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

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.2 Likelihood evaluation</td>
<td>32</td>
</tr>
<tr>
<td>20.3 Invertibility of MA polynomial</td>
<td>33</td>
</tr>
<tr>
<td>21 Monte Carlo experimentation</td>
<td>33</td>
</tr>
</tbody>
</table>
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$(p, d, q)\dagger$ 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.

---

\dagger Part 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 -DN0_DLL

The -DN0_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.
#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:

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

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>#obs</th>
<th>#miss</th>
<th>min</th>
<th>mean</th>
<th>max</th>
<th>std.dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPI_UK</td>
<td>160</td>
<td>0</td>
<td>11.3</td>
<td>56.563</td>
<td>153.8</td>
<td>47.409</td>
</tr>
<tr>
<td>Inflat</td>
<td>159</td>
<td>1</td>
<td>-0.0082305</td>
<td>0.01642</td>
<td>0.090573</td>
<td>0.015168</td>
</tr>
</tbody>
</table>
A Package for Estimating, Forecasting and Simulating ARFIMA Models: ARFIMA package 1.06 for OX

--- Maximum likelihood estimation of ARFIMA(0,d,0) model ---

Iterating:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>f</th>
<th>df</th>
<th>e1</th>
<th>e2</th>
<th>step</th>
</tr>
</thead>
<tbody>
<tr>
<td>it0</td>
<td>4.420666</td>
<td>0.07155</td>
<td>0.1002</td>
<td>0.005111</td>
<td>1</td>
</tr>
<tr>
<td>it1</td>
<td>4.421729</td>
<td>0.01305</td>
<td>0.01780</td>
<td>0.0004044</td>
<td>0.5</td>
</tr>
<tr>
<td>it2</td>
<td>4.421764</td>
<td>0.0003887</td>
<td>0.0005324</td>
<td>0.0004028</td>
<td>1</td>
</tr>
<tr>
<td>it3</td>
<td>4.421764</td>
<td>2.442e-006</td>
<td>3.345e-006</td>
<td>1.165e-005</td>
<td>1</td>
</tr>
</tbody>
</table>

Strong convergence

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

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Std.Error</th>
<th>t-value</th>
<th>t-prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>d parameter</td>
<td>0.369581</td>
<td>0.05098</td>
<td>7.25</td>
</tr>
</tbody>
</table>

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.ARMAd(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_MAXMPLIK);
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).

frccest5.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: frccest6.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 frccest7.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, frccest8.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-process, 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 \textit{(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: \textit{(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 Ox-Pack 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 $\Delta \log(RPI\_UK)$ 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:

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:

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

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):

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

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

![Figure 2: Forecasts](image)

9 Some simulation examples

The following simulation programs\(^2\) use the Arfima simulation class \texttt{ArfimaSim}, which derives from the base simulation class \texttt{Simulation}. The main (constructor) function \texttt{ArfimaSim()} sets up a sim-

\(^2\)The 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 ($M = 250, T = 100$):

<table>
<thead>
<tr>
<th>$\phi$ = 0.7</th>
<th>$\phi$ = 0.2</th>
<th>$\phi$ = -0.3</th>
<th>$\phi$ = -0.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d$ = -0.3</td>
<td>-9.7</td>
<td>3.4</td>
<td>-14.1</td>
</tr>
<tr>
<td>$d$ = 0</td>
<td>-12.4</td>
<td>5.2</td>
<td>-26.9</td>
</tr>
<tr>
<td>$d$ = 0.3</td>
<td>-16.8</td>
<td>8.5</td>
<td>-33.3</td>
</tr>
</tbody>
</table>

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 $d$.  

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

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

\texttt{\textbf{static Acf}(\textbf{const cT}, \textbf{const vP}, \textbf{const cAR}, \textbf{const cMA},
\textbf{const bPrintErr});}

\text{cT} \quad \text{in: int, number of observations, } T
\text{vP} \quad \text{in: } s\text{-vector with coefficients in order } d, \text{ AR,}
\text{MA, } s \geq 1 + p + q
\text{cAR} \quad \text{in: int, number of AR parameters, } p
\text{cMA} \quad \text{in: int, number of MA parameters, } q
\text{bPrintErr} \quad \text{in: int, TRUE: print fracsigma error code if an}
\text{error occurs (only when } d \neq 0)\text{]

\textbf{Return value}

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

\textbf{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: \( |\rho_1| \geq 0.9999 \), AR root inside the unit circle;
6: failed to find roots of AR polynomial.
7: there are identical roots.

