

# Cobra: A package for co-breaking analysis

Michael Massmann\*

Institute of Econometrics, University of Bonn, and  
Lincoln College, University of Oxford

<http://www.uni-bonn.de/~mmassma1>

second version: 21 July 2003<sup>†</sup>

## Abstract

The purpose of this paper is to introduce **Cobra**, an econometric software package designed for **CO-BR**eaking **Analysis**. **Cobra** is programmed in Ox, see [Doornik \(2002\)](#), and consists of three modules: Firstly, the **CobraDgp** module is a class derived from the **Database** class and enables the user to generate a multivariate time series that is subject to multiple breaks in its intercept as well as linear trend. Secondly, the **Cobra** module is derived from the **Modelbase** class and implements two algorithms for the estimation of co-breaking relationships, as outlined in [Massmann \(2003\)](#). Taking advantage of the capabilities provided by **Modelbase**, **Cobra** may be loaded into **OxPack** and used with **GiveWin** as front end, see [Doornik & Hendry \(2001\)](#). Finally, **CobraSim** is derived from the **Simulation** class and wrapped around **CobraDgp** and **Cobra** to allow a straightforward implementation of Monte Carlo experiments.

After a brief introduction of the concept of co-breaking, as suggested by [Hendry \(1996\)](#), the paper proceeds to illustrate **Cobra** by means of an empirical example as well as a simple Monte Carlo. Excerpts of Ox code as well as screenshots are provided.

---

\*I am very grateful to the members of the **ox-users** discussion group, above all to Richard Lewney, for their prompt and constructive response to my queries. A special thank goes to Charles Bos for his help in the development of **Cobra** as **OxPack** module. All remaining errors are of course mine.

<sup>†</sup>**This is a preliminary version of the paper.** Please do not quote without the author's consent. Comments and suggestions are very welcome.

# 1 Introduction

The purpose of this paper is to introduce **Cobra**, an econometric software package designed for **CO-BReaking Analysis**. The idea behind co-breaking is that some economic processes may be seen as subject to structural breaks in their intercept or trend although a linear combination of them, i.e. the so-called co-breaking relationship, is constant. Co-breaking may thus be seen as the equivalent to cointegration when the non-stationarity of the process in question is seen to arise from a deterministic rather than a stochastic source. The concept was introduced by [Hendry \(1996\)](#) and may be regarded as a special case of common features, see [Engle & Kozicki \(1993\)](#).

The question of how co-breaking relationships could be estimated in a multivariate system has not yet received much attention in the literature. [Krolzig & Toro \(2001\)](#) suggest looking at the issue in the context of an autoregressive model. Their co-breaking relationships are termed *conditional* by [Massmann \(2003\)](#) since they are estimated conditional on lagged values of the process. As opposed to that, the latter author pursues the question of estimating *unconditional* co-breaking vectors. He suggests two hypothesis tests with the help of which the number of co-breaking relationships may be ascertained. Given that number, [Massmann](#) argues that standard procedures may then be used to estimate the actual co-breaking vectors. The present **Cobra** software package implements these algorithms.

**Cobra** is programmed in Ox, see [Doornik \(2002\)](#), and consists of three modules: Firstly, the **CobraDgp** module is a class derived from the **Database** class and enables the user to generate a multivariate time series that is subject to multiple breaks in its intercept or linear trend. Secondly, the **Cobra** module is derived from the **Modelbase** class and implements the aforementioned two algorithms for the estimation of co-breaking relationships, as outlined in [Massmann \(2003\)](#). Taking advantage of the capabilities provided by **Modelbase**, **Cobra** may be loaded into OxPack and used with GiveWin as front end, see [Doornik & Hendry \(2001\)](#). Finally, **CobraSim** is derived from the **Simulation** class and wrapped around **CobraDgp** and **Cobra** to allow a straightforward implementation of Monte Carlo experiments.

The outline of the paper is thus as follows. Sections 2 to 4 document the three modules **CobraDgp**, **Cobra** and **CobraSim**, respectively. Excerpts of Ox code provide illustrative examples. Section 5 then illustrates the use of the **Cobra** OxPack module by replicating the empirical analysis in [Massmann \(2003\)](#). Screenshots of the package in action are provided. Section 6 concludes.

## 2 Generating a multiple time series subject to structural breaks

This section describes how the **CobraDgp** class may be used to generate a multiple time series that is subject to an arbitrary number of breaks in its intercept and/or trend. Excerpts of sample code are given, as are some

illustrative graphs of generated series.

The general data generating process implemented in the `CobraDgp` class is given by

$$X_t = \phi_t + u_t \quad (1)$$

$$\phi_t = \tau_{c,0} + \tau_c D_{c,t} + \tau_{l,0}t + \tau_l D_{l,t} \quad (2)$$

$$u_t = \Pi_1 u_{t-1} + \Pi_2 u_{t-2} + \Pi_3 u_{t-3} + \varepsilon_t \quad (3)$$

$$\varepsilon_t \sim \text{NID}(0, \Sigma). \quad (4)$$

The equation for  $X_t$  in (1) describes an unobserved components process, made up of a deterministic component  $\phi_t$  and a stochastic component  $u_t$ . The deterministic component  $\phi_t$ , see (2), consists of an intercept term ( $\tau_{c,0} + \tau_c D_{c,t}$ ) and a linear trend term ( $\tau_{l,0} + \tau_l D_{l,t}$ ). The stochastic component  $u_t$ , see (3) and (4), is an autoregressive process of maximal order 3, with its disturbance term  $\varepsilon_t$  having an independent and identical Gaussian distribution with variance matrix  $\Sigma$ .

The dimension of  $X_t$  is  $(p \times 1)$  in general, implying that the autoregressive parameters  $\Pi_i$ ,  $i = 1, 2, 3$ , as well as the disturbance variance matrix  $\Sigma$  are  $(p \times p)$ . The deterministic term  $\phi_t$  contains a baseline intercept  $\tau_{c,0}$  as well as a baseline trend  $\tau_{l,0}$  and may be subject to  $m$  intercept breaks and  $n$  trend breaks. The latter are modelled by the dummy matrices  $D_{c,t}$  and  $D_{l,t}$ , respectively, which are of dimension  $(m \times 1)$  and  $(n \times 1)$ . The baseline parameters are  $(p \times 1)$  while the break coefficients  $\tau_c$  and  $\tau_l$  are, respectively,  $(p \times m)$  and  $(p \times n)$ . The intercept breaks or trend breaks occur, say, in time periods  $T_{c,1} < \dots < T_{c,m}$  and  $T_{l,1} < \dots < T_{l,n}$ , respectively, such that  $D_{c,t}$  and  $D_{l,t}$  are defined as

$$D_{c,t} = (1_{\{T_1+1 \leq t \leq T_2\}} : \dots : 1_{\{T_m+1 \leq t\}})' \quad (5)$$

$$D_{l,t} = (\min(t - T_1, T_1) 1_{\{T_1+1 \leq t\}} : \dots : \min(t - T_n, T_n) 1_{\{T_n+1 \leq t\}})' \quad (6)$$

If  $T$  observations are generated the full sample values of  $D_c = (D_{c,1} : \dots : D_{c,T})$  and  $D_l = (D_{l,1} : \dots : D_{l,T})$  may be illustrated as in Box 1.

