ECONSTOR
Make Your Publications Visible.

A Service of

ZBW Leibniz-Informationszentrum Wirtschaft
Leibniz Information Centre for Economics

Nielsen, Morten Ørregaard; Popiel, Michał Ksawery

Working Paper

A Matlab program and user's guide for the fractionally cointegrated VAR model

Queen's Economics Department Working Paper, No. 1330

Provided in Cooperation with:
Queen’s University, Department of Economics (QED)

Suggested Citation: Nielsen, Morten Ørregaard; Popiel, Michał Ksawery (2014) : A Matlab program and user's guide for the fractionally cointegrated VAR model, Queen's Economics Department Working Paper, No. 1330, Queen's University, Department of Economics, Kingston (Ontario)

This Version is available at:
https://hdl.handle.net/10419/122041

Standard-Nutzungsbedingungen:

Die Dokumente auf EconStor dürfen zu eigenen wissenschaftlichen Zwecken und zum Privatgebrauch gespeichert und kopiert werden.

Sie dürfen die Dokumente nicht für öffentliche oder kommerzielle Zwecke vervielfältigen, öffentlich ausstellen, öffentlich zugänglich machen, vertreiben oder anderweitig nutzen.

Sofern die Verfasser die Dokumente unter Open-Content-Lizenzen (insbesondere CC-Lizenzen) zur Verfügung gestellt haben sollten, gelten abweichend von diesen Nutzungsbedingungen die in der dort genannten Lizenz gewährten Nutzungsrechte.

Terms of use:

Documents in EconStor may be saved and copied for your personal and scholarly purposes.

You are not to copy documents for public or commercial purposes, to exhibit the documents publicly, to make them publicly available on the internet, or to distribute or otherwise use the documents in public.

If the documents have been made available under an Open Content Licence (especially Creative Commons Licences), you may exercise further usage rights as specified in the indicated licence.

WWW.ECONSTOR.EU

Mitglied der Leibniz-Gemeinschaft

Queen's Economics Department Working Paper No. 1330

A Matlab program and user's guide for the fractionally cointegrated VAR model

Morten Ørregaard Nielsen
Queen's University and CREATES

Michał Ksawery Popiel
Queen's University

Department of Economics
Queen's University
94 University Avenue
Kingston, Ontario, Canada
K7L 3N6

10-2014

A Matlab program and user's guide for the fractionally cointegrated VAR model*

version 1.0

Morten Ørregaard Nielsen†
Queen's University and CREATES
Email: mon@econ.queensu.ca

Michał Ksawery Popiel
Queen's University
Email: popielm@econ.queensu.ca

October 24, 2014

Abstract

This manual describes the usage of the accompanying freely available Matlab program for estimation and testing in the fractionally cointegrated vector autoregressive (FCVAR) model. This program replaces an earlier Matlab program by Nielsen and Morin (2014), and although the present Matlab program is not compatible with the earlier one, we encourage use of the new program.

JEL codes: C22, C32.

Keywords: cofractional process, cointegration rank, computer program, fractional autoregressive model, fractional cointegration, fractional unit root, Matlab, VAR model.

*We are grateful to Federico Carlini, Andreas Noack Jensen, Søren Johansen, Maggie Jones, James MacKinnon, Jason Rhinelander, and Daniela Osterrieder for comments, and to the Canada Research Chairs program, the Social Sciences and Humanities Research Council of Canada (SSHRC), and the Center for Research in Econometric Analysis of Time Series (CREATES, funded by the Danish National Research Foundation) for financial support.

†Corresponding author. If you find any bugs or other problems, please let us know.

Contents

1 Obtaining and using the software

1.1 Disclaimer

We have done our best to make this program as functional and free from errors as possible, but no warranty is given whatsoever. We cannot guarantee that we have been 100% successful in eliminating bugs, so if you find any please let us know.

1.2 Obtaining the Matlab program

The Matlab program can be downloaded from the first author's website at Queen's University:

http://www.econ.queensu.ca/faculty/mon/software/

It is freely available for non-commercial, academic use.

Although the Matlab program can run standalone, one of the functions, `RankTests.m`, makes an external system call to a separately installed program, `fdpval`. This external program is the C++ implementation of a Fortran program used to obtain simulated P-values from MacKinnon and Nielsen (2014). If the user would like P-values for the cointegration rank tests to be automatically calculated, we recommend obtaining this companion program, which is made available by Jason Rhinelander and can be downloaded from:

https://github.com/jagerman/fracdist/releases

It can be either installed or downloaded in a compressed folder. It is important to note where the program is stored or installed, because the Matlab program requires the program location as an input in the estimation options. For example, if the program is stored in the folder `/usr/bin/` on a Linux system, the location variable is defined as follows, `progLoc = '"/usr/bin/fdpval"'`. For details see Sections 4.1 and 4.4.1.

1.3 Citation

If you use this program, or any program or computer codes based on it, we ask that you please cite this document. For example, you could add "The results were obtained using the computer program by Nielsen and Popiel (2014)" in the main text or in a footnote of your paper and then add the following citation to your list of references:

Nielsen, M. Ø. and M. K. Popiel (2014), "A Matlab program and user's guide for the fractionally cointegrated VAR model," QED working paper 1330, Queen's University.

1.4 Using the Matlab program

The use of this program requires a functioning installation of Matlab. Any recent version should work. Unzip the contents of the zip file into any directory which will be the working directory of the program.

The next section describes the FCVAR model and the restricted models that can be estimated with this program. Section 3 describes the functioning of the main program, which is a replication of one of the tables of results in Jones et al. (2014). Importantly, this is the only file that would need to be changed to apply the program for other empirical analyses. Section 4 describes how each of the seven major program files work (each in a separate subsection): (1) defining options, (2) estimation, (3) lag order selection, (4) cointegration rank tests, (5) hypothesis tests, (6) forecasting, and (7) serial correlation tests.

1.5 Version history

v1.0 (October 24, 2014): First publicly available version.

2 The fractionally cointegrated VAR model

The fractionally cointegrated vector autoregressive (FCVAR) model was proposed in Johansen (2008) and analyzed by, e.g., Johansen and Nielsen (2010, 2012). For a time series X_t of dimension p, the fractionally cointegrated VAR model is given in error correction form as

$$\Delta^d X_t = \alpha\beta'\Delta^{d-b}L_b X_t + \sum_{i=1}^{k}\Gamma_i\Delta^d\ L_b^i X_t + \varepsilon_t, \tag{1}$$

where ε_t is p-dimensional $i.i.d.(0,\Omega)$, $d \geq b > 0$, Δ^d is the fractional difference operator, and $L_b = 1-\Delta^b$ is the fractional lag operator.[1]

Model (1) includes the Johansen (1995) CVAR model as the special case $d = b = 1$. Some of the parameters are well-known from the CVAR model and these have the usual interpretations also in the FCVAR model. The most important of these are the long-run parameters α and β, which are $p \times r$ matrices with $0 \leq r \leq p$. The rank r is termed the cointegration, or cofractional, rank. The columns of β constitute the r cointegration (cofractional) vectors such that $\beta' X_t$ are the cointegrating combinations of the variables in the system, i.e. the long-run equilibrium relations. The parameters in α are the adjustment or loading coefficients which represent the speed of adjustment towards equilibrium for each of the variables. The short-run dynamics of the variables are governed by the parameters $\Gamma = (\Gamma_1, \ldots, \Gamma_k)$ in the autoregressive augmentation.

The FCVAR model has two additional parameters compared with the CVAR model, namely the fractional parameters d and b. Here, d denotes the fractional integration order of the observable time series and b determines the degree of fractional cointegration, i.e. the reduction in fractional integration order of $\beta' X_t$ compared to X_t itself. These parameters are estimated jointly with the remaining parameters. This model thus has the same main structure as in the standard CVAR model in that it allows for modeling of both cointegration and adjustment towards equilibrium, but is more general since it accommodates fractional integration and cointegration.

In the next four subsections we briefly describe the accommodation of deterministic terms as well as estimation and testing in the FCVAR model.

2.1 Deterministic terms

There are several ways to accommodate deterministic terms in the FCVAR model (1). The inclusion of the so-called restricted constant was considered in Johansen and Nielsen (2012), and the so-called unrestricted constant term was considered in Dolatabadi et al. (2014). A general formulation that encompasses both models is[2]

$$\Delta^d X_t = \alpha\Delta^{d-b}L_b(\beta' X_t + \rho') + \sum_{i=1}^{k}\Gamma_i\Delta^d\ L_b^i X_t + \xi + \varepsilon_t. \tag{2}$$

The parameter ρ is the so-called restricted constant term (since the constant term in the model is restricted to be of the form $\alpha\rho'$), which is interpreted as the mean level of the long-run equilibria when these are stationary, i.e. $E\beta' X_t + \rho' = 0$. The parameter ξ is the unrestricted constant term, which gives rise to a deterministic trend in the levels of the variables. When $d = 1$ this trend is linear. Thus, the model (2) contains both a restricted constant and an unrestricted constant. In the usual CVAR model, i.e. with $d = b = 1$, the former would be absorbed in the latter, but in the fractional model they can both be present and are interpreted differently. For the representation theory related to (2), and in particular for additional interpretation of the two types of constant terms, see Dolatabadi et al. (2014).

An alternative formulation of deterministic terms was suggested by Johansen and Nielsen (2014), albeit in a simpler model, with the aim of reducing the impact of pre-sample observations of the process. This model is

$$\Delta^d(X_t - \mu) = \alpha\beta'\Delta^{d-b}L_b(X_t - \mu) + \sum_{i=1}^{k}\Gamma_i\Delta^d\ L_b^i(X_t - \mu) + \varepsilon_t, \tag{3}$$

[1] Both the fractional difference and fractional lag operators are defined in terms of their binomial expansion in the lag operator, L. Note that the expansion of L_b has no term in L^0 and thus only lagged disequilibrium errors appear in (1).

[2] In Dolatabadi et al. (2014) the constants are included as $\rho'\pi_t(1)$ and $\xi\pi_t(1)$, where $\pi_t(u)$ denotes coefficients in the binomial expansion of $(1-z)^{-u}$. This is mathematically convenient, but makes no difference in terms of the practical implementation.

which can be derived easily from the unobserved components formulation

$$X_t = \mu + X_t^0, \quad \Delta^d X_t^0 = L_b \alpha\beta' X_t^0 + \sum_{i=1}^{k} \Gamma_i \Delta^d \; L_b^i X_t^0 + \varepsilon_t. \tag{4}$$

The formulation (3), or equivalently (4), includes the restricted constant, which may be obtained as $\rho' = \beta'\mu$. More generally, the level parameter μ is meant to accommodate a non-zero starting point for the first observation on the process, i.e., for X_1. It has the added advantage of reducing the bias arising due to pre-sample behavior of X_t, at least in simple models, even when conditioning on no initial values (see below). For details, see Johansen and Nielsen (2014).