Arfima::Arfima

\texttt{\textbf{Arfima}();}

\textit{No return value.}

\textbf{Description}

Constructor function.

Arfima::ARMA

\texttt{\textbf{ARMA}(\textbf{const cAR}, \textbf{const cMA});}

\text{cAR} \quad \text{in: int, no of AR paramaters, } p
\text{cMA} \quad \text{in: int, no of MA paramaters, } q

\textit{No return value.}

\textbf{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

\texttt{\textbf{Estimate}();}

\textit{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 $1 \times 3$ 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 $\frac{2\pi j}{T}, j = 1, \ldots, \text{iTrunc}$. In the notation of Robinson (1995b), c.f. Beran (1994, §4.6), and Robinson (1995a), $n = T$, $m = \text{iTrunc}$, $l = 1$.


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();
A PACKAGE FOR ESTIMATING, FORECASTING AND SIMULATING ARFIMA MODELS: ARFIMA
package 1.06 for OX

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 partialed (concentrated) out of the likelihood
(this will only work for linear \(X\beta\), see §16.3. The overridden version should return FALSE.
(Note: when Z or W variables are present, X’s are never partialed out.)
GetXBeta returns the \(T \times 1\) matrix \(f(X,\beta)\), 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 \(f(X,\beta)\) is desired instead of just \(X\beta\).
Examples are in fracest6.ox and fracest7.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 $d$FixMean, unless $d$FixMean 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_MAXMPLIK, 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;
- \( \text{ArfimaSim} \) 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 \( \text{FixMean}(0) \); to get the old default insert a call to \( \text{UseSampleMean} \).
- AIC now computed as \(-2\ell + 2s\) (was negative: \(2\ell - 2s\)). 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

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

assuming either \(\varepsilon_t \sim \text{NID}(0, \sigma^2_{\varepsilon})\), or \(\mathbb{E}[\varepsilon_t] = 0\) and \(\mathbb{E}[\varepsilon_t^2] = \sigma^2_{\varepsilon}\). Using lag polynomials and introducing a mean \(\mu\) 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

\[ \Phi(L)(1-L)^d(y_t - \mu) = \Theta(L)\varepsilon_t. \tag{1} \]

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

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

defines the variance matrix of the joint distribution of \(y = (y_1, \ldots, y_T)'\):

\[ V[y] = \begin{pmatrix} c(0) & c(1) & c(2) & \cdots & c(T-1) \\ c(1) & c(0) & c(1) & \ddots & \vdots \\ c(2) & c(1) & c(0) & \ddots & c(2) \\ \vdots & \ddots & \ddots & \ddots & c(1) \\ c(T-1) & \cdots & c(2) & c(1) & c(0) \end{pmatrix} = T[c(0), \ldots, c(T-1)] = \Sigma, \tag{2} \]

which is a Toeplitz matrix, denoted by \(T\). Under normality:

\[ y \sim \mathcal{N}_T(\mu, \Sigma). \tag{3} \]

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

\[ r = [r(0) \cdots r(T-1)]' = \sigma^{-2}_\varepsilon [c(0) \cdots c(T-1)]'. \]

15.1 Autocovariance function

An algorithm for the computation of the ACovF of an ARFIMA process is derived in Sowell (1992):

\[ c(i) = \sigma^2_{\varepsilon} \sum_{k=-q}^{q} \sum_{j=1}^{p} \psi_k \zeta_j C(d, p + k - i, \rho_j), \tag{4} \]
where
\[ \psi_k = \sum_{s=|k|}^{q} \theta_s \theta_{s-|k|}, \quad \zeta_j^{-1} = \rho \left[ \prod_{i=1}^{p} (1 - \rho_i \rho_j) \prod_{m \neq j} (\rho_j - \rho_m) \right], \tag{5} \]
and
\[ C(d,h,\rho) = \frac{\Gamma(1 - 2d)}{[\Gamma(1-d)]^2} \left( \frac{(d)_h}{(1-d)_h} \right)^2 \left[ \rho^{2p} F(d+h;1-d+h;\rho) + F(d-h;1-d-h;\rho) - 1 \right]. \tag{6} \]

