Discussion Papers
Department of Economics
University of Copenhagen

No. 13-04

# A fast fractional difference algorithm

Andreas Noack Jensen & Morten Ørregaard Nielsen

Øster Farimagsgade 5, Building 26, DK-1353 Copenhagen K., Denmark
Tel.: +45 35 32 30 01 – Fax: +45 35 32 30 00
http://www.econ.ku.dk

ISSN: 1601-2461 (E)

# A fast fractional difference algorithm*

Andreas Noack Jensen
University of Copenhagen

Morten Ørregaard Nielsen†
Queen's University and CREATES

First version April, 2013. This version May 23, 2013.

**Abstract**

We provide a fast algorithm for calculating the fractional difference of a time series. In standard implementations, the calculation speed (number of arithmetic operations) is of order $T^2$, where $T$ is the length of the time series. Our algorithm allows calculation speed of order $T \log T$. For moderate and large sample sizes, the difference in computation time is substantial.

**JEL Codes:** C22, C63, C87.

**Keywords:** Circular convolution theorem, fast Fourier transform, fractional difference.

## 1 Introduction

In the estimation or simulation of fractionally integrated (or fractional) time series models, the computational cost is almost exclusively associated with the calculation of fractional differences. Indeed, the computational cost of these calculations can be so great that estimation or simulation of fractional models is infeasible when the sample size is very large.

In this paper, we derive an algorithm for the calculation of fractional differences based on circular convolutions. The advantage of our algorithm is that it is designed to exploit very efficient implementations of the discrete Fourier transform, i.e. the fast Fourier transform (Cooley and Tukey, 1965). The number of arithmetic operations required, and hence the calculation speed, of standard implementations of the fractional difference operation is of order $T^2$, where $T$ is the length of the time series, i.e. the sample size. Note that we use "order" to denote the tight (asymptotic) bound, that is, $f(T)$ is of order $g(T)$ if for some $T_0$ and $K_u > K_l > 0$ then $K_l|g(T)| \leq |f(T)| \leq K_u|g(T)|$ whenever $T > T_0$.

*We are grateful to Jurgen Doornik, Uwe Hassler, Søren Johansen, James MacKinnon, and Rocco Mosconi for comments and to the Canada Research Chairs program, the Social Sciences and Humanities Research Council of Canada (SSHRC), and the Center for Research in Econometric Analysis of Time Series (CREATES, funded by the Danish National Research Foundation) for financial support.

†Corresponding author. Postal address: Department of Economics, Dunning Hall, Queen's University, 94 University Avenue, Kingston, Ontario K7L 3N6, Canada. Email address: mon@econ.queensu.ca

In contrast, our algorithm is able to achieve order $T \log T$. For large sample sizes, the difference in computation time is substantial.

As an example, suppose we observe a sample of size $T = 100,000$, which is not at all unreasonable for applications in, e.g., finance. As illustrated below, in a standard MATLAB® implementation (other languages provide similar timings), calculating the fractional difference just one time requires about 2.7 seconds of CPU time on an Intel Core i5-2400 3.1GHz desktop. In comparison, a MATLAB implementation of our algorithm is able to calculate the same fractional difference in only 0.012 seconds of CPU time. In the estimation of even the simplest fractional time series model, based on, e.g., a conditional-sum-of-squares criterion, one would expect to calculate 5-6 fractional differences for each iteration in the numerical optimization (one to evaluate the objective function and a few more to evaluate the gradient and Hessian numerically). If 15-20 iterations are required to locate an optimum of the objective function, that suggests that roughly 100 fractional differences would need to be calculated. Thus, for $T = 100,000$, the difference in estimation time for the standard implementation versus our implementation of the fractional difference algorithm is of the order of 4.5 minutes versus 1.2 seconds. The computational costs with standard implementations seems prohibitive for bootstrap or simulation procedures with large sample sizes. On the other hand, such procedures remain quite feasible with our implementation of the fractional difference operator.