2.2 Maximum likelihood estimation

It is assumed that a sample of length $T+N$ is available on X_t, where N denotes the number of observations used for conditioning, for details see Johansen and Nielsen (2014). The models (1), (2), and (3) are estimated by conditional maximum likelihood, conditional on N initial values, by maximizing the function

$$\log L_T(\lambda) = -\frac{Tp}{2}(\log(2\pi) + 1) - \frac{T}{2} \log\det \left\{ T^{-1} \sum_{t=N+1}^{T+N} \varepsilon_t(\lambda)\varepsilon_t(\lambda)' \right\}, \tag{5}$$

where the residuals are defined as

$$\varepsilon_t(\lambda) = \Delta^d X_t - \alpha\Delta^{d-b} L_b(\beta' X_t + \rho') - \sum_{i=1}^{k} \Gamma_i \Delta^d \; L_b^i X_t - \xi, \quad \lambda = (d, b, \alpha, \beta, \Gamma, \rho, \xi), \tag{6}$$

for model (2), and hence also for submodels of model (2), such as (1), with the appropriate restrictions imposed on ρ and ξ. For model (3) the residuals are

$$\varepsilon_t(\lambda) = \Delta^d (X_t - \mu) - \alpha\beta'\Delta^{d-b} L_b (X_t - \mu) - \sum_{i=1}^{k} \Gamma_i \Delta^d \; L_b^i (X_t - \mu), \quad \lambda = (d, b, \alpha, \beta, \Gamma, \mu). \tag{7}$$

It is shown in Johansen and Nielsen (2012) and Dolatabadi et al. (2014) how, for fixed (d, b), the estimation of model (2) reduces to regression and reduced rank regression as in Johansen (1995). In this way the parameters $(\alpha, \beta, \Gamma, \rho, \xi)$ can be concentrated out of the likelihood function, and numerical optimization is only needed to optimize the profile likelihood function over the two fractional parameters, d and b. In model (3) we can similarly concentrate the parameters (α, β, Γ) out of the likelihood function resulting in numerical optimization over (d, b, μ), making the estimation of model (3) slightly more involved numerically than that of model (2).

For model (2) with $\xi = 0$, Johansen and Nielsen (2012) shows that asymptotic theory is standard when $b < 0.5$, and for the case $b > 0.5$ asymptotic theory is non-standard and involves fractional Brownian motion of type II. Specifically, when $b > 0.5$, Johansen and Nielsen (2012) shows that under i.i.d. errors with suitable moment conditions, the conditional maximum likelihood parameter estimates $(\hat{d}, \hat{b}, \hat{\alpha}, \hat{\Gamma}_1, \ldots, \hat{\Gamma}_k)$ are asymptotically Gaussian, while $(\hat{\beta}, \hat{\rho})$ are locally asymptotically mixed normal. These results allow asymptotically standard (chi-squared) inference on all parameters of the model, including the cointegrating relations and orders of fractionality, using quasi-likelihood ratio tests. As in the CVAR model, see Johansen (1995), the same results hold for the same parameters in the full models (2) and (3), whereas the asymptotic distribution theory for the remaining parameters, ξ and μ, is currently unknown.

2.3 Cointegration rank tests

Letting $\Pi = \alpha\beta'$, the likelihood ratio (LR) test statistic of the hypothesis $\mathcal{H}_r : \text{rank}(\Pi) = r$ against $\mathcal{H}_p : \text{rank}(\Pi) = p$ is of particular interest because it deals with an important empirical question. This statistic is often denoted the "trace" statistic. Let $\theta = (d, b)$ for model (2) and $\theta = (d, b, \mu)$ for model (3) denote the parameters for which the likelihood is numerically maximized. Then let $L(\theta, r)$ be the profile likelihood function given rank r, where (α, β, Γ), and possibly (ρ, ξ) if appropriate, have been concentrated

out by regression and reduced rank regression; see Johansen and Nielsen (2012) and Dolatabadi et al. (2014) for details.

The profile likelihood function is maximized both under the hypothesis $\mathcal{H}_r$ and under $\mathcal{H}_p$ and the LR test statistic is then $\mathrm{LR}_T(q) = 2\log(L(\hat{\theta}_p, p)/L(\hat{\theta}_r, r))$, where

$$L(\hat{\theta}_p, p) = \max_{\theta} L(\theta, p), \quad L(\hat{\theta}_r, r) = \max_{\theta} L(\theta, r),$$

and $q = p - r$. This problem is qualitatively different from that in Johansen (1995) since the asymptotic distribution of $\mathrm{LR}_T(q)$ depends qualitatively (and quantitatively) on the parameter b. In the case with $0 < b < 1/2$ (sometimes known as "weak cointegration"), $\mathrm{LR}_T(q)$ has a standard asymptotic distribution, see Johansen and Nielsen (2012, Theorem 11(ii)), namely

$$\mathrm{LR}_T(q) \xrightarrow{D} \chi^2(q^2), \quad 0 < b < 1/2. \tag{8}$$

On the other hand, when $1/2 < b \leq d$ ("strong cointegration"), asymptotic theory is nonstandard and

$$\mathrm{LR}_T(q) \xrightarrow{D} \mathrm{Tr}\left\{ \int_0^1 \mathsf{d}W(s) F(s)' \left(\int_0^1 F(s)F(s)'\mathsf{d}s \right)^{-1} \int_0^1 F(s)\mathsf{d}W(s)' \right\}, \quad b > 1/2, \tag{9}$$

where the vector process $\mathsf{d}W$ is the increment of ordinary (non-fractional) vector standard Brownian motion of dimension $q = p - r$. The vector process F depends on the deterministics in a similar way as in the CVAR model in Johansen (1995), although the fractional orders complicate matters. The following cases have been derived in the literature:

1. When no deterministic term is in the model, $F(u) = W_b(u)$, where $W_b(u) = \Gamma(b)^{-1} \int_0^u (u-s)^{b-1} \mathsf{d}W(s)$ is vector fractional Brownian motion of type II, see Johansen and Nielsen (2012, Theorem 11(i)).

2. When only the restricted constant term is included in model (2), $F(u) = (W_b(u)', u^{-(d-b)})'$, see Johansen and Nielsen (2012, Theorem 11(iv)) for the result with $d = b$ and an earlier working paper version for the general result.

3. In model (3) the same result as in bullet 2. holds because $\beta'\mu = \rho'$ is the restricted constant and $\beta_\perp'\mu$ has no influence on the asymptotic distribution (in a similar way to X_0 in a random walk).

4. When both the restricted and unrestricted constants are included in model (2) with $d = 1$,

$$F_i(u) = W_{b,i}(u) - \int_0^1 W_{b,i}(u)\mathsf{d}u, \ i = 1, ..., q-1,$$
$$F_q(u) = u^b - \int_0^1 u^b \mathsf{d}u = u^b - 1/(b+1),$$
$$F_{q+1}(u) = u^{b-1} - \int_0^1 u^{b-1}\mathsf{d}u = u^{b-1} - 1/b,$$

see Dolatabadi et al. (2014).

Importantly, the asymptotic distribution (9) of the test statistic $\mathrm{LR}_T(q)$ depends on both b and $q = p - r$. The dependence on the unknown (true value of the) scalar parameter b complicates empirical analysis compared to the CVAR model. Generally, the distribution (9) would need to be simulated on a case-by-case basis. However, for model (1) and for model (2) with $d = b$ and $\xi = 0$, and hence also for model (3) with $d = b$ in light of bullet 3. above, computer programs for computing asymptotic critical values and asymptotic P values for the LR cointegration rank tests based on numerical distribution functions, are made available by MacKinnon and Nielsen (2014). Their computer programs are incorporated in the present program for the relevant cases/models as discussed and illustrated below.

2.4 Restricted models

Note that a reduced rank restriction has already been imposed on models (1)–(3), where the coefficient matrix $\Pi = \alpha\beta'$ has been restricted to rank $r \leq p$. Other restrictions on the model parameters can be considered as in Johansen (1995). The most interesting restrictions from an economic theory point of view would likely be restrictions on the adjustment parameters α and cointegration vectors β.

However, rather than using a switching algorithm as in Johansen (1995, Chapter 7), we formulate hypotheses as

$$R_\psi \psi = r_\psi, \tag{10}$$
$$R_\alpha \text{vec}(\alpha) = 0, \tag{11}$$
$$R_\beta \text{vec}(\beta^*) = r_\beta, \tag{12}$$

with $\beta^* = (\beta', \rho')'$, and optimize the likelihood function numerically subject to the restrictions.

The only limitation on the linear restrictions that can be imposed on (d, b, α, β^*) in (10)–(12) is that only homogenous restrictions can be imposed on $\text{vec}(\alpha)$ in (11). Otherwise, any combination of linear restrictions can be imposed on these parameters. For now, the remaining parameters cannot be restricted.

Note that, when the restricted constant term ρ is included in the model, restrictions on β and ρ must be written in the form given by (12). This is without loss of generality.

The restrictions in (10)–(12) above can be implemented individually or simultaneously in the Matlab program. The next section provides an example session illustrating the use of the program with a step-by-step description of a typical empirical analysis, including several restricted models in Section 3.6.

3 Example session: replication_JNP2014.m

The main file is `replication_JNP2014.m` and it serves as an example of what a typical session of estimation, testing, and forecasting can include. This code replicates "Table 4: FCVAR results for Model 1" from Jones et al. (2014) and follows the empirical procedure developed in that paper. This procedure includes the following steps:

1. Importing data
2. Choosing estimation options
3. Lag selection
4. Cointegration rank selection
5. Model estimation
6. Hypothesis testing
7. Forecasting (not included in JNP, 2014)

It is important to note that all necessary commands for file execution and option modification can be called from this script. All other files contained in the package (described in detail in the next section) do not require any modification by the user.

To accommodate the sequential nature of the procedure, the main file is broken up into *code sections*[3]. These *code sections*, known as *cells* in previous versions of Matlab, allow the user to execute specific parts of a script individually. Each of the *code sections* are delimited by a double comment `%%` and the section header.

3.1 Importing data

The first step is importing the data. Executing the code in Listing 1, shown below, assigns the data from the file `data_JNP2014.csv` to a matrix called `data`.

Listing 1: Importing data

```
% -------- Import Data ----------%
clear all;
data = csvread('data_JNP2014.csv',1); % skip first row because var names.

% data for each model.
x1 = data(:, [1 3 5]);
x2 = data(:, [2 3 5]);
x3 = data(:, [1 2 3 5]);
x4 = data(:, [1 3 4 5 6]);
x5 = data(:, [2 3 4 5 6]);
x6 = data(:, [1 2 3 4 5 6]);
```

The columns contain the following variables: (1) aggregate support for the Liberal party, (2) aggregate support for the Conservative party, (3) Canadian 3-month T-bill rates, (4) US 3-month T-bill rates, (5) Canadian unemployment rate, and (6) US unemployment rate. Since each of the models in JNP (2014) contain different combinations of these variables, the relevant columns of `data` for each model are assigned to different matrices of variables named `x1` through `x6`.

3.2 Choosing options

Once the data is imported, the user sets the program options. The script contains two sets of options: variables set for function arguments in the script itself and model/estimation related options. Listing 2 shows the first of set of options.

