Package 'dse1'

July 11, 2007

Title Dynamic Systems Estimation (time series package)

Description Multivariate Time Series - ARMA and State Space models. See ?dse.Intro for more details.

Depends R (>= 2.5.0), tframe (>= 2007.5-3), setRNG (>= 2004.4-1)

Version 2007.7-1

Date 2007-07-11

LazyLoad yes

License Free. See the LICENCE file for details.

Author Paul Gilbert <pgilbert@bank-banque-canada.ca>

Maintainer Paul Gilbert <pgilbert@bank-banque-canada.ca>

URL http://www.bank-banque-canada.ca/pgilbert

R topics documented:

`00.dse.Intro` *Dynamic Systems Estimation - Multivariate Time Series Package*

Description

Functions for multivariate time series modeling

Details

See `dse-package` (in the help system use package?dse or ?"dse-package") for an overview.

`acf` *Calculate the acf for an object*

Description

Calculate the acf for an object.

Usage

```
## S3 method for class 'TSdata':
acf(x, ...)
## S3 method for class 'TSestModel':
acf(x, ...)
```

Arguments

`x`	an object.
`...`	additional arguments passed to stats::acf.

Details

The default method uses `stats::acf`. The method for TSdata extracts the output data and then uses acf. The method for TSestModel calculates the residuals (prediction minus data) and then uses acf.

Value

see stats::acf.

Author(s)

Paul Gilbert

See Also

`stats::acf`,

`addPlotRoots` *Add Model Roots to a plot*

Description

Calculate and plot roots of a model.

Usage

```
addPlotRoots(v, pch='*', fuzz=0)
```

Arguments

`v`	An object containing a TSmodel.
`pch`	Character to use for plotting.
`fuzz`	If non-zero then roots within fuzz distance are considered equal.

Value

The eigenvalues of the state transition matrix or the inverse of the roots of the determinant of the AR polynomial are returned invisibly.

Side Effects

The roots are addeded to an existing plot.

See Also

`plot.roots`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- estVARXls(eg1.DSE.data.diff)
plot(roots(model))
addPlotRoots(toSS(model))
```

ARMA — *ARMA Model Constructor*

Description

Constructs an ARMA TSmodel object as used by the DSE package.

Usage

```
ARMA(A=NULL, B=NULL, C=NULL, TREND=NULL,
      constants=NULL,
      description=NULL, names=NULL, input.names=NULL, output.names=NULL)
is.ARMA(obj)
```

Arguments

`A`	The auto-regressive polynomial, an axpxp array.
`B`	The moving-average polynomial, an bxpxp array.
`C`	The input polynomial, an cxpxm array. C should be NULL if there is no input
`TREND`	A matrix, p-vector, or NULL.
`constants`	NULL or a list of logical arrays with the same names as arrays above, indicating which elements should be considered constants.
`description`	An arbitrary string.
`names`	A list with elements input and output, each a vector of strings. Arguments input.names and output.names should not be used if argument names is used.
`input.names`	A vector of strings.
`output.names`	A vector of strings.
`obj`	Any object.

Details

The ARMA model is defined by:

A(L)y(t) = B(L)w(t) + C(L)u(t) + TREND(t)

where

A (axpxp) is the auto-regressive polynomial array.

B (bxpxp) is the moving-average polynomial array.

C (cxpxm) is the input polynomial array. C should be NULL if there is no input

y is the p dimensional output data.

u is the m dimensional control (input) data.

TREND is a matrix the same dimension as y, a p-vector (which gets replicated for each time period), or NULL.

This is sometime called a vector ARMA (VARMA) model, but the univariate case is also handled by this structure. VAR models are a special case where `B(L) = I`. ARIMA models are also special cases where the polynomial arrays have unit roots, but these are not distinguished in a separate term as is sometimes done in other programs.

The name of last term, `TREND`, is misleading. If it is NULL it is treated as zero. If it is a p-vector, then this constant vector is added to the the p-vector `y(t)` at each period. For a stable model this would give the none zero mean, and might more appropriately be called the constant or intercept rather than trend. If the model is for differenced data, then this mean is the trend of the undifferenced model. The more general case is when `TREND` is a time series matrix of the same dimension as `y`. In this case it is added to `y`. This allows for a very general deterministic component, which may or may not be a traditional trend.

By default, elements in parameter arrays are treated as constants if they are exactly 1.0 or 0.0, and as parameters otherwise. A value of 1.001 would be treated as a parameter, and this is the easiest way to initialize an element which is not to be treated as a constant of value 1.0. Any array elements can be fixed to constants by specifying the list `constants`. Arrays which are not specified in the list will be treated in the default way. An alternative for fixing constants is the function `fixConstants`.

The function `ARMA` sets up a model but does not estimate it. See `estVARXls` for one possibility for estimating VAR models and `estMaxLik` for one possibility for estimating ARMA models.

Value

An ARMA TSmodel

See Also

`TSmodel`, `simulate.ARMA`, `fixConstants` `estVARXls` `estMaxLik`

Examples

```
mod1 <- ARMA(A=array(c(1,-.25,-.05), c(3,1,1)), B=array(1,c(1,1,1)))
AR   <- array(c(1, .5, .3, 0, .2, .1, 0, .2, .05, 1, .5, .3) ,c(3,2,2))
VAR  <- ARMA(A=AR, B=diag(1,2))
C    <- array(c(0.5,0,0,0.2),c(1,2,2))
VARX <- ARMA(A=AR, B=diag(1,2), C=C)
MA   <- array(c(1, .2, 0, .1, 0, 0, 1, .3), c(2,2,2))
ARMA  <- ARMA(A=AR, B=MA, C=NULL)
ARMAX <- ARMA(A=AR, B=MA, C=C)
```

`balanceMittnik` *Balance a state space model*

Description

Balance a state space model a la Mittnik.

Usage

```
balanceMittnik(model, n=NULL)
SVDbalanceMittnik(M, m, n=NULL)
```

Arguments

`model`	An TSmodel object.
`M`	a matrix. See details in `MittnikReduction`.
`m`	an integer indicating the number of input series in the model.
`n`	see details

Details

`balanceMittnik` calculate a state space model balance a la Mittnik. `n` is intended primarily for producing a state space model from the markov parameters of an `ARMA` model, but if it is supplied with an `SS` model the result will be a model with state dimension n based on the n largest singular values of the svd of a Hankel matrix of markov parameters generated by the original model. If `n` is not supplied then the singular values are printed and the program prompts for `n`. `balanceMittnik` calls `SVDbalanceMittnik`

`SVDbalanceMittnik` calculates a nested-balanced state space model by svd a la Mittnik. If state dim n is supplied then svd criteria are not calculated and the given n is used. Otherwise, the singular values are printed and the program prompts for n. M is a matrix with p x (m+p) blocks giving the markov parameters, that is, the first row of the Hankel matrix. It can be generated from the model as in the function markovParms, or from the data, as in the function estSSMittnik. m is the dimension of input series, which is needed to decompose M. The output dimension p is taken from nrow(M).

See also `MittnikReduction` and references.

Value

A state space model in a TSestModel object.

References

See references for `MittnikReduction`.

See Also

`estVARXls`, `estVARXar` `MittnikReduction`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- toSS(TSmodel(estVARXls(eg1.DSE.data.diff)))
## Not run: newmodel <-balanceMittnik(model)
# this prints information about singular values and prompts with
#Enter the number of singular values to use for balanced model:
# 18 might be a good choice in this example.
newmodel <-balanceMittnik(model, n=18)
```

`bestTSestModel` *Select Best Model*

Description

Select the best model.

Usage

```
bestTSestModel(models, sample.start=10, sample.end=NULL,
 criterion='aic', verbose=TRUE)
```

Arguments

`models`	a list of TSestModels.
`sample.start`	the starting point to use for calculating information criteria.
`sample.end`	the end point to use for calculating information criteria.
`criterion`	Criterion to be used for model selection. see `informationTestsCalculations`. 'taic' would be a better default but this is not available for VAR and ARMA models.
`verbose`	if TRUE then additional information is printed.

Details

Information criteria are calculated and return the best model from ... according to criterion models should be a list of TSestModel's. models[[i]]$estimates$pred is not recalculated but a subsample identified by sample.start and sample.end is used and the likelihood is recalculated. If sample.end=NULL data is used to the end of the sample. taic might be a better default selection criteria but it is not available for ARMA models.

Value

A TSestModel

See Also

`estBlackBox1`, `estBlackBox2` `estBlackBox3` `estBlackBox4` `informationTestsCalculations`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
models <- list(estVARXls(eg1.DSE.data.diff), estVARXar(eg1.DSE.data.diff))
z <-  bestTSestModel(models)
```

`checkBalanceMittnik` *Check Balance of a TSmodel*

Description

Calculate the difference between observability and reachability gramians of the model transformed to Mittnik's form.

Usage

```
checkBalanceMittnik(model)
## S3 method for class 'ARMA':
checkBalanceMittnik(model)
## S3 method for class 'SS':
checkBalanceMittnik(model)
## S3 method for class 'TSestModel':
checkBalanceMittnik(model)
```

Arguments

`model`	An object of class TSmodel.

Details

Balanced models should have equal observability and reachability gramians.

Value

No value is returned.

Side Effects

Differences between the observability and reachability gramians are printed.

See Also

`checkBalance` `MittnikReduction`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- toSS(estVARXls(eg1.DSE.data.diff))
checkBalanceMittnik(model)
```

checkBalance *Check Balance of a TSmodel*

Description

Calculate the difference between observability and reachability gramians.

Usage

```
checkBalance(model)
## S3 method for class 'SS':
checkBalance(model)
## S3 method for class 'ARMA':
checkBalance(model)
## S3 method for class 'TSestModel':
checkBalance(model)
```

Arguments

`model`	A TSmodel object.

Details

Balanced models should have equal observability and reachability gramians.

Value

No value is returned.

Side Effects

Differences between the observability and reachability gramians are printed.

See Also

`checkBalanceMittnik` `MittnikReduction`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- toSS(estVARXls(eg1.DSE.data.diff))
checkBalance(model)
```

checkConsistentDimensions

Check Consistent Dimensions

Description

Check that dimensions of a model and data agree.

Usage

```
checkConsistentDimensions(obj1, obj2=NULL)
## Default S3 method:
checkConsistentDimensions(obj1, obj2=NULL)
## S3 method for class 'ARMA':
checkConsistentDimensions(obj1, obj2=NULL)
## S3 method for class 'SS':
checkConsistentDimensions(obj1, obj2=NULL)
## S3 method for class 'TSdata':
checkConsistentDimensions(obj1, obj2=NULL)
## S3 method for class 'TSestModel':
checkConsistentDimensions(obj1, obj2=NULL)
```

Arguments

`obj1`	An object containing a TSmodel, TSdata, or TSestModel, depending on the method
`obj2`	Another object containing TSdata corresponding to the TSmodel in obj1, or a TSmodel corresponding to the TSdata in obj1.

Details

Check that dimensions of a model and data agree. If `obj1` is a `TSestModel` then if `obj2` is `NULL, TSdata` is taken from `obj1`.

Value

logical

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- estVARXls(eg1.DSE.data.diff)
checkConsistentDimensions(model)
```

checkResiduals *Autocorrelations Diagnostics*

Description

Calculate autocorrelation diagnostics of a time series matrix or TSdata or residuals of a TSestModel

Usage

```
checkResiduals(obj, ...)
## Default S3 method:
checkResiduals(obj, ac=TRUE, pac=TRUE, select=seq(nseries(obj)),
               drop=NULL, plot.=TRUE, graphs.per.page=5, verbose=FALSE, ...)
## S3 method for class 'TSdata':
checkResiduals(obj, ...)
## S3 method for class 'TSestModel':
checkResiduals(obj, ...)
```

Arguments

- `obj` An TSestModel or TSdata object.
- `ac` If TRUE the auto-correlation function is plotted.
- `pac` If TRUE the partial auto-correlation function is plotted.
- `select` Is used to indicate a subset of the residual series. By default all residuals are used.
- `drop` Is used to indicate a subset of the residual time periods to drop. All residuals are used with the default (NULL).Typically this can be used to get rid of bad initial conditions (eg. drop=seq(10)) or outliers.
- `plot.` If FALSE then plots are not produced.
- `graphs.per.page` Integer indicating number of graphs to place on a page.
- `verbose` If TRUE then the auto-correlations and partial auto-correlations are printed if they are calculated.
- `...` arguments passed to other methods.

Details

This is a generic function. The default method works for a time series matrix which is treated as if it were a matrix of residuals. However, in a Box-Jenkins type of analysis the matrix may be data which is being evaluated to determine a model. The method for a TSestModel evaluates the residuals calculated by subtracting the output data from the model predictions.

Value

A list with residual diagnostic information: residuals, mean, cov, acf= autocorrelations, pacf= partial autocorrelations.

Side Effects

Diagnostic information is printed and plotted if a device is available. Output graphics can be paused between pages by setting par(ask=TRUE).

See Also

informationTests, Portmanteau

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- estVARXls(eg1.DSE.data.diff)
checkResiduals(model)
```

---

coef.TSmodel *Extract or set Model Parameters*

Description

Set or extract coefficients (parameter values) of model objects.

Usage

```
## S3 method for class 'TSmodel':
coef(object, ...)
## S3 method for class 'TSestModel':
coef(object, ...)
coef(object) <- value
## S3 replacement method for class 'default':
coef(object) <- value
```

Arguments

object	An object of class TSmodel or TSestModel.
value	value to be assigned to object.
...	(further arguments, currently disregarded).

Value

A vector of parameter values.

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- estVARXls(eg1.DSE.data.diff)
coef(model)
coef(model) <- 0.1 + coef(model)
```

combine — *Combine two objects.*

Description

This is a generic method to combine two objects of the same class to make a single object of that class.

Usage

```
combine(e1, e2)
## Default S3 method:
combine(e1, e2)
```

Arguments

`e1, e2`	TSdata objects.