The `CobraDgp` class provides a straightforward implementation for either or both of  $\tau_c$  and  $\tau_l$  in (2) to be of reduced rank such that  $\text{rk}(\tau_c) = r_c \leq \min(p, m)$  and  $\text{rk}(\tau_l) = r_l \leq \min(p, n)$ . As a consequence,  $\tau_c$  and  $\tau_l$  may be decomposed into two full rank matrices each, i.e.

$$\tau_c = \xi \eta' \quad (7)$$

$$\tau_l = \zeta \theta' \quad (8)$$

where  $\xi$  is  $(p \times r_c)$ ,  $\eta$  is  $(m \times r_c)$ ,  $\zeta$  is  $(p \times r_l)$  and  $\theta$  is  $(n \times r_l)$ .

Consider Box 2 for an excerpt of an (incomplete) programme using the `CobraDgp` class. The source code of the class is imported by means of the `#import "CobraDgp"` statement in the header of the file (see line 2) while an object is created in the standard fashion by the `new` command (see line 7). It is assigned to the variable `dgp` and, at the end of the programme in line 19, deleted. The five function calls in lines 11 to 17 form the core



```

1  #include <oxstd.h>
2  #import "CobraDgp"
3
4  main()
5  {
6      decl dgp, mX;
7      dgp = new CobraDgp();
8
9      ...
10
11     dgp.Create(cP, cM, cN, cT);
12     dgp.ChooseXParameters(mTauC0, mXi, mEta,
13                          mTauL0, mZeta, mTheta);
14     dgp.ChooseUParameters(&mPi1, &mPi2, &mPi3,
15                          &mSigma);
16     dgp.ChooseBreakPoints(vIntcBreaks, vTrendBreaks);
17     mX = dgp.GenerateData(cT);
18
19     delete dgp;
20 }

```

Box 2: Excerpt of an Ox programme, illustrating how the `CobraDgp` class is invoked and showing the core function calls.

```

1  mTauC0 = <3; 2; 1>;
2  mXi = <4; 3; 2>;
3  mEta = <5; 10>;
4  mTauL0 = <>;
5  mZeta = <>;
6  mTheta = <>;
7
8  mPi1 = mPi2 = mPi3 = <>;
9  mSigma = unit(3);
10
11 vIntcBreaks = <30, 70>;
12 vTrendBreaks = <>;
13
14 cT = 100;
15 cP = rows(mSigma);
16 cM = rows(mNu);
17 cN = rows(mTheta);

```

Box 3: Excerpt of an Ox programme, exemplifying a parameterisation of the data-generating process.

```

1  dgp.SetSave(sName);
2  dgp.SetPrint(1);
3
4  dgp.Create(...);
5  dgp.ChooseXParameters(...);
6  dgp.ChooseUParameters(...);
7  dgp.ChooseBreakPoints(...);
8  mX = dgp.GenerateData(...);
9
10 dgp.DGPinfo();
11 dgp.DGPplot();
12
13 mDc = dgp.GetIntcDummy();
14 mDl = dgp.GetTrendDummy();

```

Box 4: Excerpt of an Ox programme, showing the data output capabilities.

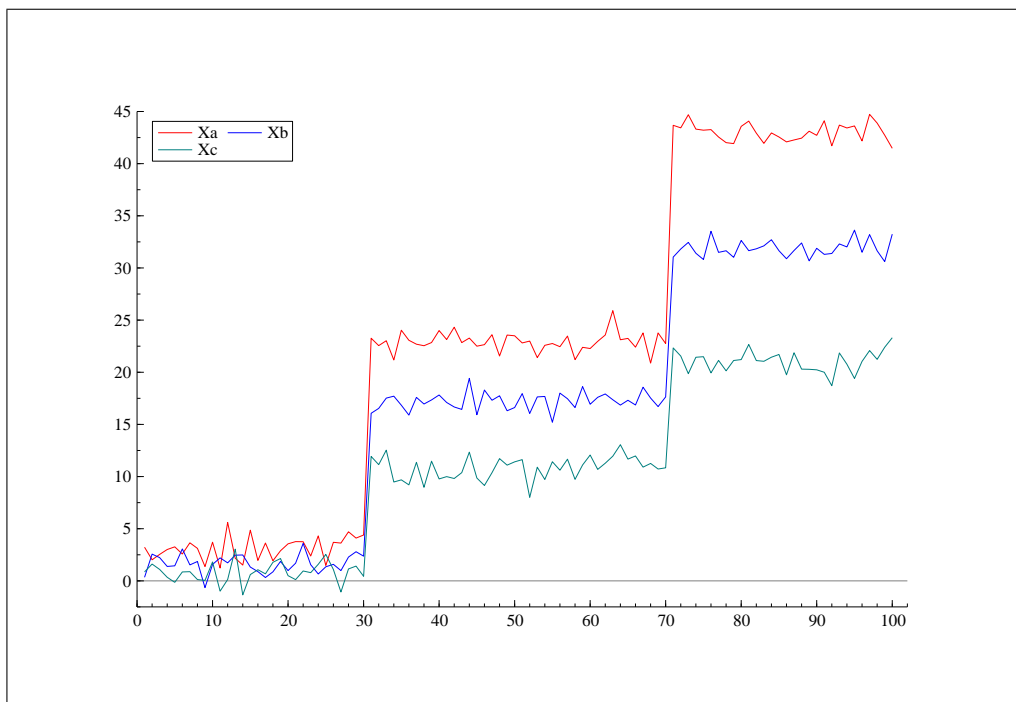


Figure 1: A realisation of the DGP whose parameters are given in Box 3

of the `CobraDgp` class. They may be divided into three groups. Firstly, `Create(...)` sets up the database for the to-be-created series, taking account of the dimensions of the  $X_t$ ,  $D_{c,t}$  and  $D_{l,t}$  variables. Secondly, the three `Choose...` commands define the parameters of the data generating process. In particular, `ChooseXParameters(...)` determines the parameters of the deterministic component  $\phi_t$  of  $X_t$ , see equations (2), (7) and (8) above, while `ChooseUParameters(...)` fixes the parameters of the stochastic component  $u_t$ , cf. equations (3) and (4). The time periods of potential intercept or trend breaks are chosen by the `ChooseBreakPoints(...)` command in line 16. Thirdly, the available information is used by `GenerateData(...)` to generate and return the series. The data are then assigned to the `mX` variable.

