

A Package for Estimating, Forecasting and Simulating Arfima Models: Arfima package 1.06 for Ox

BY JURGEN A. DOORNIK AND MARIUS OOMS

Nuffield College, Oxford OX1 1NF, UK, Free University, Amsterdam, The Netherlands,

December 19, 2012

Contents

1	Introduction	3
2	Disclaimer	3
3	Availability and Citation	4
4	Installation	4
5	Running Arfima programs	4
6	Some estimation and forecasting examples	4
7	Treatment of the mean	7
8	Arfima in OxPack	7
9	Some simulation examples	10
10	Arfima summary	12
11	Arfima member functions	14
12	ArfimaSim member functions	22
13	Changes from previous versions	24
14	Technical Summary	25
15	The Arfima model	25
	15.1 Autocovariance function	25
16	Estimation	26
	16.1 Regressors in mean	26
	16.2 Initial values	26
	16.3 Exact maximum likelihood (EML)	27
	16.4 Modified profile likelihood (MPL)	29
	16.5 Non-linear least squares (NLS)	29
	16.6 Variance-covariance matrix estimates	29
17	Estimation output	30
18	Estimation options	30
	18.1 Sample mean versus known mean	30
	18.2 Fixing parameters	30
	18.3 Weighted estimation	31
	18.4 Z variables	31
19	Forecasting	31
20	Some notes on computation	32
	20.1 Autocovariance function	32

A PACKAGE FOR ESTIMATING, FORECASTING AND SIMULATING ARFIMA MODELS: ARFIMA
PACKAGE 1.06 FOR OX 2

20.2	Likelihood evaluation	32
20.3	Invertibility of MA polynomial	33
21	Monte Carlo experimentation	33

1 Introduction

This documentation describes the `Arfima` package version 1.06 for Ox 7 or later, see [Doornik \(2013\)](#). The `Arfima` package has a class for estimation and forecasting of ARFIMA(p, d, q)¹ and ARMA(p, q) models.

The available estimation methods are exact maximum likelihood (EML), modified profile likelihood (MPL), and nonlinear least squares (NLS). The mean of the process the ARFIMA process can be a (nonlinear) function of regressors. This makes it straightforward to model (nonlinear) deterministic trends and additive outliers. Missing observations can easily be estimated.

Regressors can also be used to model the innovations of the process. This allows ARFIMA distributed lag modelling, an extension of autoregressive distributed lag (ARDL) modelling. Innovative outliers can be estimated.

We have managed to make the storage requirement of order T (see below), so that very large samples can be used without major problems.

The `Arfima` package is written in Ox, a fast object-oriented matrix programming language. The package is used by writing small Ox functions which create and use an `Arfima` object. Some knowledge of Ox is useful; although this new version of the package can be used interactively in conjunction with **OxPack** for **OxMetrics** (see [Doornik and Hendry, 2013](#)). `Arfima` users may also be interested in the OxMetrics version of X12arima ([Findley, Monsell, Bell, Otto, and Chen, 1998](#)), which allows estimation (EML and NLS) and forecasting of (seasonal) ARIMA models, see www.pcgive.com.

The `Arfima` class derives from the `Modelbase` class, which in turn derives from `DataBase`. The `DataBase` class admits simple loading of data sets in various formats and easy selection of variables and samples. The `Modelbase` class contains standard functions for the organisation of estimation input and the presentation of estimation output. An additional simulation class, `ArfimaSim`, allows Monte Carlo experimentation of the facilities in the `Arfima` class.

The organization of the documentation is as follows. After discussing installation we present 8 example programs in §6, which show the estimation and forecasting facilities of `Arfima`. Some of these features are also illustrated in §8 in the discussion of the corresponding interactive environment. Section 9 presents simulation programs that use `ArfimaSim`. Running the programs of this section will show the speed of the computations and the effectiveness of the modified profile likelihood for bias correction. Sections 10, 11 and 12 document the main functions of the classes `Arfima` and `ArfimaSim`. The remaining sections provide a summary of the implementation details of the procedures in the `Arfima` package, complementing the exposition in [Ooms and Doornik \(1998\)](#). The notation necessary for understanding the output of the sample programs is presented in §15.

We discussed computational aspects in [Doornik and Ooms \(2003\)](#), while inference and forecasting within an empirical example was considered in [Doornik and Ooms \(2004\)](#).

2 Disclaimer

This package is functional, but no warranty is given whatsoever. The most appropriate forum to discuss problems and issues related to the `Arfima` package is the `ox-users` discussion group (see www.mailbase.ac.uk/lists/ox-users). Please report suggestions for improvement to Marius Ooms at ooms@econometriclinks.com.

¹Part of the underlying code is a rewritten version of the Fortran procedure by Falaw Sowell (see [Sowell, 1992](#)), we thank him for permission to use his code. Without the original code, writing the current set of procedures would have been much harder.

3 Availability and Citation

The Arfima package is available for downloading through <http://www.doornik.com/download.html>. The Unix DLLs must be downloaded separately.

To facilitate replication and validation of empirical findings, cite this documentation and [Doornik and Ooms \(2004\)](#) in all reports and publications involving the application of the Arfima package.

4 Installation

1. Make sure you have properly installed Ox version 4.00 or later. The Arfima package does not work fully with earlier versions of Ox. Type `oxl` at the command prompt to check.
2. Create an `arfima` subdirectory in the `ox\packages` folder and put `arfima.zip` in that subdirectory, then `unzip arfima.zip`.
3. Read the `read.me` file for info on the last updates.
4. If Ox has been installed properly, this will allow using the Arfima package from any directory. To use the package in your code, add the command

```
#import <packages/arfima/arfima>
```

at the top of all files which require it.

5 Running Arfima programs

The package has a section of code in a dynamic link library for optimum speed. This version can be run under Windows and most Unix versions.

To run the examples in §6 under Windows/Unix, type

```
oxl fracest1
```

at the command prompt. Alternatively, use `oxrun` to run the §6 program. `OxRun` requires `OxMetrics` to show the output and graphs on screen.

To run the programs without using the DLL (any platform; this is at least three times slower) use:

```
oxl fracest1 -DNO_DLL
```

The `-DNO_DLL` version is so slow because it contains a literal translation of the Fortran code to Ox; no attempt has been made to vectorize this.

6 Some estimation and forecasting examples

This section discusses eight example programs provided as `fracest1.ox`, ..., `fracest8.ox` in the standard installation. The first three programs demonstrate the selection of dataset, model orders, estimation sample, forecast horizon, model restrictions, regressors, and estimation method. The fourth program illustrates forecasting extensions. The fifth program shows the use of popular semiparametric estimates of d . Programs 6 and 7 make clear how to perform nonlinear regression with ARFIMA-disturbances. `fracest8.ox` shows how to deal with innovative outliers and additive outliers.

The code below (provided as `fracest1.ox`) estimates an ARIMA(1, d , 1) model on the `OxMetrics` data set `rpi_uk.in7` (UK retail price index). Some possible changes to this code are shown in the following programs.

```

                                                                    fracest1
#include <oxstd.h>
#include <oxfloat.h>    // required for M_NAN
#import <packages/arfima/arfima>

main()
{
    decl arfima, dly;
    // create an object of class Arfima
    arfima = new Arfima();

    // load the data file
    arfima.LoadIn7("rpi_uk.in7");
    // translate RPI into inflation (delta log RPI)
    // setting first value to missing value
    dly = diff0(log(arfima.GetVar("RPI_UK")), 1, M_NAN);
    // store in database
    arfima.Append(dly, "Inflat", 0);
    arfima.Info();

    // formulate arfima model, select "Y" as Y_VAR
    // from lag 0 to lag 0 (i.e. current only)
    arfima.Select(Y_VAR, { "Inflat", 0, 0 } );
    // specify an ARMA(0,d,0) model, estimate by exact ML
    arfima.ARMA(0,0);
    arfima.SetMethod(M_MAXLIK);
    arfima.UseSampleMean();
    // select the maximum sample period
    arfima.SetSelSample(-1, 1, -1, 1);

    // print compact iteration output every iteration
    MaxControl(-1,1,1);

    // estimate, automatically prints the results
    println("\nIterating:");
    arfima.Estimate();

    // done with arfima: delete the object
    delete arfima;
}
    
```

Which generates output:

```

                                                                    fracest1.out
Arfima package version 1.01, object created on 20-06-2001

---- Database information ----
Sample:    1955 (1) - 1994 (4) (160 observations)
Frequency: 4
Variables: 2

Variable    #obs  #miss    min      mean      max      std.dev
RPI_UK      160    0       11.3     56.563    153.8     47.409
Inflat      159    1   -0.0082305  0.01642  0.090573  0.015168
    
```

```

----- Maximum likelihood estimation of ARFIMA(0,d,0) model -----
Iterating:
it0  f=      4.420666 df=    0.07155 e1=    0.1002 e2=  0.005111 step=1
it1  f=      4.421729 df=    0.01305 e1=    0.01780 e2=  0.0004044 step=0.5
it2  f=      4.421764 df=  0.0003887 e1=  0.0005324 e2=  0.0004028 step=1
it3  f=      4.421764 df=2.442e-006 e1=3.345e-006 e2=1.165e-005 step=1
Strong convergence

The estimation sample is: 1955 (2) - 1994 (4)
The dependent variable is: Inflat
(in deviation from sample mean)

          Coefficient  Std.Error  t-value  t-prob
d parameter          0.369581    0.05098    7.25    0.000

log-likelihood      477.449176
no. of observations      159  no. of parameters          2
AIC.T                -950.898353  AIC                -5.98049279
mean(Inflat)         0.0164204  var(Inflat)         0.000230057
sigma                 0.011965  sigma^2             0.00014316

BFGS using numerical derivatives (eps1=0.0001; eps2=0.005):
Strong convergence
Used starting values:
0.40000
    
```

The Arfima package allows for fixing parameters, as well as forecasting. The following section of code, from `fracest2.ox`, illustrates:

```

arfima.ARMA(4,0);
arfima.FixAR(<1:3>); // omit AR1..AR3 terms in AR polynomial

arfima.SetSelSample(-1, 1, 1993, 4); // keep 1 year
arfima.UseSampleMean();
arfima.Estimate();
arfima.Forecast(8); // 4 in sample, 4 out-of-sample
    
```

In addition, regressors can be added to the model specification, and estimation method changed. The program `fracest3.ox` experiments with these, for example adding a constant term and estimating by modified profile likelihood:

```

arfima.Select(Y_VAR, { "Inflat", 0, 0 } );
arfima.Select(X_VAR, { "Constant", 0, 0 } );
arfima.SetSelSample(-1, 1, -1, 1);
arfima.SetMethod(M_MAXMLIK);
arfima.Estimate();
    
```

The estimation method can be switched to non-linear least squares. By default, the package obtains starting values using the methods set out in §16.2. After model formulation and sample selection, you can specify your own starting values as follows:

```

arfima.SetStartPar(<-0.1,0.1>); // order is: d, AR1
arfima.Estimate();
    
```

The argument to `SetStartPar` is a vector, with an entry for each *free* coefficient.