The remainder of the paper is laid out as follows. In the next section we describe the fractional difference operation in more detail and derive our proposed algorithm. Section 3 provides numerical results, and some further discussion and conclusions are given in section 4.

## 2 Fast fractional difference algorithm

Consider the time series $X_t$, which is observed for $t = 1, \ldots, T$. Suppose we want to calculate the fractional difference

$$Y_t = \Delta_+^d X_t = \sum_{j=0}^{t-1} \pi_j(-d) X_{t-j}, \qquad t = 1, \ldots, T, \tag{1}$$

where the fractional coefficients $\pi_j(u)$ are defined as the coefficients in an expansion of $(1-z)^{-u}$, which are

$$\pi_j(u) = \frac{u(u+1)\cdots(u+j-1)}{j!}, \qquad j = 0, 1, \ldots. \tag{2}$$

Note that the summation in (1) is truncated at $t-1$ because we only observe $X_t$ starting at time $t = 1$. The subscript "+" on the fractional difference operator thus indicates that only observations on $X_t$ with a positive time index are included in the summation. If we had pre-sample observations (initial values) on $X_t$ that we wanted to include, then the summation would be extended to include those as well; see Johansen and Nielsen (2012a,b). However, such considerations are not essential to the developments in this paper, and therefore we do not consider this possibility further.[1]

[1]The convention applied here is that of a so-called "type II" fractional process, see e.g. Marinucci and Robinson (1999). While this is certainly not the only type of fractional process, these definitions are not

The standard calculation of the fractional difference in (1) is done as a linear convolution of the two series $X=(X_t)_{t=1}^T$ and $q=(\pi_{t-1}(-d))_{t=1}^T$. That is, the time series $Y=(Y_t)_{t=1}^T$ with $t$'th element given in (1) can be written as

$$Y_t=\sum_{j=1}^t q_j X_{t-j+1}, \qquad t=1,\ldots,T. \tag{3}$$

Because the number of arithmetic operations required in each sum in (3) is of order $t$, the whole linear convolution operation for $t=1,\ldots,T$ is of order $T^2$.

Our algorithm for the fractional difference operator takes advantage of a frequency-domain transformation, and we therefore define the discrete Fourier transform $f=(f_j)_{j=1}^T$ of a series $a=(a_t)_{t=1}^T$ as the solution to the equation $a=T^{-1}Ff$, where $F$ is the Fourier matrix with $(j,k)$'th element $(F)_{jk}=w_T^{(j-1)(k-1)}$ and $w_T=e^{\mathrm{i}2\pi/T}$ with $\mathrm{i}=\sqrt{-1}$ denoting the imaginary unit. Each element of $a$ can therefore be expressed in terms of the Fourier coefficients $f_j$ and powers of $w_T$ as

$$a_t=\frac{1}{T}\sum_{j=1}^T f_j w_T^{(t-1)(j-1)}, \qquad t=1,\ldots,T. \tag{4}$$

Since $F$ is symmetric and $F\overline{F}=TI_T$, where the bar denotes complex conjugation, the inverse operation is $(T^{-1}F)^{-1}=\overline{F}$, i.e. the complex conjugate of each element in $F$, such that $f_j=\sum_{t=1}^T a_t w_T^{-(t-1)(j-1)}$. Thus, the matrix $\overline{F}$ represents the discrete Fourier transform whereas $T^{-1}F$ is the inverse transform.

The circular convolution of two series $a$ and $b$ of length $T$ is denoted $a\circledast b$ and is defined such that it contains $T$ elements, each calculated from a sum of $T$ terms. Formally, define

$$(a\circledast b)_t=\sum_{j,k\in J_t^T} a_j b_k, \qquad t=1,\ldots,T, \tag{5}$$

where $J_t^T=\{(j,k)|\,j,k=1,\ldots,T\wedge j+k-1\equiv t \pmod T\}$. Finally, for any $T\times 1$ vector $a$, let

$$\tilde{a}=[a',0'_{T-1}]' \tag{6}$$

denote the $(2T-1)\times 1$ vector consisting of $a$ extended with $T-1$ zeros.