[3]For more information see http://www.mathworks.com/help/matlab/matlab_prog/run-sections-of-programs.html

Listing 2: Initialization of local variables

```
%% -------- INITIALIZATION ----------%
p                = size(x1, 2); % system dimension.
kmax             = 3;     % maximum number of lags for VECM.
order            = 12;    % number of lags for white noise test in lag selection.
printWNtest      = 1;     % to print results of  white noise tests post-estimation.
```

The variable `kmax` determines the highest lag order for the sequential testing that is performed in the lag selection, whereas `p` is the dimension of the system. The other variables are self-explanatory.

The next set of initialization commands, shown in Listing 3, assign values to the variables contained in object `opt` defined by the class `EstOptions`.

Listing 3: Choosing estimation options

```
% -------- Choosing estimation options ----------%
opt = EstOptions; % Define variable to store Estimation Options (object).
opt.dbMin        = 0.01; % lower bound for d,b.
opt.dbMax        = 2.00; % upper bound for d,b.
opt.unrConstant  = 0; % include an unrestricted constant? 1 = yes, 0 = no.
opt.rConstant    = 0; % include a restricted constant? 1 = yes, 0 = no.
opt.levelParam   = 1; % include level parameter? 1 = yes, 0 = no.
opt.constrained  = 0; % impose restriction dbMax >= d >= b >= dbMin ?
                      % 1 = yes, 0 = no.
opt.restrictDB   = 1; % impose restriction d=b ? 1 = yes, 0 = no.
opt.db0          = [.8 .8]; % set starting values for optimization algorithm.
opt.N            = 0; % number of initial values to condition upon.
opt.print2screen = 1; % print output.
opt.printRoots   = 1; % print roots of characteristic polynomial.
opt.plotRoots    = 1; % plot roots of characteristic polynomial.
opt.gridSearch   = 1; % For more accurate estimation, perform the grid search.
                                          % This will make estimation take longer.
opt.plotLike     = 1; % Plot the likelihood (if gridSearch = 1).
opt.progress     = 1; % Show grid search progress indicator waitbar.
opt.updateTime   = 5; % How often progress is updated (seconds).

% Linux example:
opt.progLoc = '"/usr/bin/fdpval"';  % location path with program name
                                    % of fracdist program, if installed
                                    % Note: use both single (outside) and double
                                    % quotes (inside). This is especially
                                       important
                                    % if path name has spaces.
DefaultOpt = opt; % Store the options for restoring them in between hypothesis
    tests.
```

The first line initializes the object `opt` and assigns all of the default options set in `EstOptions`. The user can see the full set of options by typing `EstOptions` (or `opt` after initialization) in the command line. Listing 3 shows how to easily change any of the default options. Defining the program options in this way allows the user to create and store several option objects with different attributes. This can be very convenient when, for example, performing the same hypothesis tests on different data sets.

The set of available options can be broken into several categories: numerical optimization, model deterministics and restrictions, output, grid search, and P-values for the rank test. We recommend that only advanced users make changes to the numerical optimization options. Adding deterministics requires setting the variable corresponding to the type of deterministic component to 1. For instance, in the present example, a model estimated with options `opt` will include the level parameter μ but no restricted or unrestricted constant. Output variables refer to either printing or plotting various information post-estimation and usually take values 1 or 0 (on or off). For example, if the user is not interested in the estimates of Γ, they can be suppressed by setting `opt.printGammas = 0`.

An important feature in this package is the ability to pre-estimate by using a grid search. If the user selects this option, they can view progress by setting `opt.progress` to 1 (waitbar) or 2 (output in command line). The minimum frequency of these updates is set by `opt.updateTime`. The user also has the option (`opt.plotLike`) to view a plot of the likelihood over d and/or b after the grid search completes.

In order to automatically obtain P-values for cointegration rank tests when $b > 0.5$, the user needs to download and install the necessary program (see Section 1). The last option, `opt.progLoc`, identifies the location of that program. After all options have been set, line 43 stores them in `DefaultOpt` so that the user can recall them at any point in the estimation. This is particularly useful if the user wants to change only a few options in between estimations.

3.3 Lag-order selection

Once the options are set, the user moves to the next step, which involves choosing the appropriate lag order. The relevant information is obtained with a call to `LagSelect.m`, shown in Listing 4, which performs estimation of models with lag-orders from 0 to `kmax`. The program performs lag selection on the full-rank unrestricted model.

Listing 4: Lag selection

```
45 %% --------- LAG SELECTION ---------- %
46 LagSelect(x1, kmax, p, order, opt);
```

The output generated by this function is shown below.

```
-------------------------------------------------------------------------------------------
                          Lag Selection Results
-------------------------------------------------------------------------------------------

k r   d     b       LogL      LR     pv     AIC       BIC     pmvQ pQ1  pLM1 pQ2  pLM2 pQ3  pLM3
3 3 0.256 0.256   461.22   16.91 0.050  -842.44   -692.21 0.00 0.45 0.40 0.48 0.87 0.63 0.35
2 3 0.581 0.581   452.77   20.59 0.015  -843.53   -727.11 0.00 0.69 0.45 0.29 0.75 0.54 0.40
1 3 1.043 1.043   442.47   56.99 0.000  -840.94   -758.31 0.00 0.75 0.52 0.15 0.58 0.34 0.18
0 3 1.036 1.036   413.97    0.00 0.000  -801.95   -753.12 0.00 0.01 0.01 0.00 0.08 0.37 0.17
```

Estimates of d and b are reported for each lag (k) with rank (r) set to the number of variables in the system. Note that in this example the restriction $d = b$ has been imposed. The log-likelihood for each lag is shown in column `LogL`. The likelihood ratio test-statistic `LR` is for the null hypothesis $\Gamma_k = 0$ with P-value reported in column `pv`. This is followed by AIC and BIC information criteria. The next set of columns provides P-values for white noise tests on the residuals. The first P-value, `pmvQ`, is for the multivariate Q-test followed by univarite Q-tests as well as LM tests on the p individual residuals; that is, `pQ1` and `pLM1` are the P-values for the residuals in the first equation, `pQ2` and `pLM2` are for the residuals in the second equation, and so on.

3.4 Cointegration rank testing

The user now chooses the lag-order based on the information provided above and can move to the next step, which is cointegration rank testing. The next code section is shown in listing 5. The user first assigns the lag augmentation, $k = 2$ in this case, and then calls the function `RankTests.m`.

Listing 5: Cointegration rank testing

```
47 %% --------- COINTEGRATION RANK TESTING ---------- %
48 k = 2;
49 RankTests(x1, k, opt);
```

Executing the code in Listing 5 produces the following output.

```
-------------------------------------------------------------------------------
                Likelihood Ratio Tests for Cointegrating Rank
-------------------------------------------------------------------------------
```

```
Dimension of system:       3     Number of observations in sample:          316
Number of lags:            2     Number of observations for estimation:     316
Restricted constant:      No     Initial values:                              0
Unestricted constant:     No     Level parameter:                           Yes
-------------------------------------------------------------------------------
Rank    d     b    Log-likelihood  LR statistic  P-value
 0    0.643 0.643         440.040        25.454    0.026
 1    0.569 0.569         451.174         3.186    0.828
 2    0.576 0.576         452.707         0.120    0.948
 3    0.581 0.581         452.767          ----    ----
-------------------------------------------------------------------------------
```

The first block of output provides a summary of the model specification. The second block provides the test results relevant for selecting the appropriate rank. The table is meant to be read sequentially from lowest to highest rank, i.e. from top to bottom. Since we can reject the null of rank 0 against the alternative of rank 3 we move to the test of rank 1 against rank 3. This test fails to reject with a P-value of 0.828, so this is the appropriate choice in this case.

3.5 Unrestricted model estimation

With the rank and lag selected, the user can now move to the next code section, shown in Listing 6.

Listing 6: Unrestricted model estimation

```
%% --------- UNRESTRICTED MODEL ESTIMATION ---------- %
r=1;

opt1 = DefaultOpt;

m1 = FCVARestn(x1, k, r, opt1); % This model is now in the structure m1.

mv_wntest(m1.Residuals, order, printWNtest);
```

Here the user first specifies the choice for the rank based on the previously performed cointegrating rank tests (thus setting $r = 1$ in this example). Next, the default options set in the initialization, see Section 3.2, are assigned to `opt1`, which is used as an argument in the call to the function `FCVARestn.m`. This function is the main part of the program since it performs the estimation of the parameters, obtains model residuals and standard errors, and calculates many other relevant components such as the number of free parameters and the roots of the characteristic polynomial. If `opt1.print2screen=1` then, in addition to storing all of these results in the Matlab structure `m1`, the function outputs the estimation results to the command window. To see a list of variables stored in `m1`, the user can type `m1` in the command line. After the unrestricted model has been estimated, this code section concludes with a call to `mv_wntest.m`, which performs a series of white noise tests on the residuals and prints the output in the command window.

The program output is shown below. It begins with a table summarizing relevant model specifications and then the coefficients and their standard errors. The roots of the characteristic polynomial are displayed at the bottom.

```
-------------------------------------------------------------------------------
                    Fractionally Cointegrated VAR: Estimation Results
-------------------------------------------------------------------------------
Dimension of system:       3     Number of observations in sample:          316
Number of lags:            2     Number of observations for estimation:     316
Restricted constant:      No     Initial values:                              0
Unestricted constant:     No     Level parameter:                           Yes
-------------------------------------------------------------------------------
Cointegrating rank:            1  AIC:            -848.348
Log-likelihood:          451.174  BIC:            -746.943
log(det(Omega_hat)):     -11.369  Free parameters:      27
```

```
--------------------------------------------------------------------------------
    Fractional parameters:
--------------------------------------------------------------------------------
    Coefficient                Estimate                 Standard error
--------------------------------------------------------------------------------
         d                        0.569                      0.049
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    Cointegrating equations (beta):
--------------------------------------------------------------------------------
      Variable        CI equation 1
--------------------------------------------------------------------------------
        Var1              1.000
        Var2              0.111
        Var3             -0.240
--------------------------------------------------------------------------------
Note: Identifying restriction imposed.
--------------------------------------------------------------------------------
    Adjustment matrix (alpha):
--------------------------------------------------------------------------------
      Variable        CI equation 1
--------------------------------------------------------------------------------
        Var 1            -0.180
         SE 1         (   0.064  )
        Var 2             0.167
         SE 2         (   0.194  )
        Var 3             0.037
         SE 3         (   0.014  )
--------------------------------------------------------------------------------
Note: Standard errors in parenthesis.
--------------------------------------------------------------------------------
    Long-run matrix (Pi):
--------------------------------------------------------------------------------
      Variable         Var 1          Var 2          Var 3
--------------------------------------------------------------------------------
      Var 1           -0.180         -0.020          0.043
      Var 2            0.167          0.019         -0.040
      Var 3            0.037          0.004         -0.009
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
    Level parameter (mu):
--------------------------------------------------------------------------------
        Var 1            -0.345
         SE 1         (   0.069  )
        Var 2            11.481
         SE 2         (   0.548  )
        Var 3            -2.872
         SE 3         (   0.033  )
--------------------------------------------------------------------------------
Note: Standard errors in parenthesis (from numerical Hessian) but asymptotic
distribution is unknown.
--------------------------------------------------------------------------------
    Lag matrix 1 (Gamma_1):
```

```
-------------------------------------------------------------------------------
      Variable         Var 1          Var 2          Var 3
-------------------------------------------------------------------------------
      Var 1             0.276         -0.032         -0.510
       SE 1        (    0.160  )   (    0.026  )   (    0.513  )
      Var 2            -0.148          1.126         -3.285
       SE 2        (    0.378  )   (    0.196  )   (    1.975  )
      Var 3            -0.052          0.008          0.711
       SE 3        (    0.022  )   (    0.005  )   (    0.170  )
-------------------------------------------------------------------------------
Note: Standard errors in parenthesis.
-------------------------------------------------------------------------------
    Lag matrix 2 (Gamma_2):
-------------------------------------------------------------------------------
      Variable         Var 1          Var 2          Var 3
-------------------------------------------------------------------------------
      Var 1             0.566          0.106          0.609
       SE 1        (    0.182  )   (    0.045  )   (    0.612  )
      Var 2             0.493         -0.462          0.450
       SE 2        (    0.562  )   (    0.198  )   (    2.627  )
      Var 3            -0.039         -0.020          0.318
       SE 3        (    0.032  )   (    0.008  )   (    0.143  )
-------------------------------------------------------------------------------
Note: Standard errors in parenthesis.
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
    Roots of the characteristic polynomial
-------------------------------------------------------------------------------
    Number     Real part    Imaginary part       Modulus
-------------------------------------------------------------------------------
       1         -2.893          0.000            2.893
       2         -1.522          0.000            1.522
       3          1.010          0.927            1.371
       4          1.010         -0.927            1.371
       5          1.108          0.000            1.108
       6          1.000          0.000            1.000
       7          1.000          0.000            1.000
       8          0.944          0.261            0.980
       9          0.944         -0.261            0.980
-------------------------------------------------------------------------------
```