The sample program `fracest4.ox` compares exact maximum likelihood (EML) with modified profile likelihood (MPL), and illustrates forecasting issues:

- Producing log-level forecasts from a second differenced dependent variable, this is also shown graphically in §8 below;

- Producing level forecasts from log-level forecasts. Note the extra forecast bias correction due to the data transformation, see e.g. [Granger and Newbold \(1986, p.311\)](#).

`fracest5.ox` applies two semiparametric methods for inference on the order of integration d : log periodogram regression using `EstimateGPH` and Gaussian semiparametric estimation with `EstimateGSP`.

It is somewhat more involved to estimate an ARFIMA model where the mean is a general function of regressors: `fracest6.ox` shows how to override the virtual functions for the mean function in-sample and out-of-sample by a simple linear time trend. The code in `fracest7.ox` extends this example to estimate and forecast a smooth logistic function of time. [Eisinga, Franses, and Ooms \(1999\)](#) recently used a logistic trend with stationary ARFIMA-disturbances to model and test convergence in opinion polls.

Finally, `fracest8.ox` illustrates that both weighted estimation and innovation dummies can be used to downweight the effect of outliers, and compares these approaches with the estimation of additive dummies. In order to do so, the program uses two extra variable types, `W_VAR` and `Z_VAR`, in addition to the usual `Y_VAR` and `X_VAR`. See also §18.3 and §18.4 below.

7 Treatment of the mean

There are several reasons to choose different ways to estimate the overall mean of a stationary ARFIMA-proces, see §18.1 below. Therefore the `Arfima` package implements three ways to allow for the mean component:

1. Use a constant term as a regressor.

```
arfima.Deterministic(FALSE); // create Constant in database
arfima.Select(X_VAR, { "Constant", 0, 0 } ); // add to model
arfima.FixMean(0); // no mean adjustment (the default)
```

The last statement fixes the mean at zero, which implies that no mean adjustment is made (this is done through the constant term instead). `FixMean(0)` is the default behaviour, so the line could be omitted.

2. Estimate in deviation from sample mean.

```
arfima.UseSampleMean();
```

In this case, the dependent variable is demeaned prior to estimation. This is noted in the output by the addition of (*in deviation from sample mean*) after the name of the dependent variable (see the output in §6 above).

3. Impose a known mean (which could be zero).

```
arfima.FixMean(1.6);
```

If a non-zero value is used, this is noted in the output: (*in deviation from imposed mean 1.6*). The default is `FixMean(0)`.

Under MS Windows operating systems it is possible to experiment interactively with many of the options discussed in the previous sections. This is done in the `Arfima` package in `OxPack`, which requires `Ox Professional`.

8 Arfima in OxPack

`Arfima` in `OxPack` has an effective graphical user interface for interactive data and model selection, estimation, forecasting, diagnostics and testing.

Installing Arfima in OxPack

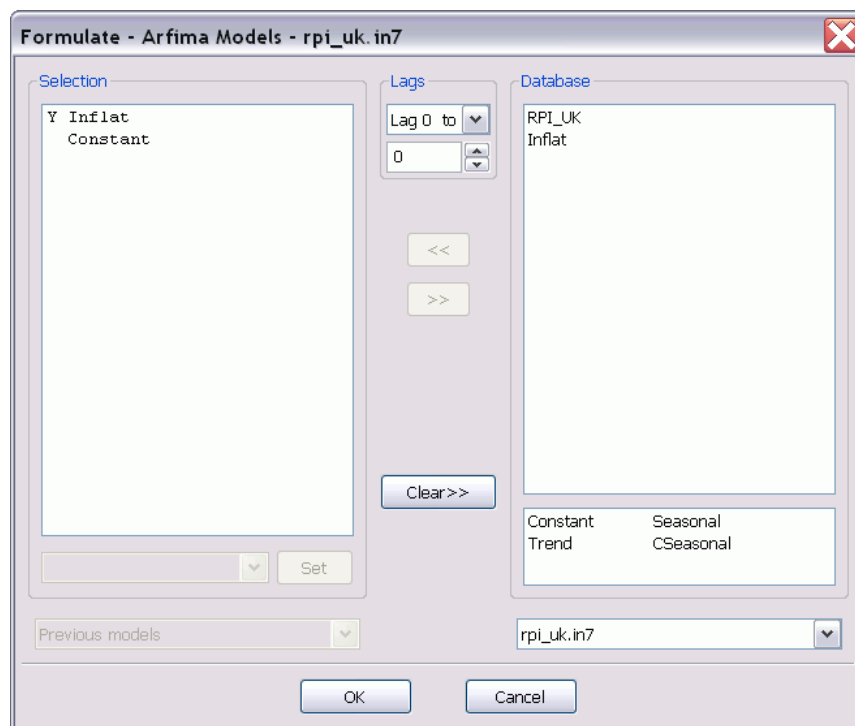
Installation of the interactive version of Arfima:

1. Install Arfima into `ox/packages/arfima` as described above.
2. Start OxMetrics, and then OxPack from the OxMetrics Run menu or the workspace. From the OxPack Package menu Choose Add/Remove Package. Locate `arfima.oxo` (in the `arfima` folder) using the Browse button, enter `Arfima` as the class name and press Add.

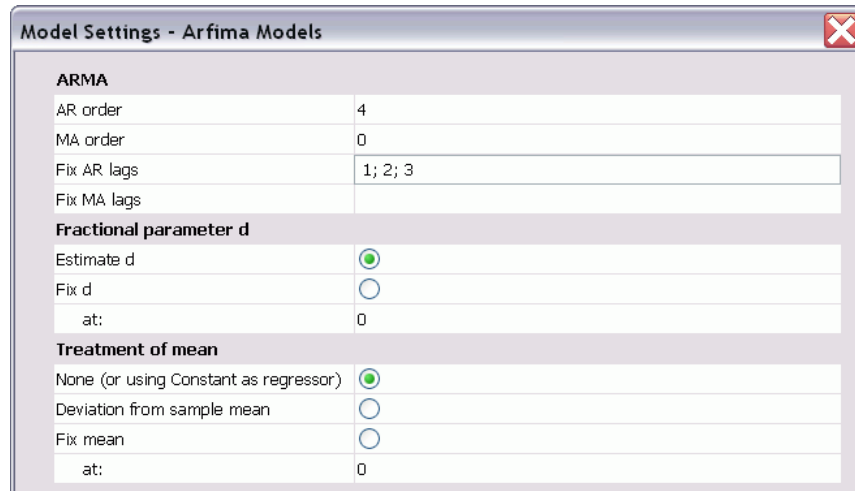
Sample session using Arfima in OxPack

OxPack is now ready to use Arfima. As an example, we use again `rpi_uk.in7` which is also in the `arfima` folder:

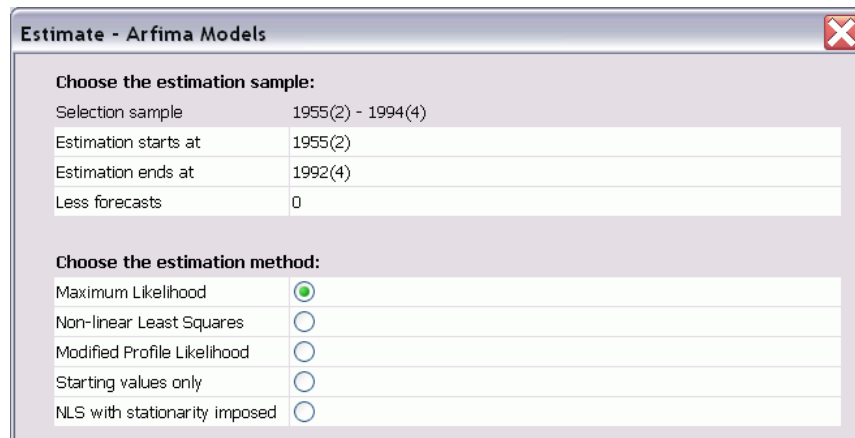
1. Load the data `rpi_uk.in7` in OxMetrics.
2. From the OxPack Package menu choose Arfima. The title bar of the OxPack window shows Arfima is loaded and the message Arfima package version 1.05, object created on ... is displayed in the GiveWin Results window.
3. Create *Inflat* as $\Delta \log(\text{RPI_UK})$ with the OxMetrics Calculator. This is done in two obvious steps. Compare also `fracest4.ox`. Alternatively, use the supplied Algebra file `rpi_uk.alg`.
4. From the OxPack Model menu choose Formulate, and select *Inflat* and a *Constant*:



5. Specify the model as an AR(4), but restrict lags 1 – 3 to zero:



6. In the estimation dialog, choose maximum likelihood (for example), and reduce the sample by two years:



7. Estimation is nearly instantaneous. Various options are available in the Test menu. For example, Graphic Analysis presents the following diagnostic output:

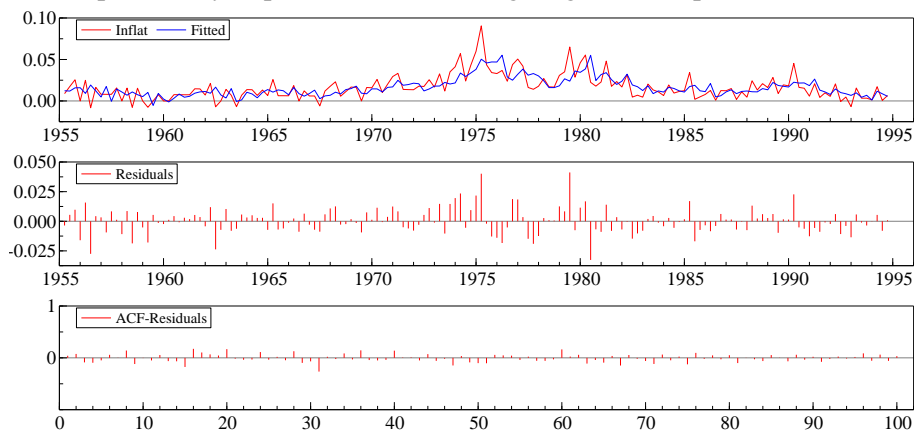
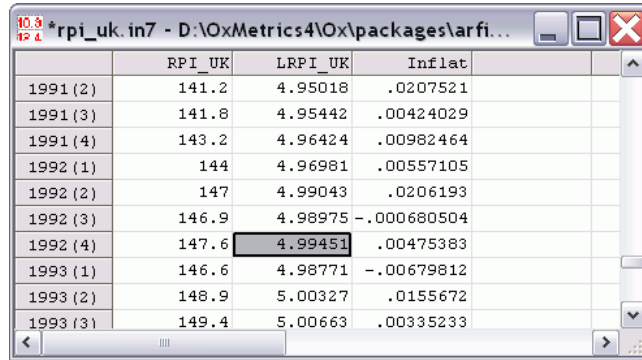


Figure 1 Graphic Analysis

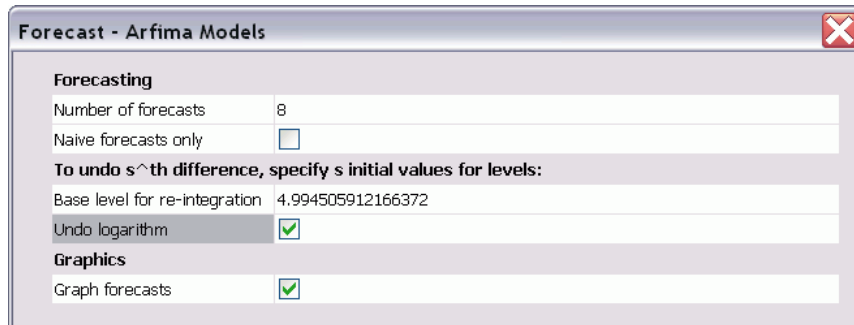
8. Finally, we consider forecasting. From the Test menu, select Forecasting, choosing 8 forecasts. The dependent variable is $i = \Delta \log(P)$, but it is possible to see the forecasts in terms of P . To

re-integrate i , Arfima needs to know the base level from which to start. Since the last observation used in estimation was 1992(4), the start is $\log(P)_{1992(4)}$. Copy this from the database (select the observation by dragging the left mouse button in the cell, copy, e.g. using the right mouse button context-menu):



	RPI_UK	LRPI_UK	Inflat
1991 (2)	141.2	4.95018	.0207521
1991 (3)	141.8	4.95442	.00424029
1991 (4)	143.2	4.96424	.00982464
1992 (1)	144	4.96981	.00557105
1992 (2)	147	4.99043	.0206193
1992 (3)	146.9	4.98975	-.000680504
1992 (4)	147.6	4.99451	.00475383
1993 (1)	146.6	4.98771	-.00679812
1993 (2)	148.9	5.00327	.0155672
1993 (3)	149.4	5.00663	.00335233

and paste it (right click on the field, and select Paste) as the base value. Also select ‘undo logarithm’, and graphics:



Forecast - Arfima Models

Forecasting

Number of forecasts: 8

Naive forecasts only:

To undo s^h difference, specify s initial values for levels:

Base level for re-integration: 4.994505912166372

Undo logarithm:

Graphics

Graph forecasts:

This results in three graphs. Note that the last graph does not have the normal statistical interpretation, see the discussion of `fracest4.ox` in §6 above.

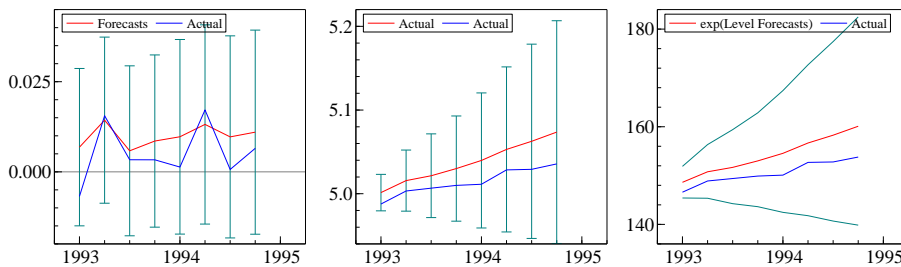


Figure 2 Forecasts

9 Some simulation examples

The following simulation programs² use the Arfima simulation class `ArfimaSim`, which derives from the base simulation class `Simulation`. The main (constructor) function `ArfimaSim()` sets up a sim-

²The simulation results were obtained with Ox 3. Ox 4 outcomes will differ somewhat because a different default random number generator is used.

ulation experiment, which is determined by the selection of a data generating process, an estimator and a (set of) tests. More details are given in §12. Running these programs shows that online evaluation of the estimation methods is now possible. Also see Hauser (1997) for extensive Monte Carlo results for ARFIMA models.

We provide the following sample programs (also available in the `arfima` folder):

- `fracsim1.ox` simulates an ARFIMA(0, -0.3, 0) with $M = 1000, T = 100$. First the mean is fixed at zero, then estimation is in deviation from sample mean. In the first case the bias is about -1.2%, in the second -3.3%.
- `fracsim2.ox` simulates an ARFIMA(0, d , 0).
- `fracsim3.ox` simulates various AR(1) and MA(1) processes.
- `fracsim4.ox` extends `fracsim1` by adding AR and MA parameters. It also does exact maximum likelihood (EML) with a constant, and modified profile likelihood (MPL) with a constant. The biases, as a percentage are ($M = 250, T = 100$):

	$\phi = 0.7$		$\phi = 0.2$		$\phi = -0.3$		$\phi = -0.8$	
	d ,	ϕ	d ,	ϕ	d ,	ϕ	d ,	ϕ
Estimating in deviation from sample mean, EML								
$d = -0.3$	-9.7,	3.4	-14.1,	10.0	-6.0,	3.6	-3.0,	2.3
$d = 0$	-12.4,	5.2	-26.9,	21.4	-8.4,	5.3	-5.8,	2.9
$d = 0.3$	-16.8,	8.5	-33.3,	27.5	-12.5,	9.1	-7.1,	3.3
No mean adjustment (known mean 0), EML								
$d = -0.3$	-2.2,	-2.2	-5.5,	2.7	-3.0,	1.9	-2.2,	2.1
$d = 0$	-3.4,	-1.4	-9.2,	6.0	-3.1,	2.0	-2.3,	2.2
$d = 0.3$	-7.5,	2.4	-10.6,	7.7	-3.7,	2.4	-2.8,	2.3
Constant term in model, EML								
$d = -0.3$	-10.2,	3.6	-18.4,	13.8	-7.8,	4.7	-5.8,	2.9
$d = 0$	-12.4,	5.2	-29.3,	23.3	-8.2,	5.0	-6.1,	3.0
$d = 0.3$	-17.0,	8.6	-32.9,	27.1	-12.5,	9.2	-7.2,	3.3
Constant term in model, MPL								
$d = -0.3$	-1.0,	-2.0	-4.7,	1.6	-2.8,	1.6	-2.3,	2.1
$d = 0$	-2.8,	-0.7	-5.9,	2.6	-2.7,	1.5	-2.3,	2.1
$d = 0.3$	-10.1,	5.2	-11.8,	8.7	-4.0,	2.7	-2.6,	2.2

MPL clearly improves on EML, with biases comparable to EML with known constant. MPL has the highest number of rejected experiments: in most cases less than 2%. The notable exceptions are MPL($d = 0.3, \phi = 0.7$): 30% rejections, and MPL($d = 0.3, \phi = 0.3$): 20% rejections.

- `fracsim_anbl.ox` replicates the experiment of An and Bloomfield (1993), comparing the bias of EML and MPL. Run this with OxRun to see the graphs, for example from the first two experiments. Again, MPL does well in correcting the bias and provides reliable inference on d .

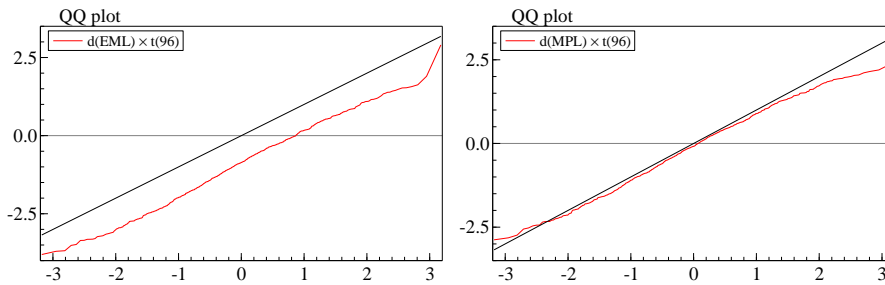


Figure 3 Simulation

10 Arfima summary

The Arfima class derives from Modelbase, which in turn derives from Database. Some of the functions below are in the base class (marked with *) or override a base-class virtual function (marked with +), but documented because they will be commonly used when estimating an ARFIMA model. Consult the header file arfima.h for definitions of member variables, and undocumented functions, such as those for communication with OxPack.