Each of the arguments of these five functions needs to be specified explicitly. In particular, while the arguments of `Create(...)` and `GenerateData(...)` are integers, those of the remaining three functions are of matrix type. An example specification of the parameters is given in the excerpt displayed in Box 3. Note that the parameterisation is such that the rank of  $\xi$  and  $\eta$  is  $r_c = 1 < 3 = p$  so that  $\tau_c$  is of reduced rank. A full rank specification, would be obtained, for instance, by setting  $\tau_c = \eta'$  and  $\xi = I_p$ .

A few data output functions are available. As Box 4 illustrates the data may be printed, plotted and saved as an `.in7` data file. In particular, the `SetSave(sName)` command ensures that the dataset as well as the data plots are saved under the name `sName`. Output is printed by calling `SetPrint(...)`, with the value of the argument determining the amount of detail in increasing order. Calling the functions `DGPinfo()` and `DGPplot()` leads to information on the DGP to be printed and to the generated data series  $X_t$  to be plotted. In some situations it may be useful to retrieve the two dummy matrices  $D_c$  and  $D_l$  that are used for generating the data, see equation (2) above. This is achieved by calling the two `Get...` functions which return the variables of interest. Note that the `Set...` commands must be called before `GenerateData(...)` to have an effect, while the remaining four functions only make sense after the generation of the data. As an example, a realisation of the three-dimensional DGP whose parameters are displayed in Box 3 above is shown in Figure 1.

### 3 Estimating co-breaking relationships

This section illustrates how the `Cobra` package may be used to estimate co-breaking vectors. Two general statistical models are provided for by the class: full rank regression models and reduced rank regression models, each with a disturbance term that is either white noise or autocorrelated. In particular, denote a general  $p$ -dimensional regression model by

$$Y_t = \kappa_1 Z_{1,t} + \kappa_2 Z_{2,t} + v_t \quad (9)$$

where  $v_t = \sum_{i=1}^k A_i v_{t-i} + \epsilon_t$  and  $\epsilon_t \sim \mathbf{N}(0, \Omega)$ . The regressor  $Z_{1,t}$  is of dimension  $(q_1 \times 1)$  while  $Z_{2,t}$  is  $(q_2 \times 1)$ . If  $A_i = 0, \forall i$ , then the disturbance term is white noise while if  $A_i \neq 0$ , for some  $i$ , it is autocorrelated. As long as

both  $\kappa_1$  and  $\kappa_2$  are of full rank the regression may be consistently estimated by standard procedures, e.g. OLS when  $u_t$  is white noise and GLS otherwise.

More interestingly, a reduced rank regression model results when one of the two coefficient matrices, say  $\kappa_2$ , has rank  $\text{rk}(\kappa_2) = s < \min(p, q_2)$ . If  $s$  is known then the parameters may be consistently estimated by means of, for instance, maximum likelihood. Moreover, a reduced-rank  $\kappa_2$  implies that it may be decomposed into two full rank matrices  $\nu$  and  $\rho$ , where  $\nu \sim (p \times s)$  and  $\rho' \sim (s \times q_2)$  such that  $\kappa_2 = \nu\rho'$ , cf. equations (7) and (8) in the previous section. Then the orthogonal complement of  $\nu$  may be found to be  $\nu_\perp$ , of dimension  $(p \times (p - s))$ , such that the linear combinations  $\nu'_\perp Y_t$  no longer depends on  $Z_{2,t}$ . If the latter regressor is a dummy variable representing breaks in the intercept or the trend of  $Y_t$  then  $\nu_\perp$  is called a co-breaking vector and  $\nu'_\perp Y_t$  are  $(p - s)$  co-breaking combination. The idea of co-breaking was introduced by [Hendry \(1996\)](#) and may be seen as a special case of the general concept of common features discussed by [Engle & Kozicki \(1993\)](#).

If the rank of  $\kappa_2$  is not known it seems natural to use a hypothesis test to make inference about it. As long as the disturbance  $u_t$  in (9) is white noise this is achieved by applying a standard likelihood ratio test based on the partial canonical correlations between  $Y_t$  and  $Z_{2,t}$  as developed by [Box \(1949\)](#) and [Anderson \(1951\)](#). It turns out, however, that when the disturbance is autocorrelated this procedure may no longer be appropriate. [Massmann \(2003\)](#) discusses two approaches to rank testing in such models: One is based on a re-parameterisation of the model in (9) suggested by [Johansen, Mosconi & Nielsen \(2000\)](#) and will be referred to as the JMN test in the remainder of the paper. The other approach follows a suggestion by [Reinsel & Velu \(1998\)](#) and will be termed the RV test. Indeed, the foremost reason behind coding `Cobra` is the implementation of these two testing procedures.

Consider [Box 5](#). The `Cobra` class is activated by `#import "Cobra"` in line 2 and an object is created by `new Cobra()` in line 10. The dataset of interest is loaded in line 8 while its container is created by means of the `Create(...)` command and filled with the variables, appropriately named, in `Append(...)`, see lines 11 - 12. These features are in fact inherited from the `Database` class. The core of the `Cobra` functionality is illustrated by the commands in lines 14 - 19. In particular, a `Select(...)` command chooses the model variables from the database and assigns them to a group. In this instance, the model contains only endogenous variables, as indicated by the single identifier `Y_VAR`. Each variable is included with lags 0 to 2. Exogenous variables may in principle also be incorporated by selecting variables into a `X_VAR` group. The deterministic variables are chosen by the `SetDummy(...)` command in line 16. In the present setting, the integer constant `B_SLB` means that a constant intercept, a constant linear trend, intercept breaks as well as slope breaks are created. The number of breaks and the break points are chosen by the `SetBreaks(...)` command in line 17. Since both the coefficient of the intercept break dummy and that of the slope break dummy may conceivably be of reduced rank, the function `SetReducedRank(...)` determines which coefficient the rank test is to be applied to. Here, it is the trend (or slope) coefficient. Finally, `SetMethod(...)` is used to determine

```

1  #include <oxstd.h>
2  #import "Cobra"
3
4  main()
5  {
6      decl est, mdata;
7
8      mdata = loadmat(...);
9
10     est = new Cobra();
11     est.Create(1, 1, 1, rows(mdata), 1);
12     est.Append(mdata, {"Xa", "Xb", "Xc"});
13
14     est.Select(Y_VAR,
15         {"Xa", 0, 2, "Xb", 0, 2, "Xc", 0, 2});
16     est.SetDummy(B_SLB);
17     est.SetBreaks(<30; 50; 80>);
18     est.SetReducedRank(CB_TR);
19     est.SetMethod(M_JMN);
20
21     est.Estimate();
22
23     delete est;
24 }

```

Box 5: Excerpt of an Ox programme that uses **Cobra** to estimate co-breaking relationships.

```

1  est.SetArTest(<1, 2>);
2  est.SetNormTest();
3
4  est.SetOutput(3);
5  est.SetPlot(TRUE);
6  est.SetSave(FALSE);
7  est.SetDbNameRoot(oxfilename(2) ~ "_model1");
8
9  est.Estimate();

```

Box 6: This excerpt shows some of the output options of **Cobra**.

the estimation and testing algorithm, i.e. either OLS or GLS in a full rank regression or the algorithm involving the JMN or RV rank test in the case of a reduced rank regression.

