OxMetrics is a powerful system for econometric, statistical, and financial econometric modelling and forecasting. OxMetrics 8 continues a long tradition of reliable and easy-to-use software. The user-friendly OxMetrics front-end is shared between all OxMetrics modules, thus considerably reducing the learning curve when using the full range of econometric and statistical functionality of the OxMetrics system. The modules have been developed by experts in their field: many of their advances have been published in learned journals, while at the same time benefiting the OxMetrics system. We first present an overview of the components of the entire OxMetrics system, before more detailed descriptions of each module.

1 OxMetrics overview

OxMetrics is a family of software packages providing an integrated solution for the econometric analysis of time series, forecasting, financial econometric modelling, or statistical analysis of cross-section and panel data. OxMetrics consists of a front-end program called OxMetrics, and individual application modules such as Ox, CATS, PcGive, STAMP and G@RCH. OxMetrics Enterprise is a single product that includes all the important components: OxMetrics desktop, Ox Professional, CATS, PcGive and STAMP and G@RCH

The OxMetrics front end provides facilities to manage and transform the databases that are used in the statistical analysis. Output from OxMetrics modules is displayed in the form of graphs and reports. Graphs can be edited in preparation for publication. Multiple plots within a graph are possible, with automatic adjustments for smaller sizes. OxMetrics databases for macro-economic analysis have a fixed frequency. Databases for financial econometrics are usually ‘dated’, which allows for daily or timed data. Aggregation facilities are provided, e.g. to convert daily data into monthly data. Transformations can be done using a calculator, or with algebra code. A batch language allows for repetition of tasks, and the batch code is recorded in the background while interactive modelling proceeds. All modelling components have been written in the Ox language.

Ox is an object-oriented statistical and econometric development system. At its core is a powerful matrix language, which is complemented by a comprehensive matrix and statistical library. Among the special features of Ox are its speed, well-designed syntax, and graphical facilities. Ox can read and write many data formats, including spreadsheets and OxMetrics files. Ox is at the core of OxMetrics: most of the interactive modules of OxMetrics (such as PcGive, STAMP, G@RCH) are implemented with the Ox language.

PcGive aims to give an operational and structured approach to econometric modelling and forecasting using the most sophisticated yet user-friendly software. The accompanying books transcend the old ideas of ‘textbooks’ and ‘computer manuals’ by linking the learning of econometric methods and concepts to the outcomes achieved when they are applied. The econometric techniques of PcGive include: VAR, cointegration, simultaneous equations models, Markov Switching, ARFIMA, logit, probit, GARCH modelling, static and dynamic panel data models, X2tARIMA, and more. PcGive uses Autometrics for automatic model selection. PcGive incorporates PcNaive to interactively design and run Monte Carlo experiments.

CATS is dedicated to the cointegrated vector autoregression. Both (I(1) and (I(2)) models can be estimated using the most sophisticated algorithms. Restrictions can be imposed on the cointegrating vectors and their loadings. Both Bartlett corrections and bootstrap versions of tests are available. Models can be estimated recursively. Random samples can be drawn from the estimated models, making it easier to implement simulation experiments. CATSmining helps with identification of the cointegrating space.

G@RCH is dedicated to the estimation and forecasting of univariate and multivariate ARCH-type models. It also allows the estimation of univariate and multivariate non-parametric estimators of quadratic variation and integrated volatility. G@RCH provides a menu-driven easy-to-use interface, as well as graphical features. For repeated tasks, the models can be estimated via the Batch Editor of OxMetrics or using the Ox language together with the ‘Garch’, ‘MGarch’ and ‘Realized’ classes.

STAMP stands for Structural Time series Analyser, Modeller and Predictor. The models are set up in terms of unobserved components such as trends, seasonals and cycles, which all have a direct interpretation. STAMP gives access to sophisticated algorithms through an easy-to-use interface. Univariate decompositions can be specified with higher-order trends and cycles to force these components to be more smooth; the signal-to-noise ratio can be fixed but also estimated. This option in STAMP is regularly used for extracting business and financial cycles from macroeconomic time series, even when portions of the time series are missing. Realized-variance time series can be decomposed into multiple stochastic components such as long-term variance (random walk or smooth trend), trading-day effects (seasonal component), stationary volatility (autoregressive) and noise (irregular component). The multiple components can be extracted from the data for analysis and forecasting. Multivariate decompositions of macroeconomic time series are relevant for economic policy making when it is recognized that the variables are endogenous. This is fully supported in STAMP.

SsfPack is a library to carry out computations involving the statistical analysis of univariate and multivariate models in state space form. SsfPack routines can be called from Ox, while the library itself is written in C. SsfPack is not a member of OxMetrics Enterprise Edition.

Versions

- OxMetrics Enterprise Edition is a single product that includes and integrates all the important components for theoretical and empirical research in econometrics, time series analysis and forecasting, applied economics and financial time series: OxMetrics front-end, Ox Professional, PcGive, G@RCH, CATS, and STAMP.
- OxMetrics Modules are available individually, always including the OxMetrics front-end and Ox Professional.
- SsfPack is a separate program. Recent editions of macOS, Windows and Linux are supported.
2 OxMetrics

Data formats
- **OxMetrics** data files (.in7/.bnr) — this format is designed to make reading and writing of data very efficient;
- **Excel** spreadsheet files (.xlsx);
- **Spreadsheet** Comma-separated files (.csv);
- **Other** Stata .dta files; formatted text files.