Constructor

Arfima Constructor

Model formulation

ARMA	Specify the AR and MA orders
DeSelect *	Removes the model selection
FixAR	Fix AR orders at 0
FixD	Fix d parameter at a specified value
FixMA	Fix MA orders at 0
FixMean	Use to set known mean (0 is the default)
FreeARMA	Estimate all ARMA parameters freely (default)
FreeD	Estimate d freely (default)
Select *	Selects a variable into the model
SetMethod *	Sets the estimation method
SetPrint *	Switches printing on or off (default is on)
SetSelSample *	Sets the estimation sample
UseSampleMean	Estimate in deviation from the sample mean

Model estimation

Estimate *	Estimate the model
SetStartPar +	Specify starting values (overrides the default)

Post estimation

GetD	Returns estimated value of d
GetFreePar *	Returns the current values of the free parameters
GetMean	Returns the fixed mean or the sample mean
GetNaiveResiduals	Get naive residuals (uses NLS filter)
GetPar *	Returns the current values of all parameters
GetResiduals +	Get the residuals
GetResult *	Gets the return code from MaxBFGS
GetSigma2	Gets the residual variance

TestGraphicAnalysis + Plots the graphic analysis

TestSummary + Prints a test summary

Forecasting

Forecast Get forecasts

SetFreePar * Sets the values of the free parameters (when not estimating)

SetSigma2 Sets a value for the residual variance (when not estimating)

TestForecastGraphics + Plots forecasts (after Forecast)

General

Acf ARFIMA ACF

EstimateGPH Estimate d using the log-periodogram method

EstimateGSP Estimate d using the Gaussian semi-parametric method

SolveAR Solve the Yule-Walker equations for AR values

SolveMA Solve for MA values using the Tunnicliffe-Wilson method

Filter Applies the ARFIMA filter to a variable

FilterNaive Applies the naive (NLS) filter to a variable

Custom versions

GetX... Used to customize the X-regressor components

GetZ... Used to customize the Z-regressor components

11 Arfima member functions

This section documents the main member functions of `Arfima` and the base class `Modelbase` in alphabetical order.

Arfima::Acf

```
static Acf(const cT, const vP, const cAR, const cMA,
           const bPrintErr);
    cT          in: int, number of observations,  $T$ 
    vP          in:  $s$ -vector with coefficients in order  $d$ , AR,
                MA,  $s \geq 1 + p + q$ 
    cAR        in: int, number of AR parameters,  $p$ 
    cMA        in: int, number of MA parameters,  $q$ 
    bPrintErr   in: int, TRUE: print fracsigma error code if an
                error occurs (only when  $d \neq 0$ )
```

Return value

Returns the $1 \times T$ autocorrelation function corresponding to the specified ARFIMA(p, d, q) model.

Description

The possible error codes are:

- 1: $|d| > 5$ (allowing up to 5 for maximization, but note that $d \geq 0.5$ will cause problems);
- 4: $|\rho_i| \geq 0.9999$, AR root inside the unit circle;
- 6: failed to find roots of AR polynomial.
- 7: there are identical roots.

Arfima::Arfima

```
Arfima();
```

No return value.

Description

Constructor function.

Arfima::ARMA

```
ARMA(const cAR, const cMA);
    cAR      in: int, no of AR paramaters,  $p$ 
    cMA      in: int, no of MA paramaters,  $q$ 
```

No return value.

Description

Formulates the length of the AR and MA polynomial, the default is an ARIMA(0, d , 0) model. To fix specific parameters, use `FixAR`, `FixMA`. To omit the fractional part use `FixD(0)`.

Modelbase::Estimate

```
Estimate();
```

No return value.

Description

Finds starting values and estimates the formulated model. Prints the results, unless this is switched off by `SetPrint`.

Use `SetSelSample` to select an estimation sample and `Select` to select a dependent variable (`Y_VAR`) or regressors (`X_VAR`).

Calls `InitData` and `InitPar` if necessary.

Arfima::EstimateGPH, Arfima::EstimateGSP

```
static EstimateGPH(const mY, const iTrunc, const fPrint);
static EstimateGSP(const mY, const iTrunc, const fPrint);
    mY          in:  $T \times 1$  matrix, dependent variable observation in time
                domain
    iTrunc       in: int, truncation parameter in the frequency domain
                number of low frequency periodogram points used in
                estimation
    fPrint       in: int, TRUE: print results
```

Return value

Returns a 1×3 vector with \hat{d} , $SE(\hat{d})$, and the p -value for two-sided testing of $d = 0$.

Description

Periodogram points are evaluated at Fourier frequencies $\frac{2\pi j}{T}$, $j = 1, \dots, iTrunc$. In the notation of [Robinson \(1995b\)](#), c.f. [Beran \(1994, §4.6\)](#), and [Robinson \(1995a\)](#), $n = T$, $m = iTrunc$, $l = 1$. `EstimateGPH` implements the log-periodogram regression method for estimating d as discussed in [Geweke and Porter-Hudak \(1983\)](#). Zero periodogram points are omitted, see [Ooms and Hassler \(1997\)](#).

`EstimateGSP` implements the Gaussian semi-parametric method for estimating d as discussed in [Robinson and Henry \(1998\)](#).

Arfima::FixAR, Arfima::FixD, Arfima::FixMA

```
FixAR(const iOrder);
FixD(const dD);
FixMA(const iOrder);
    iOrder      in: int, index of AR or MA orders to set to 0; or matrix with
                orders to omit
    dD          in: double, value of  $d$ 
```

No return value.

Description

These functions fix certain parameters. The value of d can be fixed at zero or another value. To fix the first AR parameter (on y_{t-1}) at zero use `FixAR(1)`, similarly `FixMA(1)` for the first MA parameter, etc. The last AR parameter may not be fixed (this would result in an inverse root of infinity and computational problems in `Acf`, see [§20.1](#)).

Arfima::FixMean

```
FixMean(const dYmean);
```

`dYmean` in: double, value for fixed mean

No return value.

Description

Used to set a known mean. The default is `FixMean(0)`, also see §7.

`UseSampleMean` is available to estimate in deviation from the sample mean.

Arfima::Forecast

`Forecast(const cTforc);`

`Forecast(const cTforc, const vYlevel);`

`Forecast(const cTforc, const vYlevel, const bNaiveOnly);`

`cTforc` in: int, number of forecasts

`vYlevel` in: (optional argument) $s \times 1$ matrix with initial values to integrate the forecasts to levels, use `<>` to omit level forecasts (default: no level forecasts)

`bNaiveOnly` in: (optional argument) int, TRUE: only do naive forecasts (default: do both)

No return value.

Description

Prints the forecast results with `cTforc` forecasts after the estimation sample using the current parameter values (so not necessarily after estimation).

The second argument is used to integrate the forecasts back to the level. When $s = 1$, the forecasts are integrated once, when $s = 2$ twice, etc. (an example is in `fracest4.ox`).

Forecasting with Z variables is not yet implemented. Weights (W variable) are ignored during forecasting.

Arfima::FreeD, Arfima::FreeARMA

`FreeD();`

`FreeARMA();`

No return value.

Description

`FreeD` can be used to free the d parameter after using `FixD`. The default is to estimate d freely.

`FreeARMA` frees all ARMA coefficients after previous calls to `FixAR` and `FixMA`.

Arfima::GetD

`GetD();`

Return value

Returns the current value of d (the estimated value after `Estimate`).

Modelbase::GetFreePar

`GetFreePar();`

Return value

Returns a column-vector with the current value of the free parameters (the estimated values after Estimate).

Description

The parameters are ordered as follows: d , AR parameters, MA parameters, parameters on regressors. Any fixed parameters are omitted from the returned value.

Arfima::GetMean

GetMean();

Return value

Returns the mean of the dependent variable: either the fixed mean (set in FixMean) or the sample mean (when using UseSampleMean).

Arfima::GetNaiveResiduals

GetNaiveResiduals();

Return value

Returns the residuals using the current parameters, applying the naive (NLS) filter (regardless of the estimation method). After weighted estimation (W variable) the weighted residuals are returned.

Modelbase::GetPar,

GetPar();

Return value

Returns a column-vector with the current value of all coefficients, including the fixed coefficients.

Description

The parameters are ordered as follows: d , AR parameters, MA parameters, parameters on regressors. Fixed parameters are included in the returned value.

Arfima::GetResiduals

GetResiduals();

Return value

Returns the residuals from the estimated model. After weighted estimation (W variable) the weighted residuals are returned.

Modelbase::GetResult

GetResult();

Return value

Returns the return code from MaxBFGS.

Arfima::GetSigma2

GetSigma2();

Return value

Returns the estimated residual variance.

Arfima::GetXBeta, Arfima::GetX...

```
virtual GetXNames();
virtual GetXBeta(const vP);
virtual GetXBetaForc(const vP, const cTforc);
virtual GetXBetaStart(const mY);
virtual GetXPartial();
virtual GetXSizeInit();
    vP          in:  $s \times 1$  matrix with all coefficients, in order:
                  d,AR,MA,X,Z
    mY          in:  $T \times 1$  matrix, data variable to derive starting for
    cTforc      in: number of forecasts,  $H$ 
```

Return value

GetXNames returns an array of strings with the names of the X parameters.

GetXSizeInit returns an integer with the number of X parameters. This is called after the Modelbase member variables m_cX and m_mX are initialized, and the return value could increment m_cX. It is not necessary that the number of columns of the regressor matrix m_mX equals the number of parameters in the non-linear regression term, m_cX. After the call to GetXSizeInit, m_cX will be changed to the returned value.

GetXBetaStart is called from InitPar and should return a (column) vector of starting values for the m_cX X parameters.

GetXPartial returns TRUE if the X's should be partialled (concentrated) out of the likelihood (this will only work for linear $\mathbf{X}\beta$), see §16.3. The overridden version should return FALSE. (Note: when Z or W variables are present, X's are never partialled out.)

GetXBeta returns the $T \times 1$ matrix $\mathbf{f}(\mathbf{X}, \beta)$, which runs over the estimation sample determined by ModelBase members: m_iT1Est...m_iT2Est. The input argument vP is the full coefficient vector, with the X parameters located at indices 1+cAR+cMA : cAR+cMA+m_cX.

