Journal of Statistical Software

December 2007, Volume 23, Issue 5. http://www.jstatsoft.org/

Algorithms for Linear Time Series Analysis: With R Package

A. Ian McLeod
University of
Western Ontario

Hao Yu
University of
Western Ontario

Zinovi L. Krougly
University of
Western Ontario

Abstract

Our **ltsa** package implements the Durbin-Levinson and Trench algorithms and provides a general approach to the problems of fitting, forecasting and simulating linear time series models as well as fitting regression models with linear time series errors. For computational efficiency both algorithms are implemented in C and interfaced to R. Examples are given which illustrate the efficiency and accuracy of the algorithms. We provide a second package **FGN** which illustrates the use of the **ltsa** package with fractional Gaussian noise (FGN). It is hoped that the **ltsa** will provide a base for further time series software.

Keywords: exact maximum likelihood estimation, forecasting, fractional Gaussian noise, inverse symmetric Toeplitz matrix, long memory and the Nile river minima, time series regression, time series simulation.

1. Introduction

Let $z_t,\ t = 1, \ldots, n$, denote n successive observations from an ergodic covariance stationary Gaussian time series with autocovariance function (acvf) $\gamma_k = \text{Cov}\,(z_t, z_{t-k}), k = 0, 1, \ldots, n-1$ and mean μ. The general linear process (GLP) may be specified by its autocovariance sequence, γ_k, or equivalently in terms of its autocorrelation sequence (acf), $\rho_k = \gamma_k/\gamma_0$ or coefficients ψ_k in the infinite moving-average (MA),

$$z_t = \mu + a_t + \psi_1 a_{t-1} + \psi_2 a_{t-2} + \ldots, \tag{1}$$

where $a_t \sim \text{NID}\,(0, \sigma_a^2)$ and $\psi_1^2 + \psi_2^2 + \ldots < \infty$. Notice that both acf and infinite MA model specifications also require the innovation variance σ_a^2. The condition $\psi_1^2 + \psi_2^2 + \ldots < \infty$ ensures that the acvf exists and that the GLP is stationary. For sufficiently large Q we may

approximate using a MA of order Q,

$$z_t = \mu + a_t + \psi_1 a_{t-1} + \psi_2 a_{t-2} + \ldots \psi_Q a_{t-Q}, \tag{2}$$

Most parametric time series models may be specified so that either the autocovariances, γ_k, or the MA coefficients, ψ_k, are functions of a small number of parameters, β. Both theory and experience suggests that exact maximum likelihood are preferable to other methods. Please see the Appendix for further discussion about the GLP.

The covariance matrix of $z_t,\ t = 1, \ldots, n$, denoted by Γ_n, is given by $\Gamma_n = (\gamma_{i-j})$, where the (i, j)-entry in the $n \times n$ matrix is indicated. The minimum-mean-square linear predictor of z_t given $z_s, s = 1, \ldots, t-1$, where $t \leq n$, may be written

$$\hat{z}_t = \phi_{t-1,1} z_1 + \ldots + \phi_{t-1,t-1} z_{t-1}, \tag{3}$$

where $\phi^{(t)} = (\phi_{t-1,1}, \ldots, \phi_{t-1,t-1})$ is determined by the linear equations

$$\Gamma_t \phi^{(t)} = (\gamma(1), \ldots, \gamma(t))', \tag{4}$$

and the variance of the predictor is given by

$$\sigma_k^2 = \gamma(0) - \phi_{t-1,1}\gamma(1) - \ldots - \phi_{t-1,t-1}\gamma(t-1). \tag{5}$$

The covariance determinant is,

$$|\Gamma_t| = \prod_{k=0}^{k=t-1} \sigma_k^2. \tag{6}$$

The Durbin-Levinson algorithm (Golub and Loan 1996, Algorithm 5.7-1) provides an efficient algorithm for solving Equations (4) and (5) in $O(n^2)$ flops. The Trench algorithm (Golub and Loan 1996; Trench 1964) may be derived by determining the parameters in an AR $(n-1)$ with autocovariances $\gamma_0, \ldots, \gamma_{n-1}$ using the Durbin-Levinson algorithm. After the AR coefficients have been found, Γ_n^{-1} is obtained using the method given by Siddiqui (1958). The Trench algorithm evaluates the matrix inverse in $O(n^2)$ flops. Our R function `TrenchInverse` uses the C interface to provide an efficient implementation of the Trench algorithm. On a typical current PC it takes less than a fraction of a second to evaluate this inverse for matrices with $n = 1000$. In the special case of ARMA time series, Zinde-Walsh (1988) obtained an explicit expression for computing the elements in Γ_n^{-1}. This method is suitable for symbolic computation (McLeod 2005).

The Durbin-Levinson and Trench algorithms provide a convenient method for computing the exact likelihood function of a general linear Gaussian time series model (Li 1981; Sowell 1992; Brockwell and Davis 1991). In a suitable quantitative programming environment such as R the likelihood function may be explored graphically, optimized to obtain exact MLE or integrated for Bayesian estimates (McLeod and Quenneville 2001). For the well-known ARMA family of time series models there are many algorithms for the computation of the exact likelihood function which require only $O(n)$ flops per function evaluation Box and Luceño (1997, Section 12B) as opposed to the $O(n^2)$ flops required in the Durbin-Levinson or Trench algorithm. For long ARMA time series, a superfast likelihood algorithm requiring only $O(1)$ flops per log-likelihood evaluation is available (McLeod and Zhang 2007). The advantage of the Durbin-Levinson or Trench algorithm is that they are more general and with current

computing technology, MLE using this algorithm is sufficiently fast provided that n is not too large. When the ARMA model is extended to include long-memory alternatives such as the ARFIMA model Brockwell and Davis (1991, Section 13.2) and other ARMA long-memory extensions (Baillie 1996), the Durbin-Levinson likelihood algorithm is as computationally efficient as other commonly used exact likelihood methods such as the innovation algorithm or Kalman filter. A brief discussion and comparison with the superfast algorithm (Chen, Hurvich, and Lu 2006) is given in Section 2.3.

For linear parametric time series models, it is assumed that the acvf or MA coefficients, $\psi_k, k = 0, 1, \ldots,$ are uniquely determined by p parameters, $\beta = (\beta_1, \ldots, \beta_p)$. An example we will discuss in Section 3 is the fractional Gaussian noise time series model which is characterized by its autocorrelation function Hipel and McLeod (1994, page 340),

$$\rho_k = (|k+1|^{2H} - 2|k|^{2H} + |k-1|^{2H})/2, \qquad 0 < H < 1. \tag{7}$$

In addition to fitting and forecasting linear time series models, there are many other applications for our algorithms in time series analysis including regression with autocorrelated error, an example of which is discussed in Section 3.4. Another example where the inverse matrix is needed is for power computations in intervention analysis (McLeod and Vingilis 2005).

2. Main package

2.1. Package overview

The R functions in the **ltsa** package are shown in Table 1. The `TrenchInverse` function is especially useful for efficient exact MLE for regression with autocorrelated error and for the mean, μ, as well for computing and updating forecasts. The log-likelihood may be computed using either the Durbin-Levinson or Trench algorithm. Residuals are useful for checking model adequacy and these may also be computed using the Durbin-Levinson algorithm. A linear time series may be simulated using the Durbin-Levinson recursion, the Davies-Harte algorithm (Davies and Harte 1987) or another method using the fast Fourier transform that is given in Section 2.6. In the next section we discuss in more detail `TrenchInverse` and in the following sections the remaining functions are discussed. The **FGN** package discussed in Section 3 describes how the **ltsa** functions may be used to develop a package for fractional Gaussian noise (FGN) time series modeling.

2.2. `TrenchInverse`

The `TrenchInverse` function in R is interfaced to a C function for maximum speed. R memory management C functions are used so there is complete compatibility with all R platforms. Our package has been tested in the Windows, Mac OS X and Debian Linux environments.

The function `TrenchInverse` in this package inverts a positive-definite Toeplitz matrix utilizing the interface provided in R to call a C function. If the matrix is not a positive definite, a suitable error message is returned.