Value

An object of the same class as the argument but containing both e1 and e2.

See Also

tbind, combine.TSdata, combine.forecastCov

Examples

```
data("eg1.DSE.data.diff", package="dse1")
data("eg1.DSE.data", package="dse1")
new.data.set <- combine(eg1.DSE.data.diff, eg1.DSE.data)
```

combine.TSdata — *Combine series from two TSdata objects.*

Description

Combine series from two TSdata objects.

Usage

```
## S3 method for class 'TSdata':
combine(e1, e2)
```

Arguments

`e1, e2`	TSdata objects.

Value

An object of class TSdata which includes series from both e1 and e2.

See Also

tbind

Examples

```
data("eg1.DSE.data.diff", package="dse1")
data("eg1.DSE.data", package="dse1")
new.data.set <- combine(eg1.DSE.data.diff, eg1.DSE.data)
```

DSEflags — *Flags to Indicate Use of Compiled Code*

Description

Determines if compiled code should be used or not.

Usage

```
.DSEflags(new)
```

Arguments

`new`	A list which must have elements COMPILED and DUP.

Details

Setting flags with this function is primarily for debugging. It should not normally be needed by users. If called with not arguments, `.DSEflags()` returns the current setting. Several **dse** functions which call compiled fortran or C code will use the equivalent S/R version if the `.DSEflags()$COMPILED` returns `FALSE`. `.DSEflags()$DUP` is passed to fortran calls and controls whether R duplicates arguments passed to the fortran code. The safe setting is `TRUE`. Setting `FALSE` saves some memory but does not seem to give much speed gain.

Side Effects

The flag settings affect whether and how compiled fortran or C code is called.

Examples

```
.DSEflags(list(COMPILED=TRUE, DUP=TRUE))
.DSEflags()$COMPILED
```

`dse-package` *Dynamic Systems Estimation - Multivariate Time Series Package*

Description

Functions for time series modeling, including multi-variate state-space and ARMA (VAR, ARIMA, ARIMAX) models.

Details

A *Brief User's Guide* is distributed with the **dse** bundle in the directory 'dse/dse1/inst/doc/dse-guide.pdf'. The package implements an R/S style object oriented approach to time series modeling. This means that different model and data representations can be implemented with fairly simple extensions to the package.

The package includes methods for simulating, estimating, and converting among different model representations. These are mainly in **dse1**. Package **dse2** has methods for studying estimation techniques and for examining the forecasting properties of models. There are also functions for forecasting and for evaluating the performance of forecasting models, as well as functions for evaluating model estimation techniques.

Bundle:	dse
Contains:	tframe dse1 dse2
Depends:	R, setRNG, tframe
License:	free, see LICENSE file for details.
URL:	http://www.bank-banque-canada.ca/pgilbert

The main objects are:

TSdata time series input and output data structure

TSmodel a DSE model structure

TSestModel model, data and some estimation information

The main general methods are:

TSdata create, extract a DSE data structure

TSmodel create, extract a DSE model structure

simulate simulate a model to produce artifical data

toSS convert to a state-space model

toARMA convert to an ARMA model

ARMA construct an ARMA model

SS construct a state-space model

l evaluate a model with data

smoother calculate the smoothed state estimate

The main estimation methods are:

estVARXls estimate an ARMA model with least squares

estVARXar estimate an ARMA model with ar

estSSfromVARX calculate a state-space model from an estimated VAR model

bft a (usually) good “black-box” estimated model

estMaxLik estimate a model using maximum likelihood

The main diagnositic methods are:

checkResiduals autocorrelation diagnostics

informationTests calculate several information tests for a model

McMillanDegree calculate the McMillanDegree of a model

stability calculate the stability of a model

roots calculate the roots of a model

The methods for producing and evaluating forecasts are:

l evaluate a model with data (and simple forecasts)

forecast calculate forecasts

featherForecasts calculate forecasts starting at different periods

horizonForecasts calculate forecasts at different horizons

forecastCov calculate the covariance of forecasts

MonteCarloSimulations multiple simulations

The methods for evaluating estimation methods are:

EstEval evaluate estimation methods

The functions described in the *Brief User’s Guide* and examples in the help pages should work fairly reliably (since they are tested regularly), however, the code is distributed on an “as-is” basis. This is a compromise which allows me to make the software available with minimum effort. This software is not a commercial product. It is the by-product of ongoing research. Error reports, constructive suggestions, and comments are welcomed.

Usage

library("dse1")

library("dse2")

Author(s)

Paul Gilbert <pgilbert@bank-banque-canada.ca>

Maintainer: Paul Gilbert <pgilbert@bank-banque-canada.ca>

References

Anderson, B. D. O. and Moore, J. B. (1979) *Optimal Filtering*. Prentice-Hall.

Gilbert, P. D. (1993) State space and ARMA models: An overview of the equivalence. Working paper 93-4, Bank of Canada. Available at `www.bank-banque-canada.ca/pgilbert`

Gilbert, P. D. (1995) Combining VAR Estimation and State Space Model Reduction for Simple Good Predictions. *J. of Forecasting: Special Issue on VAR Modelling.* **14**:229–250.

Gilbert, P.D. (2000) A note on the computation of time series model roots. *Applied Economics Letters*, **7**, 423–424

Jazwinski, A. H. (1970) *Stochastic Processes and Filtering Theory*. Academic Press.

See Also

TSdata, TSmodel, TSestModel.object

DSEutilities *DSE Utilities*

Description

These functions are used by other functions in DSE and are not typically called directly by the user.

Usage

```
estVARXmean.correction(X, y, bbar, fuzz=sqrt(.Machine$double.eps), warn=TRUE)
fake.TSestModel.missing.data(model,data, residual, max.lag, warn=TRUE)
printTestValue(x, digits=16)
svd.criteria(sv)
criteria.table.heading()
criteria.table.legend()
criteria.table.nheading()
DSE.ar(data, ...)
```

DSEversion *Print Version Information*

Description

Print version information.

Usage

```
DSEversion()
```

Examples

```
DSEversion()
```

`eg1.DSE.data` *Four Time Series used in Gilbert (1993)*

Description

Data is for Canada. The series start in March 1961 (April 1961 for `eg1.DSE.data.diff`) and end in June 1991, giving 364 observations on each variable (363 for `eg1.DSE.data.diff`).

The input series is 90-day interest rates (R90) in both `eg1.DSE.data` and `eg1.DSE.data.diff`.

The output series are M1, GDP lagged two months, and CPI. M1, GDP and CPI were all seasonally adjusted data. These are not transformed in `eg1.DSE.data` and are first difference of logs in `eg1.DSE.data.diff`.

GDP is lagged because it is not available on as timely a basis. (The data was used in an example where the intent was to build a model for timely monitoring.)

The Statistics Canada series identifiers are B14017, B1627, I37026, and B820200.

The data for M1 (B1627) were taken prior to revisions made in December 1993.

The file `eg1.dat` contains the same data as `eg1.DSE.data` in a simple ASCII file.

Usage

```
data(eg1.DSE.data)
data(eg1.DSE.data.diff)
```

Format

The objects `eg1.DSE.data` and `eg1.DSE.data.diff` are `TSdata objects`. The file `eg1.dat` is an ASCII file with 5 columns, the first enumerating the observations, the second giving the input series, and the third to fifth giving the output series. The input series name is "R90" and the output series names are "M1", "GDPl2" and "CPI". GDPl2 is GDP lagged two months

Source

Statistics Canada, Bank of Canada.

References

Gilbert, P.D. (1993) State Space and ARMA Models: An Overview of the Equivalence. Bank of Canada Working Paper 93-4. Also available at `www.bank-banque-canada.ca/pgilbert`.

See Also

`TSdata`

`egJofF.1dec93.data` *Eleven Time Series used in Gilbert (1995)*

Description

Data is for Canada unless otherwise indicated. The series start in February 1974 and end in September 1993 (236 observations on each variable).

The input series is 90 day interest rates (R90) and the ten output variables are CPI, GDP, M1, long run interest rates (RL), the Toronto stock exchange 300 index (TSE300), employment, the Canada/US exchange rate (PFX), a commodity price index in US dollars, US industrial production, and US CPI.

R90, RL and TSE are differenced. All other variables are in terms of percent change.

R90 is the 3 month prime corporate paper rate. While it is not set directly by the Bank of Canada, Bank policy influences it directly and it is often thought of as a proxy "policy variable."

The Statistics Canada identifiers are B14017 (R90), P484549 (CPI), I37026 (GDP), B1627 (M1), B14013 (RL), B4237 (TSE300), D767608 (employment), B3400 (PFX).

M.BCPI (commodity price index) is published by the Bank of Canada. JQIND (US industrial production), and CUSA0 (US CPI) are DRI identifiers.

The data for M1 (B1627) were taken prior to revisions made in December 1993.

Usage

```
data(egJofF.1dec93.data)
```

Format

This data is a TSdata object. The input series name is "R90" and the output series names are "CPI", "GDP", "M1", "RL", "TSE300", "employment", "PFX", "commod.price index", "US ind.prod." and "US CPI"

Source

Statistics Canada, Bank of Canada, DRI.

References

Gilbert, P.D. 1995 "Combining VAR Estimation and State Space Model Reduction for Simple Good Predictions" *J. of Forecasting: Special Issue on VAR Modelling.* 14:229-250

See Also

`TSdata`

`estBlackBox1` *Estimate a TSmodel*

Description

Estimate a TSmodel.

Usage

```
estBlackBox1(data, estimation="estVARXls",
    reduction="MittnikReduction",
    criterion="taic", trend=FALSE, subtract.means=FALSE,
    verbose=TRUE, max.lag=6)
```

Arguments

`data`	Data in an object of class TSdata.
`estimation`	Initial estimation method to be used.
`reduction`	Reduction method to be used.
`criterion`	Criterion to be used for model selection. see `informationTestsCalculations`.
`trend`	logical indicating if a trend should be estimated.
`subtract.means`	logical indicating if the mean should be subtracted from data before estimation.
`verbose`	logical indicating if information should be printed during estimation.
`max.lag`	integer indicating the maximum number of lags to consider.

Value

A state space model in an object of class TSestModel.

Side Effects

If verbose is TRUE then estimation information is printed and checkResiduals is run, which gives plots of information about the residuals.

See Also

`informationTestsCalculations`

Examples

```
data("egJofF.1dec93.data", package="dse1")
goodmodel <- estBlackBox1(egJofF.1dec93.data)
```

estBlackBox2 *Estimate a TSmodel*

Description

Estimate a TSmodel.

Usage

```
estBlackBox2(data, estimation='estVARXls',
      lag.weight=.9,
      reduction='MittnikReduction',
      criterion='taic',
      trend=FALSE,
      subtract.means=FALSE,  re.add.means=TRUE,
      standardize=FALSE, verbose=TRUE, max.lag=12)
```

Arguments

`data`	a TSdata object.
`estimation`	a character string indicating the estimation method to use.
`lag.weight`	weighting to apply to lagged observations.
`reduction`	character string indicating reduction procedure to use.
`criterion`	criterion to be used for model selection. see `informationTestsCalculations`.
`trend`	if TRUE include a trend in the model.
`subtract.means`	if TRUE the mean is subtracted from the data before estimation.
`re.add.means`	if subtract.means is TRUE then if re.add.means is TRUE the estimated model is converted back to a model for data without the mean subtracted.
`standardize`	if TRUE the data is transformed so that all variables have the same variance.
`verbose`	if TRUE then additional information from the estimation and reduction procedures is printed.
`max.lag`	The number of lags to include in the VAR estimation.

Details

A model is estimated and then a reduction procedure applied. The default estimation procedure is least squares estimation of a VAR model with lagged values weighted. This procedure is discussed in Gilbert (1995).

Value

A TSestModel.

References

Gilbert, P.D. (1995) Combining VAR Estimation and State Space Model Reduction for Simple Good Predictions *J. of Forecasting: Special Issue on VAR Modelling*, **14**, 229–250.

See Also

estBlackBox1, estBlackBox3 estBlackBox4 informationTestsCalculations

Examples

```
data("eg1.DSE.data.diff", package="dse1")
z <-  estBlackBox2(eg1.DSE.data.diff)
```

---

estBlackBox3 *Estimate a TSmodel*

Description

Estimate a TSmodel.

Usage

```
estBlackBox3(data, estimation='estVARXls',
       lag.weight=1.0,
       reduction='MittnikReduction',
       criterion='aic',
       trend=FALSE,
       subtract.means=FALSE,  re.add.means=TRUE,
       standardize=FALSE, verbose=TRUE, max.lag=12, sample.start=10)
```

Arguments

`data`	A TSdata object.
`estimation`	A character string indicating the estimation method to use.
`lag.weight`	Weighting to apply to lagged observations.
`reduction`	Character string indicating reduction procedure to use.
`criterion`	Criterion to be used for model selection. see `informationTestsCalculations`. taic might be a better default selection criteria but it is not available for ARMA models.
`trend`	If TRUE include a trend in the model.
`subtract.means`	If TRUE the mean is subtracted from the data before estimation.
`re.add.means`	If subtract.means is TRUE then if re.add.means is T the estimated model is converted back to a model for data without the mean subtracted.
`standardize`	If TRUE the data is transformed so that all variables have the same variance.
`verbose`	If TRUE then additional information from the estimation and reduction procedures is printed.
`max.lag`	The number of lags to include in the VAR estimation.
`sample.start`	The starting point to use for calculating information criteria.

Details

VAR models are estimated for each lag up to the specified max.lag. From these the best is selected according to the specified criteria. The reduction procedure is then applied to this best model and the best reduced model selected. The default estimation procedure is least squares estimation of a VAR model.

Value

A TSestModel.

See Also

estBlackBox1, estBlackBox2 estBlackBox4 informationTestsCalculations

Examples

```
data("eg1.DSE.data.diff", package="dse1")
z <-  estBlackBox3(eg1.DSE.data.diff)
```

---

estBlackBox4 *Estimate a TSmodel*

Description

Estimate a TSmodel with Brute Force Technique.