GetXBetaForc is as GetXBeta but for the forecast period: m_iT2Est+1...m_iT2Est+cTforc.

Description

These are virtual functions, which can be replaced in a derived class when a non-linear regressor term $\mathbf{f}(\mathbf{X}, \beta)$ is desired instead of just $\mathbf{X}\beta$.

Examples are in frcest6.ox and frcest7.ox.

Arfima::GetZGamma, Arfima::GetZ...

```
virtual GetZNames();
virtual GetZGamma(const vP);
virtual GetZGammaStart(const mY);
virtual GetZSizeInit();
```

Description

These are Z-variable analogues to the GetX... functions. The Z parameters are located at indices 1+cAR+cMA+m_cX : cAR+cMA+m_cX+m_cZ in the coefficient vector.

Modelbase::Select

```
Select(const iGroup, const aSel);
    iGroup    in: int, group indicator: Y_VAR or X_VAR
    aSel      in: array, specifying database name, start lag, end lag
```

No return value.

Description

Selects variables by name and with specified lags, and assigns the iGroup status to the selection. The aSel argument is an array consisting of sequences of three values: name, start lag, end lag. Some examples:

```
// select CONS from lag 0 to 0 as dependent variable, and a Constant
// as regressor (use Deterministic to create the Constant):
Select(Y_VAR, {"CONS", 0, 0});
Select(X_VAR, {"Constant", 0, 0});

// select CONS as dependent variable and
// select CONS lagged, INC and INC lagged as regressor:
DeSelect();
Select(Y_VAR, {"CONS", 0, 1});
Select(X_VAR, {"INC", 0, 1});
```

Each Select adds to the current selection. Use DeSelect to start afresh. *Note:* Select also requires a SetSelSample afterwards.

Two additional types of variable are available: W_VAR to do weighted estimation (see §18.3), and Z_VAR to add Z variables (see §18.4). Example fracest8.ox illustrates that an impulse dummy as Z variable equals the inverted dummy as W variable (but the dummy as X is different).

Modelbase::SetFreePar

```
SetFreePar(const vParFree);
    vParFree    in: int, vector with values for the free coefficients
```

No return value.

Description

Sets the free coefficients.

The parameters are ordered as follows: *d*, AR parameters, MA parameters, parameters on regressors. Any fixed parameters should be omitted from vParFree.

Modelbase::SetMethod

```
SetMethod(const iMethod);
    iMethod    in: int, one of:
                M_MAXLIK – exact maximum likelihood,
                M-NLS – non-linear least squares,
                M_MAXMPLIK – modified profile maximum likelihood,
                M_INITONLY – starting values only,
                M-NLS_STATIONARY – NLS, non-stationarity imposed.
```

No return value.

Description

The default estimation is maximum likelihood (M_MAXLIK). Use `SetMethod` to switch estimation method.

Modelbase::SetPrint

```
SetPrint(fPrint);  
    fPrint    in:  int, TRUE or FALSE
```

No return value.

Description

Switches printing on (TRUE) or off (FALSE). By default printing is on, but for simulations it must be switched off.

Modelbase::SetSelSample

```
SetSelSample(const iYear1, const iPeriod1, const iYear2,  
             const iPeriod2);
```

Description

See under the Database class.
Each `Select` adds to the current selection. Use `DeSelect` to start afresh.

Arfima::SetSigma2

```
SetSigma2(const dSigma);  
    dSigma    in:  double, residual variance,  $\hat{\sigma}_\epsilon^2$ 
```

No return value.

Description

This function is only useful when forecasting without estimating. Then, after the model has been formulated, the coefficients can be set with `SetFreePar` or `SetStartPar`, and the residual variance with `SetSigma2`.

Modelbase::SetStartPar

```
SetStartPar(const vP);  
    vP        in:  int, vector with values for the free parameters
```

No return value.

Description

Sets starting values for the free parameters (so excluding those which are fixed). Calls `InitData` if necessary, so the model must be formulated, and the sample selected before this function can be used.

`InitPar` is automatically called if `SetStartPar` is not used.

Arfima::TestForecastGraphics, Arfima::TestGraphicAnalysis, Arfima::TestSummary

```
virtual TestForecastGraphics(const bExp);  
virtual TestGraphicAnalysis();  
virtual TestSummary();
```

`bExp` `in`: int, TRUE: also take exponentials

No return value.

Description

`TestSummary` prints a test summary. Because these are computed from the residuals, they are perhaps better interpreted as descriptive statistics:

- Normality test, [Doornik and Hansen \(1994\)](#);
- ARCH test, see e.g. [Hendry and Doornik \(2001, §18.4\)](#);
- Portmanteau test, [Ljung and Box \(1978\)](#).

`TestGraphicAnalysis` plots the graphic analysis: actual and fitted, residuals and ACF.

`TestForecastGraphics` graphs the forecasts, must be preceded by a call to `Forecast`.

Arfima::UseSampleMean

`UseSampleMean()` ;

No return value.

Description

This will free the mean (i.e. use the sample mean), after it has been fixed previously using `FixMean` (which fixes the mean to a known value, and is the default).

12 ArfimaSim member functions

The `ArfimaSim` class derives from `Simula`. This section describes the main functions of `ArfimaSim`, again in alphabetical order. See the header file `arfimasim.h` for undocumented functions.

ArfimaSim::AddTrend

```
AddTrend(const dTrendCoeff);
    dTrendCoeff      in: double, DGP coefficient of the Trend
```

No return value.

Description

Adds a trend to the DGP and the model.

ArfimaSim::ArfimaSim

```
ArfimaSim(const mcT, const dDgpYmean, const dEpsVar, const dD,
    const vAr, vMa, const fFixD, const fUseMean,
    const dFixMean, const cRep, const iMethod);
```

```
~ArfimaSim();
    mcT          in: matrix with sample sizes for experiments,
    dDgpYmean    in: double, DGP mean of generated dependent
                    variable,
    dEpsVar      in: double, DGP error variance,
    dD           in: double, DGP value of  $d \in (-1, 1.5)$ ,
    vAr          in:  $1 \times p$  matrix, DGP values of AR parameters
    vMa          in:  $1 \times q$  matrix, DGP values of MA parameters
                    (these are flipped to ensure invertibility)
    fFixD        in: int, TRUE: do not estimate d (is fixed at DGP
                    value in model)
    fUseMean     in: int, TRUE: estimation is in deviation from
                    fixed mean dFixMean, unless dFixMean is
                    the missing value, in which case estimation
                    is in deviation from the sample mean
                    FALSE: a constant is added to the model
    dFixMean     in: int, value of known mean
    cRep         in: int, number of replications,  $M$ 
    iMethod      in: int, estimation method, one of: M_MAXLIK,
                    M_NLS, M_MAXMLIK, M_INITONLY,
                    M_NLS_STATIONARY.
```

No return value.

Description

Constructor function which designs the experiment. The estimation object resides in the `m_arfima` data member, which is created in `ArfimaSim` by calling the virtual function `CreateObject`.

`~ArfimaSim` is the destructor, which also prints how long the experiment took.

ArfimaSim::CreateObject

```
virtual CreateObject();
```

No return value.

Description

The estimation object resides in the `m_arfima` data member, which is created through the virtual function `CreateObject`. By default it is of type `Arfima`. Overriding `CreateObject` in a class derived from `ArfimaSim` allows this object to be of a class derived from `Arfima`.

ArfimaSim::DoCoefTstats

```
DoCoefTstats();
```

No return value.

Description

Generate t-values.

ArfimaSim::Generate, ArfimaSim::GetCoefficients, ArfimaSim::Get...

```
virtual Generate(const iRep, const cT, const mxT); // generate replication  
GetCoefficients(); // returns coefficient estimates  
GetPvalues(); // returns empirical p-values  
GetTestStatistics(); // return test statistics
```

Description

These functions implement Simulation class virtual functions.

ArfimaSim::SaveIn7

```
SaveIn7(const sFilename);  
    sFilename    in: string, file name
```

Return value

TRUE if anything was stored.

Description

Stores the simulation results in the named file, after the experiment has finished. Requires a call to the (Simulation class function) `SetStore(TRUE)` before the experiment is started.

ArfimaSim::SetMethod, ArfimaSim::SetStartFromDgp

```
SetMethod(iMethod);  
SetStartFromDgp();  
    iMethod    in: int, estimation method (see ArfimaSim)
```

No return value.

Description

`SetMethod` allows for changing the estimation method after the constructor has been called.

`SetStartFromDgp` lets each estimation start from the DGP values. Otherwise, the starting values are generated using the default procedure.

13 Changes from previous versions

Release history Arfima package:

- version 1.06: Dec 2012: recompiled for Ox 7
- version 1.05: June 2008 (Ox), Sept 2006 (PcGive); Minimum Ox version: Ox 4.00;
- version 1.04: March 2006; Minimum Ox version: Ox 4.00;
- version 1.01: June 2000; Minimum Ox version: Ox 3.00;
- version 1.0: May 1999; Minimum Ox version: Ox 2.10;
- version 0.77: October 1997; for Ox 1.20;
- version 0.76: July 1996; for Ox 1.08.

changes made in version 1.05:

- Forecast: only calling Grow if extension required.

changes made in version 1.01:

- Arfima unnecessarily created seasonals, which was a problem when using 27000 observations with frequency 400.
- NLS log-likelihood now using RSS/T instead of $(RSS/(T-k))$.
- Fixed labelling error in forecast graph.

Improvements made in version 1.00:

- now using Durbin's algorithm for likelihood evaluation;
- improved starting values;
- removed singularity for a single root at zero;
- using adjusted version of Durbin's algorithm for data generation;
- optimal forecasting for arbitrary $d > 1$
- "additive" X, "innovative" Z, and "weighting" W variables;
- removal of non-invertable MA's;
- facility to simulate initial estimates;
- concentrating out regressors in EML;
- addition of modified profile likelihood estimation;
- ArfimaSim class;
- general functions for the mean, possibly nonlinear in parameters;
- functions for communication with OxPack.