The `TrenchInverse` function takes a single matrix argument Γ_n. The built-in R function, `toeplitz`, may be used to create this matrix Γ_n from the vector input $(\gamma_0, \ldots, \gamma_{n-1})$. For maximum computational efficiency one could work with just this vector, and this is how in fact

Function	Purpose
DHSimulate	Simulate using Davies-Harte method
DLAcfToAR	AR parameters, variances, pacf
DLLoglikelihood	Exact concentrated log-likelihood
DLResiduals	Standardized prediction residuals
DLSimulate	Simulate using DL recursion
SimGLP	Simulate general linear process
tacvfARMA	Acvf of ARMA
TrenchInverse	Toeplitz matrix inverse
ToeplitzInverseUpdate	Updates the inverse
TrenchLoglikelihood	Exact concentrated log-likelihood
TrenchMean	Exact MLE for mean
TrenchForecast	Exact forecast and variance

Table 1: The principal functions in **ltsa**.

the underlying C algorithm is implemented. As a brief illustrative example of this package, in the code below, we subtract the product of a Toeplitz matrix and its inverse from the identity matrix and compute the largest absolute error in the result:

```
R> phi <- 0.8
R> n <- 1000
R> r <- (1 / (1 - phi^2)) * phi^(0:(n-1))
R> G <- toeplitz(r)
R> Gi <- TrenchInverse(G)
R> id <- matrix(0, nrow=n, ncol=n)
R> diag(id) < -1
R> max(abs(id - G%*%Gi))

[1] 6.661338e-16
```

We investigated the timings for the function `TrenchInverse` and compared them with the general purpose matrix inverse function `solve` in R. The timings reported in this section were done on a 3.6 GHz Pentium 4 PC running Windows XP. For these timings we first generated 25 uniform random values on the interval $(-1, 1)$, denoted by $\phi_k, k = 1, \ldots, 25$. These values were used to generate 25 Toeplitz matrices of the form, $\Gamma_n = (\phi_k^{|i-j|}/(1-\phi_k^2))_{n\times n}$, $k = 1, \ldots, 25$, for each $n = 400, 800, 1200, 1600, 2000$. The code for generating this table is included in the package documentation for `TrenchInverse`. We conclude from Table 2 that `TrenchInverse` is significantly faster than `solve`.

In forecasting applications that we will discuss below in Section 2.5, the successive inverses of Γ_{n+k}, $k = 1, 2, \ldots$, are needed. Then Γ_{n+k}^{-1} may be more efficiently computed using the result for the inverse of partitioned matrices Graybill (1983, Section 8.3). When $k = 1$, we obtain,

$$\Gamma_{n+1}^{-1} = \begin{pmatrix} \Gamma_n^{-1}(\mathcal{I}_n + \Gamma_n^{-1}/a) & f \\ f' & e \end{pmatrix}, \tag{8}$$

n	TrenchInverse	solve
400	0.08	0.43
800	0.22	4.57
1200	0.44	18.73
1600	0.74	51.15
2000	1.16	114.30

Table 2: CPU time in seconds for `TrenchInverse` and the R function `solve`.

where $a = ehh'$, $e = 1/(\gamma_0 - h'\Gamma_n^{-1}h)$, $h = (\gamma_1, \ldots, \gamma_n)'$, $f = -e\Gamma_n^{-1}h$ and $\mathcal{I}_n$ is the $n \times n$ identity matrix. Then Equation (8) may be applied repeatedly to obtain Γ_{n+k}^{-1}, $k = 1, 2, \ldots$.

In Table 3, we compare the timing and accuracy of the updating approach described in Equation (8), implemented in `ToeplitzInverseUpdate`, with direct inversion in `TrenchInverse`. The function `TrenchInverse` is implemented using R interface to C, whereas `ToeplitzInverseUpdate` is implemented entirely in R. For the comparison we used the hyperbolic decay autocovariance function $\gamma_k = 1/(k+1)$, $k \geq 0$, and computed Γ_{n+k}^{-1}, $k = 1, 2, \ldots, 100$ and $n = 100, 500, 1000, 2000$. As a check, the absolute error for the difference between the two matrices was computed and it was found to be negligible. The CPU times are shown in Table 3. The updating algorithm is about 3 to 4 times as fast and this factor does not seem to change much with n. This might be expected since both algorithms require $O(n^2)$ flops.

2.3. Log-likelihoood

In this section we discuss the functions `DLLoglikelihood` and `TrenchLoglikelihood`.

Assuming the mean is known and that z_t has been mean-corrected, the log-likelihood function for parameters (β, σ_a^2) given data $z = (z_1, \ldots, z_n)'$ may be written, after dropping the constant term,

$$L(\beta, \sigma_a^2) = -\frac{1}{2}\log(\det(\Gamma_n)) - z'\Gamma_n^{-1}z/2. \tag{9}$$

Letting $M_n = \Gamma_n/\sigma_a^2$ and maximizing L over σ_a^2, the concentrated log-likelihood may be

n	ToeplitzInverseUpdate	TrenchInverse
100	0.09	0.61
500	2.14	7.89
1000	7.85	29.47
2000	33.07	110.00

Table 3: CPU time in seconds for direct computation using `TrenchInverse` and `ToeplitzInverseUpdate`. First the inverse of a matrix of order n is computed and then the inverses for matrices of orders $n+1, \ldots, n+100$ are computed using `TrenchInverse` and `ToeplitzInverseUpdate`.

written,

$$L_c(\beta) = -\frac{n}{2}\log(S(\beta)/n) - \frac{1}{2}\log(g_n), \tag{10}$$

where $S(\beta) = z'M_n^{-1}z$ and $g_n = \det(M_n)$. Note that L_c is unchanged if we simply replace M_n in $S(\beta)$ and g_n by the autocorrelation matrix $R_n = (\rho_{|i-j|})_{n\times n}$. Using the Durbin-Levinson algorithm,

$$S(\beta) = \sum_{t=1}^{n}(z_t - \hat{z}_t)^2/\sigma_k^2, \tag{11}$$

where $\hat{z}_t$ and σ_k^2 are given in Equation (3) and (5). The C functions we developed can evaluate $S(\beta)$ as well as $\log(g_n)$ using the Durbin-Levinson method as well as the more direct Trench algorithm. These C functions are interfaced to `DLLoglikelihood` and `TrenchLoglikelihood`. Either can then be used with the optimization functions provided with R to obtain the MLE. In practice, the Durbin-Levinson is somewhat faster as can be seen in Table 4.

Chen *et al.* (2006) have implemented a superfast method of solving linear Toeplitz systems of the form $\Gamma_n x = b$, where Γ_n is an order n symmetric positive-definite Toeplitz matrix, b is a known vector of length n and x is the vector of unknowns. With their method, x can be found in $O(n\log^{5/2} n)$ flops. This provides an alternative and, in principle, computationally more efficient method of evaluating the loglikelhood. However, unless n is very large, the gain may not be significant. The time will also depend on the computer and the specific implementation. Timings they gave for their superfast algorithm, `ML-PCG (S-PLUS)`, are reported in Table 4 and Table 5 along with timings for their S-PLUS version of the Durbin-Levinson method, `ML-Levinson (S-PLUS)`. Apart from coding, their `ML-Levinson (S-PLUS)` is equivalent to our `DLLoglikelihood, useC=TRUE`. Our timings for the Durbin-Levison algorithm are much faster and this is no doubt due to language/machine differences. In Table 5 we used a Windows PC with a 3.6 Ghz Pentium processor and in Table 4 we used a newer PC running Debian Linnux since our Windows PC could not handle such large matrices. (Chen *et al.* 2006, Table 4) used a Sun Workstation running the Solaris OS. Tables 4 and 5 suggest that, for many purposes, the current implementation of our algorithms has a satisfactory performance with respect to computer time.

The superfast algorithm of Chen *et al.* (2006) is an iterative method which is more complicated to program and has several other practical limitations. If only the inverse matrix is required then the Trench algorithm is always computationally more efficient, since the superfast algorithm only solves a set of linear equations and does not directly compute the matrix

Algorithm	Computer/OS	n	
		10000	15000
`DLLoglikelihood, useC=FALSE`	PC/Linnux	6.11	14.34
`DLLoglikelihood, useC=TRUE`	PC/Linnux	0.52	1.17
`TrenchLoglikelihood`	PC/Linnux	3.73	9.35
`ML-Levinson (S-PLUS)`	Sun/Solaris	168.4	379.8
`ML-PCG (S-PLUS)`	Sun/Solaris	6.3	9.4

Table 4: CPU time in seconds for $d = 0.45$, $n = 10,000$ and $n = 15,000$.

Algorithm	Computer/OS	d			
		-0.45	-0.25	0.25	0.45
`DLLoglikelihood, useC=FALSE`	PC/WinXp	3.42	3.39	3.40	3.39
`DLLoglikelihood, useC=TRUE`	PC/WinXP	0.17	0.17	0.17	0.17
`TrenchLoglikelihood`	PC/WinXP	2.88	2.61	2.70	2.60
`DLLoglikelihood, useC=FALSE`	MacBook/OS X	2.45	2.51	2.48	2.48
`DLLoglikelihood, useC=TRUE`	MacBook/OS X	0.14	0.14	0.14	0.14
`TrenchLoglikelihood`	MacBook/OS X	1.40	1.14	1.13	1.13
`DLLoglikelihood, useC=FALSE`	PC/Linnux	1.64	1.63	1.64	1.64
`DLLoglikelihood, useC=TRUE`	PC/Linnux	0.13	0.13	0.13	0.13
`TrenchLoglikelihood`	PC/Linnux	0.91	0.90	0.93	0.92
`ML-Levinson (S-PLUS)`	Sun/Solaris	42.8	42.6	42.3	42.6
`ML-PCG (S-PLUS)`	Sun/Solaris	5.50	4.40	3.80	4.70

Table 5: Comparison of CPU time in seconds for $n = 5000$.

inverse. A further advantage of the Trench algorithm is that as a byproduct the exact value of the determinant is also obtained.