Graphics
- **Actual series** with optional transformations
  - Create separate Plots
  - Style
    - Lines
    - Symbols
    - Lines and symbols
    - Index line: plot first series as index
    - Index line and symbols: plot first series as index
    - Bars: plot all series as bars
    - Shading: use shading where this variable is 1, no shading otherwise
    - First as bar: plot only the first as bar chart
  - Transformation:
    - Logarithms: natural logs of the series: $\log(y_t)$
    - Growth rates: $\Delta y_t = y_t - y_{t-1}$
    - First differences: $\Delta y_t = y_t - y_{t-1}$
    - Seasonal growth rates: $\log(y_t) - y_{t-s}, s = 4$ for quarterly data, $s = 12$ for monthly data,
    - Seasonal differences: $\Delta y_t = y_t - y_{t-s}$
  - Use log scale.
- **Multiple series** with optional transformations
  - Match series by
    - None: no matching is done
    - Mean & range matched to first series
    - Second series on right scale
    - Start = 100
  - Style: as above
  - Transformation: as above
  - Use log scale
- **Scatter plots**
  - Y against X
  - Y against X, labels along the axes
  - Scatter plot with regression line
  - With cubic spline smooth, automatic bandwidth
  - All scatter plots
- **Distribution plots**
  - Estimated density and histogram, optionally with normal reference
  - Estimated distribution against normal: a QQ plot
  - Frequencies and/or cumulative frequencies
  - QQ plot against Uniform, normal, t, F, or $\chi^2$ distribution
  - Box plot
- **Time-series plots**
  - Autocorrelation function
  - Partial autocorrelation function
  - Cross-correlation function

Graph editing and saving
Each graph consists of a collection of objects, which in most cases can be manipulated, moved or deleted.
Graphs can be saved as:

<table>
<thead>
<tr>
<th>extension</th>
<th>format</th>
</tr>
</thead>
<tbody>
<tr>
<td>.eps</td>
<td>Encapsulated PostScript;</td>
</tr>
<tr>
<td>.svg</td>
<td>OxMetrics graphics file;</td>
</tr>
<tr>
<td>.pdf</td>
<td>Portable document format;</td>
</tr>
<tr>
<td>.png</td>
<td>a bitmap format;</td>
</tr>
<tr>
<td>.ps</td>
<td>PostScript;</td>
</tr>
<tr>
<td>.svg</td>
<td>SVG, supported by most browsers.</td>
</tr>
</tbody>
</table>

Transformations
The **Algebra language** enables you to transform database variables by writing mathematical formulae. Algebra code can be written interactively in the Calculator, or directly in the Algebra editor. Such algebra code can be saved, reloaded, and edited.

The **Calculator** writes its operations as algebra code to the Results window, from where it can be cut and pasted into the algebra editor. Algebra can also be run directly from the results window, by highlighting the block of algebra code, and then pressing Ctrl+A.

Modules
An increasing number of modules interacts with OxMetrics: the front-end is the ‘server’, while the modules (G@RCH, PcGive, STAMP, etc.) are the ‘clients’. While it is possible to write clients that interface directly with the server (such as OxPack, and OxRun), it is much easier to develop Ox packages to do this. This requires the use of the Modelbase class, which provides the necessary functionality.
3 PcGive

The special features of PcGive are:

1. Ease of use – all modelling can be done interactively.
2. Flexible data handling in OxMetrics.
3. Efficient modelling – fast and reliable algorithms written in Ox.
4. Automatic model selection – Autometrics is available for many model types to provide a sophisticated model selection tool.
5. Advanced graphics – graphic analysis of the estimated model, recursive graphics, forecast graphics and others.
7. Extensive batch language together with generation of Ox code for more advanced uses.
8. Well-presented output.

Capabilities

PcGive supports the following model categories (book volume in parentheses):

- **Models for cross-section data**
  - Cross-section Regression (I)

- **Models for discrete data**
  - Binary Discrete Choice (II): Logit and Probit
  - Multinomial Discrete Choice (III): Multinomial Logit
  - Count data (III): Poisson and Negative Binomial

- **Models for financial data**
  - GARCH Models (III): GARCH in mean, GARCH with Student-t, EGARCH

- **Models for panel data**
  - Static Panel Methods (III): within groups, between groups
  - Dynamic Panel Methods (III): Arellano-Bond GMM estimators

- **Models for time-series data**
  - Single-equation Dynamic Modelling (I), optionally using Autometrics
  - Multiple-equation Dynamic Modelling (II): VAR, cointegration, simultaneous equations analysis, optionally using Autometrics
  - Regime Switching (V): Markov-switching models
  - ARFIMA Models (III): exact maximum likelihood, modified-profile likelihood or non-linear least squares

- **Monte Carlo using PcNaive**
  - AR(1) Experiment using PcNaive (IV)
  - Static Experiment using PcNaive (IV)
  - Advanced Experiment using PcNaive & Ox Professional (IV)

- **Other models**
  - Nonlinear Modelling (I)
  - Descriptive Statistics (I):
    - Means, standard deviations and correlations
    - Normality tests and descriptive statistics
    - Autocorrelations (ACF) and Portmanteau statistic
    - Unit-root tests
    - Principal component analysis

- **Extensive search**: to handle correlated data,
- **Efficient search**: need to estimate many models,
- **Statistical congruence**: maintained as a search constraint,
- **Statistical properties**: extensively researched,
- **Not maximizing goodness-of-fit**: avoids overfitting,
- **Controlled through gauge**: expected number of falsely selected variables,
- **Flexible**: more variables than observations, logit models, ...
- **Robustness**: to outliers using impulse-indicator saturation (IIS) and breaks using step-indicator saturation (SIS).