There are some reasons why the results may differ slightly from version 0.77:

- improved starting values;
- removed singularity for a single root at zero;
- removal of non-invertable MA's;
- small change in the line search of BFGS method;
- the default is now `FixMean(0)`; to get the old default insert a call to `UseSampleMean`.
- AIC now computed as $-2\hat{\ell} + 2s$ (was negative: $2\hat{\ell} - 2s$). AIC also differs in that s now counts the residual variance.

14 Technical Summary

The remainder gives a summary of the implementation details of the procedures in the Arfima package. It complements [Doornik and Ooms \(2004\)](#).

15 The Arfima model

The basic ARMA(p, q) model is

$$y_t = \phi_1 y_{t-1} + \dots + \phi_p y_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \dots + \theta_q \varepsilon_{t-q}, \quad t = 1, \dots, T,$$

assuming either $\varepsilon_t \sim \text{NID}[0, \sigma_\varepsilon^2]$, or $E[\varepsilon_t] = 0$ and $E[\varepsilon_t^2] = \sigma_\varepsilon^2$. Using lag polynomials and introducing a mean μ we write:

$$\Phi(L)(y_t - \mu) = \Theta(L)\varepsilon_t.$$

With a fractional integration parameter d , the ARFIMA(p, d, q) model is written as

$$\Phi(L)(1-L)^d(y_t - \mu) = \Theta(L)\varepsilon_t. \quad (1)$$

The autocovariance function (ACovF) of a stationary ARMA process with mean μ :

$$c(i) = E[(y_t - \mu)(y_{t-i} - \mu)],$$

defines the variance matrix of the joint distribution of $\mathbf{y} = (y_1, \dots, y_T)'$:

$$\mathbf{V}[\mathbf{y}] = \begin{pmatrix} c(0) & c(1) & c(2) & \dots & c(T-1) \\ c(1) & c(0) & c(1) & \ddots & \vdots \\ c(2) & c(1) & c(0) & \ddots & c(2) \\ \vdots & \ddots & \ddots & \ddots & c(1) \\ c(T-1) & \dots & c(2) & c(1) & c(0) \end{pmatrix} = \mathcal{T}[c(0), \dots, c(T-1)] = \mathbf{\Sigma}, \quad (2)$$

which is a Toeplitz matrix, denoted by \mathcal{T} . Under normality:

$$\mathbf{y} \sim N_T(\boldsymbol{\mu}, \mathbf{\Sigma}). \quad (3)$$

The autocorrelation function, ACF: $c(i)/c(0)$, of a stationary ARMA process is discussed in many textbooks, and readily computed from the ϕ_i and θ_i using the Ox function `armavar`. We often work with the autocovariances scaled by the error variance:

$$\mathbf{r} = [r(0) \dots r(T-1)]' = \sigma_\varepsilon^{-2} [c(0) \dots c(T-1)]'.$$

15.1 Autocovariance function

An algorithm for the computaion of the ACovF of an ARFIMA process is derived in [Sowell \(1992\)](#):

$$c(i) = \sigma_\varepsilon^2 \sum_{k=-q}^q \sum_{j=1}^p \psi_k \zeta_j C(d, p+k-i, \rho_j), \quad (4)$$

where

$$\psi_k = \sum_{s=|k|}^q \theta_s \theta_{s-|k|}, \quad \zeta_j^{-1} = \rho \left[\prod_{i=1}^p (1 - \rho_i \rho_j) \prod_{m \neq j} (\rho_j - \rho_m) \right], \quad (5)$$

and

$$C(d, h, \rho) = \frac{\Gamma(1-2d)}{[\Gamma(1-d)]^2} \frac{(d)_h}{(1-d)_h} [\rho^{2p} F(d+h; 1-d+h; \rho) + F(d-h; 1-d-h; \rho) - 1]. \quad (6)$$

Here Γ is the gamma function, ρ_j are the roots of the AR polynomial (assumed distinct), and $F(a, 1; c; \rho)$ is the hypergeometric function, see e.g. [Abramowitz and Stegun \(1970, Ch. 15\)](#):

$$F(a, b; c; \rho) = \sum_{i=0}^{\infty} \frac{(a)_i (b)_i}{(c)_i} \frac{\rho^i}{i!},$$

where we use Pochhammer's symbol:

$$(a)_i = a(a+1)(a+2) \cdots (a+i-1), \quad (a)_0 = 1.$$

So $(1)_i$ equals $i!$.

In the absence of AR parameters (4) reduces to³

$$c(i) = \sigma_\varepsilon^2 \sum_{k=-q}^q \psi_k \frac{\Gamma(1-2d)}{[\Gamma(1-d)]^2} \frac{(d)_{k-i}}{(1-d)_{k-i}}.$$

16 Estimation

16.1 Regressors in mean

Any set of exogeneous regressors may be used to explain the mean:

$$\mathbf{z} = \mathbf{y} - \boldsymbol{\mu}, \quad \boldsymbol{\mu} = \mathbf{f}(\mathbf{X}, \boldsymbol{\beta}),$$

where \mathbf{X} is a $T \times k$ matrix. In the leading linear case $\mathbf{f}(\mathbf{X}, \boldsymbol{\beta}) = \mathbf{X}\boldsymbol{\beta}$ and $\boldsymbol{\beta}$ is a $k \times 1$ vector.

16.2 Initial values

Initial values for the parameter estimates are obtained in the order: regressors in mean, d , AR part, and finally MA part. The very first step is to subtract a mean from y_t : $z_t = y_t - \mu_t$. When either the sample mean or a specified (known, possibly zero) mean is used: $\mu_t = \mu$. If regressors are used, take $\mu_t = \mathbf{f}(x_t, \boldsymbol{\beta})$. In the linear case $\boldsymbol{\beta}$ is obtained by regression.

1. For the fractional integration parameter the (frequency domain) log periodogram regression of [Geweke and Porter-Hudak \(1983\)](#) is used, yielding \hat{d}_0 . We use $[T^{-1/2}]$ nonzero periodogram

³Note the typo in the equation below (8) in [Sowell \(1992, p.173\)](#): $\Gamma(d+s-l)$ in the numerator should read $\Gamma(d-s+l)$.

points, except when $p = q = 0$ when we use all available points. The initial time domain residuals are then obtained using the Ox function `diffpow`:

$$u_t = \sum_{j=0}^t \frac{(-\hat{d}_0)^j}{j!} z_{t-j}. \quad (7)$$

2. Next, AR starting values are obtained from solving the Yule-Walker equations taking the number of MA parameters into account:

$$\begin{pmatrix} \hat{\rho}(q) & \dots & \hat{\rho}(q-p+1) \\ \vdots & & \\ \hat{\rho}(q+p-1) & & \hat{\rho}(q) \end{pmatrix} \hat{\phi}_0 = \begin{pmatrix} \hat{\rho}(q+1) \\ \vdots \\ \hat{\rho}(q+p) \end{pmatrix},$$

where $\hat{\rho}(i)$ is the empirical autocorrelation of u_t . When q is zero, the matrix on the right-hand side is the Toeplitz matrix $\mathcal{T}[\hat{\rho}(0), \dots, \hat{\rho}(p-1)]$.

We use OLS to solve this system; this will also give a solution when the matrix is singular. Subsequently, the `arma0` function is used to obtain residuals u_t^* .

3. Starting values for the MA parameters are derived from u_t^* using Tunnicliffe-Wilson's method, see [Granger and Newbold \(1986, p.88\)](#). Because this iterative method is slow to converge, we choose rather loose convergence criteria. A non-invertible MA is 'flipped' to an invertible MA by inverting roots outside the unit circle. The `arma0` function is used to obtain residuals u_t^{**} .

When the initial values are used as starting values for further estimation, the following adjustments are made:

1. If d is not significant at 5%, it is set to zero. A value of \hat{d}_0 less than -0.45 is set to -0.40 , and similarly to 0.40 for a value greater than 0.45 .
2. If $q > 0$ and the solution from the Yule-Walker equations yields non-stationary AR parameters, the method is applied as if $q = 0$.

16.3 Exact maximum likelihood (EML)

Based on normality (3), and with the a procedure to compute the autocovariances in (2), the log-likelihood is simply (writing \mathbf{z} for the data vector used for maximization):

$$\log L(d, \phi, \theta, \beta, \sigma_\epsilon^2) = -\frac{T}{2} \log(2\pi) - \frac{1}{2} \log |\Sigma| - \frac{1}{2} \mathbf{z}' \Sigma^{-1} \mathbf{z}. \quad (8)$$

It is convenient to concentrate σ_ϵ^2 out of the likelihood, starting by writing $\Sigma = \mathbf{R}\sigma_\epsilon^2$:

$$\log L(d, \phi, \theta, \beta, \sigma_\epsilon^2) \propto -\frac{1}{2} \log |\mathbf{R}| - \frac{T}{2} \log \sigma_\epsilon^2 - \frac{1}{2\sigma_\epsilon^2} \mathbf{z}' \mathbf{R}^{-1} \mathbf{z}.$$

Differentiating with respect to σ_ϵ^2 , and solving yields

$$\hat{\sigma}_\epsilon^2 = T^{-1} \mathbf{z}' \mathbf{R}^{-1} \mathbf{z}, \quad (9)$$

with concentrated likelihood (CLF):

$$\ell_c(d, \phi, \theta, \beta) = -\frac{T}{2} \log(2\pi) - \frac{T}{2} - \frac{1}{2} \log |\mathbf{R}| - \frac{T}{2} \log [T^{-1} \mathbf{z}' \mathbf{R}^{-1} \mathbf{z}].$$

When $f(\mathbf{X}, \beta) = \mathbf{X}\beta$ it is more convenient to also concentrate β out of the likelihood. The resulting normal profile log-likelihood function becomes:

$$\ell_P(d, \phi, \theta) = -\frac{T}{2}(1 + \log 2\pi) - \frac{1}{2} \log |\mathbf{R}| - \frac{T}{2} \log [T^{-1} \hat{\mathbf{z}}' \mathbf{R}^{-1} \hat{\mathbf{z}}], \quad (10)$$

where

$$\hat{\mathbf{z}} = \mathbf{y} - \mathbf{X}\hat{\beta}, \quad \hat{\beta} = (\mathbf{X}'\mathbf{R}^{-1}\mathbf{X})^{-1} \mathbf{X}'\mathbf{R}^{-1}\mathbf{y}. \quad (11)$$

The function used in the maximization procedure is:

$$-\frac{1}{2} \{T^{-1} \log |\mathbf{R}| + \log \sigma_\epsilon^2\}, \quad (12)$$

from which the value for the log-likelihood (10) is easily derived. The computational procedure described in §20.2 writes

$$\sigma_\epsilon^2 = T^{-1} \mathbf{z}' \mathbf{R}^{-1} \mathbf{z} = T^{-1} \mathbf{e}' \mathbf{e},$$

with $|\mathbf{R}|$ a by-product of the procedure.