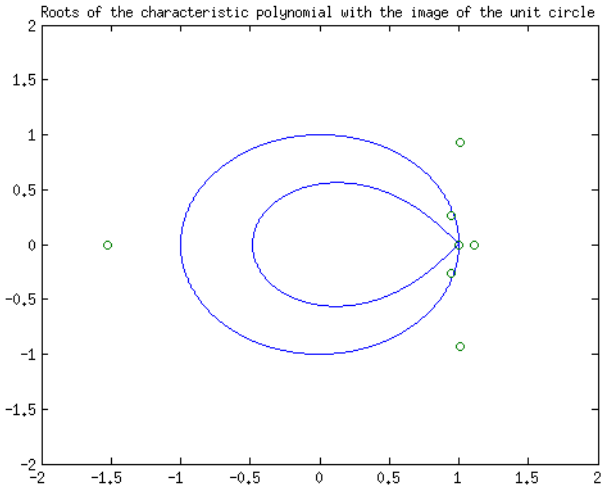
In addition to the coefficient estimates, we are also interested in testing the model residuals for serial correlation. The results of the white noise tests, called in the last line of Listing 6, are shown below. For each residual both the Q- and LM-test statistics and their P-values are reported, in addition to the multivariate Q-test and P-value in the first line of the table. From the output of this table we can conclude that there does not appear to be any problems with serial correlation in the residuals.

```
       White Noise Test Results
-------------------------------------------
Variable |       Q  P-val |      LM  P-val  |
-------------------------------------------
Multivar |  97.868  0.747 |     ----  ----  |
Var1     |   9.301  0.677 |  11.238  0.509  |
Var2     |  14.443  0.273 |   8.566  0.739  |
```

```
Var3      |  10.596   0.564 |  12.269   0.424  |
-------------------------------------------------
```

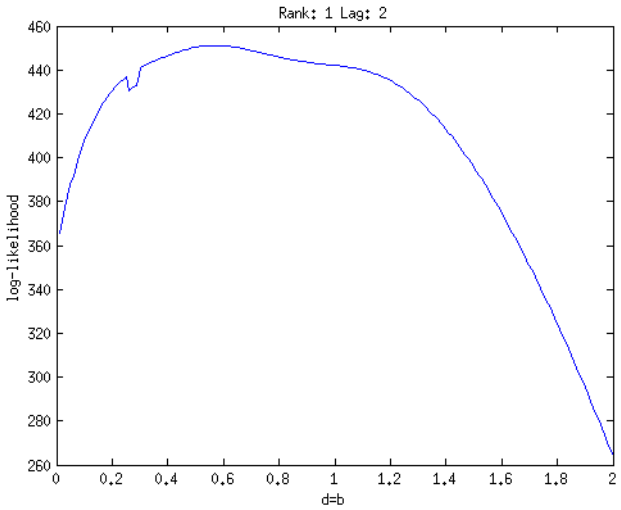
Because `opt.plotRoots = 1` in Listing 3, the roots of the characteristic polynomial is also plotted along with the unit circle and the transformed unit circle, $\mathbb{C}_{\hat{b}}$, see Johansen (2008). The plot is shown in Figure 1. Note that the axes of the plot are fixed, and therefore very large roots may not be shown in the plot (in this example, the real root `-2.893`). This should not be a problem since such roots will always be well outside the transformed unit circle.

Figure 1: Roots of characteristic polynomial



Furthermore, the estimation was performed with the grid search and the plot option selected, i.e. with `opt.gridSearch = 1` and `opt.plotLike = 1`, which produces a plot of the log-likelihood. The plot for this model is shown in Figure 2.

Figure 2: Plot of log-likelihood



The complete results for the unrestricted model are stored in the Matlab structure `m1` and can be accessed anytime. For instance, if the user would like to perform a more careful analysis of the residuals they are stored in `m1.Residuals`.

3.6 Hypothesis testing

We now move into the hypothesis testing section of the code where we can test several restricted models and perform inference. For restricted model estimation the grid search option is switched off because computation can be very slow, especially in the presence of the level parameter. However, if the user wishes to verify the accuracy of the results or if estimates are close to the upper or lower bound, the grid search option can resolve these issues and give the user additional insight about the behaviour of the likelihood.

All hypotheses are defined as shown in (10)–(12). The first hypothesis test is $\mathscr{H}_d^1$ (for precise definitions of each hypothesis, please see Jones et al. (2014)), and it is shown in Listing 7.

Listing 7: Hypothesis $\mathscr{H}_d^1$

```
%% --------- IMPOSE RESTRICTIONS AND TEST THEM ---------- %

DefaultOpt.gridSearch = 0;      % turn off grid search for restricted models
                                % because it's too intensive.

%% Test restriction that d=b=1.
opt1 = DefaultOpt;
opt1.R_psi = [1 0];
opt1.r_psi = 1;

m1r1 = FCVARestn(x1, k, r, opt1); % This restricted model is now in the structure
                                  % m1r1.

mv_wntest(m1r1.Residuals, order, printWNtest);

Hdb = HypoTest(m1, m1r1); % Test the null of m1r1 against the alternative m1 and
                          % store the results in the structure Hdb.
```

Here we test the CVAR model (null hypothesis $d = b = 1$) against the FCVAR model (alternative hypothesis $d = b \neq 1$). Since `opt1.restrictDB=1` was selected in the choice of options in Listing 3, the restriction that $d = b$ is already imposed. Thus, the user needs to only impose an additional restriction that either d or b is equal to one. In this example, the restriction that $d = 1$ is imposed by setting `opt1.R_psi = [1 0]` and `opt1.r_psi = 1`, but the result would be the same if $b = 1$ were imposed instead. The restricted model is then estimated and the results are stored in the Matlab structure `m1r1`. As before, the user can perform a series of white noise tests on the residuals by calling the `mv_wntest.m` function. The next step is to perform the actual test. With the results structures from the restricted and unrestricted models, the user can call the function `HypoTest.m` and perform an LR test. This function takes the two model result structures as inputs, automatically compares the number of free parameters to obtain the degrees of freedom, computes the LR test statistic, and displays the output. The results of this test are then stored in the Matlab structure `Hdb` and can be accessed at any time.

Since the output of the estimated model and the white noise tests are similar to the previous example, we only show the output from the hypothesis test.

```
Unrestricted log-likelihood: 451.174
Restricted log-likelihood:   442.027
Test results (df = 1):
LR statistic:   18.295
P-value:   0.000
```

The log-likelihoods from both models are reported, along with the degrees of freedom, the LR test statistic, and its P-value. In this case the test clearly rejects the null hypothesis that the model is a CVAR. For more significant digits, or to access any of these values from the command window, the user can type `Hdb`.

The next hypothesis of interest is $\mathscr{H}_\beta^1$, which is a zero restriction on the first element of the cointegration vector.

Listing 8: Hypothesis $\mathscr{H}_\beta^1$

```
%% Test restriction that political variables do not enter the cointegrating
    relation(s).
opt1 = DefaultOpt;
opt1.R_Beta = [1 0 0];

m1r2 = FCVARestn(x1, k, r, opt1); % This restricted model is now
                                  % in the structure m1r2.

mv_wntest(m1r2.Residuals, order, printWNtest);

Hbeta1 = HypoTest(m1, m1r2);    % Test the null of m1r2 against the alternative m1
                                % and store the results in the structure Hbeta1.
```

Since the object `opt1` has the restriction $d = b = 1$ stored, the first step is to reset the options to default. The restriction on β is then specified as in (12). There are two things to note here. First, the column length of R_β must equal $p_1 r$, where $p_1 = p + 1$ if a restricted constant is present and $p_1 = p$ otherwise; recall that p is the number of variables in the system and r is the number of cointegrating vectors. Second, zero restrictions are the default and automatically imposed when r_β is empty. Therefore, the user only needs to specify r_β if it includes non-zero elements. Recall that for restrictions on α only $r_\alpha = 0$ is allowed so that there is no need to specify r_α. As before, the restricted model is estimated with results stored in `m1r2`, the residuals are tested for white noise, and the model under the null is tested against the unrestricted model `m1` with results stored in `Hbeta1`.

Again, since the estimation output is similar to the first example, we only show the results of the hypothesis test here. With a P-value close to zero, this hypothesis is also strongly rejected.

```
Unrestricted log-likelihood: 451.174
Restricted log-likelihood:   444.395
Test results (df = 1):
LR statistic:   13.557
P-value:   0.000
```

Next, we move to tests on α.

Listing 9: Hypothesis $\mathscr{H}_\alpha^1$

```
%% Test restriction that political variable is long-run exogenous.
opt1 = DefaultOpt;
opt1.R_Alpha = [1 0 0];

m1r3 = FCVARestn(x1, k, r, opt1); % This restricted model is now in the structure
                                  % m1r3.

mv_wntest(m1r3.Residuals, order, printWNtest);

Halpha1 = HypoTest(m1, m1r3);   % Test the null of m1r3 against the alternative m1
                                % and store the results in the structure Halpha1.
```

Again we first reset `opt1` to the default options to clear previously imposed restrictions. Note that, if it were the case that we failed to reject $\mathscr{H}_\beta^1$ and wanted to leave it imposed while adding a restriction on α, we could either omit the first line `opt1 = DefaultOpt;`, or we could replace it with `opt1 = m1r2.options;`. The latter assignment is preferred in this case because it is explicit about which model options we are leaving imposed.