Box 6 contains an extract of code for post-estimation analysis and graphics options. In particular, for mis-specification testing of the estimated model, lines 1 and 2 show the commands to implement a test for no autocorrelation in the residuals and for normally distributed residuals, respectively. Both single equation and vector tests for the hypotheses will be computed as a result. The detail of the estimation and testing output is determined by the call of `SetOutput(...)`. `SetPlot(TRUE)` causes illustrative graphics to be plotted, and `SetSave(TRUE)` is a switch for the saving of the variable database and the graphs. The name under which this output is saved is determined by `SetDbNameRoot(...)`. Note that all commands discussed in Box 6 ought to be given *before* the call of `Estimate()`.

## 4 Examining the size of a test for reduced rank

The `CobraSim` package may be used to implement Monte Carlo experiments involving co-breaking models. It is a class derived from the `Simulation` class and thus inherits the latter's facilities to gather the results of estimation or testing procedures across replications. Some functions for the reporting and displaying of these results have been added and are outlined below.

An illustrative experiment is implemented in Box 7. As with the previous two `Cobra` modules, the `CobraSim` class is called by means of the `#import "CobraSim"` statement in line 2 of the file. Subsequently, the specification of an Monte Carlo is distributed between three functions. Firstly, the process used to generate the artificial data in the experiment is specified in `CobraSim::SimDgp(...)`. Secondly, details of the statistical model examined in the experiment are given in `CobraSim::SimModel(...)`. Thirdly, parameters pertaining to the Monte Carlo per se are left to `main()`. Note that the former two functions are members of the `CobraSim` class.

Consider the `main()` function first. An object of the `CobraSim` class is created by the `new` command in lines 12 - 13. Moreover, the three arguments of the constructor function are the sample sizes to be examined in the Monte Carlo (in the present scenario `<25,50,75,100,125,150,175,200>`), the maximum sample size (needed so that a database of sufficient dimension may be created) and the number of replications (here 10,000). The following three function calls determine the name under which output files will be saved, and whether or not results will be saved and/or plotted, see lines 14 - 16. The call to `SimDgp(...)` determines the setup of the data generating process, see lines 4 - 5 in the same box, while `SimModel(...)` specifies the statistical model to be estimated, cf. lines 6 - 7. The arguments of these two functions are the maximum sample size as well as three switches determining the printing, plotting and saving of intermediate results. `Simulate()` in line 19 sets off the recursions of the simulation proper.

```

1  #include <oxstd.h>
2  #import "CobraSim"
3
4  CobraSim::SimDgp(const cMaxT, const fPrint,
5                  const bPlot, const bSave){ ... }
6  CobraSim::SimModel(const cMaxT, const bPrint,
7                    const bPlot, const bSave){ ... }
8
9  main()
10 {
11     decl exp;
12     exp = new CobraSim(<25,50,75,100,125,150,175,200>,
13                      200, 10000);
14     exp.SetFileName(oxfilename(2));
15     exp.SetStore(TRUE);
16     exp.SetPlotRep(TRUE);
17     exp.SimDgp(200, 0, 0, 0);
18     exp.SimModel(200, 0, 0, 0);
19     exp.Simulate();
20     delete exp;
21 }

```

Box 7: This excerpt shows how the `CobraSim` module for Monte Carlo simulation is called.

```

1  m_iDummyType = B_SLB;
2  m_flRrCond = CB_IC;
3  m_cModelMaxYlag = 2;
4
5  m_est = new Cobra();
6  m_est.Create(1, 1, 1, cMaxT, 1);
7  m_est.Append(zeros(cMaxT, m_cP), {"Ya", "Yb", "Yc"});
8  m_est.SetMethod(M_JMN);
9  m_est.SetDummy(m_iDummyType);
10 m_est.SetReducedRank(m_flRrCond);
11 m_est.SetBreaks(<0.2, 0.6>);
12 m_est.SetNormTest();
13 m_est.SetArTest(<1, 2>);
14 m_est.SetPlot(bPlot);
15 m_est.SetSave(bSave);
16 m_est.SetOutput(bPrint);

```

Box 8: The `CobraSim::SimModel(...)` function specifies the statistical model to be estimated as part of the Monte Carlo.

```

1  mTauC0 = <3; 2; 1>;
2  mTauL0 = <0.8; 0.5; 0.1>;
3  mXi = <4; 3; 2>;
4  mNu = <5; 10>;
5  mZeta = <>;
6  mTheta = <>;
7  mPi1 = <0.3, 0; 0, 0.5, 0; 0, 0, 0.2>;
8  mPi2 = <0.2, 0; 0, 0.1, 0; 0, 0, 0.4>;
9  mPi3 = <>;
10 mSigma = unit(3);
11 vIntcBreaks = <0.2; 0.6>;
12 vTrendBreaks = <>;
13
14 m_cP = rows(mSigma);
15 m_cM = rows(mNu);
16 m_cN = rows(mTheta);
17
18 m_dgp = new CobraDgp();
19 m_dgp.Create(m_cP, m_cM, m_cN, cMaxT);
20 m_dgp.ChooseXParameters(mTauC0, mXi, mNu, mTauL0,
21   mZeta, mTheta);
22 m_dgp.ChooseUParameters(&mPi1, &mPi2, &mPi3, &mSigma);
23 m_dgp.ChooseBreakPoints(vIntcBreaks, vTrendBreaks);
24 m_dgp.SetPrint(fPrint);
25 m_dgp.SetPlot(bPlot);
26 m_dgp.SetSave(bSave);

```

Box 9: The `CobraSim::SimDgp(...)` function specifies the process used to generate the data in the Monte Carlo.