2.4. Estimation of the mean

Given the other parameters in the model, so that $\gamma_k, k = 0, \ldots, n-1$ is specified, the best linear unbiased estiamte estimate (BLUE) for μ is given by (Beran 1994, Section 8.2),

$$\hat{\mu} = \frac{1_n' \Gamma_n^{-1} z}{1_n' \Gamma_n^{-1} 1_n}, \tag{12}$$

where $z' = (z_1, \ldots, z_n)$ and 1_n is an n dimensional column vector whose entries are all equal to one. Notice that the autocovariances in Equation (12) may be replaced by autocorrelations. The function `TrenchMean` implements the computation in Equation (12).

An iterative algorithm may be used for the simultaneous joint MLE of μ and the other parameters β.

Step 0 Set the maximum number of iterations, $M \leftarrow 5$. Set the iteration counter, $i \leftarrow 0$. Set $\hat{\mu}^{(0)} \leftarrow \bar{z}$, where $\bar{z}$ is the sample mean. Set initial parameters to zero, $\beta^{(0)} \leftarrow 0$ or some other suitable initial estimate. Set $\ell_0 = L_c(\hat{\beta}^{(0)}, \hat{\mu}^{(0)})$.

Step 1 Obtain $\hat{\beta}^{(i+1)}$ by numerically maximizing $L_c(\beta, \hat{\mu}^{(i)})$ over ϕ. Set $\ell_{i+1} = L_c(\hat{\beta}^{(i+1)}, \hat{\mu}^{(i)})$.

Step 2 Evaluate $\hat{\mu}^{(i+1)}$ using $\hat{\beta}^{(i+1)}$ in Equation (12).

Step 3 Terminate when ℓ_{i+1} has converged or $i > M$. Otherwise set $i \leftarrow i+1$ and return to Step 1 to perform the next iteration.

Convergence usually occurs in two or three iterations.

Another topic of interest is the question of the efficiency of the sample mean. The variance of the sample mean, $\bar{z} = (z_1 + \ldots + z_n)/n$, may be written,

$$\begin{aligned} \text{Var}\,(\bar{z}) &= \text{Var}\,(1_n' z) \\ &= 1_n' \Gamma_n 1_n / n^2 \\ &= \frac{\gamma_0}{n} + \frac{2}{n}\sum_{k=1}^{n-1}(1 - \frac{k}{n})\gamma_k. \end{aligned} \tag{13}$$

It may be shown that the exact finite sample efficiency of the sample mean is (Beran 1994, Section 8.2)

$$\mathcal{E} = n^2/((1_n'\Gamma 1_n)(1_n'\Gamma^{-1}1_n)). \tag{14}$$

Although the sample mean often provides an efficient estimate, situations exist it is not very efficient. We will discuss an example of this in Section 3.2.

2.5. Forecasting

The Trench algorithm is useful for the computation of exact finite-sample forecasts and their variances. Let $z_n(k)$ denote the minimum-mean-square-error linear predictor of z_{n+k}, given the data $z' = (z_1, \ldots, z_n)$, the mean μ and autocovariances $\gamma_\ell, \ell = 0, \ldots, n-1$. Then Hamilton (1994, Section 4.3) or Hipel and McLeod (1994, Section 10.4.5),

$$z_n(k) = \mu + g_k'\Gamma_n^{-1}(z - \mu), \tag{15}$$

where $g_k' = (\gamma_{n+k-1}, \ldots, \gamma_k)$ and the variance for the forecast,

$$V_k = \gamma_0 - g_k'\Gamma_n^{-1}g_k, \tag{16}$$

For large n, $V_1 \doteq \sigma_a^2$. In Equation (15), the autocovariances may be replaced by autocorrelations, but in Equation (16) autocovariances must be used. Equations (15) and (16) may be vectorized to compute the k-step predictor for $k = 1, 2, \ldots, L$, and this is implemented in `TrenchForecast`. The computation of the forecasts and their variances using Equations (15) and (16) requires the autocovariances $\gamma_0, \ldots, \gamma_{n+k-1}$. The prediction variance may also be computed in another way as described in Section 2.9, and this is implemented in the function `PredictionVariance`.

In practice we may also be interested in updating the forecasts given a new data values $z_{n+k},\ k = 1, 2, \ldots, L$. This entails computing Γ_{n+k}^{-1} for $k = 1, 2, \ldots, L$ given Γ_n^{-1} and may be computed using the updating method described in Section 2.2. The function `TrenchForecast` implements the updating method as well as an option to compute the forecasts using direct matrix inversion. This is done in order to provide an optional check on the computations.

An example application of `TrenchForecast` to actual time series data is given in Section 3.3.

In the example below, we simulate a series of length $n = 202$ from an AR(1), then using the known parameter, we forecast starting at origin $n = 200$ for lead times $k = 1, 2, 3$, and updating the forecast origin to $n = 201, 202$. The result is a list with two 3×3 matrices. It is easy to check the forecasts and standard deviations in this case,

```
R> n <- 200
R> m <- 2
```

```
R> maxLead <- 3
R> N <- n  +m
R> phi <- 0.9
R> r <- phi^seq(0, N + maxLead - w1)
R> set.seed(19890824)
R> z <- DLSimulate(N, r)
R> out <- TrenchForecast(z, r, 0.0, n, maxLead)
R> out$Forecasts

             1           2           3
200  0.4016172  0.36145544  0.32530990
201  0.3366849  0.30301644  0.27271480
202 -0.0794845 -0.07153605 -0.06438244

R> outer(z[n:N],phi^(1:maxLead))

           [,1]        [,2]        [,3]
[1,]  0.4016172  0.36145544  0.32530990
[2,]  0.3366849  0.30301644  0.27271480
[3,] -0.0794845 -0.07153605 -0.06438244
```

It should be noted that the `predict.Arima` function in R can not be used to obtain the above results, since it only predicts for a single forecast origin time which is fixed to be at the end of the series. This hampers the use of R in forecasting experiments in time series where the data is divided into training and test samples. Examples of this technique are given in Sections 3.3 and 3.4.

Finally, it should be noted that often the Gaussian assumption may not be valid for the forecast errors. In the non-Gaussian case the prediction variances should not be used to set probability limits for the forecasts. Instead probability limits for the forecasts may be obtained by simulation or bootstrapping (McCullough 1994; Aronsson, Holst, Lindoff, and Svensson 2006).

2.6. Simulation

Simulation of time series is widely used in bootstrapping for statistical inference as well in the exploration of statistical properties of time series methods. Time series simulation is also important in engineering design and operational research (Dembo 1991; Hipel and McLeod 1994; Maceira and Damázio 2006). The Durbin-Levinson algorithm provides a convenient method for simulating a Gaussian time series, $z_1, \ldots, z_n$, with autocovariances, $\gamma_0, \ldots, \gamma_{n-1}$ (Hosking 1981; Hipel and McLeod 1994). Using the Durbin-Levinson algorithm to solve Equations (4) and (5), the series $z_1, \ldots, z_n$ may be generated for $t = 2, \ldots, n$ from

$$z_t = \phi_{t-1,1} z_1 + \ldots + \phi_{t-1,t-1} z_{t-1} + e_t \tag{17}$$

where $e_t \sim$ NID $(0, \sigma_{t-1}^2)$ and $z_1 \sim$ NID $(0, \sigma_0^2)$.

The function `DLSimulate` implements this simulation method using an interface to C. As a check the algorithm was also implemented in R and may be invoked using the optional argument `useC = FALSE`.

Davies and Harte (1987) gave an algorithm which only requires $O(n\log(n))$ flops as compared with Durbin-Levinson's $O(n^2)$ flops. This algorithm is implemented in R, in our function `DHSimulate`. However, the Davies-Harte algorithm requires a complicated non-negativity condition, and this condition may not always hold. For example, in Table 6 we generate a time series of length $n = 5000$ with fractional difference $d = 0.45$, and we found the Davies-Harte non-negativity condition failed, and so `DLSimulate` was needed.

Table 6 compares the average time needed for 100 simulations for various series lengths, n. >From this table we see that `DHSimulate` is overall faster even though it is entirely implemented in R.

Another method is useful for simulating time series with innovations from a specified non-Gaussian distribution that also uses the Fast Fourier Transform (FFT). In this case we may approximate the linear time series model as a high-order MA

$$z_t = \mu + \sum_{i=1}^{Q} \psi_i a_{t-i}. \tag{18}$$

The order Q may be quite large in some cases, and it may be chosen so that mean-square error difference,

$$\mathcal{E} = |\gamma_0 - \sigma_a^2 \sum_{i=1}^{Q} \psi_i^2| = \sigma_a^2 \sum_{i=Q+1}^{\infty} \psi_i^2, \tag{19}$$

is made sufficiently small. It may be shown that by making $\mathcal{E}$ small we can make the Kullback-Leibler discrepancy between the exact model and the MA(Q) approximation negligible McLeod and Zhang (2007, Equation 13) An example R script for determining the approximation is given in the online documentation for `DLAcfToAR`. The sum involved in Equation (18) is efficiently evaluated using the R function `convolve` which uses the FFT method. Hence the simulation requires $O(n\log(n))$ flops when n is a power of 2 and assuming $n > Q$. The built-in R function `arima.sim` may also be used, and it uses direct evaluation and drops the initial transients values. The number of initial transient values to drop is determined by the optional argument `n.start`. Only $O(n)$ flops are required by `arima.sim`. In Table 6, we took $Q = 1000$ and `n.start=1000` for `SimGLP` and `arima.sim` respectively. These two approximate methods were compared with the exact simulation methods, `DLSimulate` and `DHSimulate`, for the case of an hyperbolic decay time series with $\gamma_k = 1/\sqrt{k+1},\ k \geq 0$, for time series of lengths 100, 200, 500, 1,000, 5,000 and 10,000. For each n, the total time for 100 simulations was found.