Function (12) is maximized using BFGS with numerical derivatives. During estimation, stationarity is imposed at each step by rejecting parameter values which have:

- $d \leq -5$ or $d > 0.49999$;
- $|\rho_i| \geq 0.9999$, where ρ_i are the roots of the AR polynomial.

In addition, the procedure can fail because:

- inability to compute the roots of the AR polynomial;
- $\rho\zeta \leq 10^{-11}$, this corresponds to multiple roots, see (5).

16.4 Modified profile likelihood (MPL)

The modified profile log-likelihood, ℓ_M , for the regression model with stationary ARFIMA-errors and $\mathbf{f}(\mathbf{X}, \boldsymbol{\beta}) = \mathbf{X}\boldsymbol{\beta}$:

$$\ell_M(d, \phi, \theta) = -\frac{T}{2} (1 + \log 2\pi) - \left(\frac{1}{2} - \frac{1}{T}\right) \log |\mathbf{R}| - \frac{T-k-2}{2} \log [T^{-1} \hat{\mathbf{z}}' \mathbf{R}^{-1} \hat{\mathbf{z}}] - \frac{1}{2} \log |\mathbf{X}' \mathbf{R}^{-1} \mathbf{X}|, \quad (13)$$

see [An and Bloomfield \(1993\)](#), who applied the idea of [Cox and Reid \(1987\)](#) to reduce the bias of the EML estimator due to the presence of unknown nuisance parameters of the regressors.

The residual variance estimator now uses $T - k$, so that it is unbiased when $p = q = d = 0$:

$$\hat{\sigma}_\epsilon^2 = \frac{1}{T - k} \hat{\mathbf{z}}' \mathbf{R}^{-1} \hat{\mathbf{z}}. \quad (14)$$

16.5 Non-linear least squares (NLS)

Defining e_t as the residuals from applying the ARFIMA(p, d, q) filter to $y_t - \mu_t$, the residual variance is:

$$\sigma_\epsilon^2 = \frac{1}{T - k} \sum_{t=1}^T e_t^2. \quad (15)$$

NLS simply maximizes

$$f(d, \phi, \boldsymbol{\theta}, \boldsymbol{\beta}) = -\frac{1}{2} \log \left(\frac{1}{T} \sum_{t=1}^T e_t^2 \right). \quad (16)$$

The arfima filter is computed using the Ox function `diffpow`, see (7), followed by `arma0`. Since (7) essentially drops the first observation, $e_1 = 0$ when d is estimated.

Function (16) is maximized using BFGS with numerical derivatives, optionally with stationarity imposed.

16.6 Variance-covariance matrix estimates

Let $\boldsymbol{\vartheta}' = [d \ \phi' \ \boldsymbol{\theta}']$. The variance-covariance matrix for the EML ($\ell = \ell_P$) and MPL ($\ell = \ell_M$) estimates is computed as:

$$\begin{pmatrix} (-\partial^2 \ell(\boldsymbol{\vartheta}) / \partial \boldsymbol{\vartheta} \partial \boldsymbol{\vartheta}' |_{\hat{\boldsymbol{\vartheta}}})^{-1} & \mathbf{0} \\ \mathbf{0} & \hat{\sigma}_\epsilon^2 (\mathbf{X}' \mathbf{R}^{-1} \mathbf{X})^{-1} \end{pmatrix}.$$

The second derivative of ℓ is computed numerically.

For NLS, the variance-covariance is the inverse of minus the numerical second derivative of (16).

17 Estimation output

Estimation output consists of

- Estimated coefficients, with estimated standard errors, t-values, and p-values. The p-values are based on a $t(T - s)$ -distribution, where s is the number of estimated parameters, including the residual variance. When all parameters are freely estimated: $s = 1 + p + q + k + 1$.
- log-likelihood $\hat{\ell} =$

$$\begin{aligned} \text{EML: } & \ell_c, \\ \text{MPL: } & \ell_M, \\ \text{NLS: } & f - \frac{T}{2} (1 + \log 2\pi), \end{aligned}$$

where f for NLS is from (16).

- Akaike information criterion

$$\text{AIC} = -2\hat{\ell} + 2s,$$

where s the number of estimated parameters. When no parameters are fixed: $s = 1 + p + q + k + 1$ (the last accounts for the residual variance). The AIC/T is also reported.

- Residual variance: (9) for EML, (14) for MPL, (15) for NLS.
- Mean and variance of dependent variable.
- BFGS convergence criteria, convergence result and starting values.

18 Estimation options

18.1 Sample mean versus known mean

It has been found in early Monte Carlo experiments that, in smaller samples, using the theoretical mean could lead to more accurate estimation of d (see e.g. [Cheung and Diebold, 1994](#)). This can be seen as the most effective way to reduce the effect of a nuisance parameter on inference for the parameter of interest. Therefore, the Arfima package allows for fixing the mean at a specified value. Let y_t denote the original dependent variable, and μ_y the known mean.

The \mathbf{z} used in §16.3 for estimation when specifying a known mean is:

$$z_t = y_t - \mu_y,$$

otherwise the package uses

$$z_t = y_t - \hat{\mu}_y.$$

The specification of the mean affects the likelihood. For the last term in the log-likelihood:

$$(\mathbf{y} - \boldsymbol{\mu})' \mathbf{R}^{-1} (\mathbf{y} - \boldsymbol{\mu}) = \mathbf{y}' \mathbf{R}^{-1} \mathbf{y} - 2\boldsymbol{\mu}' \mathbf{R}^{-1} \mathbf{y} + \boldsymbol{\mu}' \mathbf{R}^{-1} \boldsymbol{\mu},$$

so the known mean case adds $\mu_y \boldsymbol{\iota}' \mathbf{R}^{-1} \mathbf{y}$, whereas the second case adds $\hat{\mu}_y \boldsymbol{\iota}' \mathbf{R}^{-1} \mathbf{y}$, and different results must be expected.

18.2 Fixing parameters

It is possible to fix d at a specific value, or drop ARMA terms using the `FixAR`, `FixD` and `FixMA` functions.

18.3 Weighted estimation

A weight variable \mathbf{w} , $w_t \geq 0$, can be used in estimation. Write $\tilde{w}_t = w_t/\bar{w}_{>0}$, where $\bar{w}_{>0}$ is the mean of the positive weights.

Then (12) for EML becomes:

$$-\frac{1}{2} \left\{ T^{-1} \log |\mathbf{R}| - T^{-1} \sum_{\tilde{w}_t > 0} \log \tilde{w}_t + \log T^{-1} \sum_{t=1}^T e_t^2 \tilde{w}_t \right\},$$

The NLS function is adjusted in a similar fashion. Weighted estimation is not available for MPL, and weights are ignored for forecasting. The weighted residuals, $\hat{e}_t \tilde{w}_t^{1/2}$, are used in the residual-based diagnostics.

18.4 Z variables

With both additive normal regressors x_t and innovative Z variables \mathbf{z}_t the ARFIMA model becomes:

$$\Phi(L)(1-L)^d(y_t - \mathbf{x}'_t \boldsymbol{\beta}) = \Theta(L)(\varepsilon_t + \mathbf{z}'_t \boldsymbol{\gamma}).$$

The notation for the Z variables in this subsection should not be confused with z_t , the demeaned y_t . After applying the normal EML or NLS filter to z_t , $\mathbf{z}'_t \hat{\boldsymbol{\gamma}}$ is subtracted at each iteration.

This model has the familiar ADL (Autoregressive Distributed-lag model) as a special case, since \mathbf{z}_t can contain different lags of the same (exogenous) variable. Whereas additive outliers (and missing observations) can be estimated using dummies for the X variables, see e.g. [Brockwell and Davis \(1993, §12.3\)](#), innovative outliers can be modelled by dummies for Z variables. Note that adding a single observation dummy for a Z variable has the same effect as giving that observation zero weight in the W variable. This is illustrated in `fracest8.ox`.

Z variables are not available for MPL.

19 Forecasting

Two methods of forecasting are supported, based on the results in [Beran \(1994, §8.7\)](#). As before let $\mathbf{z} = (z_1, \dots, z_T)'$ denote the observations over the estimation period. Assume z_t is stationary and $d > -1$. The best linear prediction of z_{T+h} is

$$\hat{z}_{T+h} = [r(T-1+h) \cdots r(h)] \{ \mathcal{T} [r(0), \dots, r(T-1)] \}^{-1} \mathbf{z} = \mathbf{q}' \mathbf{z},$$

which consists of the reversed ACovF starting from h , times the original data weighted by their correlations. The `solvetoeplitz` function is used to solve $\mathcal{T} \mathbf{x} = \mathbf{z}$ in order to keep storage requirements of order T , see §20.2. The mean square error is

$$\text{MSE}(\hat{z}_{T+h}) = \hat{\sigma}_\varepsilon^2 (r(0) - \mathbf{r}' \mathbf{q}).$$

In the presence of a mean-function $\mu_t = \mathbf{f}(x_t, \boldsymbol{\beta})$ the forecasts are:

$$\hat{y}_{T+h} = \mathbf{q}' (\mathbf{y} - \boldsymbol{\mu}) + \mu_{t+h} + \mathbf{x}'_{T+h} \hat{\boldsymbol{\beta}}.$$

The Ox code computes all requested forecasts $\hat{\mathbf{z}}_h = (z_{T+1}, \dots, z_{T+h})'$ and their joint variance-covariance matrix, $\text{Cov}(\hat{\mathbf{z}}_h)$ simultaneously. $\text{Cov}(\hat{\mathbf{z}}_h)$ is also used to derive the mean squared errors for partial sums, $\sum_{i=1}^h \hat{z}_{T+i}$, integrated partial sums etc.