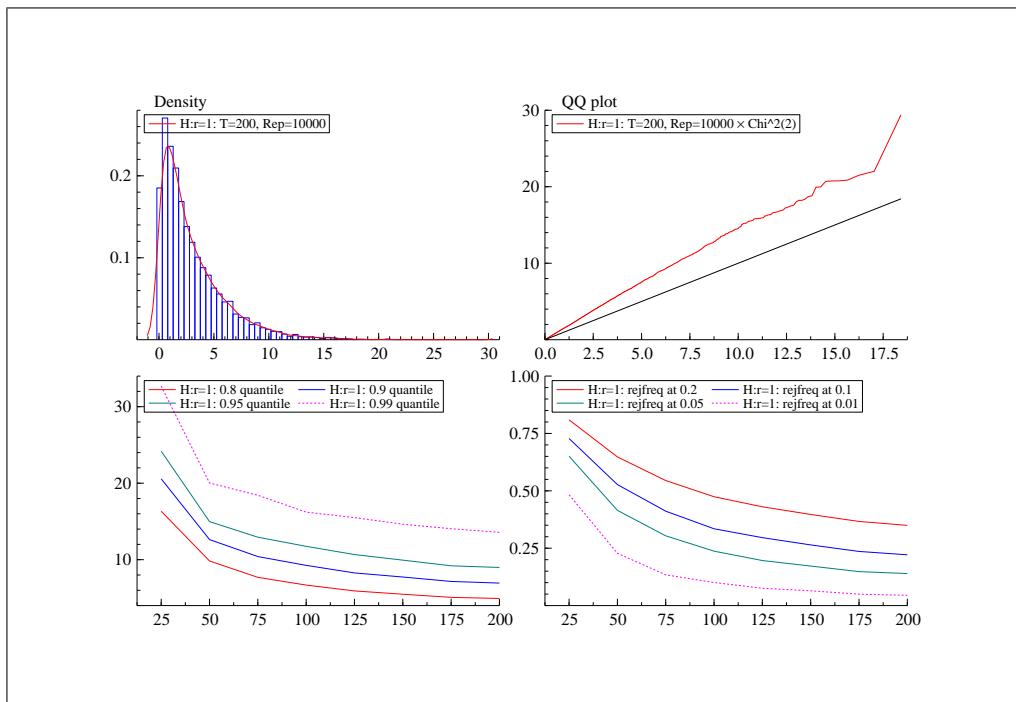


Figure 2: Some of the graphs produced by `CobraSim` as a result of the Monte Carlo experiment described in Boxes 7 - 8.

Example contents of the `CobraSim::SimModel(...)` function are presented in Box 8. In essence, an object of the `Cobra` class is created a statistical model specified in much the same way as was described in Section 3 above. Two crucial differences to Boxes 5 and 6 are to be noted, however. Firstly, the variables specifying the dummy variable type (`m_iDummyType`), the rank-restricted parameter coefficient (`m_flRrCond`), and the maximum lag of the stochastic component of the model (`m_cModelMaxYLag`) all need to be of global scope. Secondly, and importantly, the core function of the class, viz. `Estimate()`, is not called by `SimModel()` but is implicit in the definition of `CobraSim::Generate(...)`, where the latter overwrites the eponymous virtual function of the `Simulation` class and implements the individual replications of the experiments.

In order to generate the artificial data in each replication the `CobraSim` class makes use of the `CobraDgp` class. In particular, the specification of the data generating process in `CobraSim::SimDgp(...)` of Box 9 is akin to the function calls described in Section 2; see Boxes 2 - 3. Notable differences are, firstly, the dimensions  $p$ ,  $m$  and  $n$  need to be global variables since they will be used by other member functions of the class. Secondly, the call of `GenerateData(...)` is absent since it is implicit in `CobraSim::Generate(...)`.

In terms of the notation introduced in equations (1) - (4) of Section 2 the deterministic component  $\phi_t$  of the DGP  $X_t$  implemented in Box 9 is subject to intercept breaks  $D_{c,t}$ . The parameter coefficient  $\tau_c = \xi\eta'$  is of dimension  $(3 \times 2)$  yet of reduced rank  $\text{rk}(\tau_c) = 1$ . The stochastic component  $u_t$  of the process is autoregressive of order 2. Meanwhile, the statistical model set up to describe the data follows equation (9) in Section 3 and specifies  $Z_{2,t}$  to contain the intercept break dummies  $D_{c,t}$ , and  $Z_{1,t}$  to include the lagged endogenous variables as well as the baseline intercept and trend. By choosing method `M_JMN` and coefficient parameter `CB_IC` the JMN test for the rank of  $\kappa_2$  is performed. Two nested hypotheses are tested sequentially:  $H_0 : r = 0$  and, if rejected,  $H_0 : r = 1$ . The second null being true the corresponding test statistic may be shown to have a  $\chi^2$  distribution. As an illustration, Figure 2 displays the empirical distribution of that statistic, with the density depicted in the top-left panel and the QQ plot in the top-right. The bottom-left panel shows the 0.8, 0.9, 0.95 and 0.99 quantiles of the empirical distribution as a function of the sample size, and the bottom-right panel plots the empirical size of the test against the sample size, for nominal test sizes of 0.2, 0.1, 0.05 and 0.01. As may be discerned from the graphs, the test is heavily oversized even at a sample size of 200, a feature that Massmann (2003) argues is due to the autocorrelation in the data. As a remedy, he suggests choosing a smaller-than-usual type-I error.

## 5 An empirical application illustrating Cobra in OxPack

Since the `Cobra` class is derived from the `Modelbase` class the overwriting of some of the latter's virtual member functions allows the precompiled `Cobra`

source code to be usable as an OxPack module; see [Doornik \(2003\)](#). The purpose of this section is to illustrate the capabilities of the **Cobra** package by replicating the empirical co-breaking analysis performed in [Massmann \(2003\)](#).

The loading of the **Cobra** module in OxPack creates a new class object; see [Figure 3](#). The 'Data Selection' dialogue window displayed in [Figure 4](#) is called by clicking on 'Model  $\rightarrow$  Formulate'. The name of the database loaded in GiveWin is shown in the small box at the bottom right of the window while the names of the available variables appear in the column entitled 'Database'. In the present application, the database contains five variables, of which only the following three are of interest for the purpose at hand:  $w$  is an index for hourly wages in the UK,  $pwe$  are world prices in sterling, and  $pg$  is the deflator for UK GDP. The variables are plotted in [Figure 5](#). In fact, they have been used in two extensive empirical analyses of UK inflation by [Hendry \(2000, 2001\)](#); see these publications for a more detailed description of the data.

The model that is constructed is a VAR(4) in the aforementioned three variables, as rendered clear by the list in the column superscribed 'Model' in [Figure 4](#). Note that no deterministic variables are selected at this stage of the analysis. Clicking on 'OK' takes the user to the 'Model Settings' dialogue box; see [Figure 6](#). In this window the type of dummy variable to be included in the model is selected, as well as the timing of the breaks, if applicable. In the present context, the model is chosen to contain a baseline intercept, a baseline trend, and intercept break as well as slope break dummies. The break points are specified as time periods 1914, 1920, 1932, 1939, 1949, 1974 to take account of, respectively, World War I, its aftermath, the dollar coming off the gold standard, World War II, its aftermath and the oil crisis. The third element in [Figure 6](#) is the choice of coefficient parameter that is to be tested for its rank; here, it is the trend coefficient.