Here \( \Gamma \) is the gamma function, \( \rho_j \) 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):
\[ F(a,b;c;\rho) = \sum_{i=0}^{\infty} \frac{(a)_i (b)_i \rho^i}{(c)_i i!}, \]
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 to:
\[ c(i) = \sigma^2 \sum_{k=-q}^{q} \psi_k \frac{\Gamma(1 - 2d)}{[\Gamma(1-d)]^2} \left( \frac{(d)_{k-i}}{(1-d)_{k-i}} \right). \]

### 16 Estimation

#### 16.1 Regressors in mean

Any set of exogeneous regressors may be used to explain the mean:
\[ z = y - \mu, \quad \mu = f(X,\beta), \]
where \( X \) is a \( T \times k \) matrix. In the leading linear case \( f(X,\beta) = X\beta \) and \( \beta \) 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 \( \mu_t = f(x_t,\beta) \). In the linear case \( \beta \) 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

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

\[
    u_t = \sum_{j=0}^{t} \left( \frac{-\hat{d}_0}{j!} \right) z_{t-j}.
\]  

(7)

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

\[
    \begin{pmatrix}
        \hat{\rho}(q) & \ldots & \hat{\rho}(q - p + 1) \\
        \vdots & & \vdots \\
        \hat{\rho}(q + p - 1) & \hat{\rho}(q) & \hat{\rho}(q + p)
    \end{pmatrix}
    \phi_0 =
    \begin{pmatrix}
        \hat{\rho}(q + 1) \\
        \vdots \\
        \hat{\rho}(q + p)
    \end{pmatrix},
\]

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 \( T[\hat{\rho}(0), \ldots, \hat{\rho}(p - 1)] \).

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 procedure to compute the autocovariances in (2), the log-likelihood is simply (writing \( z \) for the data vector used for maximization):

\[
    \log L (d, \phi, \theta, \beta, \sigma^2_\epsilon) = -\frac{T}{2} \log(2\pi) - \frac{T}{2} \log|\Sigma| - \frac{1}{2} z' \Sigma^{-1} z. 
\]  

(8)

It is convenient to concentrate \( \sigma^2_\epsilon \) out of the likelihood, starting by writing \( \Sigma = R\sigma^2_\epsilon \):

\[
    \log L (d, \phi, \theta, \beta, \sigma^2_\epsilon) \propto -\frac{T}{2} \log|R| - \frac{T}{2} \log \sigma^2_\epsilon - \frac{1}{2\sigma^2_\epsilon} z'R^{-1}z.
\]

Differentiating with respect to \( \sigma^2_\epsilon \), and solving yields

\[
    \hat{\sigma}^2_\epsilon = T^{-1}z'R^{-1}z, 
\]  

(9)

with concentrated likelihood (CLF):

\[
    \ell_c (d, \phi, \theta, \beta) = -\frac{T}{2} \log(2\pi) - \frac{T}{2} \log|R| - \frac{T}{2} \log [T^{-1}z'R^{-1}z].
\]
When \( f(X, \beta) = X\beta \) it is more convenient to also concentrate \( \beta \) out of the likelihood. The resulting normal profile log-likelihood function becomes:

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

where

\[
\hat{z} = y - X\hat{\beta}, \quad \hat{\beta} = (X' R^{-1} X)^{-1} X' R^{-1} y.
\]

The function used in the maximization procedure is:

\[
-\frac{1}{2} \left\{ T^{-1} \log |R| + \log \sigma^2_\epsilon \right\},
\]

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

\[
\sigma^2_\epsilon = T^{-1} \hat{z}' R^{-1} \hat{z} = T^{-1} e' e,
\]

with \( |R| \) 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 \( \rho_i \) 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, \( \ell_M \), for the regression model with stationary ARFIMA-errors and \( f(X, \beta) = X\beta \):

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

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 \( p = q = d = 0 \):

\[
\hat{\sigma}_\epsilon^2 = \frac{1}{T - k} \tilde{z}' R^{-1} \tilde{z}.
\]

### 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:

\[
\sigma^2_{\epsilon} = \frac{1}{T - k} \sum_{t=1}^{T} e_t^2.
\]