Usage

```
estBlackBox4(data, estimation="estVARXls",
             lag.weight=1.0,  variable.weights=1,
             reduction="MittnikReduction",
             criterion="taic",
             trend=FALSE, subtract.means=FALSE,  re.add.means=TRUE,
             standardize=FALSE, verbose=TRUE, max.lag=12, sample.start=10, wa
bft(data, ... )
```

Arguments

data	A TSdata object.
estimation	a character string indicating the estimation method to use.
lag.weight	weighting to apply to lagged observations.
variable.weights	weighting to apply to series if estimation method is estWtVariables.
reduction	character string indicating reduction procedure to use.
criterion	criterion to be used for model selection. see informationTestsCalculations.
trend	if TRUE include a trend in the model.
subtract.means	if TRUE the mean is subtracted from the data before estimation.
re.add.means	if subtract.means is TRUE then if re.add.means is T the estimated model is converted back to a model for data without the mean subtracted.

`standardize`	if TRUE the data is transformed so that all variables have the same variance.
`verbose`	if TRUE then additional information from the estimation and reduction procedures is printed.
`max.lag`	VAR estimation is done for each lag up to max.lag.
`sample.start`	the starting point to use for calculating information criteria in the final selection.
`warn`	logical indicating if warning messages should be suppressed.
`...`	arguments passed to estBlackBox4.

Details

For each lag up to max.lag a VAR model is estimated and then a reduction procedure applied to select the best reduced model. Finally the best of the best reduced models is selected. The default estimation procedure is least squares estimation of the VAR models. This procedure is described as the brute force technique (bft) in *Gilbert (1995).*

Value

A TSestModel.

References

Gilbert, P.D. (1995) Combining VAR Estimation and State Space Model Reduction for Simple Good Predictions *J. of Forecasting: Special Issue on VAR Modelling*, **14**, 229–250.

See Also

`estBlackBox1`, `estBlackBox2` `estBlackBox3` `informationTestsCalculations`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
z <-  bft(eg1.DSE.data.diff)
```

`estBlackBox` *Estimate a TSmodel*

Description

Estimate a TSmodel.

Usage

```
estBlackBox(data,...)
```

Arguments

`data`	Data in an object of class TSdata.
`...`	Optional arguments depent on the function which is eventually called.

Details

The function makes a call the most promising procedure currently available. These tend to have names like estBlackBox1, estBlackBox2, estBlackBox4. This is an active area of ongoing research and so the actual routine called will probably change with new versions.

Value

A state space model in an object of class TSestModel.

Examples

```
data("egJofF.1dec93.data", package="dse1")
goodmodel <- estBlackBox(egJofF.1dec93.data)
```

---

estMaxLik *Maximum Likelihood Estimation*

Description

Maximum likelihood estimation.

Usage

```
estMaxLik(obj1, obj2=NULL, ...)
## S3 method for class 'TSmodel':
estMaxLik(obj1, obj2, algorithm="optim",
    algorithm.args=list(method="BFGS", upper=Inf, lower=-Inf, hessian=TRUE),
    ...)
## S3 method for class 'TSestModel':
estMaxLik(obj1, obj2=TSdata(obj1), ...)
## S3 method for class 'TSdata':
estMaxLik(obj1, obj2, ...)
```

Arguments

`obj1`	an object of class TSmodel, TSdata or TSestModel
`obj2`	TSdata or a TSmodel to be fitted with obj1.
`algorithm`	the algorithm ('optim', or 'nlm') to use for maximization.
`algorithm.args`	arguments for the optimization algorithm.
`...`	arguments passed on to other methods.

Details

One of `obj1` or `obj2` should specify a `TSmodel` and the other `TSdata`. If `obj1` is a `TSestModel` and `obj2` is `NULL`, then the data is extracted from `obj1`. The `TSmodel` object is used to specify both the initial parameter values and the model structure (the placement of the parameters in the various arrays of the TSmodel). Estimation attempts to minimimize the negative log likelihood (as returned by `l`) of the given model structure by adjusting the parameter values. A `TSmodel` can also have constant values in some array elements, and these are not changed. (See `SS`, `ARMA` and `fixConstants` regarding setting of constants.)

With the number of parameter typically used in multivariate time series models, the default maximum number of iterations may not be enough. Be sure to check for convergence (a warning is printed at the end, or use `summary` on the result). The maximum iterations is passed to the estimation algorithm with `algorithm.args`, but the elements of that list will depend on the specified optimization `algorithm` (so see the help for the alogrithm). The example below is for the default `optim` algorithm.

Value

The value returned is an object of class TSestModel with additional elements `est$converged`, which is TRUE or FALSE indicating convergence, `est$converceCode`, which is the code returned by the estimation algorithm, and `est$results`, which are detailed results returned by the estimation algorithm. The hessian and gradient in results could potentially be used for restarting in the case of non-convergence, but that has not yet been implemented.

Warning

Maximum likelihood estimation of multivariate time series models tends to be problematic, even when a good structure and good starting parameter values are known. This is especially true for state space models. Also, it seems that in-sample fit is often obtained at the expense of out-of-sample forecasting ability. If a prior model structure is not important then the `bft` estimation method may be preferable.

See Also

`optim`, `nlm`, `estVARXls`, `bft`, `TSmodel`, `l`, `SS`, `ARMA`, `fixConstants`

Examples

```
true.model <- ARMA(A=c(1, 0.5), B=1)
est.model <-  estMaxLik(true.model,  simulate(true.model))
summary(est.model)
est.model
tfplot(est.model)
est.model <-  estMaxLik(true.model,  simulate(true.model),
  algorithm.args=list(method="BFGS", upper=Inf, lower=-Inf, hessian=TRUE,
                      control=list(maxit=10000)))
```

`estSSfromVARX` *Estimate a state space TSmodel using VAR estimation*

Description

Estimate a VAR TSmodel with (optionally) an exogenous input and convert to state space.

Usage

```
estSSfromVARX(data, warn=TRUE, ...)
```

Arguments

`data`	An object with the structure of an object of class TSdata (see TSdata).
`warn`	Logical indicating if warnings should be printed (TRUE) or suppressed (FALSE).
`...`	See arguements to estVARXls

Details

This function uses the functions estVARXls and toSS.

Value

A state space model in an object of class TSestModel.

References

Gilbert, P. D. (1993) State space and ARMA models: An overview of the equivalence. Working paper 93-4, Bank of Canada. Available at www.bank-banque-canada.ca/pgilbert.

Gilbert, P. D. (1995) Combining VAR Estimation and State Space Model Reduction for Simple Good Predictions. *J. of Forecasting: Special Issue on VAR Modelling*. 14:229-250.

See Also

toSS estSSMittnik bft estVARXls estMaxLik

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <-estSSfromVARX(eg1.DSE.data.diff)
```

---

estSSMittnik *Estimate a State Space Model*

Description

Estimate a state space model using Mittnik's markov parameter estimation.

Usage

```
estSSMittnik(data, max.lag=6, n=NULL, subtract.means=FALSE, normalize=FALSE)
```

Arguments

`data`	A TSdata object.
`max.lag`	The number of markov parameters to estimate.
`n`	The state dimension.
`subtract.means`	If TRUE subtract the means from the data before estimation.
`normalize`	If TRUE normalize the data before estimation.

Details

Estimate a nested-balanced state space model by svd from least squares estimate of markov parameters a la *Mittnik (1989, p1195)*. The quality of the estimate seems to be quite sensitive to `max.lag`, and this is not properly resolved yet. If `n` is not supplied the svd criteria will be printed and `n` prompted for. If `subtract.means=T` then the sample mean is subtracted. If `normalize` is `T` the lsfit estimation is done with outputs normalize to cov=I (There still seems to be something wrong here!!). The model is then re-transformed to the original scale.

See `MittnikReduction` and references cited there. If the state dimension is not specified then the singular values of the Hankel matrix are printed and the user is prompted for the state dimension.

Value

A state space model in an object of class `TSestModel`.

References

See references for `MittnikReduction`.

See Also

`MittnikReduction` `estVARXls` `bft`

Examples

```
data("egJofF.1dec93.data", package="dse1")
## Not run: model <- estSSMittnik(egJofF.1dec93.data)
# this prints information about singular values and prompts with
#Enter the number of singular values to use for balanced model:
# the choice is difficult in this example.
model <- estSSMittnik(egJofF.1dec93.data, n=3)
```

`estVARXar` *Estimate a VAR TSmodel*

Description

Estimate a VAR TSmodel with (optionally) an exogenous input.

Usage

```
estVARXar(data, subtract.means=FALSE,  re.add.means=TRUE, standardize=FALSE,
     unstandardize=TRUE, aic=TRUE, max.lag=NULL, method="yule-walker", warn=
```

Arguments

`data` A TSdata object.

`subtract.means` If TRUE subtract the means from the data before estimation.

`re.add.means` If TRUE the model is adjusted for the non-zero mean data when returned. If subtract.means is also TRUE then the mean is added back to the data.

`standardize` Note that the mean is not subtracted unless subtract.means is TRUE. A VAR model in an object of class TSestModel.

`unstandardize` If TRUE and standardize is TRUE then the returned model is adjusted to correspond to the original data.

`aic` Passed to function ar.

`max.lag` The maximum number of lags that should be considered.

`method` Passed to function ar.

`warn` If TRUE certain warning message are suppressed.

Details

This function estimates a VAR model with exogenous variable using ar(). Residuals,etc, are calculated by evaluating the estimated model with ARMA. The procedure ar is used by combine exogeneous variables and endogenous variable and estimating as if all variables were endogenous. The estVARXar method does not support trend estimation (as in estVARXls).

If `aic=TRUE` the number of lags is determined by an AIC statistic (see ar). If an exogenous (input) variable is supplied the input and output are combined (i.e.- both treated as outputs) for estimation, and the resulting model is converted back by transposing the exogenous variable part of the polynomial and discarding inappropriate blocks. Residuals,etc, are calculated by evaluating the estimated model as a TSmodel/ARMA with the data (ie. residuals are not the residuals from the regression).

Note: ar uses a Yule-Walker approach (uses autocorrelations) so effectively the model is for data with means removed. Thus subtract.means does not make much difference and re.add.means must be TRUE to get back to a model for the original data.

The conventon for AR(0) and sign are changed to ARMA format. Data should be of class TSdata. The exog. variable is shifted so contemporaneous effects enter. the model for the exog. variable (as estimated by ar() is discarded.

Value

A TSestModel object containing an ARMA TSmodel object. The model has no MA portion so it is a VAR model.

References

Gilbert, P. D. (1993) State space and ARMA models: An overview of the equivalence. Working paper 93-4, Bank of Canada. Available at `www.bank-banque-canada.ca/pgilbert`

Gilbert, P. D. (1995) Combining VAR Estimation and State Space Model Reduction for Simple Good Predictions. *J. of Forecasting: Special Issue on VAR Modelling.* **14**:229–250.

See Also

`estSSfromVARX` `estSSMittnik` `bft` `estVARXls` `estMaxLik` `ar` `DSE.ar`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- estVARXar(eg1.DSE.data.diff)
```

`estVARXls` *Estimate a VAR TSmodel*

Description

Estimate a VAR TSmodel with (optionally) an exogenous input and (optionally) a trend.

Usage

```
estVARXls(data, subtract.means=FALSE, re.add.means=TRUE, standardize=FALSE,
 unstandardize=TRUE, max.lag=NULL, trend=FALSE, lag.weight=1.0, warn=TRUE)
```

Arguments

- `data` A TSdata object.
- `subtract.means` If TRUE subtract the means from the data before estimation.
- `re.add.means` If TRUE and subtract.means is TRUE then the mean is added back to the data and the model is adjusted for the non-zero mean data when returned.
- `standardize` If TRUE divide each series by its sample standard deviation before estimation. Note that the mean is not subtracted unless subtract.means is TRUE.
- `unstandardize` If TRUE and standardize is TRUE then the returned model is adjusted to correspond to the original data.
- `trend` If TRUE a trend is estimated.
- `max.lag` Number of lags to be used.
- `lag.weight` Weight between 0 and 1 to be applied to lagged data. Lower weights mean lagged data is less important (more noisy).
- `warn` If TRUE a warning message is issued when missing data (NA) is detected and the model predictions are reconstructed from the lsfit residuals.

Details

A VAR model is fitted by least squares regression using lsfit. The argument max.lag determines the number of lags. If it is not specified then six lags are used. This is an exceedingly naive approach, so the max.lag argument really should be specified (or see `bft` for a more complete approach to model selection.) If a trend is not estimated the function estVARXar may be preferred. Missing data is allowed in lsfit, but not (yet) by ARMA which generates the model predictions, etc., based on the estimated model and the data. (This is done to ensure the result is consistent with other estimation techniques.) In the case of missing data ARMA is not used and the model predictions, etc., are generated by adding the data and the lsfit residual. This is slightly different from using ARMA, especially with respect to initial conditions.

Value

A TSestModel object containing a TSmodel object which is a VAR model.

References

Gilbert, P. D. (1993) State space and ARMA models: An overview of the equivalence. Working paper 93-4, Bank of Canada. Available at `www.bank-banque-canada.ca/pgilbert`

Gilbert, P. D. (1995) Combining VAR Estimation and State Space Model Reduction for Simple Good Predictions. *J. of Forecasting: Special Issue on VAR Modelling.* **14**:229–250.

See Also

`estSSfromVARX` `estSSMittnik` `bft` `estVARXar` `estMaxLik`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- estVARXls(eg1.DSE.data.diff)
```

`estWtVariables` *Weighted Estimation*

Description

estWtVariables

Usage

```
estWtVariables(data, variable.weights,
                        estimation="estVARXls", estimation.args=NULL)
```

Arguments

- `data` A TSdata object.
- `variable.weights` weights to use for each output series.
- `estimation` An estimation method.
- `estimation.args` An arguments for the estimation method.