[Figure 7](#) shows the 'Estimate Model' dialogue box. What is on display are the methods available for the estimation of the model, as well as information on the maximum sample period, taking account of the dataset's size and lagged dependent variables. The actual estimation sample may be chosen to be different from the selection sample. Regarding the estimation methods, full rank OLS and full rank GLS are the standard algorithms that do not impose a reduced rank restriction on one of the parameter coefficients. The remaining two procedures, viz. 'reduced rank JMN approximation' and 'reduced rank RV approximation' are the aforementioned algorithms that, in a first stage, approximate the rank of the restricted coefficient and, in a second stage, estimate all parameters in the model given that restriction. The procedure chosen in the present application is the JMN approximation. Note that clicking on the button named 'Options...' causes a further window to be opened containing options regarding the estimation and testing output; see [Figure 8](#). The radio buttons determine the amount of detail printed after the model estimation has been carried out. They are arranged in increasing order, i.e. ranging from no output at all (i.e. top button) to the printing of the model headers, the estimation result and the data used (viz. bottom

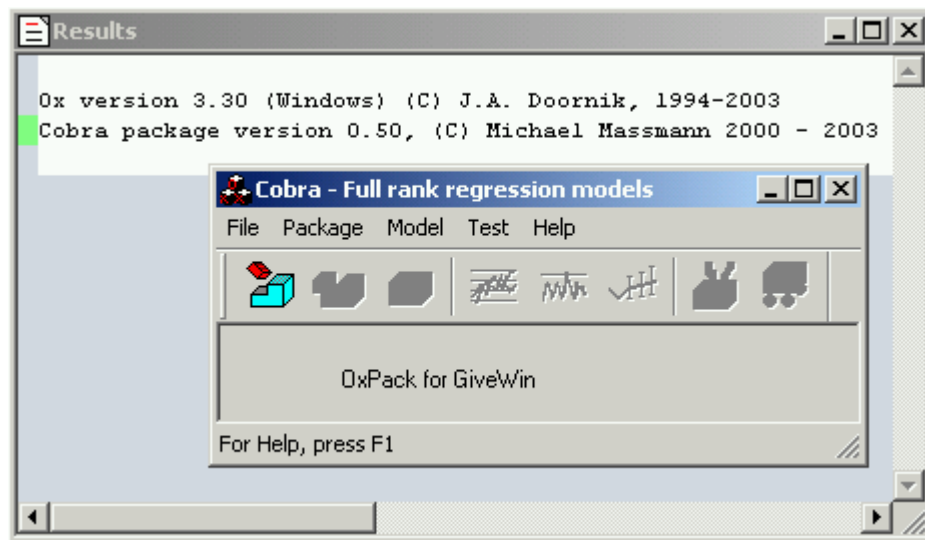


Figure 3: This is a screenshot of the OxPack window with the Cobra module loaded as well as the corresponding Results window in GiveWin.

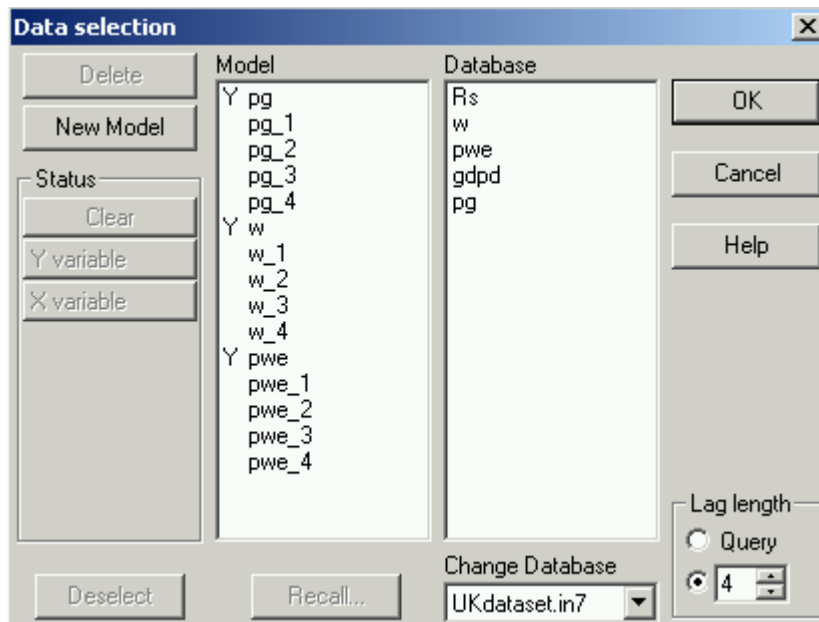


Figure 4: This screen allows the variables that are to be included in the model to be selected from the database.

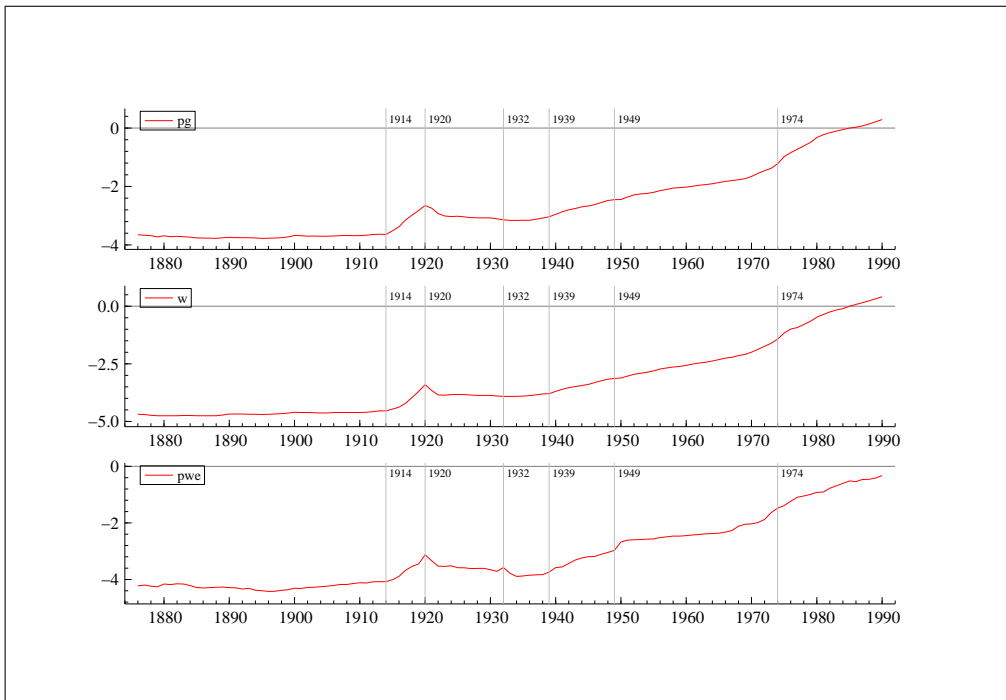


Figure 5: The three data series  $pg$ ,  $w$ ,  $pwe$  examined in the application, with the breakpoints superimposed.