'Naive' forecasts are derived from the autoregressive representation of the process, truncated at $T + h$:

$$\Theta^{-1}(L) \Phi(L) (1-L)^d z_t = \left(1 - b_1 L \cdots - b_{T+h-1} L^{T+h-1}\right) z_t = B(L) z_t.$$

In this case the z_t need not be stationary, c.f. [Beran \(1995\)](#), but $d > -0.5$. The first T coefficients in the $(1-L)^d$ polynomial can be computed using the `diffpow` function when the input is a one followed by $T-1$ zeroes; this follows from (7). For polynomial multiplication and division, `polymul` and `polydiv` are used. The naive forecasts are computed recursively:

$$\hat{z}_{T+h}^* = [b_{T+h-1} \cdots b_1] \times [\mathbf{z}' \hat{z}_{T+1} \cdots \hat{z}_{T+h-1}]',$$

and

$$\text{MSE}(\hat{z}_{T+h}^*) = \hat{\sigma}_\varepsilon^2 \left(1 + \sum_{i=1}^{h-1} a_i^2\right), \tag{17}$$

where a_i are the coefficients of $B^{-1}(L)$.

Level forecasts are computed by adding the (integrated) partial sums of forecasts to a specified starting level. The reported MSE of the integrated naive forecasts can be obtained directly from (17).

Forecasting with Z variables is not yet implemented.

20 Some notes on computation

20.1 Autocovariance function

[Sowell \(1992\)](#) gives several tricks for recursively computing various quantities needed in (4). This is further refined in the code of the Arfima package, and discussed here in detail in [Doornik and Ooms \(2003\)](#).

20.2 Likelihood evaluation

Remember that \mathbf{R} is the $T \times T$ Toeplitz matrix of the autocorrelations, and there are various ways of computing the function (12), see [Doornik and Ooms \(2003\)](#).

The Arfima package uses Durbin's algorithm. This method (see [Golub and Van Loan, 1989](#), §4.7.2) amounts to computing the Choleski decomposition of the inverted Toeplitz matrix. Durbin's method solves

$$\mathcal{T}[r(0), \dots, r(T-1)] = \mathbf{L}\mathbf{D}\mathbf{L}' = \mathbf{P}\mathbf{P}', \quad \mathbf{e} = \mathbf{D}^{-1/2}\mathbf{L}^{-1}\mathbf{z} = \mathbf{P}^{-1}\mathbf{z}.$$

with an operation count of order T^2 . So we can write:

$$\mathbf{z}'\mathbf{R}^{-1}\mathbf{z} = \mathbf{e}'\mathbf{e}.$$

By applying the factorization as it is computed, storage of the $\frac{1}{2}T(T+1)$ matrix is avoided. This method leads to a more elegant expression of the log-likelihood (in addition to being marginally faster), and is currently used in the Arfima package.

20.3 Invertibility of MA polynomial

The same likelihood pertains when the roots of the MA polynomial are inverted. Since the likelihood of a non-invertible MA can be evaluated without problems, estimation is not affected. In a Monte Carlo experiment, however, it is essential that non-invertibility is taken into account. Take an MA(1), with $\theta = 0.5$. Since $\theta = 2$ yields the same likelihood, it is thinkable that half the experiments yield $\hat{\theta} \approx 0.5$ and the other half $\hat{\theta} \approx 2$, resulting in poor average estimates from the Monte Carlo experiment.

The following table illustrates the issue ($T = 100, M = 100$). The first set of results removes the non-invertible MA (required in 19 cases), the second leaves the MA roots unchanged:

coefficients	mean	std.dev	mean bias
with MA inversion			
MA1=0.9	0.89157	0.075471	-0.0084326
MA2=0.81	0.81967	0.11363	0.0096664
without inversion			
MA1=0.9	0.93546	0.26420	0.035462
MA2=0.81	0.86154	0.13834	0.051540

21 Monte Carlo experimentation

The problem in data generation for the ARFIMA(p, d, q) process is analogue to that set out in §20.2:

- Use the naive Choleski method for likelihood evaluation. Let \mathbf{r} be the standardized autocovariances of the specified process, and $\mathcal{T}[\mathbf{r}] = \mathbf{P}\mathbf{P}'$, then

$$\mathbf{y} = \sigma_{\varepsilon}\mathbf{P}\boldsymbol{\varepsilon} + \boldsymbol{\mu},$$

where $\boldsymbol{\varepsilon}$ are drawings from the standard normal distribution. For small T , this is convenient, because \mathbf{P} only need to be computed once. Once the Choleski decomposition has been computed, generating data is only of order T^2 .

- A modified version of Durbin's algorithm is used to apply the inverted filter:

$$\mathcal{T}[r(0), \dots, r(T-1)] = \mathbf{P}\mathbf{P}', \quad \mathbf{z} = \mathbf{P}\mathbf{e}.$$

This algorithm is of order T^2 , but perhaps somewhat slower than the naive method for small T . However, it allows for simulation with a large number of observations.

References

- Abramowitz, M. and I. A. Stegun (1970). *Handbook of Mathematical Functions*. New York: Dover Publications Inc. [26]
- An, S. and P. Bloomfield (1993). Cox and Reid's modification in regression models with correlated errors. Technical report, Department of Statistics, North Carolina State University, Raleigh, NC 27695-8203, U.S.A. [11, 29]
- Beran, J. (1994). *Statistics for Long-memory Processes*. London: Chapman and Hall. [15, 31]
- Beran, J. (1995). Maximum likelihood estimation of the differencing parameter for invertible short and long memory autoregressive integrated moving average models. *Journal of the Royal Statistical Society* 57, 659–672. [32]
- Brockwell, P. J. and R. A. Davis (1993). *Time Series: Theory and Methods (2nd ed.)*. USA: Springer-Verlag, New-York. [31]
- Cheung, Y.-W. and F. X. Diebold (1994). On maximum likelihood estimation of the differencing parameter of fractionally-integrated noise with unknown mean. *Journal of Econometrics* 62, 301–316. [30]
- Cox, D. R. and N. Reid (1987). Parameter orthogonality and approximate conditional inference (with discussion). *Journal of the Royal Statistical Society Series B* 49, 1–39. [29]
- Doornik, J. A. (2013). *Object-Oriented Matrix Programming using Ox (7th ed.)*. London: Timberlake Consultants Press. [3]
- Doornik, J. A. and H. Hansen (1994). A practical test for univariate and multivariate normality. Discussion paper, Nuffield College. [21]
- Doornik, J. A. and D. F. Hendry (2013). *OxMetrics: An Interface to Empirical Modelling (7th ed.)*. London: Timberlake Consultants Press. [3]
- Doornik, J. A. and M. Ooms (2003). Computational aspects of maximum likelihood estimation of autoregressive fractionally integrated moving average models. *Computational Statistics & Data Analysis* 42, 333–348. [3, 32]
- Doornik, J. A. and M. Ooms (2004). Inference and forecasting for ARFIMA models, with an application to US and UK inflation. *Studies in Nonlinear Dynamics and Econometrics* 8. Issue 2, Article 14. [3, 4, 25]
- Eisinga, R., P. H. Franses, and M. Ooms (1999). Forecasting long memory right-left political orientations. *International Journal of Forecasting* 15, 185–199. [7]
- Findley, D. F., B. C. Monsell, W. R. Bell, W. R. Otto, and B.-C. Chen (1998). New capabilities and methods of the x-12-arima seasonal-adjustment program. *Journal of Business & Economic Statistics* 16, 127–152, discussion: 153–177. [3]
- Geweke, J. F. and S. Porter-Hudak (1983). The estimation and application of long memory time series models. *Journal of Time Series Analysis* 4(4), 221–238. [15, 26]
- Golub, G. H. and C. F. Van Loan (1989). *Matrix Computations*. Baltimore: The Johns Hopkins University Press. [32]
- Granger, C. W. J. and P. Newbold (1986). *Forecasting Economic Time Series*, (2nd ed.). New York: Academic Press. [7, 27]
- Hauser, M. A. (1997). Maximum likelihood estimators for ARFIMA models: A Monte Carlo study. Technical report, Department of Statistics, University of Economics and Business Administration, Vienna, Austria. [11]
- Hendry, D. F. and J. A. Doornik (2001). *Empirical Econometric Modelling using PcGive: Volume*

- I* (3rd ed.). London: Timberlake Consultants Press. [21]
- Ljung, G. M. and G. E. P. Box (1978). On a measure of lack of fit in time series models. *Biometrika* 65, 297–303. [21]
- Ooms, M. and J. A. Doornik (1998). Estimation, simulation and forecasting for fractional autoregressive integrated moving average models. Technical report, Econometric Intitute, Erasmus University Rotterdam, presented at the fourth annual meeting of the Society for Computational Economics, June 30, 1998, Cambridge, UK. [3]
- Ooms, M. and U. Hassler (1997). On the effect of seasonal adjustment on the log-periodogram regression. *Economics Letters* 56, 135–141. [15]
- Robinson, P. M. (1995a). Gaussian semiparametric estimation of long range dependence. *The Annals of Statistics* 23, 1630–1661. [15]
- Robinson, P. M. (1995b). Log-periodogram regression of time series with long range dependence. *Annals of Statistics* 23, 1048–1072. [15]
- Robinson, P. M. and M. Henry (1998). Long and short memory conditional heteroscedasticity in estimating the memory parameter of levels. Technical Report STIDERC Econometrics EM/98/357, London School of Economics and Political Science. [15]
- Sowell, F. (1992). Maximum likelihood estimation of stationary univariate fractionally integrated time series models. *Journal of Econometrics* 53, 165–188. [3, 25, 26, 32]