In Theorem 1 below, we state the finite version of the circular convolution theorem, which shows how the circular convolution of finite sequences (5) can be calculated by application of the discrete Fourier transform. For periodic integrable functions this result can be found in, e.g., Zygmund (2003, Theorem 1.5, p. 36). The finite version has appeared in various forms in the engineering literature as an important application of the fast Fourier transform, see e.g. Stockham (1966, p. 230) and Cooley, Lewis, and Welch (1969, p. 32). The version in Theorem 1 allows a simple proof of our main result. Because the notion of circular convolution of finite sequences seems less known in the econometrics literature we provide a brief proof of the theorem.

essential to this paper, since our focus is on fast calculation of the fractional difference operator in (1). Please see section 4 for some further remarks on this.

**Theorem 1** *Let $a = (a_t)_{t=1}^T$ and $b = (b_t)_{t=1}^T$ be two sequences. Then*

$$a \circledast b = T^{-1} F(\overline{F} a \circ \overline{F} b), \tag{7}$$

*where the symbol $\circ$ denotes element-wise matrix multiplication.*

**Proof.** Let $f = \overline{F} a$ and $g = \overline{F} b$ denote the discrete Fourier transforms of $a$ and $b$, respectively. It then needs to be shown that $a \circledast b = T^{-1} F(f \circ g)$. To do this, insert the expressions for $a$ and $b$ in terms of their discrete Fourier transforms,

$$\begin{aligned}
(a \circledast b)_t &= \sum_{j,k \in J_t^T} a_j b_k = \sum_{j,k \in J_t^T} \left( T^{-1} \sum_{s=1}^T f_s w_T^{(j-1)(s-1)} \right) \left( T^{-1} \sum_{u=1}^T g_u w_T^{(k-1)(u-1)} \right) \\
&= T^{-2} \sum_{s=1}^T \sum_{u=1}^T f_s g_u \sum_{j,k \in J_t^T} w_T^{(j+k-2)(s-1)+(k-1)(u-s)} \\
&= T^{-2} \sum_{s=1}^T \sum_{u=1}^T f_s g_u w_T^{(t-1)(s-1)} \sum_{k=1}^T w_T^{(k-1)(u-s)} = T^{-1} \sum_{s=1}^T f_s g_s w_T^{(t-1)(s-1)},
\end{aligned}$$

where the penultimate equality follows by definition of $J_t^T$ and the last equality follows because of the well-known result

$$\sum_{j=1}^T w_T^{(j-1)k} = \begin{cases} T & \text{if } k \equiv 0 \pmod T, \\ 0 & \text{if } k \not\equiv 0 \pmod T. \end{cases}$$

This shows that the Fourier coefficients of $a \circledast b$ are given by the elementwise product of the Fourier coefficients of $a$ and $b$. ■

The next theorem presents our main result, which is to show how the finite circular convolution theorem can be used to calculate the fractional difference in (1), or equivalently in (3), by the discrete Fourier transform.

**Theorem 2** *The fractionally differenced time series $Y$ in (3) can be calculated as the first $T$ elements of the $(2T-1) \times 1$ vector*

$$T^{-1} F(\overline{F} \tilde{q} \circ \overline{F} \tilde{X}). \tag{8}$$

**Proof.** By equation (7) it holds that the $t$'th element of (8) is equal to $(\tilde{q} \circledast \tilde{X})_t$. Furthermore, for $t = 1, \ldots, T$, we have

$$(\tilde{q} \circledast \tilde{X})_t = \sum_{j,k \in J_t^{2T-1}} \tilde{q}_j \tilde{X}_k = \sum_{j=1}^t \tilde{q}_j \tilde{X}_{t-j+1} + \sum_{j=t+1}^{2T-1} \tilde{q}_j \tilde{X}_{2T+t-j} = \sum_{j=0}^{t-1} \pi_j X_{t-j} \tag{9}$$

because $\tilde{q}_j = \pi_{j-1}$ for $j = 1, \ldots, T$ and $\tilde{q}_j = 0$ for $j \geq T+1$, while $\tilde{X}_{t-j+1} = X_{t-j+1}$ for $j = 1, \ldots, t$ and $\tilde{X}_{2T+t-j} = 0$ for $t+1 \leq j \leq T$ by the definition (6). ■