Algorithm	n					
	100	200	500	1000	5000	10000
`arima.sim`	0.31	0.56	1.08	2.00	9.43	18.72
`SimGLP`	0.28	0.30	0.55	0.65	0.64	15.09
`DLSimulate, useC=TRUE`	0.04	0.05	0.18	0.68	15.43	61.30
`DLSimulate, useC=FALSE`	1.59	2.93	8.25	19.46	220.36	785.11
`DHSimulate`	0.03	0.05	0.10	0.22	1.49	3.67

Table 6: CPU time in seconds for 100 simulations of a time series of length n.

In general, if it is planned to do many simulations, `DHSimulate` may be preferred provided the Davies-Harte condition is met. As described in the documentation for `DHSimulate`, one can use the function `DHSimulate` itself to test, if the Davies-Harte condition is satisfied. Both `DHSimulate` and `DLSimulate` can also be used to generate non-Gaussian time series by setting the optional argument `rand.gen` to some other distribution. However, only `SimGLP` and `arima.sim` allow one to specify the exact innovation distribution. See online documentation for examples.

2.7. Regression with autocorrelated error

Consider the regression with autocorrelated error model, $z_t = \alpha_0 + \alpha_1 x_{1,t} + \ldots + \alpha_k x_{k,t} + \xi_t$, where ξ_t's, the errors, are assumed to be generated from a general linear Gaussian mean-zero process with parameters β and σ_a^2. Given observed data $(z_t, x_{1,t}, \ldots, x_{k,t})$, $t = 1, \ldots, n$, for fixed β, the MLE for $\alpha' = (\alpha_0, \alpha_1, \ldots, \alpha_k)$ is given by

$$\hat{\alpha} = (X' R_n^{-1} X)^{-1} X' R_n^{-1} z, \tag{20}$$

where X is the $n \times (k+1)$ matrix with first column 1's and j-th column $x_{j,t}$, $t = 1, \ldots, n; j = 2, \ldots, k+1$.

Joint MLE for α, β and σ_a^2 may be obtained using the following iterative algorithm.

Step 0 Initialization. Set $i \leftarrow 0$. Set R_n to the identity matrix. Set ℓ_0 to a negative number with large absolute value.

Step 1 Use Equation (20) to obtain an estimate of α, $\hat{\alpha}^{(i)}$. Compute the residuals $\hat{\xi} = z - X\hat{\alpha}^{(i)}$.

Step 2 Taking $\hat{\xi}$ as the input time series, maximize L_m using a nonlinear optimization function to obtain $\hat{\beta}^{(i)}$ and $\ell_i = L_m(\hat{\beta}^{(i)})$.

Step 3 If $\ell_i - \ell_{i-1} < 10^{-3}$, perform Step 4. Otherwise set $i \leftarrow i + 1$, and return to Step 1 to perform the next iteration.

Step 4 Compute the MLE for $\hat{\sigma}_a^2$.

The error tolerance is set to 10^{-3}, since in terms of the log-likelihood the change in parameters is of negligible importance. With this error tolerance, convergence usually occurs in three or four iterations. An implementation of this method for multiple linear regression with FGN error is given in our **FGN** package.

2.8. Prediction residuals

The prediction residuals are defined by the difference between the observed value and the one-step ahead minimum-mean-square error forecast. These residuals may be standardized by dividing by their standard deviations. If the model is correct, these residuals should be approximately uncorrelated. It should be noted that asymptotically the prediction residuals are equivalent to the usual residuals, the estimated innovations, $\hat{a}_t$. Hence, the widely used Ljung-Box portmanteau test (Ljung and Box 1978) and other diagnostic checks (Li 2004) may be used to check the adequacy of the fitted model.

2.9. Acf to AR parameters and ARMA Acvf

Given the autocorrelations $\rho_1, \ldots, \rho_m$, the function `DLAcfToAR` uses the Durbin-Levinson recursions to obtain the parameters $\phi_1, \ldots, \phi_m$ of the best linear predictor of order m as well the partial autocorrelations $\phi_{1,1}, \ldots, \phi_{m,m}$ and the minimum mean-square-errors $\sigma_1^2, \ldots, \sigma_m^2$ corresponding to the k-th order predictor, $k = 1, \ldots, m$. For our purposes `DLAcfToAR` provides more general and useful output than the built-in R function `Acf2AR`.

As an illustrative application, consider the computation R^2 (Nelson 1976),

$$R^2 = 1 - \frac{\sigma_a^2}{\gamma_0}. \tag{21}$$

For a fitted model we may use estimates of the parameters. For sufficiently large m,

$$\hat{R}^2 \doteq 1 - \hat{\sigma}_m^2, \tag{22}$$

where $\hat{\sigma}_m^2$ is computed with `DLAcfToAR` using the fitted values, $\hat{\rho}_1, \ldots, \hat{\rho}_m$,and setting, without loss of generality, $\gamma_0 = 1$ in Equation (21). R^2 indicates the proportion of variability accounted for by the model and is sometimes called the coefficient of forecastibility.

In the brief example below, we show that for a FGN model with $H = 0.84$, $R^2 \doteq 41\%$.

```
R> library("FGN")
R> m <- 10^4
R> r < -FGNAcf(1:m, 0.84)
R> 1 - DLAcfToAR(r)[,3][m]

    10000
0.4075724
```

(Box, Jenkins, and Reinsel 1994, Chapter 7) gave algorithms for forecasting ARMA models which work well when n is not too small and the model is invertible whereas the method given in Section 2.5 is always exact although not as computationally efficient or elegant their methods. (Box *et al.* 1994, Chapter 7) showed,

$$V_k = \sigma_a^2 \sum_{j=0}^{k-1} \psi_k^2. \tag{23}$$

Given autocorrelations $\rho_1, \ldots, \rho_n$, we may fit the AR(n) linear predictor using `DLAcfToAR`, and then expand as a MA using the built-in R function `ARMAtoAR`. Our function `PredictionVariance` uses Equation (23) to compute $V_k, k = 1, \ldots, L$, where L is the maximum lead time. As an option, `PredictionVariance` also implements the exact method of Section 2.5.

Another useful function is `tacvfARMA` for the tacvf of the ARMA(p, q) which is similar to the built-in `ARMAacf` but provides autocovariances (McLeod 1975) instead of autocorrelations. Again this function generalizes the built-in R function in a useful way. The function `tacvfARMA` is needed to demonstrate that the output from our `TrenchForecast` normally agrees quite closely with the built-in `predict.Arima` for ARMA models – see Example 1 in the `TrenchForecast` documentation. This function is also useful in computing the variance of the sample mean using Equation (13).

2.10. Validation and checking of the algorithms

Many checks were done to ensure the accuracy of our results. Most of the checks described below are given in the help file documentation associated with the particular R functions.

The function `TrenchInverse` is easily checked by simply multiplying the output by the input to obtain the identity matrix. The exact concentrated log-likelihood function defined in Equation (10) and implemented in `TrenchLoglikelihood` was checked by implementing in R the Durbin-Levinson recursion to compute (10). This function, `DLLoglikelihood`, with the option `useC=FALSE` provides a slower alternative method for the evaluation of (10). The computation is also easily verified in the case of the Gaussian AR(1) model, $z_t = \phi_1 z_{t-1} + a_t$, $a_t \sim$ NID $(0, \sigma_a^2)$. Given data $z_1, \ldots, z_n$, Equation (10) for the concentrated log-likelihood reduces to

$$L_c(\beta) = \frac{1}{2}\log(1-\phi_1^2) - \frac{n}{2}\log(S/n), \tag{24}$$

where $S = (1-\phi_1^2)z_1^2 + (z_2 - \phi_1 z_1)^2 + (z_3 - \phi_1 z_2)^2 + \ldots + (z_n - \phi_1 z_{n-1})^2$. In the documentation for `DLLoglikelihood` we show numerically that these two methods of evaluating the concentrated log-likelihood for the AR(1) are equivalent.

Also, for the AR(1), the exact forecast function may be written $z_n(k) = \mu + \phi^k(z_n - \mu)$ with variance $V_k = \sigma_a^2(1-\phi^{2k})/(1-\phi^2)$. The output from `TrenchForecast` agrees with the results produced by this formula for the AR(1) case. Details are given in the package documentation for `TrenchForecast`. The function `TrenchMean` can be checked by computing the exact MLE for an AR(1) fitted to some test data and then using the function `TrenchMean` to compute the exact MLE for μ given the estimate $\hat{\phi}$ obtained from `arima`. This estimate of μ closely agrees with that given by `arima`. An illustration of this check is provided in the documentation to the function `TrenchMean`.

