described above has no success. Of course a different method could be suggested: If the negative g_k are close to zero treat them as zero. This is simple and can remove some of the problems found in our benchmarks (especially for n large) but we do not recommend this method as it seems – in contrast to the regularization method described above – difficult to get an impression about the difference between the autocorrelation function of the regularized process and the true process.

In order to compare the computer resources necessary for the different algorithm we present Table 1 that summarizes the theoretical requirements already stated in the above sections.

Table 1: Flops and storage requirements

Algorithm	flops setup+1 time-series	flops 1 time series	memory required	normal variates
1 Cholesky	$n^3/3 + O(n^2)$	$n^2 + O(n)$	$n^2/2 + O(n)$	n
2 Durbin	$4n^2 + O(n)$	$3n^2 + O(n)$	$4n + O(1)$	n
2a Durbin	$4n^2 + O(n)$	$n^2 + O(n)$	$n^2/2 + O(n)$	n
3 FFT	$8n \log_2(2n) + O(n)$	$4n \log_2(2n) + O(n)$	$3n + O(1)$	$2(n-1)$

Clearly the results of Table 1 have direct consequences for the speed of the compared algorithms. For n large it is clear that Algorithm 3 is by far the fastest but it makes sense to compare the execution times of the algorithms for small to moderate n. Our timings (in seconds) on the DEC-station for the setup and the generation of m time series are given in Table 2. Of course they depend on the computer the compiler and the random variate generator used. On a PC486/50 under Linux the algorithms were about two times slower but the relative speed of Algorithm 2a was about ten percent better than on the DEC-station.

Table 2: Execution times in seconds for m replications

n	10	50	100	200	500	1000	10^4	10^5
m	10000	2000	1000	500	200	100	10	1
1 Cholesky	0.55	1.3	2.5	12.9	52.27	274	*	*
2 Durbin	0.83	2.5	5.7	14.2	30.1	56.3	878.9	10^5
2a Durbin	0.55	1.2	2.0	5.8	14.6	29.7	*	*
3 FFT	2.2	1.8	1.9	1.9	1.6	1.7	7.7	13.9

* ...more than 500 megabyte memory necessary

The results of Table 2 mirrors the strong influence of the theoretical measures of performance on the execution times. In addition we see that for $n = 10$ Algorithms 1 and 2a have the same speed. For larger n the very slow setup makes Algorithm 1 slower than Algorithm 2a. Algorithm 3 is fastest for n larger than about 100.

6. Conclusions and some comments

For the generation of commonly used low order ARMA models the necessary presample values obeying to the correct unconditional distribution may be computed via any of the discussed methods. This saves the necessary iterations to reduce the effects of the initial conditions. The following steps are then simple recursions which are linear in flops.

For the generation of general stationary Gaussian time series the Cholesky decomposition method should not be used unless n is really small. For large n Algorithm 3 is by far the fastest alternative and should be applied but there can occur problems if the spectral density is close to zero in a region. For these cases Algorithm 2 and its regularization can be used as a slower but stable alternative. Algorithm 2a should be applied for n small to moderate if speed is of importance. Algorithm 2, on the other hand, can also be used to compute the product of the Cholesky factor of a symmetric Toeplitz matrix with an arbitrary vector without computing the Cholesky factor explicitly.

References

[1] J.R.M. Hosking, "Modeling persistence in hydrological time series using fractional differencing," *Water Resources Research*, Vol. 20, pp. 1898–1908, 1984.

[2] P.J. Brockwell and R.A. Davis, *Time Series: Theory and Methods.* 2nd edition, New York: Springer, 1991.

[3] Y.-W. Cheung, "Long memory in foreign-exchange rates," *Journal of Business and Economic Statistics*, Vol. 11, No. 1, pp. 93–101, 1993.

[4] A.S. Marchenko and V.A. Ogorodnikov, "The modelling of very long stationary gaussian sequences with an arbitrary correlation function," *USSR Comput. Maths. Math. Phys.*, Vol. 24, No. 5, pp.141–144, 1984.

[5] R.B. Davies and D.S. Harte, "Tests for Hurst effect," *Biometrika*, Vol. 74, pp. 95–101, 1987.

[6] G.H Golub and C.F. Van Loan, *Matrix Computations.* Baltimore and London: The Johns Hopkins University Press, 1989.

[7] B. D. Ripley, *Stochastic Simulation.* New York: John Wiley & Sons, 1987.

[8] J. Durbin, "The fitting of time series models," *Rev. Inst. Int. Stat.*, Vol. 28, pp. 233–243, 1960.

[9] M.A. Hauser, W. Hörmann, R. M. Kunst, and J. Lenneis, "A note on generation, estimation and prediction of stationary processes," In R. Dutter and W. Grossmann, editors, *Compstat, Proceedings in Computational Statistics*, pp. 323–328, Heidelberg: Physica-Verlag, 1994.

[10] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical Recipes in C, 2nd edition.* Cambridge University Press, 1992.

[11] W. Hörmann and G. Derflinger, "The ACR method for generating normal random variables," *OR Spektrum*, Vol. 12, pp. 181–185, 1990.

About the Authors:

MICHAEL A. HAUSER received his MS (1978), and Ph.D. (1987), all in technical mathematics from Technical University Vienna. Currently he is assistant professor at the University of Economics and Business Administration Vienna, Department for Applied Statistics. His major research area is time series analysis, especially models for macro economic and financial series.

WOLFGANG HÖRMANN received his MS (1987) and his PhD (1989) both in mathematics from the University of Vienna. Since 1989 he has been assistant professor at the department of statistics at the University of Economics and Business Administration Vienna. His main research interests are in the field of stochastic simulation, especially the generation of non-uniform random variates. In the last two years he has specialized in the development of universal algorithms for large classes of univariate and bivariate distributions.