Details

Weight series so that some series residuals are more important than others. Each output variable is scaled according to variable.weights, estimate is done, and then the estimated model unscaled. Estmation is done the method specified by estimate and any arguments specified by estimation.args. estimation.args should be NULL if no args are needed.

Value

A TSestModel.

See Also

`estVARXls` `estBlackBox` `bft` `estMaxLik`

fixConstants — *Fix TSmodel Coefficients (Parameters) to Constants*

Description

Fix specified coefficients to constant values or any coefficients within fuzz of 0.0 or 1.0 to exactly 0.0 or 1.0. This will not change the model much but will affect some estimation techniques and information criteria results, as these are considered to be constants rather than coefficients.

Usage

```
fixConstants(model, fuzz=1e-5, constants=NULL)
```

Arguments

`model`	an object of class TSmodel.
`fuzz`	absolute difference to be considered equivalent.
`constants`	NULL or a list of logical arrays.

Details

If constants is not NULL then parameters within fuzz of 0.0 or 1.0 are set as constants 0.0 or 1.0. If constants is not NULL then it should be a list with logical arrays named F, G ..., with TRUE corresponding to any array elements which are to be treated as constant.

Value

An object of class ’SS’ ’TSmodel’ with some array entries set to constants 0.0 or 1.0.

See Also

`fixF`

Examples

```
f <- array(c(.5,.3,.2,.4),c(2,2))
h <- array(c(1,0,0,1),c(2,2))
k <- array(c(.5,.3,.2,.4),c(2,2))
ss <- SS(F=f,G=NULL,H=h,K=k)
ss
coef(ss)
ss <- fixConstants(ss, constants=list(
          F = matrix(c(TRUE, FALSE, FALSE, FALSE), 2,2)))
ss
coef(ss)
data("eg1.DSE.data.diff", package="dse1")
model <- toARMA(toSS(estVARXls(eg1.DSE.data.diff)))
model <- fixConstants(model)
```

fixF — *Set SS Model F Matrix to Constants*

Description

Set any parameters of the F matrix to constants. The same values are retained but they are considered to be constants rather than parameters. This will not change the model but will affect some estimation techniques and information criteria results.

Usage

```
fixF(model)
```

Arguments

`model`	An object of class TSmodel.

Value

An SS TSmodel object.

See Also

`fixConstants`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- toSS(estVARXls(eg1.DSE.data.diff))
model <- fixF(model)
```

gmap — *Basis Transformation of a Model.*

Description

Transform the basis for the state by a given invertible matrix.

Usage

```
gmap(g, model)
```

Arguments

`g`	An invertible matrix
`model`	An object of class TSmodel.

Details

If the input model is in state space form g is a change of basis for the state. If the input model is in ARMA form then the polynomials are premultiplied by g. If g is a scalar it is treated as a diagonal matrix.

Value

An equivalent model transformed using g.

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- toSS(estVARXls(eg1.DSE.data.diff))
gmap(2, model)
```

---

informationTestsCalculations

Calculate selection criteria

Description

Calculates several model selection criteria.

Usage

```
informationTestsCalculations(lst, sample.start=1,sample.end=NULL, warn=TRUE)
```

Arguments

`lst`	One or more objects of class TSestModel.
`sample.start`	The start of the period to use for criteria calculations.
`sample.end`	The end of the period to use for criteria calculations. If omitted the end of the sample is used.
`warn`	If FALSE then some warning messages are suppressed.

Value

The calculated values are returned in a vector with names: port, like, aic, bic, gvc, rice, fpe, taic, tbic, tgvc, trice and tfpe. These correspond to values for the Portmanteau test, likelihood, Akaike Information Criterion, Bayes Information Criterion, Generalized Cross Validation, Rice Criterion, and Final Prediction Error. The preceeding 't' indicates that the theoretical parameter space dimension has been used, rather than the number of coefficient (parameter) values. Methods which select a model based on some information criterion calculated by `informationTestsCalculations` should use the name of the vector element to specify the test value which is to be used.

See Also

`informationTests`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- estVARXls(eg1.DSE.data.diff)
informationTestsCalculations(model)
```

informationTests *Tabulates selection criteria*

Description

Tabulates several model selection criteria.

Usage

```
informationTests(..., sample.start=1,sample.end=NULL, Print=TRUE, warn=TRUE)
```

Arguments

`...`	At least one object of class TSestModel.
`sample.start`	The start of the period to use for criteria calculations.
`sample.end`	The end of the period to use for criteria calculations. If omitted the end of the sample is used.
`Print`	If FALSE then printing suppressed.
`warn`	If FALSE then some warning messages are suppressed.

Value

A matrix of the value for each model on each test returned invisibly.

Side Effects

Criteria are tabulated for all models in the list.

See Also

`informationTestsCalculations`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model1 <- estVARXls(eg1.DSE.data.diff)
model2 <- estVARXar(eg1.DSE.data.diff)
informationTests(model1, model2)
```

`inputData` *TSdata Series*

Description

Extract or set input or output series in a TSdata object.

Usage

```
inputData(x, series=seqN(nseriesInput(x)))
## Default S3 method:
inputData(x, series=seqN(nseriesInput(x)))
## S3 method for class 'TSdata':
inputData(x,  series=seqN(nseriesInput(x)))
## S3 method for class 'TSestModel':
inputData(x, series=seqN(nseriesInput(x)))

outputData(x, series=seqN(nseriesOutput(x)))
## Default S3 method:
outputData(x, series=seqN(nseriesOutput(x)))
## S3 method for class 'TSdata':
outputData(x,  series=seqN(nseriesOutput(x)))
## S3 method for class 'TSestModel':
outputData(x, series=seqN(nseriesOutput(x)))

inputData(x)  <- value
outputData(x) <- value
```

Arguments

`x`	object of class TSdata.
`value`	a time series matrix.
`series`	vector of strings or integers indicating the series to select.

Value

The first usages returns the input or output series. The second usages assigns the input or output series.

See Also

`TSdata` `selectSeries`

Examples

```
data("eg1.DSE.data", package="dse1")
outputData(eg1.DSE.data)
```

l.ARMA *Evaluate an ARMA TSmodel*

Description

Evaluate an ARMA TSmodel.

Usage

```
## S3 method for class 'ARMA':
l(obj1, obj2, sampleT=NULL, predictT=NULL,result=NULL,
   error.weights=0,  compiled=.DSEflags()$COMPILED, warn=TRUE,
   return.debug.info=FALSE, ...)
```

Arguments

- `obj1` an 'ARMA' 'TSmodel' object.
- `obj2` a TSdata object.
- `sampleT` an integer indicating the number of periods of data to use.
- `predictT` an integer to what period forecasts should be extrapolated.
- `result` if non-NULL then the returned value is only the sub-element indicated by result. result can be a character string or integer.
- `error.weights` a vector of weights to be applied to the squared prediction errors.
- `compiled` indicates if a call should be made to the compiled code for computation. A FALSE value is mainly for testing purposes.
- `warn` if FALSE then certain warning messages are turned off.
- `return.debug.info` logical indicating if additional debugging information should be returned.
- `...` (further arguments, currently disregarded).

Details

This function is called by the function l() when the argument to l is an ARMA model (see ARMA). Using l() is usually preferable to calling l.ARMA directly. l.ARMA calls a compiled program unless compiled=FALSE. The compiled version is much faster.

sampleT is the length of data which should be used to calculate the one-step ahead predictions, and likelihood value for the model: Output data must be at least as long as sampleT. If sampleT is not supplied it is taken to be periods(data).

Input data must be at least as long as predictT. predictT must be at least as large as sampleT. If predictT is not supplied it is taken to be sampleT.

If `error.weights` is greater than zero then weighted prediction errors are calculated up to the horizon indicated by the length of error.weights. The weights are applied to the squared error at each period ahead.

Value

An object of class TSestModel (see TSestModel) containing the calculated likelihood, prediction, etc. for ARMA model.

See Also

ARMA l, l.SS TSmodel TSestModel.object

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- TSmodel(estVARXls(eg1.DSE.data.diff))
evaluated.model <- l(model,eg1.DSE.data.diff)
```

---

l — *Evaluate a TSmodel*

Description

Evaluate a model with data.

Usage

```
l(obj1, obj2, ...)
## S3 method for class 'TSdata':
l(obj1, obj2, ...)
## S3 method for class 'TSestModel':
l(obj1, obj2, ...)
```

Arguments

`obj1`	a TSmodel, TSdata, or TSestModel object.
`obj2`	a TSmodel or TSdata object.
`...`	arguments to be passed to other methods.

Details

For state space models `l.SS` is called and for ARMA models `l.ARMA` is called.

Value

Usually a TSestModel object is returned. Most methods allow an argument `result` which specifies that a certain part of the object is returned. (This is passed in ... to another method in most cases.) The likelihood can be returned by specifying result="like", which is useful for optimization routines.

See Also

l.SS, l.ARMA

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- toSS(TSmodel(estVARXls(eg1.DSE.data.diff)))
evaluated.model <- l(model, eg1.DSE.data.diff)
```

`l.SS` *Evaluate a state space TSmodel*

Description

Evaluate a state space TSmodel.

Usage

```
## S3 method for class 'SS':
l(obj1, obj2, sampleT=NULL, predictT=NULL, error.weights=0,
     return.state=FALSE, return.track=FALSE, result=NULL,
     compiled=.DSEflags()$COMPILED,
     warn=TRUE, return.debug.info=FALSE, ...)
```

Arguments

`obj1`	An 'SS' 'TSmodel' object.
`obj2`	A TSdata object.
`sampleT`	an integer indicating the last data point to use for one step ahead filter estimation. If NULL all available data is used.
`predictT`	an integer indicating how far past the end of the sample predictions should be made. For models with an input, input data must be provided up to predictT. Output data is necessary only to sampleT. If NULL predictT is set to sampleT.
`error.weights`	a vector of weights to be applied to the squared prediction errors.
`return.state`	if TRUE the element `filter$state` containing E[z(t)|y(t-1), u(t)] is returned as part of the result. This can be a fairly large matrix.
`return.track`	if TRUE the element `filter$track` containing the expectation of the tracking error given y(t-1) and u(t) is returned as part of the result. This can be an very large array.
`result`	if result is not specified an object of class TSestModel is returned. Otherwise, the specified element of `TSestModel$estimates` is returned.
`compiled`	if TRUE the compiled version of the code is used. Otherwise the S/R version is used.
`warn`	if FALSE then certain warning messages are turned off.
`return.debug.info`	logical indicating if additional debugging information should be returned.
`...`	(further arguments, currently disregarded).

Details

This function is called by the function l() when the argument to l is a state space model. Using l() is usually preferable to calling l.SS directly. l.SS calls a compiled program unless compiled=FALSE. The compiled version is much faster than the S version.

Output data must be at least as long as sampleT. If sampleT is not supplied it is taken to be periods(data).

Input data must be at least as long as predictT. predictT must be at least as large as sampleT. If predictT is not supplied it is taken to be sampleT.

If `error.weights` is greater than zero then weighted prediction errors are calculated up to the horizon indicated by the length of error.weights. The weights are applied to the squared error at each period ahead.

`sampleT` is the length of data which should be used for calculating one step ahead predictions. `y` must be at least as long as `sampleT`. If `predictT` is large than `sampleT` then the model is simulated to `predictT`. `y` is used if it is long enough. `u` must be at least as long as `predictT`. The default `result=0` returns a list of all the results. Otherwise only the indicated list element is return (eg. `result=1` return the likelihood and `result=3` returns the one step ahead predictions.

If `z0` is supplied in the model object it is used as the estimate of the state at time 0. If not supplied it is set to zero.

If `rootP0` is supplied in the model object then t(rootP0) %*% rootP0 is used as P0. If `P0` is supplied or calculated from rootP0 in the model object, it is used as the initial tracking error P(t=1|t=0). If not supplied it is set to the identity matrix.

Additional objects in the result are `Om` is the estimated output cov matrix. `pred` is the time series of the one-step ahead predictions, E[y(t)|y(t-1),u(t)]. The series of prediction error is given by `y - pred` If error.weights is greater than zero then weighted prediction errors are calculated up to the horizon indicated by the length of error.weights. The weights are applied to the squared error at each period ahead. `trackError` is the time series of P, the one step ahead estimate of the state tracking error matrix at each period, Covz(t)-E[z(t)|t-1] The tracking error can only be calculated if Q and R are provided (i.e. non innovations form models). Using the Kalman Innov K directly these are not necessary for the likelihood calculation, but the tracking error cannot be calculated.

Value

Usually an object of class TSestModel (see TSestModel), but see result above.

References

Anderson, B. D. O. and Moore, J. B. (1979) *Optimal Filtering*. Prentice-Hall. (note p.39,44.)