3. Application to fractional Gaussian noise

3.1. FGN package

The **FGN** package illustrates the use of the **ltsa** package. This package provides modelling capabilities for the FGN time series defined by the autocorrelation function given in Equation (7). The principal functions available in this package are shown in Table 7.

Function	Purpose
Boot	Generic function for bootstrap
`FGNAcf`	Autocorrelation of FGN
`FGNLL`	Evaluate Equation (10)
`FitFGN`	Exact MLE in FGN
`FitRegressionFGN`	Implements algorithm in Section 2.3
`GetFitFGN`	Fitting function used in `FitFGN`
`HurstK`	Hurst's estimator
`SimulateFGN`	Simulation of FGN time series

Table 7: The principal functions in **FGN**.

n	500	1000	2000	5000
`GetFitFGN`	0.02	0.05	0.19	1.07
`FitFGN`	0.35	0.33	0.50	1.47

Table 8: Average CPU time to simulate and fit an FGN series of length n.

The functions `FGNLL`, `FitFGN`, `FitRegressionFGN`, `GetFitFGN`, and `SimulateFGN` are specific implementations of the general methods discussed in Section 2.5 for the case of the FGN model.

The function `HurstK` provides a nonparametric estimator of H which is described in detail in Hipel and McLeod (1994, page 232).

The simulation function `SimulateFGN` utilizes `DHSimulate` or `DLSimulate`. It was determined empirically that the Davies-Harte non-negativity condition holds for $n \geq 50$ and $0 < H < 0.84$. So in this case `DHSimulate` is used, and `DLSimulate` is used otherwise. Table 8 shows average time taken to simulate and fit an FGN model for various n on our PC Windows XP 3.6 GHz Pentium IV computer.

The output from `FitFGN` is an S3-class object `"FitFGN"` with methods implemented for the standard R generic functions: `coef`, `plot`, `predict`, `print` and `summary`.

3.2. Efficiency of the sample mean

It is shown in Beran (1994, Section 8.2) that the asymptotic efficiency of the sample mean is always greater than 98% in the persistent case, that is, when $\frac{1}{2} < H < 1$. Table 9, obtained by evaluating Equation (14), shows the exact small sample efficiency for various lengths n. The finite sample efficiency is in good agreement with the asymptotic limit in the persistent case. Note that when $H = \frac{1}{2}$, the series is simply Gaussian white noise, and when $0 < H < \frac{1}{2}$ the series is said to be antipersistent. Such antipersistent time series can arise when differencing is used to transform a nonstationary time series to a stationary one. In the strongly antipersistent case with $H = 0.1$, the efficiency of the small mean can be quite low as seen in Table 9. Since most annual geophysical time series exhibit persistence, the sample mean may be used for such data.

n	$H = 0.1$	$H = 0.3$	$H = 0.7$	$H = 0.9$
50	0.6086	0.9492	0.9872	0.9853
500	0.5684	0.9455	0.9866	0.9847
1000	0.5657	0.9453	0.9866	0.9847
2000	0.5643	0.9451	0.9866	0.9847

Table 9: Efficiency of the sample mean in FGN

3.3. Fitting FGN model with ARMA comparison

The Nile minima time series consists of $n = 663$ observations of the annual minimum flow from 622 AD to 1284 AD. It is an often cited example of a geophysical time series exhibiting long-

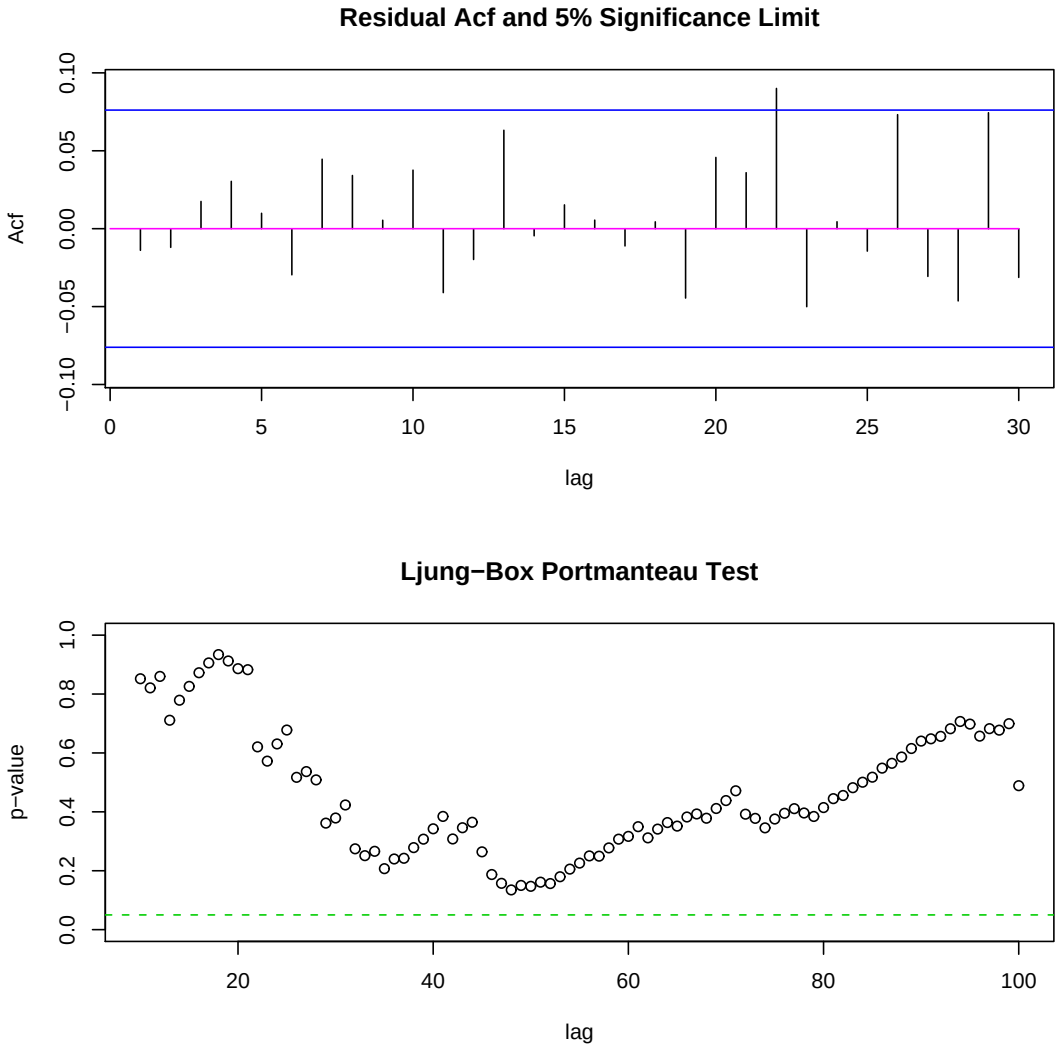

Figure 1: Diagnostic plots produced by `plot` for the fitted FGN model to the `NileMin` series.

term dependence. (Beran 1994, page 118) obtained $\hat{H} = 0.84$ using the Whittle approximate MLE. The MLE using `FitFGN` is $\hat{H} = 0.831$ which agrees with the Whittle estimate as well as with the Hurst K nonparametric estimate which was $K = 0.825$.

```
R> data("NileMin")
R> out < -FitFGN(NileMin)
R> out

H = 0.831,  R-sq = 38.46%
```

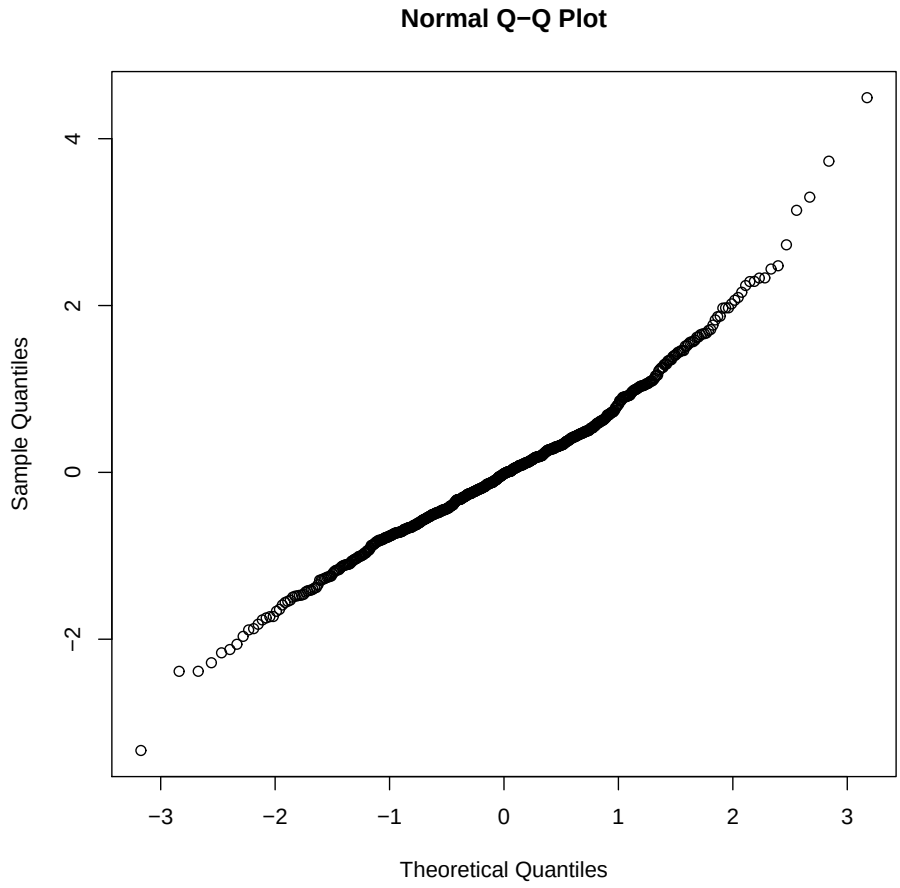

Figure 2: Normal probability plot of the standardized prediction residuals of the fitted FGN model to the `NileMin` series.

```
length of series = 663 ,  number of parameters = 2
loglikelihood = 236.52 ,  AIC = -469 ,  BIC =  -460

R> coef(out)

          MLE         sd Z-ratio
H   0.8314782 0.03028091  10.947
mu 11.4812519 0.33459046  34.314

R> plot(out)
R> qqnorm(resid(out))
```