Also see Hendry and Doornik (2014), *Empirical Model Discovery and Theory Evaluation*, MIT Press.

**PcNaive**

PcNaive is an interactive program for Monte Carlo study of econometric methods. Experiments allow the finite-sample properties of dynamic econometric methods to be evaluated in relevant settings. Experiments based on the simple AR(1) DGP and static DGP can be formulated in PcGive, and run directly in OxMetrics. Advanced experiments, on the other hand, are formulated interactively. PcNaive then writes the Ox code which is executed in OxMetrics by OxRun.

**Example**

This example uses the UK unemployment rate (Ur) and the real interest rate minus the real growth rate (Rr, see Clements & Hendry (2006) Ch.12 in *Handbook of Economic Forecasting*, Vol.1).

After loading the database in OxMetrics, start modelling using PcGive:

![PcGive - Models for time-series data](image)

Formulate the model with two lags:

![Formulate - Single-equation Dynamic Modelling](image)

using IIS and SIS, and keeping back 8 forecasts:
Estimation with Autometrics finds 4 impulses and 5 steps. The steps, however, combine to effectively make dummies, so we rerun with IIS only. This leaves 7 impulse dummies, three before 1890, two in the 1920's, and finally 1930 and 1939:

Modelling Ur by OLS
The dataset is: UKUnempData.in7
The estimation sample is: 1863 - 2006

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Std. Error</th>
<th>t-value</th>
<th>t-prob</th>
<th>Part. R^2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ur_1</td>
<td>1.26941</td>
<td>0.06397</td>
<td>19.8</td>
<td>0.0000</td>
</tr>
<tr>
<td>Ur_2</td>
<td>-0.371891</td>
<td>0.05731</td>
<td>-6.49</td>
<td>0.2418</td>
</tr>
<tr>
<td>Rr</td>
<td>0.160550</td>
<td>0.02089</td>
<td>7.69</td>
<td>0.0000</td>
</tr>
<tr>
<td>Rr_1</td>
<td>-0.0929132</td>
<td>0.02214</td>
<td>-4.20</td>
<td>0.1576</td>
</tr>
<tr>
<td>I:1879</td>
<td>0.0312359</td>
<td>0.009103</td>
<td>3.43</td>
<td>0.0008</td>
</tr>
<tr>
<td>I:1880</td>
<td>-0.048728</td>
<td>0.009531</td>
<td>-5.09</td>
<td>0.0000</td>
</tr>
<tr>
<td>I:1884</td>
<td>0.0453289</td>
<td>0.009084</td>
<td>-4.99</td>
<td>0.1587</td>
</tr>
<tr>
<td>I:1921</td>
<td>0.0527939</td>
<td>0.01042</td>
<td>5.06</td>
<td>0.0000</td>
</tr>
<tr>
<td>I:1922</td>
<td>-0.0487495</td>
<td>0.01040</td>
<td>-4.69</td>
<td>0.1426</td>
</tr>
<tr>
<td>I:1930</td>
<td>0.036300</td>
<td>0.009096</td>
<td>4.01</td>
<td>0.1086</td>
</tr>
<tr>
<td>I:1939</td>
<td>-0.0349234</td>
<td>0.009160</td>
<td>-3.81</td>
<td>0.0992</td>
</tr>
<tr>
<td>Constant U</td>
<td>0.00475903</td>
<td>0.001469</td>
<td>3.24</td>
<td>0.0015</td>
</tr>
</tbody>
</table>

The estimation sample is: 1863 - 2014
The dataset is: UKUnempData.in7
Modelling Ur by OLS
The estimation sample is: 1863 - 2006

While the forecasts are dynamic in the unemployment rate, they are conditional on the growth adjusted interest rate. So they are not genuinely out of sample.