See Also

`SS` `l` `l.ARMA` `TSmodel` `TSestModel` `TSestModel.object` `state` `smoother`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- toSS(TSmodel(estVARXls(eg1.DSE.data.diff)))
lmodel <- l(model,eg1.DSE.data.diff)
summary(lmodel)
tfplot(lmodel)
lmodel <- l(model,eg1.DSE.data.diff, return.state=TRUE)
tfplot(state(lmodel, filter=TRUE))
```

`makeTSnoise` *Generate a random time series*

Description

Generate a random time series (matrix). This is a utility typically used in a time series model simulate method and not called directly by the user.

Usage

```
makeTSnoise(sampleT,p,lags,noise=NULL, rng=NULL, Cov=NULL, sd=1,
          noise.model=NULL, noise.baseline=0,
          tf=NULL, start=NULL,frequency=NULL)
```

Arguments

- `sampleT` an integer indicating the number of periods.
- `p` an integer indicating the number of series.
- `lags` an integer indicating the number of periods prior to the sample (initial data w0) for which random numbers should be generated. This is useful in ARMA models.
- `noise` Noise can be supplied. Otherwise it will be generated. If supplied it should be a list as described below under returned value.
- `Cov` The covariance of the noise process. If this is specified then sd is ignored. A vector or scalar is treated as a diagonal matrix. For an object of class TSestModel, if neither Cov nor sd are specified, then Cov is set to the estimated covariance (`model$estimates$cov`).
- `sd` The standard deviation of the noise. This can be a vector.
- `noise.model` A TSmodel to be used for generating noise (not yet supported by SS methods).
- `noise.baseline` a constant or matrix to be added to noise. Alternately this can be a vector of length p, each value of which is treated as a constant to add to the coresponding noise series.
- `rng` The random number generator information needed to regenerate a simulation.
- `tf` a time frame to use for the generated matrix. (alternately use start and frequency)
- `start` a time start date to use for the generated matrix.
- `frequency` a time frequency to use for the generated matrix.

Value

A time series matrix.

markovParms — *Markov Parameters*

Description

Construct a Matrix of the Markov Parameters

Usage

```
markovParms(model, blocks=NULL)
```

Arguments

`model`	An ARMA or SS TSmodel.
`blocks`	Number of blocks to calculate.

Details

Construct a matrix with partitions [M0|...|Mi] giving the Markov parameters Mi, i+1 = blocks where each Mi is a p by (m+p) matrix, (m is the dimension of the exogeneous series and p is the dimension of endogeneous series) ie. y(t) = e(t) + M [u'(t)|y'(t-1) | u'(t-1)|y'(t-2)]' This requires that models be transformed so that lagged endogeneous variables are inputs. See Mittnik p1190. If blocks=NULL (the default) then at least 3 blocks are generated, and up to n+1, but the series is truncated if the blocks are effectively zero. This will affect the size of the Hankel matrix.

Value

A matrix

References

See references for `MittnikReduction`.

See Also

`SVDbalanceMittnik`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- estVARXls(eg1.DSE.data.diff)
markovParms(model)
```

McMillanDegree *Calculate McMillan Degree*

Description

Calculate the McMillan degree of an ARMA TSmodel.

Usage

```
McMillanDegree(model,  ...)
## S3 method for class 'ARMA':
McMillanDegree(model, fuzz=1e-4, verbose=TRUE, warn=TRUE, ...)
## S3 method for class 'SS':
McMillanDegree(model, fuzz=1e-4, ...)
## S3 method for class 'TSestModel':
McMillanDegree(model,  ...)
```

Arguments

`model`	An object of class TSmodel.
`fuzz`	Roots within fuzz distance are counted as equivalent.
`verbose`	If TRUE roots are printed.
`warn`	If FALSE then warnings about unit roots added for TREND are not printed.
`...`	arguments to be passed to other methods.

Value

A list with elements gross and distinct containing all roots and distinct roots.

Side Effects

The number of roots and distinct roots is printed if verbose is TRUE.

See Also

`stability`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- estVARXls(eg1.DSE.data.diff)
McMillanDegree(model)
```

MittnikReducedModels *Reduced Models via Mittnik SVD balancing*

Description

Reduced Models via Mittnik SVD balancing.

Usage

```
MittnikReducedModels(largeModel)
```

Arguments

`largeModel`	An SS TSmodel.

Details

The largeModel is balanced by the SVD method promoted by Mittnik (see MittnikReduction) and then models for every state dimension up to the state dimension of the largeModel are return. Note that this procedure does not result in smaller models which are balanced.

Value

A list of state space TSmodels with smaller state dimensions.

See Also

`MittnikReduction`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
    z <-  MittnikReducedModels(toSS(estVARXls(eg1.DSE.data.diff)))
```

MittnikReduction *Balance and Reduce a Model*

Description

Balance and reduce the state dimension of a state space model a la Mittnik.

Usage

```
MittnikReduction(model, data=NULL, criterion=NULL, verbose=TRUE,warn=TRUE)
MittnikReduction.from.Hankel(M, data=NULL, nMax=NULL,
   criterion=NULL, verbose=TRUE, warn=TRUE)
```

Arguments

`model`	An object of class TSmodel or TSestModel.
`data`	If the supplied model is of class TSestModel and data is not supplied then it is taken from the model. If the model is of class TSmodel then data must be supplied.
`criterion`	Criterion to be used for model selection. see `informationTestsCalculations`.
`verbose`	logical indicating if information should be printed during estimation.
`warn`	logical indicating if some warning messages should be suppressed.
`M`	a matrix. See details.
`nMax`	integer indicating the state dimension of the largest model considered.

Details

`MittnikReduction` gives nested-balanced state space model using reduction by svd of the Hankel matrix generated from a model. If a state space model is supplied the max. state dimension for the result is taken from the model. If an ARMA model is supplied then singular values will be printed and the program prompts for the max. state dimension. criterion should be the name of one of the values returned by informationTests, that is, one of ("port","like","aic","bic","gvc","rice","fpe","taic", "tbic","tgvc","trice","tfpe"). If criteria is not specified then the program prompts for the state dimension (n) to use for the returned model. The program requires data to calculate selection criteria. (The program balanceMittnik calculates svd criteria only and can be used for reduction without data.)

The function `MittnikReduction.from.Hankel` is called by `MittnikReduction` and typically not by the user, but there are situations when the former might be called directly. It selects a reduced state space model by svd a la Mittnik. Models and several criteria for all state dimensions up to the max. state dim. specified are calculated. (If nMax is not supplied then svd criteria are printed and the program prompts for nMax). The output dimension p is taken from nrow(M). M is a matrix with p x (m+p) blocks giving the markov parameters, that is, the first row of the Hankel matrix. It can be generated from the model as in the function markovParms, or from the data, as in the function estSSMittnik.

`data` is necessary only if criteria (AIC,etc) are to be calculated.

Value

A state space model balance a la Mittnik in an object of class TSestModel.

References

Gilbert, P. D. (1993) State space and ARMA models: An overview of the equivalence. Working paper 93-4, Bank of Canada. Available at `www.bank-banque-canada.ca/pgilbert`.

Gilbert, P. D. (1995) Combining VAR Estimation and State Space Model Reduction for Simple Good Predictions. *J. of Forecasting: Special Issue on VAR Modelling*, **14**, 229-250.

Mittnik, S. (1989), Multivariate Time Series Analysis With State Space Models. *Computers Math Appl.* **17**, 1189–1201.

Mittnik, S. (1990), Macroeconomic Forecasting Experience With Balance State Space Models. *International Journal Of Forecasting*, **6**, 337–348.

Mittnik, S. (1990), Forecasting With Balanced State Space Representations of Multivariate Distributed Lag Models. *J. of Forecasting*, **9**, 207–218.

See Also

estVARXls bft balanceMittnik informationTests informationTestsCalculations

Examples

```
data("egJofF.1dec93.data", package="dse1")
model <- toSS(estVARXls(egJofF.1dec93.data))
newmodel <-MittnikReduction(model, criterion="taic")
```

---

nseriesInput — *Number of Series in in Input or Output*

Description

Number of input or output series in a TSdata object.

Usage

```
nseriesInput(x)
## Default S3 method:
nseriesInput(x)
## S3 method for class 'TSdata':
nseriesInput(x)
## S3 method for class 'SS':
nseriesInput(x)
## S3 method for class 'ARMA':
nseriesInput(x)
## S3 method for class 'TSestModel':
nseriesInput(x)

nseriesOutput(x)
## Default S3 method:
nseriesOutput(x)
## S3 method for class 'TSdata':
nseriesOutput(x)
## S3 method for class 'SS':
nseriesOutput(x)
## S3 method for class 'ARMA':
nseriesOutput(x)
## S3 method for class 'TSestModel':
nseriesOutput(x)
```

Arguments

x	Object of class TSdata, TSmodel or TSestModel.

Value

An integer indicating the number of series.

See Also

seriesNamesInput seriesNamesOutput

Examples

```
data("eg1.DSE.data", package="dse1")
nseriesOutput(eg1.DSE.data)
```

nstates *State Dimension of a State Space Model*

Description

Extract the state dimension of a state space model object.

Usage

```
nstates(x)
## S3 method for class 'SS':
nstates(x)
## S3 method for class 'ARMA':
nstates(x)
## S3 method for class 'TSestModel':
nstates(x)
```

Arguments

x	Object of class TSmodel or TSestModel.

Value

An integer indicating the state dimension.

See Also

nseriesInput

observability *Calculate Model Observability Matrix*

Description

Calculate the singular values of the observability matrix of a model.

Usage

```
observability(model)
## S3 method for class 'ARMA':
observability(model)
## S3 method for class 'SS':
observability(model)
## S3 method for class 'TSestModel':
observability(model)
```

Arguments

`model`	An object containing a TSmodel.

Details

If all singular values are significantly different from zero the model is observable.

Value

The singular values of the observability matrix.

See Also

`reachability`, `stability` `McMillanDegree`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- toSS(estVARXls(eg1.DSE.data.diff))
observability(model)
```

`percentChange.TSdata` *Calculate percent change*

Description

Calculate the percent change relative to the data lag periods prior.

Usage

```
## S3 method for class 'TSdata':
percentChange(obj, base=NULL, lag=1, cumulate=FALSE, e=FALSE, ...)
## S3 method for class 'TSestModel':
percentChange(obj, base=NULL, lag=1, cumulate=FALSE, e=FALSE, ...)
```

Arguments

`obj`	An object of class TSdata or TSestModel
`e`	see the default method.
`base`	see the default method.
`lag`	see the default method.
`cumulate`	see the default method.
`...`	arguments passed to other methods.

Details

See `percentChange`.

Value

For an object of class TSdata the percent change calculation is done with the output data and the result is an object of class TSdata (or a list of objects of class TSdata). For an object of class TSestModel the percent change calculation is done with `estimates$pred` and the result is an object of class TSdata (or a list of objects of class TSdata).

See Also

`percentChange` `ytoypc`

Examples

```
data("eg1.DSE.data", package="dse1")
z <- percentChange(outputData(eg1.DSE.data))
```

`periodsInput` *TSdata Periods*

Description

Apply a method to the input or output data.

Usage

```
periodsInput(x)
## S3 method for class 'TSdata':
periodsInput(x)
## S3 method for class 'TSestModel':
periodsInput(x)

periodsOutput(x)
## S3 method for class 'TSdata':
periodsOutput(x)
## S3 method for class 'TSestModel':
periodsOutput(x)

startInput(x)
```

```
## S3 method for class 'TSdata':
startInput(x)
## S3 method for class 'TSestModel':
startInput(x)

startOutput(x)
## S3 method for class 'TSdata':
startOutput(x)
## S3 method for class 'TSestModel':
startOutput(x)

endInput(x)
## S3 method for class 'TSdata':
endInput(x)
## S3 method for class 'TSestModel':
endInput(x)

endOutput(x)
## S3 method for class 'TSdata':
endOutput(x)
## S3 method for class 'TSestModel':
endOutput(x)

frequencyInput(x)
## S3 method for class 'TSdata':
frequencyInput(x)
## S3 method for class 'TSestModel':
frequencyInput(x)

frequencyOutput(x)
## S3 method for class 'TSdata':
frequencyOutput(x)
## S3 method for class 'TSestModel':
frequencyOutput(x)
```

Arguments

`x`	An object containing TSdata.

Details

Apply a method to the input or output data so, for example, periodsInput(x) in theory does periods(inputData(x)), which returns the number of periods in input data. The actual implementation may not do periods(inputData(x)). For example, with TSPADIdata inputData(x) requires a database retrieval which may be fairly slow, while the number of periods may be available much more quickly.

Value

Depends.

Examples

```
data("eg1.DSE.data.diff", package="dse1")
periodsOutput(eg1.DSE.data.diff)
```

`periods.TSdata` *Specific Methods for tframed Data*

Description

See the generic function description.

Usage

```
## S3 method for class 'TSdata':
periods(x, ...)
## S3 method for class 'TSestModel':
periods(x)
## S3 method for class 'TSdata':
start(x, ...)
## S3 method for class 'TSestModel':
start(x, ...)
## S3 method for class 'TSdata':
end(x, ...)
## S3 method for class 'TSestModel':
end(x, ...)
## S3 method for class 'TSdata':
frequency(x, ...)
## S3 method for class 'TSestModel':
frequency(x, ...)
```

Arguments

`x`	a time series object.
`...`	(further arguments, currently disregarded).

See Also

`periods`, `start`, `end`, `frequency`

`plot.roots` *Plot Model Roots*

Description

Calculate and plot roots of a model.

Usage

```
## S3 method for class 'roots':
plot(x, pch='*', fuzz=0, ...)
```

Arguments

`x`	An object of class roots (a vector of complex (or real) values as returned by the function roots).
`pch`	character to be used for the plot (passed to plot.default).
`fuzz`	If non-zero then roots within fuzz distance are considered equal.
`...`	(further arguments, currently disregarded).

Value

The eigenvalues of the state transition matrix or the inverse of the roots of the determinant of the AR polynomial are returned invisibly.

Side Effects

The roots and a unit circle are plotted on the complex plane.

See Also

`addPlotRoots` `roots` `stability` `McMillanDegree`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- estVARXls(eg1.DSE.data.diff)
plot(roots(model))
```

`Polynomials` *Polynomial Utilities*

Description

Polynomial utility functions used by DSE.

Usage

```
characteristicPoly(a)
companionMatrix(a)
polyvalue(coef, z)
polydet(a)
polyprod(a, b)
polysum(a, b)
polyrootDet(a)
```

Arguments

`a`	An array representing a matrix polynomial.
`b`	An array representing a matrix polynomial.
`coef`	Coefficients of a polynomial.
`z`	Value at which the polynomial is to be evaluated.

Details

These are utility functions used in some ARMA model calculations such as root and stability calculations.

Value

depends

See Also

polyroot roots stability

Portmanteau *Calculate Portmanteau statistic*

Description

Calculate Portmanteau statistic.