The significance of Theorem 2 lies in the fact that the discrete Fourier transform can be calculated very efficiently by means of the fast Fourier transform algorithm, where

the number of arithmetic operations required is proportional to $T \log T$, see Cooley and Tukey (1965). Because the operation in (8) only applies three discrete Fourier transforms and one element-wise multiplication of two vectors (which is of order $T$), the fractional difference algorithm (8) in Theorem 2 is itself of order $T \log T$.

Note that our result in Theorem 2 provides an exact calculation of the fractional difference in (1), and that no approximation is involved. Finally, also note that, depending on the particular implementation of the fast Fourier transformation applied, it may be necessary in practice to extend the series $X$ and $q$ to a length greater than the $2T-1$ used in (6). Specifically, some implementations of the fast Fourier transform require the length to be a power of two, and in that case $\tilde{X}$ and $\tilde{q}$ should be extended with zeros to length equal to the smallest power of two that is at least $2T-1$.

## 3 Numerical results

In this section we illustrate numerically the difference in computational cost between the standard implementation in (1) or (3) and the algorithm (8) in Theorem 2 for a range of sample sizes, $T$.

The baseline algorithm computes the convolution (3) directly, where the number of required arithmetic operations is of order $T^2$. The computation time is expected to be proportional to the number of arithmetic operations, and consequently also of order $T^2$. These timings are compared to algorithms of order $T \log T$ based on the fast Fourier transform. The algorithms differ in the number of arithmetic operations required and therefore the results should be independent of programming language. However, in practice this may not be true because of, e.g., different implementations of the fast Fourier transform. We present results in the three popular languages MATLAB (MathWorks, 2012), Ox (Doornik, 2007), and R (R Core Team, 2013) in order to exemplify the time gains in practical application.

MATLAB does not ship with a function for fractional differencing. Instead we have written the function `fracfilter` that computes the fractional difference by direct linear convolution through the MATLAB function `filter`. The function `fracdiff` uses the fast Fourier transform to compute the convolution as in Theorem 2, and the time series is padded with zeros such the total length of the series is a power of two. Ox, on the other hand, has the built-in function `diffpow` for fractional differencing, see the `arfima` package v1.04 by Doornik and Ooms (2003), and we use that as benchmark. The function for the fast Fourier transform in Ox automatically pads the input with sufficient zeros, and hence we do not need to do so in our code. Finally, R also has a package for analyzing fractionally differenced data, namely `fracdiff` by Maechler (2012) (not to be confused with our algorithm of the same name), which has the built-in function `diffseries` for fractional differencing. We compare that with our implementation, which again uses the fast Fourier transform and padding with zeros such that the length of the time series is a power of two. All these algorithms are presented in Listings 1, 2, and 3, for MATLAB, Ox, and R, respectively, and are downloadable from the authors' websites.

Listing 1: Matlab code

```
function [dx] = fracfilter(x, d)
    T = size(x, 1);
```

```
    k = (1:T-1)';
    b = [1;cumprod((k-d-1)./k)];
    dx = filter(b, 1, x);
end

function [dx] = fracdiff(x, d)
    T = size(x, 1);
    np2 = 2.^nextpow2(2*T-1);
    k = (1:T-1)';
    b = [1;cumprod((k-d-1)./k)];
    dx = ifft(fft(x, np2).*fft(b, np2));
    dx = dx(1:T, :);
end
```

Listing 2: Ox code

```
fracdiff(const x, const d)
{
    decl T, k, b, dx;
    T = rows(x);
    k = range(1, T-1)';
    b = 1|cumprod(((k-d-1)./k));
    dx = fft(cmul(fft(b'~zeros(1, T-1)), fft(x'~zeros(1, T-1))), 2);
    return dx[:T-1]';
}
```