One approach is to reformulate the model as a vector autoregression. Starting with two lags as before, it is possible to select Autometrics with IIS and SIS again, this time using 1%:

Choose a model type:
- Unrestricted system
- Contegrated VAR
- Simultaneous equations model
- Constrained simultaneous equations model

Choose the Autometrics options:
- Target size
- Pre-search log reduction
- Outlier and break detection
- Large residuals
- Saturation estimation
- Step indicator saturation (SD)
- Impulse indicator saturation (IS)
- Differenced IIS (DI)
- Trend saturation (TS)
- Advanced Autometrics settings

The selected model has the same variables in each equation. To obtain a more parsimonious representation, rerun Autometrics at 0.1% on the simultaneous equations model with both equations in their unrestricted reduced form. Estimation is by FIML:

Choose the estimation sample:
Selection sample: 1863 - 2014
Estimation starts at: 1953
Estimation ends at: 1980
No. of observations: 144
No. of parameters: 12

Autometrics
- Full Information Maximum Likelihood (FIML)
- Three-stage Least Squares (LS)
- Two-stage Least Squares (LS)
- Equations by Equation (OLS)
- Automatic maximization
- Standard errors
- Standard

The first equation (using \(	ext{Ur} = \beta_0 + \beta_1 \text{Ur}_{t-1} + \beta_2 \text{Ur}_{t-2} + \epsilon_t\) written by PcGive):

\[
\begin{align*}
\text{Ur} &= 1.26 \text{Ur}_{t-1} - 0.374 \text{Ur}_{t-2} + 0.0306 \text{t}1879_t \\
&\quad - 0.0597 \text{t}1880_t + 0.0455 \text{t}1884_t - 0.132 \text{t}1922_t \\
&\quad + 0.0325 \text{t}1930_t - 0.0354 \text{t}1939_t - 0.0935 \text{t}1950_t \\
&\quad + 0.091 \text{t}1992_t + 0.00597 \\
&\quad (0.055) (0.052) (0.0087) (0.011) (0.0086) (0.016) (0.011) (0.0017)
\end{align*}
\]

The final model does have some non-normality left, but the forecasts are genuinely ex ante now:
CATS

The basic model of CATS is the \( p \)-dimensional vector autoregressive (VAR) model with Gaussian errors and \( k \) lags
\[
X_t = A_1 X_{t-1} + \cdots + A_k X_{t-k} + \Phi D_t + \epsilon_t, \quad t = 1, \ldots, T,
\]
where \( X_0, \ldots, X_{-k+1} \) are fixed, \( \epsilon_1, \ldots, \epsilon_T \) are iid \( N_p(0, \Omega) \) and \( D_t \) is a vector of deterministic variables such as a constant, linear trend, and seasonal or intervention dummies.

The VAR(p) model is reformulated as (taking \( k = 2 \)):
\[
\Delta X_t = \Pi X_{t-1} + \Gamma_1 \Delta X_{t-1} + \Phi D_t + \epsilon_t, \quad t = 1, \ldots, T.
\]
The hypothesis of cointegration is formulated as a reduced rank condition on the \( \Pi \) matrix,
\[
\mathcal{H}(r): \Pi = a \beta',
\]
where \( a \) and \( \beta \) are \( p \times r \) matrices of full column rank. This is the \( I(1) \) model.

The VAR in second order differences with \( \mathcal{H}(r) \) imposed is:
\[
\Delta^2 X_t = \alpha \beta' X_{t-1} + \Gamma_2 \Delta X_{t-1} + \Phi D_t + \epsilon_t, \quad t = 1, \ldots, T.
\]
The \( I(2) \) model imposes an additional rank reduction on the model:
\[
\alpha \beta' \Gamma = \xi \eta',
\]
where \( \xi \) and \( \eta \) both are \((p - r \times s_1)\)-dimensional matrices.

CATS features

Here is a brief summary of new features in the \( I(1) \) part of CATS

1. Much more efficient computations (can be several orders of magnitude faster than previous implementations) in Bartlett correction and recursive estimation;
2. Bartlett correction always included when valid;
3. Improved beta-switching algorithm;
4. New alpha-beta-switching algorithm allowing linear restrictions on alpha and not requiring identification;
5. Bootstrap of rank test;
6. Bootstrap of restrictions;
7. More Monte Carlo facilities: draw from estimated model, either with estimated or with specified coefficients;
8. General-to-specific CATSmining;
9. Automatic generation of Ox code;
10. New convenient way to express restrictions;
11. Most algorithms are QR based.

And for the \( I(2) \) part of CATS:

1. Improved tau-switching algorithm;
2. New delta-switching algorithm;
3. New triangular-switching algorithm allowing linear restrictions on alpha, beta, tau and not requiring identification;
4. Estimation with \( \hat{\delta} = 0 \);
5. Bootstrap of rank test;
6. Simulation of asymptotic distribution of rank test;
7. Bootstrap of restrictions;
8. More Monte Carlo facilities: draw from estimated model, either with estimated or with specified coefficients;
9. Automatic tests of unit vectors and variables;
10. CATSmining;
11. Improved computation of standard errors;
12. Automatic generation of Ox code;
13. All algorithms are QR based.

Documentation

The CATS documentation, written by Katarina Juselius and Jurgen Doornik, presents the models together with tutorials. The contents are

- The Multivariate Cointegration Model
- An \( I(1) \) Analysis
- CATSmining
- The Cointegrated \( I(2) \) model
- An \( I(2) \) Analysis

Formulating linear restrictions

One of the most important features of CATS is that it allows you to test or impose restrictions on the parameters \( \alpha \) and \( \beta \). Linear restrictions on the cointegrating vectors can include the desired normalization.

There are two ways, in general, to express the linear restrictions on a \( p \times 1 \) vector. First as
\[
\beta = H \psi,
\]
where \( H \) is a known \( p \times q \) matrix, with \( q < p \) and \( \psi \) is a column vector of length \( q \). A second way to express restrictions is in the form \( R' \beta = 0 \), where \( R \) is \((p - q) \times p, R = H, \) moves between the representations.