The diagnostic plots produced by `plot` are shown in Figure 1. Figure 2 shows the normal probability of the residuals. These diagnostic plots confirm that the FGN is a reasonable model.

Model	k	L_m	AIC	BIC
FGN	1	236.52	−471.0	−466.5
ARMA$(2,1)$	3	237.61	−469.2	−455.7

Table 10: Comparison of models fitted to the Nile minima time series. L_m is the concentrated log-likelihood, BIC $= -2L_m + 2k$, BIC $= -2L_m + k\log(n)$, $n = 663$ is the series length and and k is the number of parameters.

An ARMA$(2,1)$ was fit using the R function `arima` and was found to fit the observed series very well. The exact likelihood for the fitted ARMA model was determined using the `DLLoglikleihood` function. Table 10 summarizes the fits in terms of L_m. We see that the ARMA$(2,1)$ is slightly better in terms of the log-likelihood but requires more parameters than FGN so that FGN is better in terms of both the AIC and BIC criteria.

In Table 11, we compare the forecasts at origin time $n = 663$ for lead times $k = 1, \ldots, 5$ for the FGN and ARMA models. The standard deviations of the forecasts were also computed. The differences between the models seem minor. In the case of the ARMA model we compared the built-in R function `predict.Arima` with our `TrenchForecast` for the fitted ARMA model. As expected there is almost no difference in this case.

A further forecasting experiment compares the quality of the forecasts using `TrenchForecast` with the built-in R function `predict` for the ARMA$(2,1)$ model. In addition, the forecast for the fitted FGN model was included to compare with the ARMA$(2,1)$. In each case, the model was fit to all the time series up to time $t = n_1 + k$ for each k, $k = 1, \ldots, K$, where we took $n_1 = n - K$, $K = 100$, and $n = 663$ is the length of the series `NileMin`. For each t, $t = n_1+1, \ldots, n$, we compute the forecasts $z_t(\ell), \ell = 1, 2, 3$, and their errors $e_t(\ell) = z_{t+\ell} - z_t(\ell)$. The empirical root-mean-square errors (RMSE), were computed for $\ell = 1, 2, 3$ and are shown in Table 12. It is interesting that the FGN outperformed ARMA and that at $\ell = 2$ the `TrenchForecast` was more accurate than `predict.Arima`. The R script to produce Table 12 is available with our paper.

It is common practice to divide the data in two parts: a training sample and a test sample. The model is fit to the training sample and then its performance is evaluated on the test sample. In time series analysis, experiments are often reported in which an initial portion

		Lead-time of forecast				
Model	Algorithm	1	2	3	4	5
		Forecast				
FGN	predict.FitFGN	11.34	11.46	11.51	11.54	11.56
ARMA	predict.Arima	11.40	11.53	11.57	11.58	11.58
ARMA	TrenchForecast	11.40	11.53	11.57	11.58	11.58
		Standard deviation of forecast				
FGN	predict.FitFGN	0.70	0.76	0.78	0.79	0.80
ARMA	predict.Arima	0.70	0.76	0.78	0.79	0.79
ARMA	TrenchForecast	0.70	0.76	0.78	0.79	0.80

Table 11: Forecasts and their standard deviations for `NileMin`.

Lead	FGN	ARMA(2, 1)	ARMA(2, 1)
1	0.378	0.381	0.381
2	0.558	0.562	0.610
3	0.668	0.675	0.677

Table 12: RMSEs for forecasts for the last 100 values in the `NileMin` dataset. In this case, the model was refit to each series for each new data value and the forecast for the updated model was computed. For comparison, the sample standard deviation of the series over the forecasting period was 0.733.

of the time series is used for fitting the model, and the second portion is used to evaluate the out-of-sample forecast performance (Noakes, McLeod, and Hipel 1985; Jaditz and Sayers 1998). In the case of ARIMA models the built-in function `predict` can not do this, since it is necessary to update the fitted model for the next forecast. A script was written to fit ARMA(p, q) and FGN time series models to one part of the series and to compute the RMSE for the second part. This script is available in the online documentation for the dataset `NileMin`. Selecting $p = 2, q = 1$, we fit the ARMA(p, q) as well as the FGN to all but the last $m = 100$ observations, and then compared the forecasts with the actual values for the remaining 100 values. The RMSE for the forecasts at lead times $\ell = 1, 2, 3$ was computed. The forecasts for both models were computed using `TrenchForecast`. As shown in Table 13, the FGN model slightly outperforms the ARMA$(2, 1)$ model. Notice that because the model is not updated, the forecast errors are larger in Table 13 than in Table 12.

Lead	FGN	ARMA(2, 1)
1	0.568	0.579
2	0.659	0.678
3	0.686	0.706

Table 13: RMSEs for forecasts for the last 100 values in the `NileMin` dataset. In this case, the model was fit only once to the first 563 values and then its forecasting performance was evaluated on the next 100 values.

3.4. Simulation and bootstrapping experiments

The computation results reported in this section were carried out using the **Rmpi** package (Yu 2002) on a Beowulf cluster with 48 CPUS running Debian Linux. This reduced the time needed for our computations by a factor of about 30. The R scripts used are available and provide a template for bootstrapping and simulating with **Rmpi**.

Properties of MLE for Hurst coefficient

We investigated the asymptotic properties of the MLE, $\hat{H}$, using the function `GetFitFGN`. We simulated 10^5 replications of FGN with various parameter settings, H and n, shown in Table 14 and determined the empirical asymptotic bias, $\sqrt{n}(\hat{H} - H)$, and asymptotic variance, $n \operatorname{Var}(\hat{H})$. The variance depends on the long-memory parameter H. The last column in Table 14 is used in our function `FitFGN` to obtain an approximate standard error of $\hat{H}$.

	n									
H	100	200	500	1000	2000	100	200	500	1000	2000
	Asymptotic Bias					Asymptotic Variance				
0.1	0.04	0.03	0.02	0.02	0.01	0.14	0.14	0.13	0.13	0.13
0.2	0.14	0.10	0.08	0.06	0.05	0.26	0.25	0.24	0.24	0.23
0.3	0.21	0.16	0.11	0.08	0.07	0.35	0.33	0.32	0.31	0.31
0.4	0.26	0.20	0.14	0.10	0.08	0.42	0.40	0.37	0.36	0.36
0.5	0.30	0.23	0.16	0.12	0.09	0.47	0.43	0.41	0.40	0.40
0.6	0.33	0.25	0.18	0.13	0.10	0.50	0.47	0.44	0.43	0.42
0.7	0.37	0.27	0.19	0.14	0.11	0.51	0.48	0.46	0.45	0.44
0.8	0.41	0.31	0.21	0.16	0.12	0.50	0.48	0.46	0.46	0.45
0.9	0.49	0.37	0.25	0.18	0.14	0.42	0.42	0.44	0.45	0.45

Table 14: Properties of the MLE for H.

Bootstrapped forecasting experiment

The FGN model was also fit to the entire `NileMin` series, and then three independent bootstrap versions of the series were generated using `Boot`. The forecasting experiment discussed in the last paragraph of Section 3.3 was then repeated on each of the bootstrap series but in addition to fitting the FGN and ARMA$(2, 1)$ models, we also used FGN model with the true parameter value $H = 0.831$ in order to allow us to investigate the effect of parameter estimation errors on the FGN forecast.

Then the bootstrapping was iterated $B = 10^4$ times for all three models. The average RMSE is shown in Table 15. As expected when the parameter H is known, the forecast is then, without question, the optimal RMSE forecast and as expected it did better than the other two methods using estimated parameters. The FGN only slightly outperforms the ARMA$(2, 1)$ model at lead-times 2 and 3. The whole experiment was repeated six times, and each time results comparable to Table 15 were produced. If desired, the jackknife could be used to estimate the standard deviation or the experiment could be re-run as a paired comparison, and a suitable confidence interval for the difference given but we feel confident enough about the overall general conclusion. The R script used for Table 15 is provided for the interested reader.