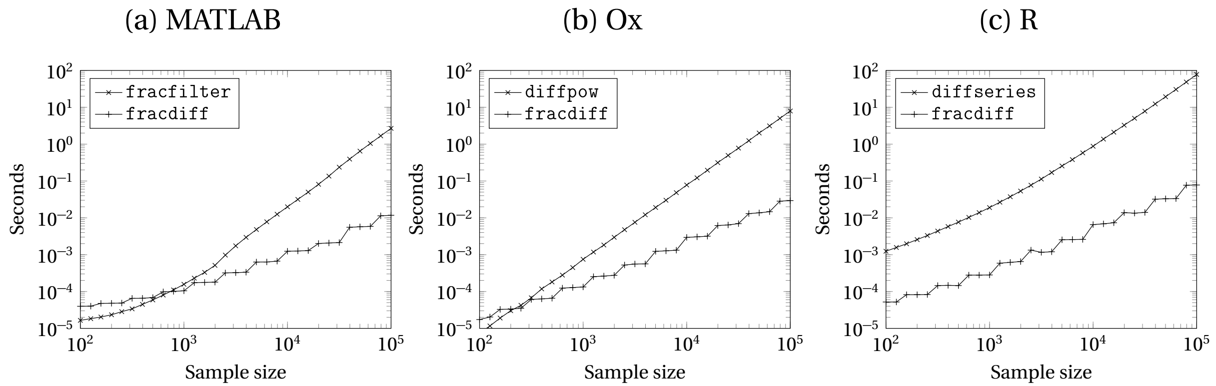
Listing 3: R code

```
fracdiff <- function(x, d){
  iT <- length(x)
  np2 <- nextn(2*iT - 1, 2)
  k <- 1:(iT-1)
  b <- c(1, cumprod((k - d - 1)/k))
  dx <- fft(fft(c(b, rep(0, np2-iT)))*
        fft(c(x, rep(0, np2-iT))), inverse=T)/np2;
  return(Re(dx[1:iT]))
}
```

The computations are run on a desktop with an Intel Core i5-2400 3.1GHz processor running Ubuntu 13.04. The software versions are MATLAB 2012b, Ox Professional 6.2, and R 3.0.0. The timings are computed for sample sizes ranging from 100 to 100,000. For each sample size the fractional differences of 10 random series are calculated and the fastest computing time of the 10 series is saved. We used the fastest computing time rather than the average time to clean the results from computing time used for background tasks, etc.[2] Results for the average computing time are nearly identical. The resulting computation times are plotted with logarithmic axes in Figure 1. This clearly

[2] Unlike MATLAB and Ox, the timing functions in R are not sufficiently accurate to allow for measurement of one fractional difference when the sample size is relatively small or the fast algorithm is used. Therefore, for $T < 3000$ the timings in R are actually the minima over 10 replications of the total time to calculate 100 fractional differences for `diffseries` and 1000 for `fracdiff`, divided by 100 and 1000, respectively. For $T \geq 3000$ we used 1 and 100 fractional differences for `diffseries` and `fracdiff`, respectively.

Figure 1: Computation times in seconds against sample size




Note: The figure displays computation times in seconds for a range of sample sizes. Panels (a), (b), and (c) show the timings for MATLAB, Ox, and R, respectively. In each panel, both axes are logarithmic.

Table 1: Examples of computing time

| | | Sample size | | | |
|---|---|---|---|---|---|
| | | 100 | 1,000 | 10,000 | 100,000 |
| MATLAB | `fracfilter` | 0.0165 | 0.157 | 19.95 | 2682.4 |
| | `fracdiff` | 0.0397 | 0.105 | 1.24 | 11.7 |
| Ox | `diffpow` | 0.0075 | 0.751 | 77.92 | 7943.1 |
| | `fracdiff` | 0.0174 | 0.133 | 2.95 | 29.4 |
| R | `diffseries` | 1.2400 | 19.010 | 878.00 | 77842.0 |
| | `fracdiff` | 0.0520 | 0.282 | 6.57 | 77.9 |