CATS presents a more intuitive way to express restrictions that reflects how we report our research. For example, homogeneity between \( X_2, X_3 \) and \( X_4 \) is formulated as
\[
a \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix} \beta = \begin{bmatrix} a_0 & a_1 & \cdots & a_q \end{bmatrix}'.
\]

Example dialogs

After selecting the variables in the VAR, the settings are used to specify the lag length and choose between \( I(1) \) and \( I(2) \) modelling. Here an \( I(2) \) model is specified:

Several types of tests are pre-programmed:
5 G@RCH

G@RCH is an OxMetrics application dedicated to the estimation and forecasting of univariate and multivariate ARCH-type models. G@RCH provides a user-friendly interface with rolling menus as well as many graphical features. For repeated tasks, the models can be estimated via the ‘Batch Editor’ of OxMetrics or the Ox programming language (several example files using the G@RCH class are provided).

G@RCH capabilities

- Conditional mean: ARMA, ARFIMA, ARCH-in-Mean, Explanatory Variables;
- Univariate conditional variance: GARCH, EGARCH, GJR, APARCH, IGARCH, RiskMetrics, FIGARCH, FIEGARCH, FIAPARCH, HYGARCH, GAS; Explanatory Variables;
- Multivariate conditional variance: MGARCH: scalar BEKK, diagonal BEKK, full BEKK, DCC, cDCC, CCC, DECO, OGARCH, GO-GARCH, Principal Components, RiskMetrics, Variance Targeting;
- (Quasi-)Maximum Likelihood: Normal, Student, GED or skewed-Student distribution;
- Constrained Maximum Likelihood, Simulated Annealing;
- Value-at-Risk, Expected shortfall, Backtesting (Kupiec LRT, Dynamic Quantile test);
- Forecasting, Realized volatility;
- Tests for jumps;
- RE@LIZED non-parametric estimators of quadratic variation, integrated volatility and jumps using intraday data.
- $h$-steps-ahead forecasts of both equations;
- Univariate and multivariate misspecification tests (Nyblom, Sign Bias Tests, Pearson goodness-of-fit, Box-Pierce, Residual-Based Diagnostic for conditional heteroscedasticity, Hosking’s portmanteau test, Li and McLeod test, constant correlation test, and more).
- Three univariate models of the class of Generalized Autoregressive Score (GAS) models, i.e. the GAS, Exponential GAS, and Asymmetric Exponential GAS models with a Normal, Student-$t$, GED and Skewed-Student distribution.
- Spline-GARCH and Spline-GJR models.

Multivariate GARCH (MGARCH)

From a financial-econometric perspective, MGARCH models enable better decision tools in many areas, e.g. asset pricing, portfolio selection, option pricing, hedging, and risk management. However, implementation of such models is not easy, making G@RCH a potentially valuable tool for financial institutions.

MGARCH models for $N$ stochastic processes can be described as

$$y_t = \mu(t) + \epsilon_t$$

where $\mu$ is a finite vector of parameters, $\mu(t)$ is the conditional mean vector and,

$$\epsilon_t = H_t^{1/2}(\theta)z_t,$$

with $H_t^{1/2}$ a positive definite matrix.

Several MGARCH models are available in G@RCH since version 5.0. The most popular one is perhaps the dynamic conditional correlation (DCC) model of Rob Engle. DCC defines:

$$H_t = D_t R_t D_t$$

where $D_t = \text{diag}(h_{11,t}, \ldots, h_{NN,t})$ and $h_{ii,t}$ can be taken as any univariate GARCH model. $R_t$ is a correlation matrix derived from a covariance matrix $Q_t = (q_{ij,t})$:

$$R_t = \text{diag}(q_{11,t}^{-1/2}, \ldots, q_{NN,t}^{-1/2})Q_t\text{diag}(q_{11,t}^{-1/2}, \ldots, q_{NN,t}^{-1/2}),$$

specified as

$$Q_t = (1 - \alpha - \rho)Q + \alpha uu_t' + \rho QQ_t - I.$$ 

A convenient feature of DCC models is that the parameters governing the variance and correlation dynamics can be estimated separately.

Aielli (2013, Journal of Business & Economic Statistics) presents a corrected DCC model, which addresses the inconsistency in the estimation of the unconditional variance matrix. This is available in G@RCH as cDCC.

MGARCH Example

The bivariate GARCH(1,1)-cDCC example is based on the Dow Jones and Nasdaq indices. After loading the DJNQ.xls data set in OxMetrics, activate the model dialog selecting MGARCH models with G@RCH:

** MGARCH(2) SPECIFICATIONS **


Strong convergence using numerical derivatives

Log-likelihood = -9835.07

Please wait: Computing the Std Errors ...

Robust Standard Errors (Sandwich formula)

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Std.Error</th>
<th>t-value</th>
<th>t-prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>rho_21</td>
<td>0.718938</td>
<td>0.066626</td>
<td>10.79</td>
</tr>
<tr>
<td>alpha</td>
<td>0.040522</td>
<td>0.02061874</td>
<td>6.549</td>
</tr>
<tr>
<td>beta</td>
<td>0.0591955</td>
<td>0.00959539</td>
<td>99.59</td>
</tr>
</tbody>
</table>
| No. Observations | 3913 No. Parameters | 13 No. Series | 2 Log Likelihood | -9835.071 Elapsed Time | 0.149 seconds (or 0.00248333 minutes).