Lead	FGN	FGN	ARMA$(2, 1)$
1	0.545	0.546	0.546
2	0.594	0.596	0.597
3	0.610	0.612	0.613

Table 15: Average RMSE in $B = 10^4$ bootstrap iterations.

3.5. Nile river intervention analysis

In 1903 the Aswan Dam went into operation and it is of hydrological interest to quantify the effect of this dam on downstream annual riverflow (Hipel, Lennox, Unny, and McLeod 1975).

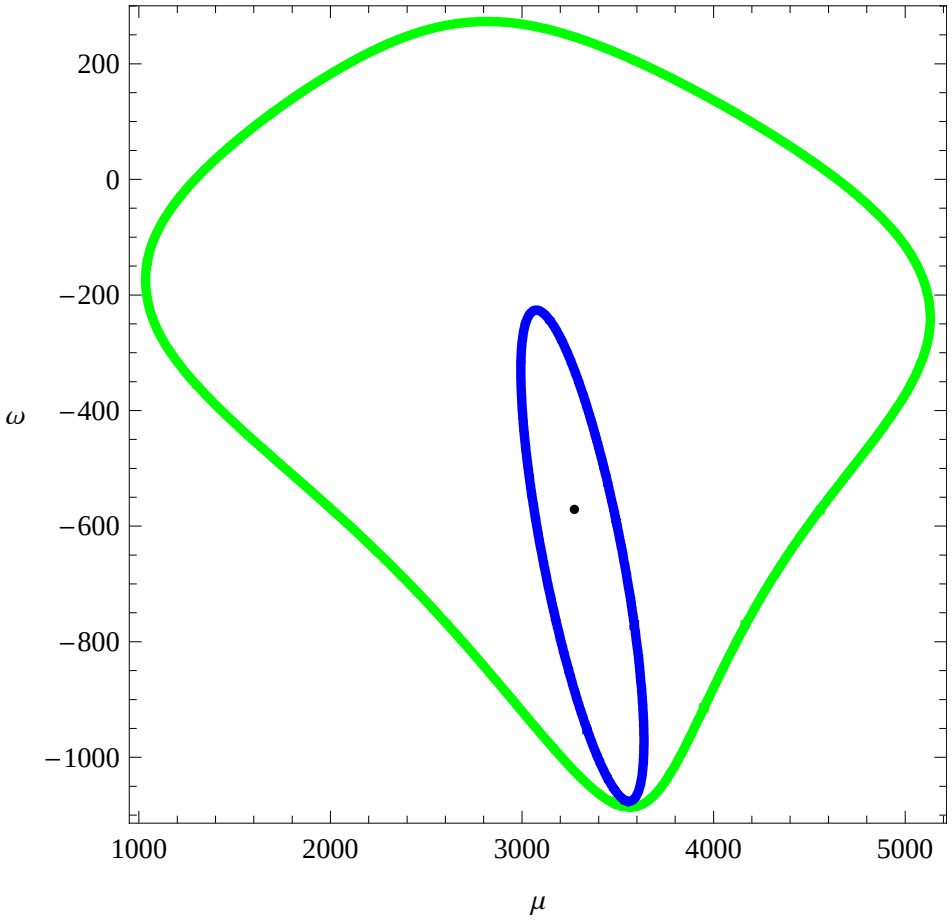

Figure 3: Comparison of 95% confidence regions for (μ, ω) with AR(1) and FGN disturbances. The confidence regions were determined using a likelihood ratio test. The larger region corresponds to the FGN case. This graph is interesting because it shows that a mis-specified ARMA may may drastically underestimate the uncertainty involved in some situations. This figure was produced using a Mathematica version of the **ltsa** and **FGN** packages.

An intervention analysis model was described in Hipel and McLeod (1994, Section 19.2.4) for the average annual riverflow in cubic meters per second for the Nile river at Aswan, 1870–1945. The model fit by Hipel and McLeod (1994, Section 19.2.4) may be written, $z_t = \mu + \omega S_t^{(T)} + \xi_t$, where $\xi_t = \phi \xi_{t-1} + a_t$ follows an AR(1) and $S_t^{(T)}$ is the unit step function defined by,

$$S_t^{(T)} = \begin{cases} 0 & \text{if } t < T \\ 1 & \text{if } t \geq T \end{cases} \tag{25}$$

It is assumed that $T = 33$. This model was fit using the algorithm outlined in Section 2.7 for both the AR(1) and the FGN case. The parameter estimates are shown in Table 16. The AR(1) fits slightly better in terms of L_m, but the relative plausibility, defined by the likelihood ratio (Royall 1997; Sprott 2000), of the FGN versus the AR(1) is about 8%. So the FGN error model cannot be ruled out. Furthermore, some scientists feel that long-memory models are often more suitable for many annual geophysical time series (Beran 1994; Hampel 1998), and

Parameter	AR(1)	FGN
μ	3343.11	3273.88
ω	-699.863	-571.082
β	0.391	0.781
L_m	-449.855	-452.363

Table 16: Intervention models fit the annual Nile riverflow series. The parameter $\beta = \phi$ or H correspond to the AR(1) and FGN models.

so the FGN model may be preferred on these grounds. Figure 3 shows the confidence regions for both models. The larger region corresponds to FGN errors. This is due to the stronger form of dependence present in the FGN case. In conclusion, we see that the uncertainty in the effect of the intervention is much greater with FGN errors.

4. Concluding remarks

The **ltsa** package is implemented in R (R Development Core Team 2007) and C and is available from the Comprehensive R Archive Network at `http://CRAN.R-project.org/`. The **FGN**, available as well from CRAN, illustrated how this package can be used to provide a building block for other time series software. It is intended to develop a package for another paper which implements the ARFIMA model and some of its generalizations. Another possible extension is to the parametric model suggested by Bloomfield (1973). In this case a fast method is needed to evaluate the required autocovariances. It is hoped to report on these developments in another paper.

S-PLUS, Mathematica and MATLAB are some other computing environments that are very popular for teaching and research and it is hoped that others may find it useful to port the **ltsa** package to these and other computer environments.

Acknowledgments

Support from NSERC Discovery and Equipment Grants held by A. Ian McLeod and Hao Yu is acknowledged with thanks. We would also like to thank Duncan Murdoch for helpful comments about R. Thanks also to Achim Zeileis for comments on the technical typesetting of this document.

References

Aronsson M, Holst L, Lindoff B, Svensson A (2006). "Bootstrap Control." *IEEE Transactions on Automatic Control*, **51**, 28–37.

Baillie RT (1996). "Long Memory Processes and Fractional Integration in Econometrics." *Journal of Econometrics*, **73**, 5–59.

Beran J (1994). *Statistics for Long-Memory Processes*. Chapman Hall, New York.

Bloomfield P (1973). "An Exponential Model for the Spectrum of a Scalar Time Series." *Biometrika*, **60**, 217–226.

Box GEP, Jenkins GM, Reinsel GC (1994). *Time Series Analysis: Forecasting and Control*. Holden-Day, San Francisco, 3rd edition.

Box GEP, Luceño A (1997). *Statistical Control by Monitoring and Feedback Adjustment*. Wiley, New York.

Brockwell PJ, Davis RA (1991). *Time Series: Theory and Methods*. Springer, New York, 2nd edition.

Chen WW, Hurvich C, Lu Y (2006). "On the Correlation Matrix of the Discrete Fourier Transform and the Fast Solution of Large Toeplitz Systems for Long-memory Time Series." *Journal of the American Statistical Association*, **101**, 812–822.

Davies RB, Harte DS (1987). "Tests for Hurst Effect." *Biometrika*, **74**, 95–101.

Dembo S (1991). "Scenario Optimization." *Annals of Operations Research*, **30**, 63–80.

Golub G, Loan CV (1996). *Matrix Computations*. John Hoptkins University Press, Baltimore, 3rd edition.

Graybill FA (1983). *Matrices with Applications in Statistics*. Wadsworth, Belmont.

Hamilton JD (1994). *Time Series Analysis*. Princeton University Press, New Jersey.

Hampel F (1998). "Is Statistics too Difficult?" *Canadian Journal of Statistics*, **26**, 497–513.

Hannan EJ (1970). *Multiple Time Series*. Wiley, New York.

Hannan EJ, Heyde CC (1972). "On Limit Theorems for Quadratic Functions of Discrete Time Series." *The Annals of Mathematical Statistics*, **43**, 2058–2066.

Hannan EJ, Kavalieris L (1991). "The Convergence of Autocorrelations and Autoregressions." *Australian Journal of Statistics*, **25**, 287–297.

Hipel KW, Lennox WC, Unny TE, McLeod AI (1975). "Intervention Analysis in Water Resources." *Water Resources Research*, **11**, 855–861.

Hipel KW, McLeod AI (1994). *Time Series Modelling of Water Resources and Environmental Systems*. Elsevier, Amsterdam. Electronic reprint available, http://www.stats.uwo.ca/faculty/aim/1994Book/.

Hosking JRM (1981). "Fractional Differencing." *Biometrika*, **68**, 165–176.