Note: Entries are computing times in milliseconds for the calculation of one fractional difference for a variety of sample sizes and for the algorithms given in Listings 1, 2, and 3 as well as the benchmark algorithms. The reported times are the fastest of 10 calculations (except for R, cf. footnote 2).

shows the different orders of the algorithms. The graphs for the benchmark algorithms are nearly straight lines with slope two except for the shortest samples. For our `fracdiff` algorithm, the graphs appear like step functions with jumps at each power of two, due to the application of the fast Fourier transform and the padding with zeros to a length of powers of two. Overall, Figure 1 clearly shows the advantage of the algorithm in Theorem 2 in terms of computation speed, especially when recalling that the axes are logarithmic.

In Table 1 we give some examples of the actual computing time in milliseconds required to calculate one fractional difference for sample sizes $T = 100, 1000, 10000, 100000$ using both the standard implementations as well as our fast fractional difference algorithm, `fracdiff`, in Listings 1, 2, and 3.

For samples of $T = 100$ the computation times are all very small, at least in MATLAB and Ox, and even though the new `fracdiff` algorithm is actually slower than the bench-

mark in MATLAB and Ox, they are all very fast and in practice there will hardly be any noticable difference between the implementations. Already for samples of 1,000 there is a substantial difference in favor of our algorithm, especially for Ox and R. For $T = 10,000$ and $T = 100,000$ the differences in computation times are enormous. For $T = 100,000$, our proposed `fracdiff` algorithm is about 230 times faster in MATLAB, about 270 times faster in Ox, and about 1000 times faster in R compared to the baseline algorithms.

## 4 Discussion and conclusions

In this paper we have provided a fast algorithm for calculating the fractional difference of a time series based on the circular convolution theorem and the fast Fourier transform. The required number of arithmetic operations for our algorithm is of order $T \log T$ compared to $T^2$ for standard implementations, and similarly for the computation time. For large sample sizes, the difference in computation time is very substantial and can easily be the difference between feasible and infeasible estimation with moderate to large sample sizes. Moreover, the much faster calculation of the fractional difference achieved by our algorithm opens up new possibilities for bootstrap or simulation methods to be applied to fractional time series models with moderate to large sample sizes.

Of course, large data sets are common in many fields such as meteorology and finance. For example, in Carlini, Manzoni, and Mosconi (2010) and Bollerslev, Osterreider, Sizova, and Tauchen (2013), the authors apply the fractional cointegration model of Johansen and Nielsen (2012a) to large data sets in finance. More specifically, Carlini *et al.* (2010) analyze supply and demand imbalances on stock prices using high-frequency observations. In the estimation, the authors use only a small subset of $T = 110,000$ observations from a data set with a total of $T = 5.8$ million observations, citing the "extreme computational burden" of the estimation. Indeed, extrapolating from Table 1, the time (in MATLAB) required to compute just one fractional difference with the standard implementation with $T = 5.8$ million would be approximately 2.7 seconds times $(5,800,000/100,000)^2 = 58^2$, which is roughly 2.5 hours. On the other hand, with our algorithm, the same calculation of one fractional difference would take only 0.012 seconds times $58 \log(5,800,000)/\log(100,000)$, which is 0.94 second. Thus, the computation time of our algorithm is over 9500 times faster, which we conjecture is sufficiently fast to allow estimation with the full sample.

In a related strand of literature on the so-called "type I" fractional processes, there has been some focus on efficient simulation of fractional processes. An early algorithm by Davies and Harte (1987), see also Craigmile (2003), Doornik (2006), and Chen, Hurvich, and Lu (2006), applies the circulant embedding method to show that, when a time series of length $T$ is to be generated with autocovariances $\gamma_0, \gamma_1, \dots, \gamma_{T-1}$, the fast Fourier transform can be used to simulate the time series such that the operation is of order $T \log T$. However, this method and related methods require a non-negativity condition on the Fourier coefficients of the autocovariance function and they also require that $d < 1/2$ such that the autocovariances are in fact well-defined. Also, an idea similar to our Theorem 1 appears in Sowell (1992, p. 170), using the continuous rather than the discrete Fourier transform, as an approximation device for the untruncated fractional difference operator for type I processes.