The next graph shows the estimated conditional variances, with the conditional correlations in the bottom graph:
Realized volatility Example

RE@LIZED provides a full set of procedures to compute non-parametric estimates of the quadratic variation, integrated volatility and jumps from intraday data. This includes univariate and multivariate versions of the realized volatility, bi-power-variation and realized outlyingness weighted variance. Daily and intraday tests for jumps are also implemented. These facilities are accessible through the rolling menus of G@RCH, as well as from Ox.

After loading the data file from the Ox folder (OxMetrics8/ox/packages/Realized/samples/data), activate RE@LIZED:

```oxt
#include <oxstd.oxh>
#include <packages/Realized/Realized>
main()
{
    decl model = new Realized();
    model.Load("C:\Program Files\OxMetrics8\ox\packages\Realized\samples\data\"Simulated_cont_GARCH_jumps_FFF_K_0.2_M_0.3.in7");
    for (decl i = 1; i <= 288; ++i)
        model.Select("Y", {sprint("Ret_", i)});
    model.SetModelClass(Realized::MC_RV);
    model.RV(1);
    model.IV(1);
    model.OPTIONS_JUMPS_TEST_BV(1,0,0,0.999);
    model.SetSelSampleByDates( dayofcalendar(1987, 1, 5), dayofcalendar(1998, 7, 3));
    model.Estimate();
    delete model;
}
```

All 288 variables Ret_1 to Ret_288, one for each 5 minute interval, are added to the model.
6 STAMP

STAMP is designed to model and forecast time series, based on structural time series models. These models use advanced estimation techniques, such as Kalman filtering, but are formulated in STAMP using convenient dialogs — at the most basic level all that is required is some appreciation of the concepts of trend, seasonal and irregular. The hard work is done by the program, leaving the user free to concentrate on formulating models, then using them to make forecasts.

Structural time series modelling can be applied to a variety of problems in time series. Macro-economic time series like gross national production, inflation and consumption can be handled effectively, but also financial time series, like interest rates and stock market volatility, can be modelled using STAMP. Further, STAMP is used for modelling and forecasting time series in medicine, biology, engineering, marketing and in many other areas.

STAMP features

- Univariate State Space Models
- Multivariate State Space Models
- Time-varying regression coefficients
- Autoregressive processes of order 1 and 2
- High order Smooth Cycles
- Automatic outlier and break detection
- A battery of equation mis-specification tests
- Post-sample and within sample predictive testing
- Easy to use, menu driven interface
- Advanced graphics capabilities

Example

We will consider the basic structural time-series model (BSM) with unobserved components (UC) trend, seasonal and irregular and use it for the analysis of a time series with missing entries. After loading ENERGYmiss.in7 in OxMetrics, select the ofuELl variable. This is quarterly UK electricity consumption between 1960 and 1986 (millions of useful therms for 'Other final users').

Accept the model settings as they are by default:

The model graph shows the decomposition in trend, seasonal, and irregular:

We can see how STAMP recreated the missing values. First use Store in Database from the Test menu

To store the trend, seasonal and irregular under their default names of Level, Seasonal, Irregular in the database. Next, use Algebra code to construct:

\[ y = \text{Level} + \text{Seasonal} + \text{Irregular}; \]

The top of the following graph shows ofuELl as a solid blue line, and the reconstructed variable as a dotted line.

In this particular example, we actually know the full series. This is shown in the bottom part of the graph above. It shows that the reconstruction worked remarkably well.

Generating batch code

After STAMP has estimated the parameters of the BSM, the standard output is sent to the Text/Results and the Graphics/Model windows. The graphical output is as presented as the first figure. The batch option is activated from the Model/Batch option (Alt-B). The Batch code as given by

```c
module("STAMP");
package("UCstamp");
usedata("ENERGYmiss.in7");
system
{
    Y = ofuELl;
}
```
The code represents the model in STAMP after maximum likelihood estimation. The commands module and package are required to start the STAMP module in OxMetrics. The command usedata loads the data file "ENERGYmiss.in7" while the command system assigns the variable ofuELl as the dependent variable that we want to analyze. The unobserved components model is constructed by the commands setcmp which introduces the components level with slope (trend), seasonal and irregular. The third arguments are the estimated variances for the components. The model formulation for our unobserved components model is completed by the command setmodel. In case we prefer a deterministic seasonal component, we can fix the seasonal variance by setting the variance to zero. The modified line is:

```cpp
setcmp("seasonal", 4, 0, 0, 0);
```

When the variance is set to zero in the setcmp command, STAMP will treat the component as deterministic (the component is fixed over time). A part of the STAMP output in the Results window reports the maximum likelihood estimates of the variances:

<table>
<thead>
<tr>
<th>Components</th>
<th>Value</th>
<th>(q-ratio)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level</td>
<td>0.000000</td>
<td>( 0.0000)</td>
</tr>
<tr>
<td>Slope</td>
<td>1.32900e-06</td>
<td>( 0.0002261)</td>
</tr>
<tr>
<td>Seasonal</td>
<td>0.000000</td>
<td>( 0.0000)</td>
</tr>
<tr>
<td>Irregular</td>
<td>0.00587911</td>
<td>( 1.000)</td>
</tr>
</tbody>
</table>

It follows that the Level component is also estimated as zero which leads to a smooth trend component in STAMP. Other useful Batch commands are intervention for including interventions in the model, forecast for generating forecasts from the model, store for storing residuals and estimated components from STAMP and teststate for printing the estimated state vector. These commands are documented in the STAMP manual.