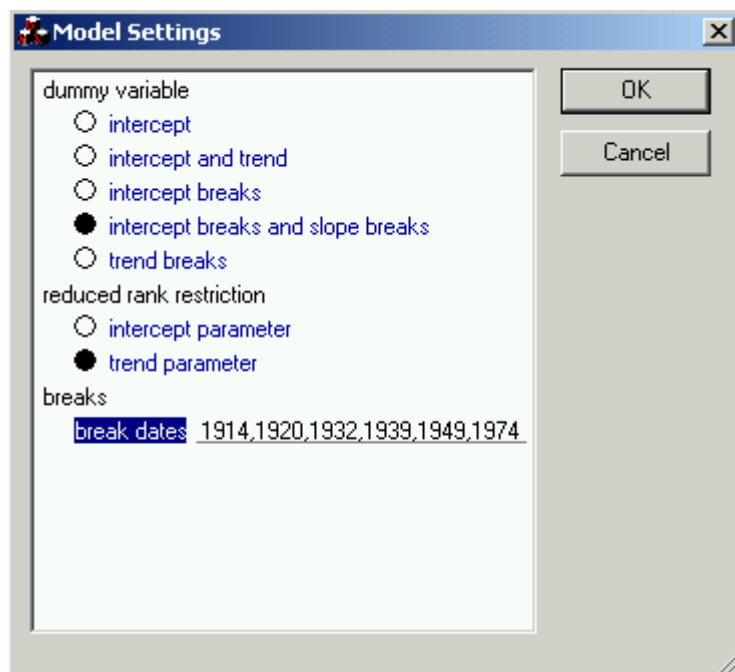


Figure 6: This OxPack dialogue box provides three choices: firstly, the type of dummy variable to be included in the model may be selected; secondly, the parameter coefficient upon which a reduced rank restriction is to be imposed; thirdly, the time periods in which the intercept and/or the trend is to break.

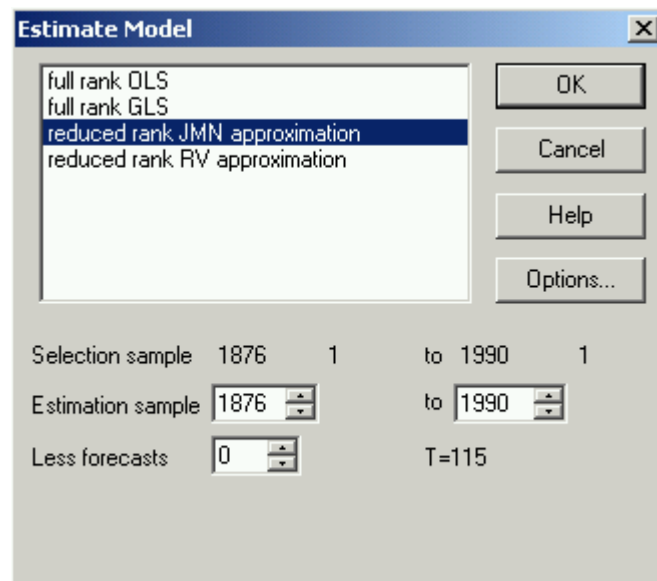


Figure 7: This is the OxPack dialogue box that allows the user to specify the estimation method as well as the estimation sample. Note the 'Options...' button on the right-hand side.

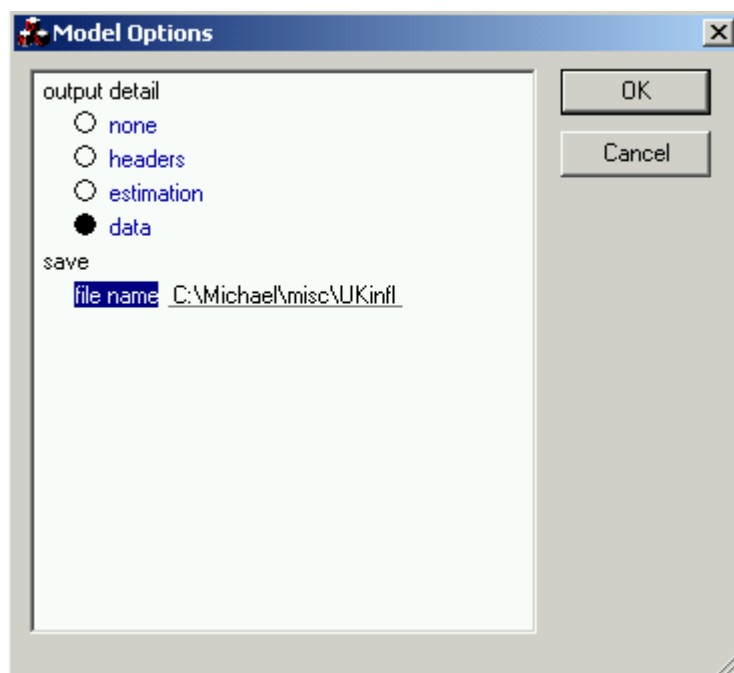


Figure 8: The OxPack 'Model Options' window permits the choice of less frequently used options.

```

----- canonical correlation analysis -----

vector of estimated non-zero eigenvalues: mLambdaHat =
      0.21468      0.17751      0.064874

rank test:
      statistic      df      tail prob
H0: r=0      57.9775      18.0000      0.0000
H0: r=1      30.1863      10.0000      0.0008
H0: r=2       7.7135       4.0000      0.1027

estimated rank = number of canonical variates = 2

----- co-breaking vectors -----

estimated trend CB vectors =
      -7.9099      4.7230      1.0000

```

Box 10: This extract displays a portion of the estimation output. In particular, the result of the JMN test shows the estimated rank of the slope dummy's coefficient as 2. As a result, there is one co-breaking vector.

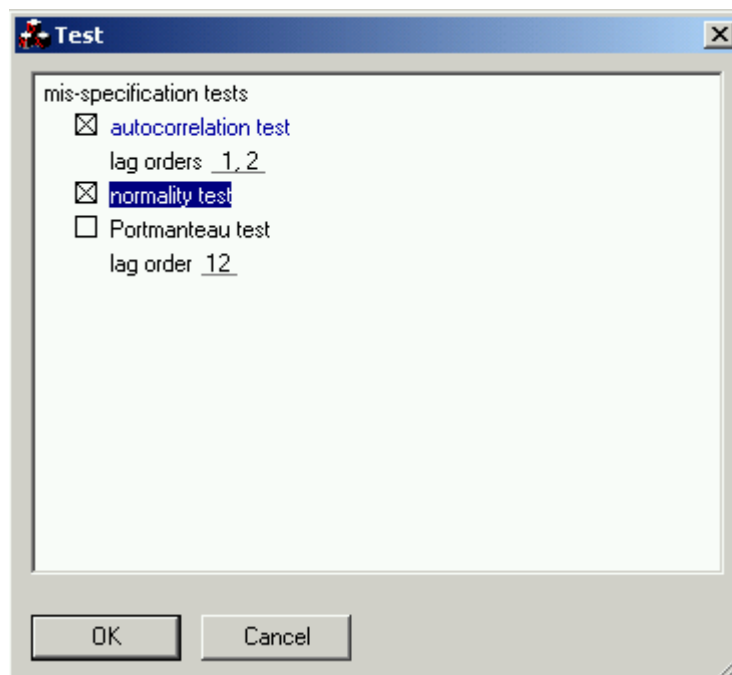


Figure 9: The 'Test' dialogue presently contains three mis-specification tests that may be called as part of a post-estimation analysis.