The hypothesis $\mathscr{H}_\alpha^1$ is tested in the exact same way as before, only now we are changing the variable R_α instead of R_β. The results are shown below and we can see that this hypothesis is also rejected.

```
Unrestricted log-likelihood: 451.174
Restricted log-likelihood:   446.086
```

```
Test results (df = 1):
LR statistic:   10.176
P-value:   0.001
```

We next move to the remaining long-run exogeneity tests, $\mathscr{H}_\alpha^2$ and $\mathscr{H}_\alpha^3$, shown in Listings 10 and 11, respectively. The results of the tests are shown below each listing.

Listing 10: Hypothesis $\mathscr{H}_\alpha^2$

```
%% Test restriction that interest-rate is long-run exogenous.
opt1 = DefaultOpt;
opt1.R_Alpha = [0 1 0];

m1r4 = FCVARestn(x1, k, r, opt1); % This restricted model is now in the
                                  % structure m1r4.

mv_wntest(m1r4.Residuals, order, printWNtest);

Halpha2 = HypoTest(m1, m1r4);   % Test the null of m1r4 against the alternative m1
                                % and store the results in the structure Halpha2.
```

Output:

```
Unrestricted log-likelihood: 451.174
Restricted log-likelihood:   450.857
Test results (df = 1):
LR statistic:   0.633
P-value:   0.426
```

Listing 11: Hypothesis $\mathscr{H}_\alpha^3$

```
%% Test restriction that unemployment is long-run exogenous.
opt1 = DefaultOpt;
opt1.R_Alpha = [0 0 1];
k=2; r=1;
m1r5 = FCVARestn(x1, k, r, opt1); % This restricted model is now in the structure
                                  % m1r5.

mv_wntest(m1r5.Residuals, order, printWNtest);

Halpha3 = HypoTest(m1, m1r5);   % Test the null of m1r5 against the alternative m1
                                % and store the results in the structure Halpha3.
```

Output:

```
Unrestricted log-likelihood: 451.174
Restricted log-likelihood:   446.184
Test results (df = 1):
LR statistic:   9.979
P-value:   0.002
```

The only hypothesis that we fail to reject is $\mathscr{H}_\alpha^2$, under which interest rates are long-run exogenous. Since this is the final restricted model, we provide the full estimation output. Note from the output that $\alpha_2 = 0$ as imposed by the restriction.

```
---------------------------------------------------------------------------
                    Fractionally Cointegrated VAR: Estimation Results
---------------------------------------------------------------------------
Dimension of system:       3      Number of observations in sample:          316
```

```
Number of lags:            2      Number of observations for estimation:     316
Restricted constant:      No      Initial values:                              0
Unestricted constant:     No      Level parameter:                           Yes
-------------------------------------------------------------------------------
Cointegrating rank:            1  AIC:            -849.715
Log-likelihood:          450.857  BIC:            -752.065
log(det(Omega_hat)):     -11.367  Free parameters:      26
-------------------------------------------------------------------------------
    Fractional parameters:
-------------------------------------------------------------------------------
    Coefficient                Estimate                 Standard error
-------------------------------------------------------------------------------
         d                        0.575                       0.048
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
    Cointegrating equations (beta):
-------------------------------------------------------------------------------
      Variable        CI equation 1
-------------------------------------------------------------------------------
        Var1             1.000
        Var2             0.106
        Var3            -0.182
-------------------------------------------------------------------------------
    Adjustment matrix (alpha):
-------------------------------------------------------------------------------
      Variable        CI equation 1
-------------------------------------------------------------------------------
        Var 1           -0.188
         SE 1         (   0.065  )
        Var 2            0.000
         SE 2         (   0.000  )
        Var 3            0.039
         SE 3         (   0.014  )
-------------------------------------------------------------------------------
Note: Standard errors in parenthesis.
-------------------------------------------------------------------------------
    Long-run matrix (Pi):
-------------------------------------------------------------------------------
      Variable         Var 1          Var 2          Var 3
-------------------------------------------------------------------------------
      Var 1           -0.188         -0.020          0.034
      Var 2            0.000          0.000          0.000
      Var 3            0.039          0.004         -0.007
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
    Level parameter (mu):
-------------------------------------------------------------------------------
        Var 1           -0.310
         SE 1         (   0.067  )
        Var 2           11.538
         SE 2         (   0.553  )
        Var 3           -2.873
         SE 3         (   0.033  )
```

Note: Standard errors in parenthesis (from numerical Hessian) but asymptotic distribution is unknown.

Lag matrix 1 (Gamma_1):

Variable	Var 1	Var 2	Var 3
Var 1	0.269	-0.032	-0.512
SE 1	(0.157)	(0.026)	(0.507)
Var 2	-0.013	1.115	-3.001
SE 2	(0.345)	(0.189)	(1.909)
Var 3	-0.053	0.008	0.694
SE 3	(0.022)	(0.005)	(0.164)

Note: Standard errors in parenthesis.

Lag matrix 2 (Gamma_2):

Variable	Var 1	Var 2	Var 3
Var 1	0.570	0.104	0.586
SE 1	(0.184)	(0.044)	(0.606)
Var 2	0.685	-0.371	0.223
SE 2	(0.508)	(0.159)	(2.510)
Var 3	-0.043	-0.020	0.330
SE 3	(0.032)	(0.008)	(0.138)

Note: Standard errors in parenthesis.

Roots of the characteristic polynomial

Number	Real part	Imaginary part	Modulus
1	-2.710	0.000	2.710
2	-1.498	0.000	1.498
3	1.129	0.939	1.469
4	1.129	-0.939	1.469
5	1.098	0.000	1.098
6	1.000	0.000	1.000
7	1.000	-0.000	1.000
8	0.934	0.281	0.976
9	0.934	-0.281	0.976

White Noise Test Results

Variable	Q	P-val	LM	P-val
Multivar	97.665	0.752	----	----
Var1	9.084	0.696	11.267	0.506

```
Var2     |  14.931  0.245 |   9.338  0.674  |
Var3     |  10.729  0.552 |  12.241  0.426  |
-------------------------------------------
```

Sometimes it is the case that the model output is not normalized with respect to the user's variable of interest. For this reason, we also include a code section that normalizes the output, i.e. imposes an identity matrix in the first $r \times r$ block of β. Of course, this code section should also be executed if it does not interfere with any restrictions imposed on the model.

Listing 12: Normalizing output

```
%% RESTRICTED MODEL OUTPUT - print normalized beta and alpha for model m1r4.
modelRstrct = m1r4;
G = inv(modelRstrct.coeffs.betaHat(1:r,1:r));
betaHatR = modelRstrct.coeffs.betaHat*G;
% alphaHat is post multiplied by G^{-1} so that pi= a(G^{-1})Gb' = ab'
alphaHatR = modelRstrct.coeffs.alphaHat*inv(G)';

display(betaHatR);
display(alphaHatR);
```

The user assigns the model of interest to the variable `modelRstrct`, in this case the restricted model is `m1r4`, and executes the cell. The output is shown below. Since in this case β_1 was already equal to 1, the output is the same as the estimation output (but with more significant digits).

```
betaHatR =
   1.000000000000000
   0.105719248546476
  -0.182450552211581

alphaHatR =
  -0.187668779488160
                   0
   0.038564004272104
```

As an example of when this feature can be useful, consider model $\mathscr{H}_\alpha^3$. A part of the output is shown below, where we notice that the cointegrating vector has not been normalized.

```
-------------------------------------------------------------------------
    Fractional parameters:
-------------------------------------------------------------------------
    Coefficient                Estimate                 Standard error
-------------------------------------------------------------------------
         d                        0.630                      0.056
-------------------------------------------------------------------------
-------------------------------------------------------------------------
    Cointegrating equations (beta):
-------------------------------------------------------------------------
      Variable        CI equation 1
-------------------------------------------------------------------------
        Var1             1.023
        Var2             0.074
        Var3            -0.044
-------------------------------------------------------------------------
    Adjustment matrix (alpha):
-------------------------------------------------------------------------
      Variable        CI equation 1
```

```
--------------------------------------------------------------------------
        Var 1            -0.188
         SE 1        (    0.063  )
        Var 2             0.101
         SE 2        (    0.186  )
        Var 3             0.000
         SE 3        (    0.000  )
--------------------------------------------------------------------------
```

Executing the "Restricted Model Output" code section for this model (`m1r5`), yields the following output:

```
betaHatR =
    1.0000
    0.0719
   -0.0430

alphaHatR =
   -0.1924
    0.1028
         0
```

3.7 Forecasting

Although not part of Jones et al. (2014), the last code section in the main file, shown in Listing 13, performs recursive one-step ahead forecasts for each of the variables as well as the equilibrium relation.

Listing 13: Forecasting

```
128 %% --------- FORECAST ---------- %
129
130 % Forecast from the final restricted model.
131 NumPeriods = 12; % forecast horizon set to 12 months ahead.
132
133 % Assign the model whose coefficients will be used for forecasting.
134 modelF = m1r1;
135
136 xf = FCVARforecast(x1, modelF, NumPeriods);
137
138 % Series including forecast.
139 seriesF = [x1; xf];
140
141 % Equilibrium relation including forecasts.
142 equilF = seriesF*modelF.coeffs.betaHat;
143
144 T = size(x1,1);
145 yMaxS  = max(max(seriesF));
146 yMinS  = min(min(seriesF));
147 yMaxEq = max(max(equilF));
148 yMinEq = min(min(equilF));
149
150 figure
151 subplot(2,1,1);
152 plot(seriesF),
153 title('Series including forecast.'), xlabel('t');
154 line([T T], [yMinS yMaxS], 'Color','k');
155 subplot(2,1,2);
156 plot(equilF),
157 title('Equilibrium relation including forecasts.'), xlabel('t');
158 line([T T], [yMinEq yMaxEq], 'Color','k');
```