Jaditz T, Sayers CL (1998). "Out-of-Sample Forecast Performance as a Test for Nonlinearity in Time Series." *Journal of Business & Economic Statistics*, **16**, 110–117.

Kabaila PV (1980). "An Optimality Property of the Least-squares Estimate of the Parameter of the Spectrum of a Purely Nondeterministic Time Series." *Annals of Statistics*, **8**, 1082–1092.

Li WK (1981). *Topics in Time Series Analysis*. Ph.D. thesis, University of Western Ontario, London, Ontario, Canada.

Li WK (2004). *Diagnostic Checks in Time Series*. Chapman & Hall/CRC, New York.

Li WK, McLeod AI (1987). "ARMA Modelling with Non-Gaussian Innovations." *Journal of Time Series Analysis*, **9**, 155–168.

Lin JW, McLeod AI (2007). "Portmanteau Tests for Randomness and Diagnostic Check in Stable Paretian AR Models." *Journal of Time Series Analysis*.

Ljung GM, Box GEP (1978). "On a Measure of Lack of Fit in Time Series Models." *Biometrika*, **65**, 297–303.

Maceira MEP, Damázio JM (2006). "Use Of The Par(p) Model In The Stochastic Dual Dynamic Programming Optimization Scheme Used In The Operation Planning Of The Brazilian Hydropower System." *Probability in the Engineering and Informational Sciences*, **20**, 143–156.

Mandelbrot BB, Hudson RL (2004). *The Misbehavior of Markets*. Basic Books, New York.

McCullagh P, Nelder JA (1989). *Generalized Linear Models*. Chapman and Hall, London, 2nd edition.

McCullough BD (1994). "Bootstrapping Forecast Intervals: An application to AR(p) models." *Journal of Forecasting*, **13**, 51–66.

McLeod AI (1975). "Derivation of the Theoretical Autocorrelation Function of Autoregressive Moving-Average Time Series." *Applied Statistics*, **24**, 255–256.

McLeod AI (2005). "Inverse Covariance Matrix of ARMA(p, q) Time Series Models." *Mathematica* Information Center, *MathSource*, http://library.wolfram.com/infocenter/MathSource/5723/.

McLeod AI, Quenneville B (2001). "Mean Likelihood Estimators." *Statistics and Computing*, **11**, 57–65.

McLeod AI, Vingilis ER (2005). "Power Computations for Intervention Analysis." *Technometrics*, **47**, 174–180.

McLeod AI, Zhang Y (2007). "Faster ARMA Maximum Likelihood Estimation." *Computational Statistics and Data Analysis*, **52**, 2166–2176.

Nelson CR (1976). "The Interpretation of R^2 in Autoregressive-Moving Average Time Series Models." *The American Statistician*, **30**, 175–180.

Noakes DJ, McLeod AI, Hipel KW (1985). "Forecasting Seasonal Hydrological Time Series." *The International Journal of Forecasting*, **1**, 179–190.

Parzen E (1962). *Stochastic Processes*. Holden-Day, San Francisco.

Rao CR (1962). "Efficient Estimates and Optimum Inference Procedures in Large Samples." *Journal of the Royal Statistical Society B*, **24**, 46–72.

R Development Core Team (2007). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Royall RM (1997). *Statistical Evidence: A Likelihood Paradigme*. Chapman and Hall, New York.

Siddiqui (1958). "On the Inversion of the Sample Covariance Matrix in a Stationary Autoregressive Process." *Annals of Mathematical Statistics*, **29**, 585–588.

Sowell F (1992). "Maximum Likelihood Estimation of Stationary Univariate Fractionally Integrated Time Series Models." *Journal of Econometrics*, **53**, 165–188.

Sprott DA (2000). *Statistical Inference in Science*. Springer, New York.

Taleb NN (2007). *The Black Swan: The Impact Of The Highly Improbable*. Random House, New York.

Taniguchi M (1983). "On the Second Order Asymptotic Efficiency of Estimators of Gaussian ARMA Processes." *The Annals of Statistics*, **11**, 157–169.

Trench WF (1964). "An Algorithm for the Inversion of Finite Toeplitz Matrices." *SIAM Journal*, **12**, 515–522.

Whittle P (1962). "Gaussian Estimation in Stationary Time Series." *Bulletin of the International Statistical Institute*, **39**, 105–129.

Yu H (2002). "**Rmpi**: Parallel Statistical Computing in R." *R News*, **2**(2), 10–14.

Zinde-Walsh V (1988). "Some Exact Formulae for Autoregressive Moving Average Processes." *Econometric Theory*, **4**, 384–402.

A. Wold decomposition

The Wold decomposition Brockwell and Davis (1991, Section 5.7) indicates that any covariance stationary time series may be expressed as the sum of a deterministic component plus an infinite moving average as in Equation (1) where a_t is white noise, that is, $\mathsf{E}(a_t) = 0$, $\mathrm{Var}\,(a_t) = \sigma_a^2$ and $\mathrm{Cov}\,(a_t, a_s) = 0,\ t \neq s$. In practice the deterministic component often taken to be the mean, μ, of the series.

The autocovariances and moving-average coefficients satisfy the equation,

$$\gamma(B) = \sigma_a^2 \psi(B)\psi(B^{-1}), \tag{26}$$

where $\psi(B) = 1 + \psi_1 B + \psi_2 B^2 + \ldots$. They are equivalent parameterizations.

B. Ergodicity

The ergodicity condition for GLP which was mentioned in the introduction means that $\gamma_k \longrightarrow 0$ as $k \longrightarrow 0$ sufficiently fast so that it is possible to estimate the mean and autocovariances from past values of the series. Sometimes this condition is referred to as mixing. See Parzen (1962) for an introductory approach and (Hannan 1970, Section IV.3) for a full discussion of ergodicity in the GLP.

C. Exact MLE

Under some further conditions, the maximum likelihood estimate (MLE) in the GLP is consistent, asymptotically normal and efficient (Hannan 1970, pages 395–398). As indicated in the introduction one of the reasons for exact MLE is that experience suggests it works better – especially for short series. It should also be pointed out that asymptotically the maximum likelihood estimators have been found to be second-order efficient (Taniguchi 1983). Not all first-order efficient estimation methods are necessarily second-order efficient (Rao 1962).

D. Gaussian efficiency and QMLE

When the assumption that $a_t \sim \mathrm{NID}\,(0, \sigma_a^2)$ is relaxed to that merely the $a_t \sim \mathrm{IID}\,(0, \sigma_a^2)$, then it can be shown that for many types of linear time series models (Hannan 1970, pages 395–398) the estimates obtained by maximizing the likelihood function under the normality assumption continue to enjoy the same properties of consistency and asymptotic normality with the same covariance matrix as in the Gaussian case (Kabaila 1980; Whittle 1962). In this situation this estimator are are known as the quasi-maximum likelihood estimator (QMLE) (McCullagh and Nelder 1989, Chapter 9). Whittle (1962) has termed this *Gaussian efficiency* since the estimators obtained by maximizing the Gaussian likelihood even when a_t are IID continue to have the same large-sample distribution as in the case of Gaussian a_t. When in fact the a_t are non-Gaussian, the true MLE may have even smaller variances than that of the Gaussian estimators (Li and McLeod 1987). So among the IID distributions with finite variance, the Gaussian case is in a sense the worst case scenario from an asymptotic viewpoint. The assumption on the innovations a_t may be relaxed even further, merely assuming that it is a sequence of martingale differences with finite fourth moment (Hannan and Heyde 1972;

Hannan and Kavalieris 1991). This is especially interesting since it shows that linear time series models may still be quite useful for modeling long financial time series which may exhibit GARCH innovations. The GARCH innovations may themselves be modeled using a linear time series model fitted to the squared values of the innovations and the usual Gaussian efficiency property will still hold. When the finite variance assumption on σ_a^2 is relaxed, the Gaussian estimates may still be consistent but the rate of convergence is even faster (Lin and McLeod 2007).

E. Outliers and gray swans

Two very interesting books about financial markets, Mandelbrot and Hudson (2004) and Taleb (2007), discuss the limitations of the normal distribution for describing many financial time series. Even though these authors are undoubtedly correct about the limitations of the normal distribution, linear time series models estimated via QMLE remains essentially valid under even these conditions. Of course, even more efficient estimators may be available if the distribution is known as was demonstrated by Lin and McLeod (2007).

In the non-Gaussian case, it is especially important that care must be taken to properly account for the uncertainty in forecasting and simulation applications. In forecasting, a prediction interval based on the normal distribution may greatly underestimate the true forecast error. Similarly in scenario generation in financial planning (Dembo 1991) the choice of the innovation distribution may be important for simulation.

Affiliation:

A. Ian McLeod, Hao Yu, Zinovi L. Krougly
Department of Statistical and Actuarial Sciences
University of Western Ontario
London, Ontario N6A 5B9, Canada
E-mail: aimcleod@uwo.ca, hyu@stats.uwo.ca, zkrougly@stats.uwo.ca
URL: http://www.stats.uwo.ca/faculty/aim/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 23, Issue 5 *Submitted:* 2005-10-14
December 2007 *Accepted:* 2007-11-11