Usage

```
Portmanteau(res)
```

Arguments

res	A matrix with time-series residuals in columns.

See Also

informationTests

Examples

```
require("stats")
Portmanteau(matrix(rnorm(200), 100,2)) # but typically with a residual
```

---

print.TSdata *Print Specific Methods*

Description

See the generic function description.

Usage

```
## S3 method for class 'TSdata':
print(x, ...)
```

Arguments

`x`	An object of class TSdata.
`...`	arguments to be passed to other methods.

See Also

`print` `summary`

`print.TSestModel` *Display TSmodel Arrays*

Description

Display TSmodel arrays.

Usage

```
## S3 method for class 'SS':
print(x, digits=options()$digits, latex=FALSE, ...)
## S3 method for class 'ARMA':
print(x, digits=options()$digits, latex=FALSE, L=TRUE, fuzz=1e-10, ...)
## S3 method for class 'TSestModel':
print(x, ...)
```

Arguments

`x`	An object of class TSmodel or TSestModel.
`digits`	the number of significant digits
`L`	logical if TRUE then ARMA model arrays are displayed as a polynomial matrix with L indicating lags. Otherwise, each lag in the array is displayed as a matrix.
`latex`	logical. If TRUE additional context is added to make the output suitable for inclusion in a latex document.
`fuzz`	ARMA model polynomial elements with absolute value less than fuzz are not displayed (i.e.-as if they are zero)
`...`	arguments passed to other methods.

Value

The object is returned invisibly.

Side Effects

The model arrays are displayed.

Note

BUG: digits cannot be controlled for some numbers (e.g.- 1.0 is printed as 0.9999999999)

See Also

print, summary

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- estVARXls(eg1.DSE.data.diff)
print(model)
print(model, digits=3)
print(model, digits=3, fuzz=0.001)
print(model, digits=3, fuzz=0.001, latex=TRUE)
```

---

reachability — *Calculate Model Reachability Matrix*

Description

Calculate the singular values of the reachability matrix of a model.

Usage

```
reachability(model)
## S3 method for class 'ARMA':
reachability(model)
## S3 method for class 'SS':
reachability(model)
## S3 method for class 'TSestModel':
reachability(model)
```

Arguments

`model`	An object containing TSmodel.

Details

If all singular values are significantly different from zero the model is controllable.

Value

The singular values of the reachability matrix.

See Also

observability, stability roots McMillanDegree

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- toSS(estVARXls(eg1.DSE.data.diff))
reachability(model)
```

residualStats *Calculate Residuals Statistics and Likelihood*

Description

Calculate the residuals statistics and likelihood of a residual.

Usage

```
residualStats(pred, data, sampleT=nrow(pred), warn=TRUE)
```

Arguments

pred	A matrix with columns representing time series.
data	A matrix with columns representing time series.
sampleT	An integer indicating the sample to use.
warn	If FALSE certain warnings are suppressed.

Details

Residuals are calculated as pred[1:sampleT,,drop=FALSE] - data[1:sampleT,,drop=FALSE] and then statistics are calculated based on these residuals. If pred or data are NULL they are treated as zero.

Value

A list with elements like, cov, pred, and sampleT. pred and sampleT are as supplied (and are returned as this is a utility function called by other functions and it is convenient to pass them along). cov is the covariance of the residual and like is a vector of four elements representing the total, constant, determinant and covariance terms of the negative log likelihood function.

See Also

l

Examples

```
residualStats(matrix(rnorm(200), 100,2), NULL) # but typically used for a residual
```

residuals.TSestModel

Calculate the residuals for an object

Description

Calculate the residuals for an object.

Usage

```
## S3 method for class 'TSestModel':
residuals(object, ...)
```

Arguments

`object`	an object.
`...`	additional arguments passed to stats::residuals.

Details

Calculates the residuals (prediction minus data).

Value

A time series matrix.

Author(s)

Paul Gilbert

Riccati

Riccati Equation

Description

Solve a Matrix Riccati Equation

Usage

```
Riccati(A, B, fuzz=1e-10, iterative=FALSE)
```

Arguments

`A`	A matrix.
`B`	A matrix.
`fuzz`	The tolerance used for testing convergence.
`iterative`	If TRUE an iterative solution technique is used.

Details

Solve Riccati equation P = APA' + B by eigenvalue decompostion of a symplectic matrix or by iteration.

Value

xxx

Note

This procedure has not been tested.

References

Vaccaro, R. J. and Vukina, T. (1993), A Solution to the Positivity Problem in the State-Space Approach to Modeling Vector-Valued Time Series. *Journal of Economic Dynamics and Control*, **17**, 401–421.

Anderson, B. D. O. and Moore, J. B. (1979) *Optimal Filtering*. Prentice-Hall. (note sec 6.7.)

Vaughan, D. (1970) A Nonrecursive Algebraic Solution for the Discrete Riccati Equation. *IEEE Tr AC*, 597–599.

Laub, A. J. (1983) Numerical Aspects of Solving Algebraic Riccati Equations *Proc IEEE conf Decision and Control*, 183–186.

Gudmundsson T., Kenney, C., and Laub, A. J. (1992) Scaling of the Discrete-Time Algebraic Riccati Equation to Enhance Stability of the Schur Solution Method *IEEE Tr AC*, **37**, 513–518.

See Also

eigen

---

roots *Calculate Model Roots*

Description

Calculate roots of a TSmodel.

Usage

```
roots(obj, ...)
## S3 method for class 'SS':
roots(obj, fuzz=0, randomize=FALSE, ...)
## S3 method for class 'ARMA':
roots(obj, fuzz=0, randomize=FALSE, warn=TRUE, by.poly=FALSE, ...)
## S3 method for class 'TSestModel':
roots(obj, ...)
```

Arguments

`obj`	An object of class TSmodel.
`fuzz`	If non-zero then roots within fuzz distance are considered equal.
`randomize`	Randomly arrange complex pairs of roots so the one with the positive imaginary part is not always first (so random experiments are not biased).
`warn`	If FALSE then warnings about unit roots added for TREND are not printed.
`by.poly`	If TRUE then roots are calculated by expanding the determinant of the A polynomial. Otherwise, they are calculated by converting to a state space representation and calculating the eigenvalues of F. This second method is preferable for speed, accuracy, and because of a limitation in the degree of a polynomial which can be handled by polyroot.
`...`	arguments passed to other methods.

Details

The equality of roots for equivalent state space and ARMA models is illustrated in *Gilbert (1993)*. The calculation of ARMA model roots is more stable if the model is converted to state space and the roots calculated from the state transition matrix (see *Gilbert,2000*). The calculation is done this way by default. If `by.poly=TRUE` then the determinant of the AR polynomial is expanded to get the roots.

Value

The eigenvalues of the state transition matrix or the inverse of the roots of the determinant of the AR polynomial are returned.

References

Gilbert, P. D. (1993) State space and ARMA models: An overview of the equivalence. Working paper 93-4, Bank of Canada. Available at `www.bank-banque-canada.ca/pgilbert`

Gilbert, P.D. (2000) A note on the computation of time series model roots. *Applied Economics Letters*, **7**, 423–424

See Also

`stability`, `McMillanDegree`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- estVARXls(eg1.DSE.data.diff)
roots(model)
```

scale.TSdata — *Scale Methods for TS objects*

Description

Scale data or a model by a given factor.

Usage

```
## S3 method for class 'TSdata':
scale(x, center=FALSE, scale=NULL)
## S3 method for class 'TSestModel':
scale(x, center=FALSE, scale=NULL)
## S3 method for class 'ARMA':
scale(x, center=FALSE, scale=NULL)
## S3 method for class 'innov':
scale(x, center=FALSE, scale=NULL)
## S3 method for class 'nonInnov':
scale(x, center=FALSE, scale=NULL)

checkScale(x, scale)
## S3 method for class 'TSestModel':
checkScale(x, scale)
## S3 method for class 'TSmodel':
checkScale(x, scale)
```

Arguments

`x`	TSdata, TSmodel or an object containing these.
`center`	to match generic arguments, not currently used.
`scale`	A list with two matrices or vectors, named input and output, giving the multiplication factor for inputs and outputs. Vectors are treated as diagonal matrices. `scale$input` can be NULL if no transformation is to be applied (or the data or model has no input.)

Value

The resulting data or model is different from the original in proportion to scale. ie. if S and T are output and input scaling matrices then y'(t) = S y(t) where y' is the new output u'(t) = S u(t) where u' is the new input

For models the result has inputs and outputs (and innovations) which are scaled as if data scaling had been applied to them as above. Thus if the input and output scales are diagonal matrices or scalars the plot of the predictions and residuals for l(scale(model,scale=somescale), scale(data, scale=somescale)) while have the same appearance as l(model, data) but will be scaled differently.

See Also

`scale`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
# This is a simple example. Usually scale would have something
# to do with the magnitude of the data.
z <- scale(eg1.DSE.data.diff,
    scale=list(input=rep(2, nseriesInput(eg1.DSE.data.diff)),
               output=rep(2,nseriesOutput(eg1.DSE.data.diff))))
model <- estVARXls(eg1.DSE.data.diff)
model <- scale(model,
    scale=list(input=rep(2, nseriesInput(eg1.DSE.data.diff)),
               output=rep(2,nseriesOutput(eg1.DSE.data.diff))))
```

---

seriesNamesInput *TSdata Series Names*

Description

Extract or set names of input or output series in a TSdata object.

Usage

```
seriesNamesInput(x)
## S3 method for class 'TSdata':
seriesNamesInput(x)
## S3 method for class 'TSmodel':
seriesNamesInput(x)
## S3 method for class 'TSestModel':
seriesNamesInput(x)

seriesNamesOutput(x)
## S3 method for class 'TSdata':
seriesNamesOutput(x)
## S3 method for class 'TSmodel':
seriesNamesOutput(x)
## S3 method for class 'TSestModel':
seriesNamesOutput(x)

seriesNamesInput(x)  <- value
seriesNamesOutput(x) <- value
```

Arguments

`x`	Object of class TSdata, TSmodel or TSestModel.
`value`	value to be assigned to object.

Value

The first usages gives a vector of strings with the series names. The second usages assigns a vector of strings to be the series names of data.

See Also

seriesNames

Examples

```
data("eg1.DSE.data", package="dse1")
seriesNamesOutput(eg1.DSE.data)
```

---

seriesNames.TSdata *Series Names Specific Methods*

Description

See the generic function description.

Usage

```
## S3 method for class 'TSdata':
seriesNames(x)
## S3 method for class 'TSmodel':
seriesNames(x)
## S3 method for class 'TSestModel':
seriesNames(x)

## S3 replacement method for class 'TSdata':
seriesNames(x) <- value
## S3 replacement method for class 'TSmodel':
seriesNames(x) <- value
## S3 replacement method for class 'TSestModel':
seriesNames(x) <- value
```

Arguments

x	an object from which series names can be extracted or to which series names are to be assigned.
value	series names to be assigned to data.

See Also

seriesNames

setArrays — *Set TSmodel Array Information*

Description

Complete parameter array information based on parameter vector settings. This function is used internally and is not normally called by a user.

Usage

```
setArrays(model, coefficients=NULL, constants=NULL)
## S3 method for class 'ARMA':
setArrays(model, coefficients=NULL, constants=NULL)
## S3 method for class 'SS':
setArrays(model, coefficients=NULL, constants=NULL)
## S3 method for class 'TSestModel':
setArrays(model, coefficients=NULL, constants=NULL)
```

Arguments

`model`	An object of class TSmodel.
`coefficients`	A vector of new values for the model coefficients (parameters).

Value

A TSmodel object.

setTSmodelParameters — *Set TSmodel Parameter Information*

Description

Complete parameter vector information based on parameter array settings. This function is used internally and is not normally called by a user.

Usage

```
setTSmodelParameters(model, constants=model$constants)
## S3 method for class 'TSestModel':
setTSmodelParameters(model, constants=TSmodel(model)$constants)
## S3 method for class 'ARMA':
setTSmodelParameters(model, constants=model$constants)
## S3 method for class 'SS':
setTSmodelParameters(model, constants=model$constants)
```

Arguments

`model`	An object of class TSmodel.
`constants`	A list of logical arrays indicating TRUE for any model array entries that should be treated as constants.

Value

An object of class 'TSmodel'.