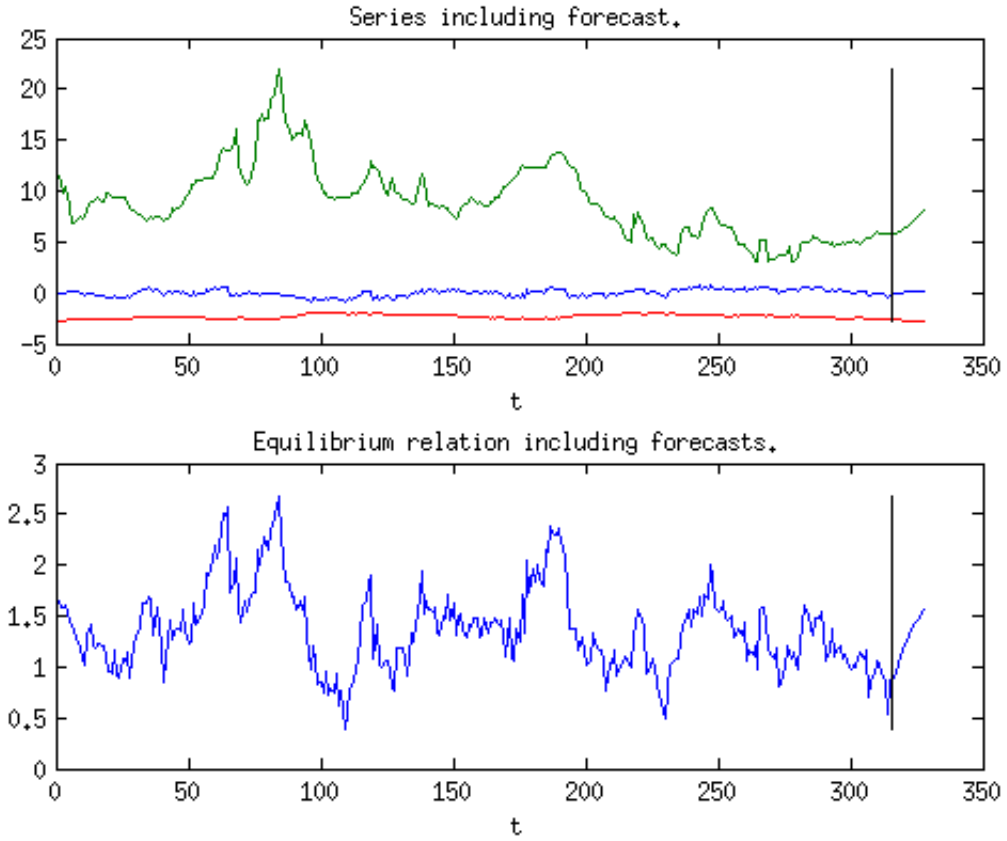
The user specifies the forecast horizon (`NumPeriods`) as well as the model (in this case, `modelF = m1r1`). These two inputs, along with the data, are used in the call to the function `FCVARforecast.m`. This function returns `xf`, a `NumPeriods` by p matrix of forecasted values of X. This code section also plots the original series and the equilibrium relation along with the forecasts. These plots are shown in Figure 3.

The forecasts can be printed to screen by typing `xf` in the command window. For this example, the forecast yields the following output:

```
xf =

  -0.143650853872867   5.857045698472417  -2.636093306380834
  -0.084879697875892   5.959876267800802  -2.654427560476592
  -0.025316897413510   6.110371912213171  -2.673504138913151
   0.023638093475328   6.291703273410405  -2.692353972906476
   0.065949410839033   6.495094604485914  -2.710793272160685
   0.101481596234305   6.712273626012697  -2.728549953757014
   0.131039943766450   6.937034957454911  -2.745463213504070
   0.155109228667221   7.164421426216199  -2.761407052913604
   0.174199823866847   7.390566368295421  -2.776295932082252
   0.188774626007768   7.612442288726599  -2.790073824966662
   0.199276573388833   7.827701550158650  -2.802710023527008
   0.206126015229692   8.034546986055656  -2.814194648028864
```

Figure 3: Forecast of final model 12 steps ahead



4 Software description

This section describes the individual components of the software package in detail. The compressed folder contains the following files:

- `data_JNP2014.csv`
- `EstOptions.m`
- `FCVARestn.m`
- `FCVARforecast.m`
- `HypoTest.m`
- `LagSelect.m`
- `mv_wntest.m`
- `RankTests.m`
- `replication_JNP2014.m`

There is one data file (`data_JNP2014.csv`), one script (`replication_JNP2014.m`), one class definition (`EstOptions.m`), and six functions. Except for `LagSelect.m` and `RankTests.m`, which call the main estimation function (`FCVARestn.m`), the functions do not have any external dependencies. All external functions required for execution are nested within each function file.

We remark again that it should not be necessary to modify any files except the script, i.e., `replication_JNP2014.m`. Only advanced users wishing to modify or extend the actual functionality of the programs will need to make any changes to the remaining files. The following subsections briefly describe the functionality of each program file.

4.1 EstOptions.m

Listing 14: EstOptions.m

```
classdef EstOptions
% classdef EstOptions
% Written by Michal Popiel and Morten Nielsen (This version 10.22.2014)
%
% DESCRIPTION: This class defines the estimation options used in the FCVAR
%       estimation procedure and the related programs. Assigning this class
%       to a variable stores the default properties defined below in that
%       variable. In addition to the properties, the methods section includes the
%       function updateRestrictions which performs several checks on the
%       user-specified options prior to estimation.
%-------------------------------------------------------------------------
```

`EstOptions` is a class definition which is assigned to an object and used in most of the functions. It contains all the model specifications and options that are available to the user. Here is an example of the contents when `EstOptions` is entered in the command line:

```
EDU>> EstOptions
ans =
  EstOptions
  Properties:
    UncFminOptions: [1x1 struct]
    ConFminOptions: [1x1 struct]
             dbMax: 2
             dbMin: 0.0100
```

```
               db0: [1 1]
       constrained: 1
        restrictDB: 1
                 N: 0
       unrConstant: 0
         rConstant: 0
        levelParam: 1
              C_db: []
              c_db: []
             R_psi: []
             r_psi: []
           R_Alpha: []
           r_Alpha: []
            R_Beta: []
            r_Beta: []
     print2screen: 1
      printGammas: 1
       printRoots: 1
        plotRoots: 1
       gridSearch: 0
         plotLike: 0
         progress: 1
       updateTime: 5
          progLoc: '"/usr/bin/fdpval"'
  Methods

Methods for class EstOptions:

EstOptions          updateRestrictions
```

4.2 FCVARestn.m

Listing 15: FCVARestn.m

```
function [ results ] = FCVARestn(x,k,r,opt)
% function [ results ] = FCVARestn(x,k,r,opt)
% Written by Michal Popiel and Morten Nielsen (This version 10.22.2014)
%
% DESCRIPTION: This function performs estimation of the FCVAR system. It is
%       the main function in the program with several nested functions, each
%       described below. It estimates the model parameters, calculates the
%       standard errors and the number of free parameters, obtains the residuals
%       and the roots of the characteristic polynomial, and prints the output.
%
% Input = x (matrix of variables to be included in the system)
%         k (number of lags)
%         r (number of cointegrating vectors)
%         opt (object containing the estimation options)
% Output = results (a Matlab structure containing estimation results)
%            - results.options       (Estimation options)
%            - results.like          (Model log-likelihood)
%            - results.coeffs        (Parameter estimates)
%            - results.rankJ         (Rank of Jacobian for
%                                       identification condition)
%            - results.fp            (Number of free parameters)
%            - results.SE            (Standard errors)
%            - results.NegInvHessian (Negative of inverse Hessian matrix)
```

```
%             - results.Residuals     (Model residuals)
%             - results.cPolyRoots (Roots of characteristic polynomial)
%-------------------------------------------------------------------------
```

This function is the central estimation function in the program. It has several nested functions which are described below. Calling this function returns a "results" Matlab structure. An example of a typical results structure is shown here:

```
m1r4 =
          options: [1x1 EstOptions]
             like: 450.8574
           coeffs: [1x1 struct]
            rankJ: 4
               fp: 26
               SE: [1x1 struct]
    NegInvHessian: [25x25 double]
        Residuals: [316x3 double]
       cPolyRoots: [9x1 double]
```

Nested functions:

4.2.1 FCVARhess.m

Listing 16: FCVARhess.m

```
function [ hessian ] = FCVARhess(x, k, r, coeffs, opt)
% function [ hessian ] = FCVARhess(x, k, r, coeffs, opt)
% Written by Michal Popiel and Morten Nielsen (This version 10.22.2014)
%
% DESCRIPTION: This function calculates the Hessian matrix of the
%       log-likelihood numerically.
%
% Input = x (matrix of variables to be included in the system)
%         k (number of lags)
%         r (number of cointegrating vectors)
%         coeffs (coefficient estimates around which estimation takes place)
%         opt (object containing the estimation options)
% Output = hessian (matrix of second derivatives)
%-------------------------------------------------------------------------
```

4.2.2 FCVARlike.m

Listing 17: FCVARlike.m

```
function [ like ] = FCVARlike(x, params, k, r, opt)
% function [ like ] = FCVARlike(x, params, k, r, opt)
% Written by Michal Popiel and Morten Nielsen (This version 10.22.2014)
%
% DESCRIPTION: This function adjusts the variables with the level parameter,
%       if present, and returns the log-likelihood given d,b.
%
% Input = x   (matrix of variables to be included in the system)
%         params (a vector of parameters d,b, and mu (if option selected))
%         k   (number of lags)
%         r   (number of cointegrating vectors)
%         opt (object containing the estimation options)
% Output = like (concentrated log-likelihood evaluated at given parameters)
%-------------------------------------------------------------------------
```

4.2.3 FCVARlikeMu.m

Listing 18: FCVARlikeMu.m

```
function [ like ] = FCVARlikeMu(y, db, mu, k, r, opt)
% function [ like ] = FCVARlikeMu(y, db, mu, k, r, opt)
% Written by Michal Popiel and Morten Nielsen (This version 10.22.2014)
%
% DESCRIPTION: This function evaluates the likelihood for a given set of
%       parameter values. It is used by the LikeGrid() function to numerically
%       optimize over the level parameter for given values of the fractional
%       parameters.
%
% Input = y  (matrix of variables to be included in the system)
%         db (fractional parameters d,b)
%         mu (level parameter)
%         k  (number of lags)
%         r  (number of cointegrating vectors)
%         opt (object containing the estimation options)
% Output = like (log-likelihood evaluated at specified parameter values)
%-------------------------------------------------------------------------
```

4.2.4 FracDiff.m

Listing 19: FracDiff.m

```
function [dx] = FracDiff(x, d)
    % function [dx] = FracDiff(x,d)
    % Andreas Noack Jensen & Morten Nielsen
    % May 24, 2013
    %
    % FracDiff(x,d) is a fractional differencing procedure based on the
    %   fast fractional difference algorithm of Jensen & Nielsen (2014, JTSA).
    %
    % input = x (vector or matrix of data)
    %         d (scalar value at which to calculate the fractional difference)
    %
    % output = vector or matrix (1-L)^d x of same dimensions as x.
    %-------------------------------------------------------
```

The function `FracDiff.m` is the implementation of the fast fractional difference algorithm by Jensen and Nielsen (2014).