Finally, depending on the particular fast Fourier transform algorithm that is being

used, we note that further (although modest) decreases in computation time can be achieved by padding the series such that the total length is a product of small prime numbers, i.e. $2^k 3^m 5^n$. In the calculation of the fast Fourier transform, this prevents excessive padding when $2T-1$ is slightly greater than a power of two. Indeed, in unreported MATLAB calculations, we found that additional (modest) decreases in computation time were attained with this procedure compared to the `fracdiff` algorithm in Listing 1 above. This alternative procedure had the further effect of generating a "smoother" line compared to the step function-type line in Figure 1(a). However, to keep focus on the (fast) calculation of the fractional difference, rather than efficient calculation of the fast Fourier transform, we have not included an in-depth discussion of the benefits of padding the series to a length given as a product of small primes.

## 5 List of references


1. Bollerslev, T., D. Osterreider, N. Sizova, and G. Tauchen (2013). Risk and return: long-run relationships, fractional cointegration, and return predictability. *Journal of Financial Economics* 108, 409–424.
2. Carlini, F., M. Manzoni, and R. Mosconi (2010). The impact of supply and demand imbalance on stock prices: An analysis based on fractional cointegration using Borsa Italiana ultra high frequency data. Working paper, Politecnico di Milano.
3. Chen, W.W., C.M. Hurvich, and Y. Lu (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.
4. Cooley, J.W., P.A.W. Lewis, and P.D. Welch (1969). The fast Fourier transform and its applications. *IEEE Transactions on Education* 12, 27–34.
5. Cooley, J.W. and J.W. Tukey (1965). An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation* 19, 297–301.
6. Craigmile, P.F. (2003). Simulating a class of stationary Gaussian processes using the Davies-Harte algorithm, with application to long memory processes. *Journal of Time Series Analysis* 24, 505–511.
7. Davies, R.B. and D.S. Harte (1987). Tests for Hurst effect. *Biometrika* 74, 95–101.
8. Doornik, J.A. (2006). Efficient ARFIMA modelling when the sample size is large. Unpublished manuscript, University of Oxford.
9. Doornik, J.A. (2007). *Object-Oriented Matrix Programming Using Ox*, 3rd ed., Timberlake Consultants Press, London, England.
10. Doornik, J.A. and M. Ooms (2003). Computational aspects of maximum likelihood estimation of autoregressive fractionally integrated moving average models. *Computational Statistics and Data Analysis* 41, 333–348.
11. Johansen, S. and M.Ø. Nielsen (2012a). Likelihood inference for a fractionally cointegrated vector autoregressive model. *Econometrica* 80, 2667–2732.
12. Johansen, S. and M.Ø. Nielsen (2012b). The role of initial values in nonstationary fractional time series models. QED working paper 1300, Queen's University.
13. Maechler, M. (2012). The `fracdiff` package for R, version 1.4-2. URL: `http://cran.r-project.org/web/packages/fracdiff/`.
14. Marinucci, D. and P.M. Robinson (1999). Alternative forms of fractional Brownian motion. *Journal of Statistical Planning and Inference* 80, 111–122.

15. MathWorks (2012). *MATLAB 2012b*, The MathWorks, Inc., Natick, MA.
16. R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL: `http://www.R-project.org`.
17. Sowell, F. (1992). Maximum likelihood estimation of stationary univariate fractionally integrated time series models. *Journal of Econometrics* 53, 165–188.
18. Stockham, T.G. (1966). High-speed comvolution and correlation. *Proceedings of the Spring Joint Computer Conference* 28, 229–233.
19. Zygmund, A. (2003). *Trigonometric Series*, vol. I and II, 3rd rev. ed., Cambridge University Press, Cambridge, England.