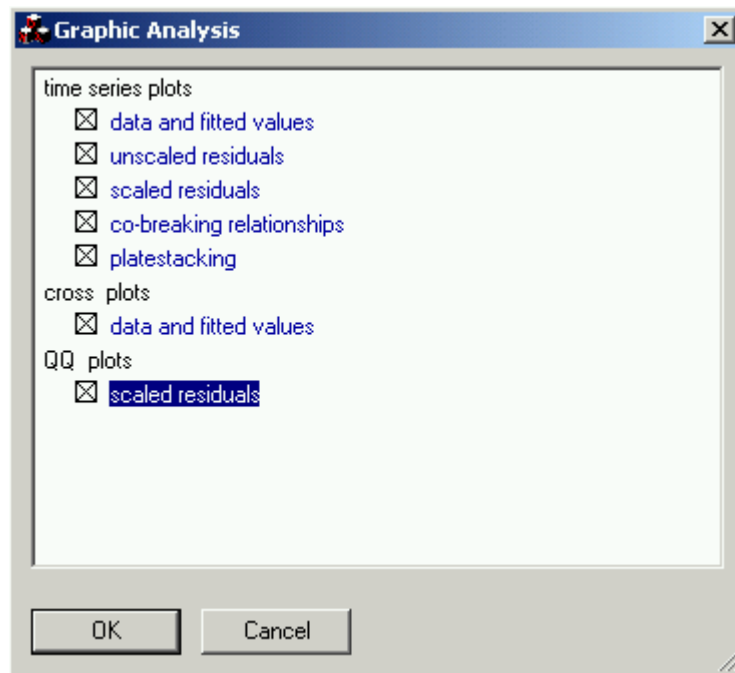


Figure 10: The 'Graphic Analysis' window allows the call of several data plots that may be used to evaluate the quality of the estimated model. Note that some options only apply to reduced rank analysis.



Figure 11: This figure shows the estimated co-breaking vector, with its deterministic component superimposed.

button). The save option allows a full path as well as a root file name to be specified with which the dataset created and the graphics that may subsequently be produced will be stored. In the present application, all output and data will be stored under the name `UKinfl`, suitably extended, in the directory `C:\Michael\misc`. Clicking on 'OK', the user is returned to the 'Estimate Model' dialogue. Pressing 'OK' in this window then triggers the actual estimation of the model, with the estimation output, as specified, appearing in the 'Results' window of GiveWin. Box 10 contains a portion of the results, namely that pertaining to the rank test.

Once the estimation of the model has terminated two entries in the 'Test' menu options are activated. In particular, the post-estimation analysis may include mis-specification testing and graphical analysis. Consider the former first, see Figure 9. The residuals of the estimated model may be tested for no autocorrelation, normality and general mis-specification, using the Portman-teau test. Ticking the relevant boxes and clicking the 'OK' button will print the output accordingly. Moreover, the 'Graphic Analysis' window allows the drawing of a variety of time series, cross and QQ plots. Some of them only have an effect in reduced rank analysis, and vice versa. For example, the button for co-breaking relationships may be ticked in both instances but is only effectual in the latter setting. Clicking on 'OK' yields the corresponding graphs which are automatically saved provided a path was specified in the 'Model options' dialogue box. In fact, Figure 5 above was produced in this fashion, and Figure 11 shows the resulting estimated co-breaking vector, with its deterministic component superimposed. Since it was the intercept break dummy whose coefficient parameter was found to be of reduced rank, see Box 10, the co-breaking relationship remains subject to breaks in its slope. Note that the shaded areas are the four time periods following each break points which are effectively dropped from the estimation to reconcile the structural breaks on the one hand with the order of the model's dynamic component on the other. They may be interpreted as transition periods. See Massmann (2003) for more details.

## 6 Conclusion and outlook

This paper documents and illustrates the `Cobra` software package, which implements several routines for co-breaking analysis, see Massmann (2003). The package is programmed in Ox and consists of three classes: `CobraDgp` allows the generation of a multiple time series subject to intercept or trend breaks, `Cobra` provides for the estimation of co-breaking relationships, while the purpose of `CobraSim` is to facilitate the easy implementation of Monte Carlo experiments. Excerpts of sample Ox code are given in this paper, as are screenshots of the `Cobra` class when used as an OxPack module. Finally, an empirical illustration replicating the analysis in Massmann (2003) is provided.

A number of extension to the existing version of `Cobra` are possible. Firstly, procedures for the testing of hypotheses on co-breaking vectors have

not yet been implemented. Secondly, an automatic selection of intercept and trend break points may be an attractive alternative to their present deterministic modelling. It is hoped, however, that the current functionality of the package will provide a useful first implementation of the co-breaking estimation and testing techniques to both the practitioner and the methodologist.

## References

- Anderson, T. W. (1951), ‘Estimating linear restrictions on regression coefficients for multivariate normal distributions’, *Annals of Mathematical Statistics* **22**, 327–351.
- Box, G. E. P. (1949), ‘A general distribution theory for a class of likelihood criteria’, *Biometrika* **36**, 317–346.
- Doornik, J. A. (2002), *Object-Oriented Matrix Programming Using Ox*, 4th edn, Timberlake Consultants Press, London. <http://www.nuff.ox.ac.uk/Users/Doornik>.
- Doornik, J. A. (2003), Ox Appendices, Ox version 3.30. Mimeo, <http://www.nuff.ox.ac.uk/Users/Doornik>.
- Doornik, J. A. & Hendry, D. F. (2001), *GiveWin Version 2: An Interface to Empirical Modelling*, Timberlake, London.
- Engle, R. F. & Kozicki, S. (1993), ‘Testing for common features’, *Journal of Business and Economic Statistics* **11**, 369–395.
- Hendry, D. F. (1996), A theory of co-breaking. Mimeo, Nuffield College, University of Oxford.
- Hendry, D. F. (2000), Does money determine UK inflation over the long run?, in R. E. Backhouse & A. Salanti, eds, ‘Macroeconomics and the Real World’, Vol. 1: Econometric Techniques and Macroeconomics, Oxford University Press, Oxford, pp. 85–114.
- Hendry, D. F. (2001), ‘Modelling UK inflation, 1875 – 1991’, *Journal of Applied Econometrics* **16**, 255–275.
- Johansen, S., Mosconi, R. & Nielsen, B. (2000), ‘Cointegration analysis in the presence of structural breaks in the deterministic trend’, *Econometrics Journal* **3**, 216–249.
- Krolzig, H.-M. & Toro, J. (2001), Testing for cointegration and super exogeneity in the presence of deterministic shifts. Mimeo, Nuffield College, University of Oxford.
- Massmann, M. (2003), Co-breaking: representation, estimation and testing. Unpublished DPhil thesis manuscript, University of Oxford.

Reinsel, G. C. & Velu, R. P. (1998), *Multivariate Reduced-Rank Regression*, Springer, New York.