4.2.5 FreeParams.m

Listing 20: FreeParams.m

```
function [ fp ] = FreeParams(k, r, p, opt, rankJ)
% function [ fp ] = FreeParams(k, r, p, opt, rankJ)
% Written by Michal Popiel and Morten Nielsen (This version 10.22.2014)
%
% DESCRIPTION: This function counts the number of free parameters based on
%       the number of coefficients to estimate minus the total number of
%       restrictions. When both alpha and beta are restricted, the rank condition
%       is used to count the free parameters in those two variables.
%
% Input = x (matrix of variables to be included in the system)
%         k (number of lags)
%         r (number of cointegrating vectors)
%         opt (object containing the estimation options)
```

```
% Output = fp (number of free parameters)
%-------------------------------------------------------------------------
```

4.2.6 FullFCVARlike.m

Listing 21: FullFCVARlike.m

```
function [ like ] = FullFCVARlike(x, k, r, coeffs, beta, rho, opt)
% function [ like ] = FullFCVARlike(x, k, r, coeffs, beta, rho, opt)
% Written by Michal Popiel and Morten Nielsen (This version 10.22.2014)
% Based on Lee Morin & Morten Nielsen (August 22, 2011)
%
% DESCRIPTION: This function returns the value of the log-likelihood
%       evaluated at the parameters provided as inputs.
%
% Input = x (matrix of variables to be included in the system)
%         k (number of lags)
%         r (number of cointegrating vectors)
%         coeffs (Matlab structure of coefficients)
%         beta (value of beta)
%         rho  (value of rho)
%         opt  (object containing the estimation options)
% Output = like (value of the log likelihood)
%-------------------------------------------------------------------------
```

4.2.7 GetParams.m

Listing 22: GetParams.m

```
function [ estimates ] = GetParams(x, k, r, db, opt)
% function [ estimates ] = GetParams(x, k, r, db, opt)
% Written by Michal Popiel and Morten Nielsen (This version 10.22.2014)
% Based on Lee Morin & Morten Nielsen (August 22, 2011)
%
% DESCRIPTION: This function uses FWL and reduced rank regression to obtain
% the estimates of Alpha, Beta, Rho, Pi, Gamma, and Omega
%
% Input = x   (matrix of variables to be included in the system)
%         k   (number of lags)
%         r   (number of cointegrating vectors)
%         db  (value of d and b)
%         opt (object containing the estimation options)
% Output = estimates (Matlab structure containing the following)
%          - estimates.db (taken directly from the input)
%          - estimates.alphaHat
%          - estimates.betaHat
%          - estimates.rhoHat
%          - estimates.piHat
%          - estimates.OmegaHat
%          - estimates.GammaHat ( p x kp matrix [GammaHat1,...,GammaHatk])
%-------------------------------------------------------------------------
```

4.2.8 GetResiduals.m

Listing 23: GetResiduals.m

```
function [ epsilon ] = GetResiduals(x, k, r, coeffs, opt)
% function [ epsilon ] = GetResiduals(x, k, r, coeffs, opt)
% Written by Michal Popiel and Morten Nielsen (This version 10.22.2014)
```

```
% Based on Lee Morin & Morten Nielsen (August 22, 2011)
%
% DESCRIPTION: This function calculates the model residuals.
%
% Input = x (matrix of variables to be included in the system)
%         k (number of lags)
%         r (number of cointegrating vectors)
%         coeffs (Matlab structure of coefficients)
%         opt (object containing the estimation options)
% Output = epsilon (matrix of residuals from model estimation evaluated at
%                     the parameter estimates specified in coeffs)
%-------------------------------------------------------------------------
```

4.2.9 Lbk.m

Listing 24: Lbk.m

```
function [ Lbkx ] = Lbk(x, b, k)
% function [ Lbkx ] = Lbk(x, b, k)
% Written by Michal Popiel and Morten Nielsen (This version 10.22.2014)
% Based on Lee Morin & Morten Nielsen (May 24, 2013)
%
% DESCRIPTION: Lbk(x, b, k) is a lag polynomial in the fractional lag operator.
%
% Input = x (vector or matrix of data)
%         b (scalar value at which to calculate the fractional lag)
%         k (number of lags)
%
% Output = matrix  [ Lb^1 x, Lb^2 x, ..., Lb^k x] where Lb = 1 - (1-L)^b.
%          The output matrix has the same number of rows as x but k times
%           as many columns.
%
% Calls the function FracDiff(x, d)
%-----------------------------------------------------
```

4.2.10 LikeGrid.m

Listing 25: LikeGrid.m

```
function [ params ] = LikeGrid(x,k,r,opt)
% function [ params ] = LikeGrid(x,k,r,opt)
% Written by Michal Popiel and Morten Nielsen (This version 10.22.2014)
%
% DESCRIPTION: This function evaluates the likelihood over a grid of values
%       for (d,b) (or phi). It can be used when parameter estimates are sensitive
%       to starting values to give a close approximation of the global max which
%       can then be used as the starting value in the numerical optimization in
%       FCVARestn().
%
% Input = x   (matrix of variables to be included in the system)
%         k   (number of lags)
%         r   (number of cointegrating vectors)
%         opt (object containing the estimation options)
% Output = params (row vector of d,b, and mu (if level parameter is selected)
%       corresponding to the global max over the grid of (d,b), or phi)
%-------------------------------------------------------------------------
```

This function allows the user to pre-estimate to obtain starting values by using a grid search. There are four types of estimation that the grid search can perform. If d and b are completely unconstrained, the

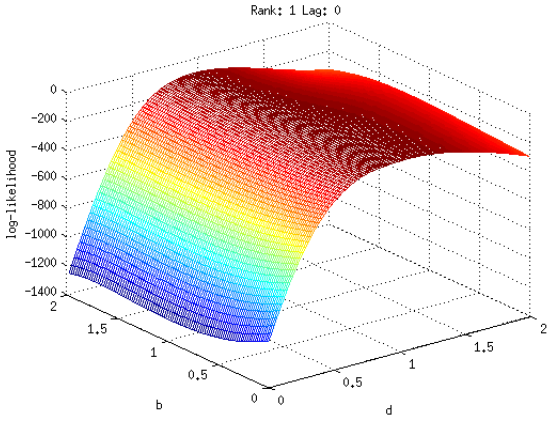
grid search is over two dimensions. An example of the likelihood obtained in an unconstrained grid search is shown in Figure 4(a). Next, if $d \geq b$ is imposed, the computation can be cut in half. An example of this likelihood is shown in Figure 4(b). If the restriction $d = b$ is imposed, then the grid search is one-dimensional as shown in Figure 4(c). Finally, if a restriction is imposed on either d or b via R_ψ and r_ψ in (10), then the grid search is one-dimensional. An example of this situation is shown in Figrue 4(d). Note that the x-axis is over the parameter ϕ, which is unrestricted. In this case, the fractional parameters are found from

$$\begin{bmatrix} d \\ b \end{bmatrix} = H\phi + h, \tag{13}$$

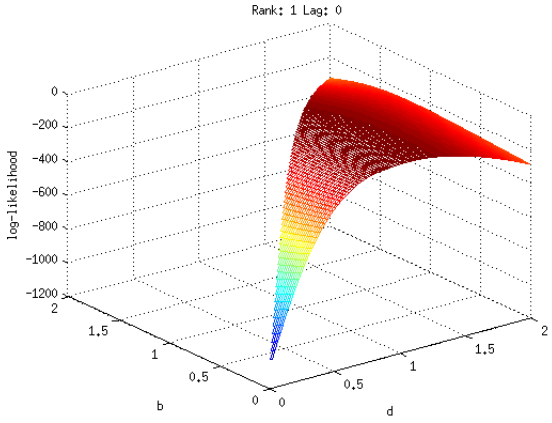
where $H = (R'_\psi)_\perp$ and $h = R'_\psi(R_\psi R'_\psi)^{-1} r_\psi$.

Figure 4: Grid search

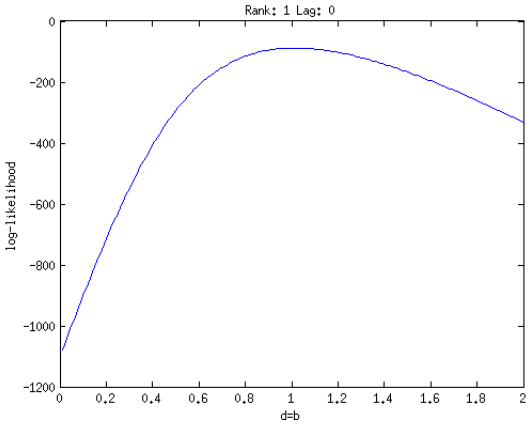
(a) Unconstrained



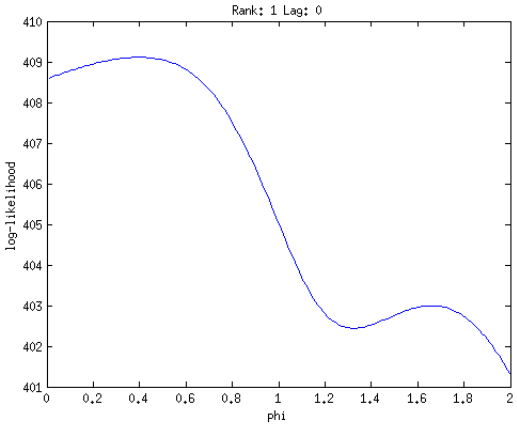
(b) $d \geq b$



(c) $d = b$



(d) Restrictions imposed



4.2.11 rLike.m

Listing 26: rLike.m

```
function [like] = rLike(coeffs, S00, S01, S11, T, p, r, opt)
% function [ like ] = rLike(coeffs, S00, S01, S11, T, p, r, opt)
% Written by Michal Popiel and Morten Nielsen (This version 10.22.2014)
%
% DESCRIPTION: This function evaluates the log-likelihood for the
```

```
%       restricted optimization function RstrctOptm(), i.e. it evaluates the
%       likelihood with the user-specified restrictions imposed.
%
% Input = coeffs (vector of coefficients alpha and beta, including rho)
%         S00, S01, S11 (product moments)
%         T (number of observations)
%         p (number of variables)
%         r (number of cointegrating vectors)
%         opt (object containing the estimation options)
% Output = like (log-likelihood evaluated at specified parameter values)
%       rLikeEst is a global structure that stores the estimates:
%           - rLikeEst.betaStar
%           - rLikeEst.alphaHat
%           - rLikeEst.OmegaHat
%-------------------------------------------------------------------------
```

4.2.12 RstrctOptm.m

Listing 27: RstrctOptm.m

```
function [betaStar, alphaHat, OmegaHat] = RstrctOptm(beta0, S00, S01, S11, T, p,
    opt)
% function [betaStar, alphaHat, OmegaHat]
%                   = RstrctOptm(beta0, S00, S01, S11, T, p, opt)
% Written by Michal Popiel and Morten Nielsen (This version 10.22.2014)
%
% DESCRIPTION: This function is used instead of the switching algorithm to
%       optimize over free parameters psi and phi directly. We translate between
%       (psi, phi) and (alpha, beta) using the relation of R*vec(alpha) = 0
%       and A*psi = alpha, and R*vec(beta) = r_beta and H*phi+h = beta.
%
% Input = beta0 (unrestricted estimate of beta)
%         S00, S01, S11 (product moments)
%         T (number of observations)
%         p (number of variables)
%         opt (object containing the estimation options)
% Output = betaStar (estimate of betaStar)
%          alphaHat (estimate of alpha)
%          OmegaHat (estimate of Omega)
%-------------------------------------------------------------------------
```