### Generating Ox code

An alternative but a more flexible method of running STAMP in batch is by means of the Ox code generator facility. It is activated by pressing Alt-O when STAMP is activated. In case a model is formulated in STAMP and parameters are estimated, the option Alt-O opens the menu window Generate Ox code in which the user has two options. The default choice is Most recent model and can be accepted. STAMP then outputs the following Ox code:

```cpp
#include <oxstd.oxh>
#include <packages/stamp/stamp_ox_uc>
main()
{
    decl model = new UCstamp();

    model.Load("C:\\Users\\...\\Documents\\OxMetrics8\\" "data\\ENERGYmiss.in7");
    model.Deterministic(-1);

    model.Select("Y", {"ofuELl", 0, 0});

    model.SetSelSample(1960, 3, 1986, 1);
    model.SetMethod("MAXSTAMP");
    // Specify components
    model.StartStamp();
    model.AddComponent(COM_LEVEL, 1, 0.000387097);
    model.AddComponent(COM_SLOPE, 1, 1.87978e-06);
    model.AddComponent(COM_SEASONAL, 4, 0.00017307);
    model.AddComponent(COM_IRREG, 0, 0.000524284);
    model.Estimate();
    delete model;
}
```

When STAMP is installed on the computer, the stamp_ox_uc library offers many Ox functions that are developed for STAMP. These functions are collected in the class UCstamp which is activated by the new command in Ox.

The command Load reads in the data file ENERGYmiss.in7 and Select takes the variable ofuELl as the dependent variable to analyze. In the Ox code, we can first select the sample and the estimation method using the commands SetSelSample and SetMethod, respectively. The command StartStamp initializes the settings for formulating an UC model. The next part of the Ox code is similar to the Batch code. The inclusion of a component is established by the command AddComponent. In case of the seasonal component, the constant COM_SEASONAL indicates that the seasonal component is included in the model. The second constant indicates that we work with quarterly data (seasonal length is 4) and the value 0.00017307 is the value of the seasonal variance, as estimated by the STAMP program. The Estimate command (re-)estimates the variances (and, possibly, other parameters). Other commands in the stamp_ox_uc library are AddIntervention for including interventions in the model, GetForecast for generating forecasts from the model, Store for storing residuals and estimated components from STAMP and PrintState for printing the estimated state vector.
## 7 Ox

Ox is an **object-oriented matrix language** with a comprehensive mathematical and statistical function library. Matrices can be used directly in expressions, for example to multiply two matrices, or to invert a matrix. The **basic syntax elements** of Ox are similar to the C++ and Java languages (however, knowledge of these languages is not a prerequisite for using Ox). This similarity is most clear in syntax items such as loops, functions, arrays and classes. A major difference is that Ox variables have **no explicit type**, and that special **support for matrices** is available.

```cpp
#include <oxstd.oxh> // include Ox standard library header

test1__ = unit(3); // assign to m1 a 3 x 3 identity matrix

test2__ = m1[0][0] = 2; // set top-left element to 2

dbl = oxstd::ln(test2__); // perform log

dbl = oxstd::max(test1__, test2__); // take maximum

dbl = oxstd::min(test1__, test2__); // take minimum

dbl = oxstd::mean(test1__, test2__); // take mean

dbl = oxstd::cumsum(test2__); // cumulative sums

dbl = oxstd::cummax(test2__); // cumulative maximum

dbl = oxstd::cummin(test2__); // cumulative minimum

dbl = oxstd::psum(test2__); // partial sums

dbl = oxstd::eval(test2__); // evaluate

main() // function main is the starting point
{
    decl m1, m2; // declare two variables, m1 and m2

    m1 = unit(3); // assign to m1 a 3 x 3 identity matrix
    m1[0][0] = 2; // set top-left element to 2
    m2 = oxstd::zeros(2, 3); // m2 is a 2 x 3 matrix, the first row consists of zeros, the second of ones

    println("two_matrices", m1, m2); // print the matrices
}
```

### Object oriented

The advantages of object-oriented programming are that it potentially improves the clarity and maintainability of the code, and reduces coding effort through inheritance. Several useful classes are provided with Ox. The following example uses the **Database class**:

```cpp
#include <oxstd.oxh>
import <database>

main()
{
    decl db = new Database();
    db.Load("data/data.in7");
    db.Info();
    delete db;
}
```

A dynamic regression model can be estimated using the **PcFiml class**:

```cpp
#include <oxstd.oxh>
import <pcfiml>

main()
{
    decl mod = new PcFiml();

    mod.Load("data/data.in7");
    mod.Deterministic(FALSE);
    mod.Select(\"Y\", \"CONS\", 0, 1); // create deterministic vars in the database for dependent and lagged dependent variables, \"X\" for the other regressors. The second argument is an array with three elements: variable name, start lag and end lag.
    mod.Select(\"Y\", \"INC\", 0, 1);
    mod.Select(\"Y\", \"Constant\", 0, 0);

    mod.SetSelSample(-1, 1, -1, 1); // max sample
    mod.Estimate(); // estimate the model

    delete mod;
}
```

This estimates a model by ordinary least squares:

\[
y_t = \beta_0 + \beta_1 y_{t-1} + \beta_2 x_t + \beta_3 x_{t-1} + \epsilon_t,
\]

where \( y_t \) is CONS, \( x_t \) is INC (income).