NLS simply maximizes

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

The arfima filter is computed using the Ox function \texttt{diffpow}, see (7), followed by \texttt{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 \( \theta' = [d \phi' \theta'] \). The variance-covariance matrix for the EML (\( \ell = \ell_P \)) and MPL (\( \ell = \ell_M \)) estimates is computed as:

\[
\begin{pmatrix}
-\frac{\partial^2 \ell (\theta)}{\partial \theta \partial \theta'} & 0 \\
0 & \hat{\sigma}_\epsilon^2 (X' R^{-1} X)^{-1}
\end{pmatrix}^{-1}.
\]

The second derivative of \( \ell \) 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 \( \hat{\ell} = \)
  
  EML: \( \ell_c \),
  
  MPL: \( \ell_M \),
  
  NLS: \( f - \frac{T}{2}(1 + \log 2\pi) \),

  
  where \( f \) for NLS is from (16).
- Akaike information criterion
  
  \[
  AIC = -2\hat{\ell} + 2s,
  \]
  
  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 \( \text{AIC}/T \) 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 \( \mu_y \) the known mean.

The \( z \) 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:

\[
(y - \mu)'R^{-1}(y - \mu) = y'R^{-1}y - 2\mu'R^{-1}y + \mu'R^{-1}\mu,
\]

so the known mean case adds \( \mu_y'\mu \) whereas the second case adds \( \hat{\mu}_y'\mu \), 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 \( w, w_t \geq 0 \), 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:

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

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 \( z_t \) the ARFIMA model becomes:

\[
\Phi(L) (1 - L)^d (y_t - x'_t \beta) = \Theta(L) (\varepsilon_t + z'_t \gamma).
\]

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 \( z_t \), \( z'_t \gamma \) is subtracted at each iteration.

This model has the familiar ADL (Autoregressive Distributed-lag model) as a special case, since \( z_t \) 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 \( z = (z_1, \ldots, z_T)' \) denote the observations over the estimation period. Assume \( z_t \) is stationary and \( d > -1 \). The best linear prediction of \( z_{T+h} \) is

\[
\hat{z}_{T+h} = \left[ r(T - 1 + h) \cdots r(h) \right] \{ T [r(0), \ldots, r(T - 1)] \}^{-1} z = q'z,
\]

which consists of the reversed ACovF starting from \( h \), times the original data weighted by their correlations. The solve toeplitz function is used to solve \( Tx = z \) in order to keep storage requirements of order \( T \), see §20.2. The mean square error is

\[
\text{MSE}(\hat{z}_{T+h}) = \hat{\sigma}_z^2 (r(0) - r'q).
\]

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

\[
\hat{y}_{T+h} = q' (y - \mu) + \mu_{T+h} + x'_{T+h} \hat{\beta}.
\]
The Ox code computes all requested forecasts $\hat{z}_h = (z_{T+1}, \ldots, z_{T+h})'$ and their joint variance-covariance matrix, $\text{Cov}(\hat{z}_h)$ simultaneously. $\text{Cov}(\hat{z}_h)$ 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:

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

and

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

(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 $R$ 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

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

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

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

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:

<table>
<thead>
<tr>
<th>coefficients</th>
<th>mean</th>
<th>std.dev</th>
<th>mean bias</th>
</tr>
</thead>
<tbody>
<tr>
<td>with MA inversion</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MA1=0.9</td>
<td>0.89157</td>
<td>0.075471</td>
<td>-0.0084326</td>
</tr>
<tr>
<td>MA2=0.81</td>
<td>0.81967</td>
<td>0.11363</td>
<td>0.0096664</td>
</tr>
<tr>
<td>without inversion</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MA1=0.9</td>
<td>0.93546</td>
<td>0.26420</td>
<td>0.035462</td>
</tr>
<tr>
<td>MA2=0.81</td>
<td>0.86154</td>
<td>0.13834</td>
<td>0.051540</td>
</tr>
</tbody>
</table>

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 $r$ be the standardized autocovariances of the specified process, and $T[r] = PP'$, then
  $$y = \sigma \varepsilon P \varepsilon + \mu,$$

  where $\varepsilon$ are drawings from the standard normal distribution. For small $T$, this is convenient, because $P$ 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:
  $$T[r(0), \ldots, r(T-1)] = PP', \quad z = Pe.$$

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


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