See Also

`setArrays`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- estVARXls(eg1.DSE.data.diff)
model <- setTSmodelParameters(model)
```

`simulate` *Simulate a TSmodel*

Description

Simulate a model to produce artificial data.

Usage

```
simulate(model, ...)
## S3 method for class 'ARMA':
simulate(model, y0=NULL, input=NULL, input0=NULL,
    start=NULL, freq=NULL, sampleT=100, noise=NULL, sd=1, Cov=NULL,
    rng=NULL, noise.model=NULL, compiled=.DSEflags()$COMPILED, ...)
## S3 method for class 'SS':
simulate(model, input=NULL,
    start=NULL, freq=NULL,sampleT=100, noise=NULL, sd=1, Cov=NULL,
    rng=NULL, compiled=.DSEflags()$COMPILED, ...)
## S3 method for class 'TSestModel':
simulate(model, input=inputData(model),
                    sd=NULL, Cov=NULL, ...)
```

Arguments

`model`	An object of class TSmodel or TSestModel.
`input`	Data for the exogenous variable if specified in the model.
`sampleT`	The length of the sample to simulate.
`start`	start date for resulting data.
`freq`	freq for resulting data.

`y0, input0` Lagged values prior to t=1 for y and u, in reverse order so y0[1,] and input0[1,]correspond to t=0. These arguments are not implemented for state space models. If not specified initial values are set to zero.

`noise` Noise can be supplied. Otherwise it will be generated. If supplied it should be a list as described below in details.

`Cov` The covariance of the noise process. If this is specified then sd is ignored. A vector or scalar is treated as a diagonal matrix. For an object of class TSestModel, if neither Cov nor sd are specified, then Cov is set to the estimated covariance (`model$estimates$cov`).

`sd` The standard deviation of the noise. This can be a vector.

`noise.model` A TSmodel to be used for generating noise (not yet supported by SS methods).

`rng` The random number generator information needed to regenerate a simulation.

`compiled` Specifies the compiled version of the code should be used (instead of the S code version).

`...` arguments passed to other methods.

Details

A state space or ARMA model (see TSmodel, ARMA, and SS for more details) is simulated with pseudo random noise (The default noise is a normally distributed processes. An object of class TSdata is returned. This can be used as input to estimation algorithms. If start and freq are specified, or if input or `noise$w` (in that order) have time series properties, these are given to the output.

If `noise` is not supplied then random values will be generated using other supplied information or defaults. The rng will be set first if it is specified.

The default noise generation will be N(0,I) If `Q` is not square in a non innovations state space model (i.e. the system noise has a dimension less than the state dimension), then it is padded with zeros, so generated noise of higher dimension has no effect. If sd is supplied, then w as describe below will be N(0,sqr(sd)). sd can be a vector of `p` elements corresponding to each of the `p` outputs.

If noise is supplied it should be a list of the necessary noise processes. For non-innovation form state space models the list must have elements `w`, `e`, and `w0`. (`w0` is `w` for `t=0` in state space model and prior lags in ARMA models.) For innovation form state space models and ARMA models with MA components the list should have elements `w` and `w0`, but if `w0` is not specified it is set to zero. For ARMA models with no MA components (i.e. VAR models) the list needs only `w`. In this case, and in the innovations form state space model with `w0=0`, a matrix may be supplied in place of a list. `w` should be a `sampleT` by `p` matrix giving the noise for `t=1` to `sampleT`. If noise is specified `sampleT` will be set to the number of periods in `w`.

If `noise$w0` is a matrix (rather than a vector) for a state space model simulation (as it is for ARMA simulations) then it is set to a vector of zeros. This provides compatability with VAR models (ARMA models with no lags in B).

Input must be specified for ARMA models with `model$C` not NULL and state space models with `model$G` not NULL..

In general ARMA and SS simulations will not produce exactly the same results because it is impossible to determine necessary transformation of initial conditions and `w0`.

Value

The value returned is an object of class TSdata which can be supplied as an argument to estimation routines. (See TSdata). In addition to the usual elements (see the description of a TSdata object) there are some additional elements: model- the generating model, rng - the initial RNG and seed, version - the version of S used (random number generators may vary) Cov as specified sd as specified noise - the noise details as provided in the argument or as generated. state - the state variable for state space models.

See Also

makeTSnoise, TSmodel, TSdata, ARMA, SS

Examples

```
mod1 <- ARMA(A=array(c(1,-.25,-.05), c(3,1,1)), B=array(1,c(1,1,1)))
AR   <- array(c(1, .5, .3, 0, .2, .1, 0, .2, .05, 1, .5, .3) ,c(3,2,2))
VAR  <- ARMA(A=AR, B=diag(1,2))
print(VAR)
simData <- simulate(VAR)

C    <- array(c(0.5,0,0,0.2),c(1,2,2))
VARX <- ARMA(A=AR, B=diag(1,2), C=C)
simData <- simulate(VARX, sampleT=150, input=matrix(rnorm(300),150,2))

MA   <- array(c(1, .2, 0, .1, 0, 0, 1, .3), c(2,2,2))
ARMA  <- ARMA(A=AR, B=MA, C=NULL)
simData <- simulate(ARMA, sampleT=200)

ARMAX <- ARMA(A=AR, B=MA, C=C)
simData <- simulate(ARMAX, sampleT=150, input=matrix(rnorm(300),150,2))

data("eg1.DSE.data.diff", package="dse1")
model <- estVARXls(eg1.DSE.data.diff)
simData <- simulate(model)

ss <- SS(F=array(c(.5, .3, .2, .4), c(2,2)),
         H=array(c(1, 0, 0, 1), c(2,2)),
         K=array(c(.5, .3, .2, .4), c(2,2)))

print(ss)
simData <- simulate(ss)

testEqual(simData, simulate(ss))
testEqual(simData, simulate(ss, rng=getRNG(simData)))

simData2 <- simulate(ss,
    noise=list(w=matrix(runif(300), 150,2), w0=runif(2)))

simData3 <- simulate(ss, noise=matrix(runif(400), 200,2))
```

`smoother` *Evaluate a smoother with a TSmodel*

Description

Evaluate a state space model.

Usage

```
smoother(model, data, compiled=.DSEflags()$COMPILED)
## S3 method for class 'nonInnov':
smoother(model, data, compiled=.DSEflags()$COMPILED)
## S3 method for class 'TSmodel':
smoother(model, data, compiled=.DSEflags()$COMPILED)
## S3 method for class 'TSestModel':
smoother(model, data=TSdata(model),
  compiled=.DSEflags()$COMPILED)
```

Arguments

`model`	An object of class 'TSestModel' or 'TSmodel' with a model of class 'nonInnov' 'SS' 'TSmodel'. If filter informatin is not provided (i.e. in a TSestModel) then smoother runs the Kalman filter (l.SS) first.
`data`	A TSdata object.
`compiled`	If TRUE the compiled version of the code is used. Otherwise the S version is used.

Details

Calculate fixed interval smoother state values for a model. Smoother first runs the filter and uses the filtered state to calculate a smoothed estimate of the state (sometimes called a two sided filter). The smoother requires an non-innovations form model. The method for a TSmodel gives an error message if the model does not inherit from class `nonInnov`.

Note: this does not allow the same option as `l.SS` for calculating over a sub-sample. Smoothing is done over the length of the available filter data (which will be calculated to the length of the data if not supplied). For models with an input, smoothing will only be done over the input data period if that is shorter than the filter data.

See `SS` for details of the model:

z(t) = Fz(t-1) + Gu(t) + Qe(t) y(t) = Hz(t) + Rw(t)

Value

An object of class TSestModel with an additional element `smooth`. `smooth` is a list of `state`, the smoothed state, and `track`, the smoothed tracking error. The result will also contain the element `filter` with `state` and `track` (which may or may not have been in the original arguement).

References

Anderson, B. D. O. and Moore, J. B. (1979) *Optimal Filtering*. Prentice-Hall.

Shumway R. H. and Stoffer D.S. (1982) An Approach to Time Series Smoothing and Forecasting Using the EM Algorithm. *J. of Time Series Analysis*, **3**, 253–264 (note appendix).

Jazwinski, A. H. (1970) *Stochastic Processes and Filtering Theory*. Academic Press.

See Also

`state`, `l`, `SS` `l.SS` `TSmodel` `TSestModel.object`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
#smoother requires an non-innovations form model
model <- TSmodel(toSSChol(estVARXls(eg1.DSE.data.diff)))
smoothed.model <- smoother(model, eg1.DSE.data.diff, compiled=FALSE)
tfplot(state(smoothed.model))
tfplot(state(smoothed.model, filter=TRUE))
#compare
tfplot(state(smoothed.model, smoother=TRUE), state(smoothed.model, filter=TRUE))
```

SS *State Space Models*

Description

Construct a

Usage

```
SS(F.=NULL, G=NULL, H=NULL, K=NULL, Q=NULL, R=NULL, z0=NULL, P0=NULL, rootP0
      constants=NULL,
      description=NULL, names=NULL, input.names=NULL, output.names=NULL)
is.SS(obj)
is.innov.SS(obj)
is.nonInnov.SS(obj)
```

Arguments

`F.`	(nxn) state transition matrix.
`H`	(pxn) output matrix.
`Q`	(nxn) matrix specifying the system noise distribution.
`R`	(pxp) matrix specifying the output (measurement) noise distribution.
`G`	(nxp) input (control) matrix. G should be NULL if there is no input.
`K`	(nxp) matrix specifying the Kalman gain.
`z0`	vector indicating estimate of the state at time 0. Set to zero if not supplied.
`rootP0`	matrix indicating a square root of the initial tracking error (e.g. chol(P0)).
`P0`	matrix indicating initial tracking error P(t=1|t=0). Set to I if rootP0 or P0 are not supplied.
`constants`	NULL or a list of logical matrices with the same names as matices above, indicating which elements should be considered constants.
`description`	String. An arbitrary description.
`names`	A list with elements input and output, each a vector of strings. Arguments input.names and output.names should not be used if argument names is used.
`input.names`	A vector of character strings indicating input variable names.
`output.names`	A vector of character strings indicating output variable names.
`obj`	an object.

Details

State space models have a further sub-class: innov or non-innov, indicating an innovations form or a non-innovations form.

The state space (SS) model is defined by:

z(t) =Fz(t-1) + Gu(t) + Qe(t)

y(t) = Hz(t) + Rw(t)

or the innovations model:

z(t) =Fz(t-1) + Gu(t) + Kw(t-1)

y(t) = Hz(t) + w(t)

Matrices are as specified above in the arguments, and

y is the p dimensional output data.

u is the m dimensional exogenous (input) data.

z is the n dimensional (estimated) state at time t, E[z(t)|y(t-1), u(t)] denoted E[z(t)|t-1]. Note: In the case where there is no input u this corresponds to what would usually be called the predicted state - not the filtered state. An initial value for z can be specified as z0 and an initial one step ahead state tracking error (for non-innovations models) as P0. In the object returned by `l.ss`, `state` is a time series matrix corresponding to z.

z0 An initial value for z can be specified as z0.

P0 An initial one step ahead state tracking error (for non-innovations models) can be specified as P0.

rootP0 Alternatively, a square root of P0 can be specified. This can be an upper triangular matrix so that only the required number of parameters are used.

K, Q, R For sub-class `innov` the Kalman gain K is specified but not Q and R. For sub-class `non-innov` Q and R are specified but not the Kalman gain K.

e and w are typically assumed to be white noise in the non-innovations form, in which case the covariance of the system noise is QQ' and the covariance of the measurement noise is RR'. The covariance of e and w can be specified otherwise in the simulate method `simulate.SS` for this class of model, but the assumption is usually maintained when estimating models of this form (although, not by all authors).

Typically, an non-innovations form is harder to identify than an innovations form. Non-innovations form would typically be choosen when there is considerable theoretical or physical knowledge of the system (e.g. the system was built from known components with measured physical values).

By default, elements in parameter matrices are treated as constants if they are exactly 1.0 or 0.0, and as parameters otherwise. A value of 1.001 would be treated as a parameter, and this is the easiest way to initialize an element which is not to be treated as a constant of value 1.0. Any matrix elements can be fixed to constants by specifying the list `constants`. Matrices which are not specified in the list will be treated in the default way. An alternative for fixing constants is the function codefixConstants

Value

An SS TSmodel

References

Anderson, B. D. O. and Moore, J. B. (1979) *Optimal Filtering*. Prentice-Hall. (note p.39,44.)

See Also

`TSmodel` `ARMA` `simulate.SS` `l.SS` `state` `smoother` `fixConstants`

Examples

```
f <- array(c(.5,.3,.2,.4),c(2,2))
h <- array(c(1,0,0,1),c(2,2))
k <- array(c(.5,.3,.2,.4),c(2,2))
ss <- SS(F=f,G=NULL,H=h,K=k)
is.SS(ss)
ss
```

`stability` *Calculate Stability of a TSmodel*

Description

Calculate roots and their modulus and indicate stability.

Usage

```
stability(obj, fuzz=1e-4, eps=1e-15, digits=8, verbose=TRUE)
## S3 method for class 'ARMA':
stability(obj, fuzz=1e-4, eps=1e-15, digits=8, verbose=TRUE)
## S3 method for class 'roots':
stability(obj, fuzz=1e-4, eps=1e-15, digits=8, verbose=TRUE)
## S3 method for class 'TSmodel':
stability(obj, fuzz=1e-4, eps=1e-15, digits=8, verbose=TRUE)
## S3 method for class 'TSestModel':
stability(obj, fuzz=1e-4, eps=1e-15, digits=8, verbose=TRUE)
```

Arguments

`obj`	An object of class TSmodel.
`fuzz`	Roots within fuzz are considered equal.
`eps`	Roots with modulus less than (1-eps) are considered stable.
`digits`	Printing precision.
`verbose`	Print roots and there moduli.

Details

`eps` prevents the indication of a stable model when the largest root is within rounding error of 1.0.

Value

TRUE or FALSE if the model is stable or not stable.

Side Effects

The eigenvalues of the state transition matrix or the roots of the determinant of the AR polynomial are printed if verbose is T.

See Also

McMillanDegree

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- estVARXls(eg1.DSE.data.diff)
stability(model)
```

---

state — *Extract State*

Description

Extract state information from estimated SS model.

Usage

```
state(obj, smoother=FALSE, filter=!smoother)
```

Arguments

obj	An object of class 'TSestModel' with state information (filter or smoother) or containing an 'SS' model from which to estimate the state.
smoother	logical indicating if the smoother state should be returned.
filter	logical indicating if the filtered state should be returned.

.

Details

One and only one of smoother and filter should be TRUE).

Value

A time series matrix of the estimated state series.

See Also

smoother, SS, l.SS

summary.TSdata — *Specific Methods for Summary*

Description

See the generic function description.

Usage

```
## S3 method for class 'TSdata':
summary(object, ...)
## S3 method for class 'SS':
summary(object, ...)
## S3 method for class 'ARMA':
summary(object, ...)
## S3 method for class 'TSestModel':
summary(object, ...)
## S3 method for class 'summary.TSdata':
print(x, digits=options()$digits, ...)
## S3 method for class 'summary.SS':
print(x, digits=options()$digits, ...)
## S3 method for class 'summary.ARMA':
print(x, digits=options()$digits, ...)
## S3 method for class 'summary.TSestModel':
print(x, digits=options()$digits, ...)
```

Arguments

`object`	an object to be summarized.
`x`	a summary object to be printed.
`digits`	number of significant digits to use for printing.
`...`	arguments passed to other methods.

See Also

`print`, `summary`

sumSqerror — *Calculate sum of squared prediction errors*

Description

Calculate a weighted sum squared prediction errors for a parameterization.

Usage

```
sumSqerror(coefficients, model=NULL, data=NULL, error.weights=NULL)
```

Arguments

`coefficients` A vector of coefficients (parameters).

`model` an object of class TSmodel which gives the structure of the model for which coefficients are used. `coef(model)` should be the same length as coefficients.

`data` an object of class TSdata which gives the data with which the model is to be evaluated.

`error.weights`
a vector of weights to be applied to the squared prediction errors.

Details

This function is primarily for use in parameter optimization, which requires that an objective function be specified by a vector of parameters.It returns only the sum of the weighted squared errors (eg.for optimization). The sample size is determined by periodsOutput(data).

Value

The value of the sum squared errors for a prediction horizon given by the length of error.weights. Each period ahead is weighted by the corresponding weight in error.weights.

See Also

`l` `l.SS` `l.ARMA`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- estVARXls(eg1.DSE.data.diff)
sumSqerror(1e-10 + coef(model), model=TSmodel(model),
        data=TSdata(model), error.weights=c(1,1,10))
```

---

testEqual.ARMA *Specific Methods for Testing Equality*

Description

See the generic function description.