**Ox**

- Estimate estimates and prints the results. How much work would this have been starting from scratch?
- Finally, when done, we **delete** the object. This calls the destructor function, and then clears the object from memory.

The inheritance structure for the PcFiml class is:

```
Sample └── Database └── Modelbase └── your class
```

- Stores sample frequency and period.
- Model formulation, extract data for modelling, initialize parameters, estimate, print a report.
- Single or multiple equations dynamic regressions, Mis-specification testing and cointegration analysis.

### Ox programming

- Ox is structured as a proper programming language:
  - Somewhat harder for very small programs;
  - Much better for larger projects.
- Can be used to teach programming to economics/statistics students
- while being able to write relevant applications
- C/C++ like structure integrates well with business environment
- Facilities to easily create interactive (possibly commercial) applications for OxMetrics.

### Mathematical, Statistical and Graphical library

- Many matrix and statistical functions.
- Maximization functions (Unconstrained: BFGS, Newton, OP, SQP, FSQP, NLE), numerical differentiation.
- Random number generations/quantiles/probabilities of many statistical distributions.
- Time-series functions.
- Many graph types.
- Load and save data files.
- Date and time functions.

### Further features

- It is possible to mix high and low level code:
  - add C or Fortran procedure libraries to Ox, e.g. time critical sections, or use available libraries (e.g. SsfPack);
  - use Ox procedures from C.
- Call Ox from other languages.
- Write interface for Ox program in another languages.
- Web hosted environment (under development).

### OxPack

The following structure

```
Database class └── Modelbase class └── your class
```

makes it easy to create an interactive version that can be used via OxPack.

Examples are: Arfima, PcNaive, DPD, G@RCH, etc.

The benefits are:

- One code base for all versions: simulation, estimation, incorporation, as well as GUI version;
- Dialogs written in Ox using code that is easy to write and maintain:
  - **Simple** code,
  - Convenient run/development cycle,
  - No need to be professional programmer.

### Documentation

- Introduction to Ox
- Ox: An Object-oriented Matrix Programming Language
- Developer's manual for Ox
### SsfPack

SsfPack is a suite of C routines for carrying out computations involving the statistical analysis of **univariate and multivariate models in state space form** with easy-to-use functions for Ox. SsfPack allows for a full range of different state space forms: from a simple time-invariant model to a complicated multivariate time-varying model.

Functions are provided to put standard models such as SARIMA, unobserved components, time-varying regressions and cubic spline models into state space form. Basic functions are available for Kalman filtering, moment smoothing and simulation smoothing. Ready-to-use functions are provided for standard tasks such as likelihood evaluation, forecasting and signal extraction.

SsfPack can be easily used for implementing, fitting and analysing Gaussian models relevant to many areas of econometrics and statistics. Furthermore it provides all relevant tools for the treatment of non-Gaussian and nonlinear state space models. In particular, tools are available to implement **simulation based estimation methods** such as importance sampling and Markov chain Monte Carlo (MCMC) methods.

### Examples of SsfPack functions

#### Models in state space form

- **GetSsfSarima()** puts Seasonal ARIMA model in state space
- **GetSsfStsm()** puts UC model with higher order trends and cycles in state space

#### General state space algorithms

These use the new more robust filtering algorithms with exact diffuse initialisations.

- **KalmanFilInit()** returns initial output the exact diffuse Kalman filter
- **KalmanFilEx()** returns output of the Kalman filter
- **KalmanSmoEx()** returns output of the basic smoothing algorithm
- **KalmanFilMeanEx()** returns Kalman filter output to implement diffuse initialization based on augmentation method
- **KalmanSmoMeanEx()** returns Kalman smoother output to implement diffuse initialization on augmentation method
- **KalmanFilSmoMeanEx()** alternative Kalman smoother output to implement diffuse initialization on augmentation method

#### Likelihood functions

These use the new more robust filtering algorithms with exact diffuse initialisations.

- **SsfLikEx()** returns log-likelihood function
- **SsfLikConcEx()** returns profile log-likelihood function
- **SsfLikScoEx()** returns score vector

#### Ready-to-use functions

These use the new more robust filtering algorithms with exact diffuse initialisations.

- **SsfMomentEstEx()** returns output from prediction, forecasting and smoothing
- **SsfCondDensEx()** returns mean or a draw from the conditional density
- **SsfForecast()** forecasting function with simplified syntax
- **SsfWeightsEx()** returns observation weights of state and signal estimates
- **SsfBootstrap()** returns bootstrap simulation draws