4.2.13 SEmat2vecU.m

Listing 28: SEmat2vecU.m

```
function [ param ] = SEmat2vecU( coeffs, k, r, p , opt)
% function [ param ] = SEmat2vecU( coeffs, k, r, p , opt)
% Written by Michal Popiel and Morten Nielsen (This version 10.22.2014)
% Based on Lee Morin & Morten Nielsen (August 22, 2011)
%
% DESCRIPTION: This function transforms the model parameters in matrix
%       form into a vector.
%
% Input = coeffs (Matlab structure of coefficients in their usual matrix form)
%         k (number of lags)
%         r (number of cointegrating vectors)
%         p (number of variables in the system)
%         opt (object containing the estimation options)
% Output = param (vector of parameters)
```

```
%-------------------------------------------------------------------------
```

4.2.14 SEvec2matU.m

Listing 29: SEvec2matU.m

```
function [ coeffs ] = SEvec2matU( param, k, r, p, opt )
% function [ coeffs ] = SEvec2matU( param, k, r, p, opt )
% Written by Michal Popiel and Morten Nielsen (This version 10.22.2014)
% Based on Lee Morin & Morten Nielsen (August 22, 2011)
%
% DESCRIPTION: This function transforms the vectorized model parameters
%       into matrices.
%
% Input = param (vector of parameters)
%         k (number of lags)
%         r (number of cointegrating vectors)
%         p (number of variables in the system)
%         opt (object containing the estimation options)
% Output = coeffs (Matlab structure of coefficients in their usual matrix form)
%-------------------------------------------------------------------------
```

4.2.15 TransformData.m

Listing 30: TransformData.m

```
function [ Z0, Z1, Z2, Z3 ] = TransformData(x, k, db, opt)
% function [ Z0, Z1, Z2, Z3 ] = TransformData(x, k, db, opt)
% Written by Michal Popiel and Morten Nielsen (This version 10.22.2014)
% Based on Lee Morin & Morten Nielsen (May 24, 2013)
%
% DESCRIPTION: Returns the transformed data required for regression and
%       reduced rank regression.
%
% Input = x   (matrix of variables to be included in the system)
%         k   (number of lags)
%         db  (fractional differencing parameters d and b)
%         opt (object containing the estimation options)
% Output = Z0, Z1, Z2, and Z3 of transformed data.
%
% Calls the function FracDiff(x, d) and Lbk(x, b, k).
% -----------------------------------------------------
```

4.2.16 CharPolyRoots.m

Listing 31: CharPolyRoots.m

```
function cPolyRoots = CharPolyRoots(coeffs, opt, k, r, p)
% function cPolyRoots = CharPolyRoots(coeffs, opt, k, r, p)
% Written by Michal Popiel and Morten Nielsen (This version 10.22.2014)
% Based on Lee Morin & Morten Nielsen (May 31, 2013)
%
% DESCRIPTION: CharPolyRoots calculates the roots of the
%     characteristic polynomial and plots them with the unit circle
%     transformed for the fractional model, see Johansen (2008).
%
% input = coeffs (Matlab structure of coefficients
%         opt (object containing the estimation options)
%         k (number of lags)
```

```
%          r (number of cointegrating vectors)
%          p (number of variables in the system)
%
% output = complex vector cPolyRoots with the roots of the characteristic
    polynomial.
%
% No dependencies.
%
% Note: The roots are calculated from the companion form of the VAR,
%       where the roots are given as the inverse eigenvalues of the
%       coefficient matrix.
%-------------------------------------------------------
```

4.3 LagSelect.m

Listing 32: LagSelect.m

```
function  LagSelect(x, kmax, r, order, opt )
% function LagSelect(x, kmax, r, order, opt )
% Written by Michal Popiel and Morten Nielsen (This version 10.22.2014)
%
% DESCRIPTION: This program takes a matrix of variables and performs lag
%       selection on it by using the likelihood ratio test. Output and test
%       results are printed to the screen.
%
% Input = x     (matrix of variables to be included in the system)
%         kmax  (maximum number of lags)
%         r     (cointegration rank = number of cointegrating vectors)
%         order (order of serial correlation for white noise tests)
%         opt   (object containing estimation options)
% Output = none (only output to screen)
%-------------------------------------------------------------------------
```

4.4 RankTests.m

Listing 33: RankTests.m

```
function [ rankTestStats ] = RankTests(x, k, opt)
% function [ rankTestStats ] = rankTests(x, k, opt)
% Lee Morin & Morten Nielsen (June 5, 2013)
% Modified by Michal Popiel (10.22.2014)
%
% DESCRIPTION: Performs a sequence of  likelihood ratio tests
%       for cointegrating rank.
%
% The results are printed to screen if the indicator print2screen is 1.
%
% input = vector or matrix x of data.
%       scalar k denoting lag length.
%       opt (object containing estimation options)
%
% output = (p+1) by 6 matrix rankTestStats of results from
%       cointegrating rank tests. Each row contains, for r=0,...,p:
%       r               (rank under null)
%       d               (estimate of d when rank is r)
%       b               (estimate of b when rank is r)
%       log-lik (maximized log-likelihood when rank is r)
%       LR stat (LR trace statistic for testing rank r against rank p)
%       P-value (P-value of LR trace test, or "999" if P-value is not available)
```

```
%-------------------------------------------------------
```

4.4.1 get_pvalues

Listing 34: get_pvalues.m

```
function [pv] = get_pvalues(q, b, consT, testStat, opt)
% Written by Michal Popiel and Morten Nielsen (This version 10.22.2014)
%
% DESCRIPTION: This function calls the program FDPVAL in the terminal and
% returns the P-value based on the user's inputs. The function's
% arguments must be converted to strings in order to interact with the
% terminal.
%
% Input = q        (number of variables minus rank)
% b        (parameter)
% consT    (boolean variable indicating whether or not there is
% constant present)
% testStat (value of the test statistic)
% opt (object containing estimation options)
% Output = pv (P-value for likelihood ratio test)
% ------------------------------------------------------------------------
```

4.5 HypoTest.m

Listing 35: HypoTest.m

```
function results = HypoTest(modelUNR, modelR)
% function results = HypoTest(modelUNR, modelR, df)
% Written by Michal Popiel and Morten Nielsen (This version 10.22.2014)
%
% DESCRIPTION: This function performs a likelihood ratio test of the null
%       hypothesis: "model is modelR" against the alternative hypothesis:
%       "model is modelUNR".
%
% Input = modelUNR (structure of estimation results created for unrestricted model
%    )
%         modelR (structure of estimation results created for restricted model)
% Output = results: a Matlab structure containing test results
%            - results.loglikUNR (loglikelihood of unrestricted model)
%            - results.loglikR   (loglikelihood of restricted model)
%            - results.df        (degrees of freedom for the test)
%            - results.LRstat    (likelihood ratio test statistic)
%            - results.p_LRtest  (P-value for test)
%-------------------------------------------------------------------------
```

4.6 FCVARforecast.m

Listing 36: FCVARforecast.m

```
function xf = FCVARforecast(data, model, NumPeriods)
% function xf = FCVARforecast(data, model, NumPeriods)
% Written by Michal Popiel and Morten Nielsen (This version 10.22.2014)
%
% DESCRIPTION: This function calculates recursive forecasts. It uses
%   FracDiff() and Lbk(), which are nested below.
%
% Input = data (T x p matrix of data)
%         model (a Matlab structure containing estimation results)
```

```
%          NumPeriods (number of steps ahead for forecast)
% Output = xf (NumPeriods x p matrix of forecasted values)
%-------------------------------------------------------------------------
```

Nested functions: FracDiff.m and Lbk.m. See description in Section 4.2 on FCVARestn.m and its nested functions.

4.7 mv_wntest.m

Listing 37: mv_wntest.m

```
function [ Q, pvQ, LM, pvLM, mvQ, pvMVQ ] = mv_wntest(x, maxlag, printResults)
% function [ Q, pvQ, LM, pvLM, mvQ, pvMVQ ] = ...
%                       mv_wntest(x, maxlag, printResults)
% Written by Michal Popiel and Morten Nielsen (This version 10.22.2014)
%
% DESCRIPTION: This function performs a multivariate Ljung-Box Q-test for
%       white noise and univariate Q-tests and LM-tests for white noise on the
%       columns of x.
%       The LM test should be consistent for heteroskedastic series, Q-test is not
    .
%
% Input = x            (matrix of variables under test, typically model residuals)
%         maxlag       (number of lags for serial correlation tests)
%         printResults (set =1 to print results to screen)
% Output = Q     (1xp vector of Q statistics for individual series)
%                  pvQ   (1xp vector of P-values for Q-test on individual series)
%          LM    (1xp vector of LM statistics for individual series)
%                  pvLM  (1xp vector of P-values for LM-test on individual series)
%          mvQ   (multivariate Q statistic)
%          pvMVQ (P-value for multivariate Q-statistic using p^2*maxlag df)
%-------------------------------------------------------------------------
```

4.7.1 LMtest

Listing 38: LMtest.m

```
function [ LMstat, pv ] = LMtest(x,q)
% Breusch-Godfrey Lagrange Multiplier test for serial correlation.
```

4.7.2 Qtest

Listing 39: Qtest.m

```
function [ Qstat, pv ] = Qtest(x, maxlag)
% (Multivariate) Ljung-Box Q-test for serial correlation, see
%       Luetkepohl (2005, New Introduction to Multiple Time Series Analysis, p.
    169).
```

References

Dolatabadi, S., M. Ø. Nielsen, and K. Xu (2014). A fractionally cointegrated VAR model with deterministic trends and application to commodity futures markets. QED working paper 1327, Queen's University.

Jensen, A. N. and M. Ø. Nielsen (2014). A fast fractional difference algorithm. *Journal of Time Series Analysis 35*, 428–436.

Johansen, S. (1995). *Likelihood-Based Inference in Cointegrated Vector Autoregressive Models*. New York: Oxford University Press.

Johansen, S. (2008). A representation theory for a class of vector autoregressive models for fractional processes. *Econometric Theory 24*, 651–676.

Johansen, S. and M. Ø. Nielsen (2010). Likelihood inference for a nonstationary fractional autoregressive model. *Journal of Econometrics 158*, 51–66.

Johansen, S. and M. Ø. Nielsen (2012). Likelihood inference for a fractionally cointegrated vector autoregressive model. *Econometrica 80*, 2667–2732.

Johansen, S. and M. Ø. Nielsen (2014). The role of initial values in nonstationary fractional time series models. QED working paper 1300, Queen's University.

Jones, M., M. Ø. Nielsen, and M. K. Popiel (2014). Innis Lecture: A fractionally cointegrated VAR analysis of economic voting and political support. Forthcoming in *Canadian Journal of Economics*.

MacKinnon, J. G. and M. Ø. Nielsen (2014). Numerical distribution functions of fractional unit root and cointegration tests. *Journal of Applied Econometrics 29*, 161–171.

Nielsen, M. Ø. and L. Morin (2014). FCVARmodel.m: a Matlab software package for estimation and testing in the fractionally cointegrated VAR model. QED working paper 1273, Queen's University.