Usage

```
## S3 method for class 'ARMA':
testEqual(obj1, obj2, fuzz=0)
## S3 method for class 'SS':
testEqual(obj1, obj2, fuzz=0)
## S3 method for class 'TSdata':
testEqual(obj1, obj2, fuzz=1e-16)
## S3 method for class 'TSmodel':
testEqual(obj1, obj2, fuzz=0)
## S3 method for class 'TSestModel':
testEqual(obj1, obj2, fuzz=0)
```

Arguments

`obj1`	see generic method.
`obj2`	see generic method.
`fuzz`	see generic method.

See Also

testEqual

`tfplot.TSdata` *Tfplot Specific Methods*

Description

See the generic function description.

Usage

```
## S3 method for class 'TSdata':
tfplot(x, ...,
    tf=NULL, start=tfstart(tf), end=tfend(tf),
    select.inputs  = seq(length=nseriesInput(x)),
    select.outputs = seq(length=nseriesOutput(x)),
    Title=NULL, xlab=NULL, ylab=NULL,
    graphs.per.page=5, mar=par()$mar, reset.screen=TRUE)
## S3 method for class 'TSestModel':
tfplot(x, ...,
    tf=NULL, start=tfstart(tf), end=tfend(tf),
    select.inputs=NULL, select.outputs=NULL,
    Title=NULL, xlab=NULL, ylab=NULL,
    graphs.per.page=5, mar=par()$mar, reset.screen=TRUE)
```

Arguments

`x`	object to be plotted.
`...`	additional objects to be plotted.
`start`	start of plot.
`end`	end of plot.
`tf`	an alternate way to specify start and end of plot.
`select.inputs`	series to be plotted. (passed to selectSeries)
`select.outputs`	series to be plotted. (passed to selectSeries)
`Title`	string to use for plot title (passed to plot - see tfplot).
`xlab`	string to use for x label (passed to plot).
`ylab`	string to use for y label (passed to plot).

graphs.per.page
: integer indicating number of graphs to place on a page.

mar
: margins passed to plot. See par.)

reset.screen
: logical indicating if the plot window should be cleared before starting. If this is not TRUE then mar values will have no effect.

See Also

tfplot

---

tframed.TSdata — *Specific Methods for tframed Data*

Description

See the generic function description.

Usage

```
## S3 method for class 'TSdata':
tframed(x, tf=NULL, names=NULL, ...)
## S3 replacement method for class 'TSdata':
tframe(x) <- value
## S3 method for class 'TSdata':
tfwindow(x, tf=NULL, start=tfstart(tf), end=tfend(tf), warn=TRUE)
## S3 method for class 'TSdata':
tbind(x, d2, ..., pad.start=TRUE, pad.end=TRUE, warn=TRUE)
## S3 method for class 'TSdata':
trimNA(x, startNAs=TRUE, endNAs=TRUE)
## S3 method for class 'TSdata':
window(x, start=NULL, end=NULL, tf=NULL, warn=TRUE, ...)
```

Arguments

x	See the generic function.
tf	a time frame. See the generic function.
value	a time frame to associate with x.
names	A list with elements input and output which are strings passed as names to the default method.
start	See the generic function.
startNAs	See the generic function.
end	See the generic function.
endNAs	See the generic function.
d2	See the generic function.
pad.start	See the generic function.
pad.end	See the generic function.
warn	logical indicating if some warning messages should be suppressed.
...	arguments passed to other functions.

Details

The generic function is applied to input and to output data.

See Also

tframed, tfwindow, tbind, trimNA

---

toARMA *Convert to an ARMA Model*

Description

Convert a state space model to an ARMA representation. The state is eliminated by a method which uses an equivalence that can be demonstrated by the Cayley Hamilton theorem. It is not very parsimonious.

Usage

```
toARMA(model, ...)
## S3 method for class 'ARMA':
toARMA(model, ...)
## S3 method for class 'SS':
toARMA(model, fuzz=1e-10, ...)
## S3 method for class 'TSestModel':
toARMA(model, ...)
```

Arguments

model	An object of class TSmodel.
fuzz	Parameters closer than fuzz to one or zero are set to 1.0 or 0.0 respectively
...	arguments to be passed to other methods.

Value

An object of class 'ARMA' 'TSmodel' containing an ARMA model.

References

See, for example, Aoki, M. (1990) *State Space Modelling of Time Series*. 2d ed. rev. and enl., Springer-Verlag.

Aoki, M. and Havenner, A. (1991) State Space Modeling of Multiple Time Series. *Econometric Reviews*, **10**, 1–59.

See Also

toSS fixConstants

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- toSS(estVARXls(eg1.DSE.data.diff))
model <- toARMA(model)
```

toSSChol — *Convert to Non-Innovation State Space Model*

Description

This function may not be working properly.

Convert to a non-innovations state space representation using the given matrix (Om) as the measurement noise covariance. Om would typically be an estimate of the output noise, such as returned in `$estimates$cov` of the function `l` (`l.SS` or `l.ARMA`). This assumes that the noise processes in the arbitrary SS representation are white and uncorrelated.

Usage

```
toSSChol(model, ...)
## S3 method for class 'TSmodel':
toSSChol(model, Om=diag(1,nseriesOutput(model)), ...)
## S3 method for class 'TSestModel':
toSSChol(model, Om=NULL, ...)
```

Arguments

`model`	An object of class TSmodel.
`Om`	a matrix to be used as the measurement noise covariance. If Om is not supplied and model is of class TSestModel then `model$estimates$cov` is used. Otherwise, Om is set to the identity matrix.
`...`	arguments to be passed to other methods.

Details

Convert to a non-innovations SS representation using a Cholesky decomposition of Om as the coefficient matrix of the output noise.

Value

An object of class 'SS' 'TSmodel' containing a state space model which is not in innovations form.

See Also

`toSSinnov`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- estVARXls(eg1.DSE.data.diff)
model <- toSSChol(model)
```

toSSinnov — *Convert to State Space Innovations Model*

Description

Convert to a state space innovations representation.

Usage

```
toSSinnov(model, ...)
```

Arguments

`model`	an object of class TSmodel.
`...`	arguments passed to other methods.

Value

If the argument is a TSmodel then the result is an object of class 'SS' 'TSmodel' If the argument is a TSestModel then the converted model is evaluated with the data an a TSestModel is returned. The TSmodel is an innovations state space representation.

This assumes that the noise processes in the arbitrary SS representation are white and uncorrelated.

See Also

`toSS`, `toSSOform` `toSSChol`

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- estVARXls(eg1.DSE.data.diff)
model <- toSSinnov(model)
summary(model)

model2 <- SS(F=diag(1,3), H=matrix(c(1,0,0,1,0,0),2,3),
   Q=diag(0.5, 3, 3), R=diag(1.1, 2,2),
    description="test model", output.names=c("output 1", "output 2"))
model2 <- toSSinnov(model2)
summary(model2)
```

toSSOform — *Convert to Oform*

Description

Convert a state space model to (observability?) form.

Usage

```
toSSOform(model)
## S3 method for class 'TSmodel':
toSSOform(model)
## S3 method for class 'TSestModel':
toSSOform(model)
```

Arguments

model	An object of class TSmodel.

Details

WARNING: This function does not work properly.

Convert to a SS innovations representation with a minimum number of parameters by converting as much of H as possible to I matrix. Any remaining reductions are done by converting part of ?? to I. It seems there should remain n(m+2p) free parameters in F,G,H,K, and Om is determined implicitly by the residual.

Value

An object of class 'SS' 'TSmodel' containing a state space model in observability form (more or less).

See Also

toSSinnov

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- estVARXls(eg1.DSE.data.diff)
```

---

toSS *Convert to State Space Model*

Description

Convert a model to state space form.

Usage

```
toSS(model, ...)
## S3 method for class 'ARMA':
toSS(model, ...)
## S3 method for class 'SS':
toSS(model, ...)
## S3 method for class 'TSestModel':
toSS(model, ...)
```

```
toSSaugment(model, ...)
## S3 method for class 'ARMA':
toSSaugment(model, fuzz=1e-14, ...)
## S3 method for class 'TSestModel':
toSSaugment(model, ...)

toSSnested(model, ...)
## S3 method for class 'ARMA':
toSSnested(model, n=NULL, Aoki=FALSE, ...)
## S3 method for class 'SS':
toSSnested(model, n=NULL, Aoki=FALSE, ...)
## S3 method for class 'TSestModel':
toSSnested(model, ...)
```

Arguments

`model`	An object of class TSmodel.
`n`	If n is specified then it is used as the state dimension when the markov parameter conversion technique is required.
`Aoki`	logical indicating if Aoki's method (which does not work in general) should be tried.
`fuzz`	if the zero lag term of polynomials A and B are within fuzz of the identitity matrix then they are not inverted. (i.e. they are assumed to be identity.)
`...`	arguments to be passed to other methods.

Details

If the order of the AR polynomial equals or exceeds the MA polynomial (and the input polynomial) then the model is converted by state augmentation. Otherwise, it is converted by approximating the markov coefficients a la Mittnik. (This may not always work very well. Compare the results to check.)

Value

A state space model in an object of class 'SS' 'TSmodel'.

Examples

```
data("eg1.DSE.data.diff", package="dse1")
model <- estVARXls(eg1.DSE.data.diff)
model <- toSS(model)
```

`TSdata.object` *time series data object*

Description

Class "TSdata" of time series data objects for use with TSmodels.

Generation

This class of objects is returned by specific methods of the function TSdata or can be built according to the description below.

Methods

The `TSdata` class of objects has methods for the generic functions `print, plot, periods, start, end, ..., testEqual, seriesNames, seriesNamesInput, seriesNamesOutput` Also, the function `is.TSdata` is supported.

Inheritance

Other data classes inherit from the class `TSdata`.

Structure

Objects are a list with class the most general class "TSdata". The native form for this package has elements `input` and `output`. Any other elements are ignored. `input` and `output` are matrices (or tframe or time series matrices) of the input and output data, with each series in a column. `TSPADIdata` objects inherit from this class but have a somewhat different structure. `TSPADIdata` makes it possible to retrieve data from an external database when it is needed. These subclass objects do not contain the actual data, but only the names of the series and the data base where they are located. The function setTSPADIdata can be used to set up an object of class c("TSPADIdata", "TSdata").

See Also

`TSdata` `TSmodel` `TSestModel.object` `TSPADIdata`

TSdata — *Construct TSdata time series object*

Description

Constructor for generating or extracting a "TSdata" object containing data for use by TSmodels.

Usage

```
TSdata(data=NULL, ...)
## Default S3 method:
TSdata(data=NULL, input=NULL, output=NULL, ...)
## S3 method for class 'TSdata':
TSdata(data, ...)
## S3 method for class 'TSestModel':
TSdata(data, ...)
is.TSdata(obj)
as.TSdata(d)
```

Arguments

`data`	object of class TSdata, TSestModel, matrix, list with input and output matrices, or another object for which a constructor or TSdata extraction method has been defined.
`input`	a matrix of time series data.
`output`	a matrix of time series data.
`...`	arguments to be passed to other methods.
`obj`	an object.
`d`	an object from which a TSdata object can be extracted. See below.

Details

Generic method to construct or extract a TSdata object. The default method constructs a TSdata object. Specific methods extract the TSdata from other objects (which must contain TSdata). The function `is.TSdata(data)` returns TRUE is data inherits from "TSdata" and FALSE otherwise.

The function `as.TSdata` uses the elements input and output directly and strips away other class information and parts of the object (and does not make use of inputData(data) or outputData(data) which may do something special for certain classes.

See Also

`TSdata.object`, `freeze`, `freeze.TSPADIdata`, `TSmodel`, `TSestModel.object` `setTSPADIdata`

Examples

```
rain  <- matrix(rnorm(86*17), 86,17)
radar <- matrix(rnorm(86*5),  86,5)
mydata <- TSdata(input=radar, output=rain)
```

`TSestModel` *Estimated Time Series Model*

Description

Object containing a time series model, data, and estimation information.

Usage

```
TSestModel(obj)
## S3 method for class 'TSestModel':
TSestModel(obj)
is.TSestModel(obj)
```

Arguments

`obj`	in the first usage an object from which a TSestModel object can be extracted (or constructed).

Details

The TSestModel class of objects are generated by estimation methods. See, for example, `estVARXls`. They contains a time series model (TSmodel), data (TSdata), and information obtained by evaluating the model with the data in an element called `estimates` containing:

like The negative log likelihood function value (a vector of the total, constant, the det part, and the cov part)

cov The estimated residual covariance.

pred The one step ahead predictions (see predictT below). These are aligned with output data so that residuals are pred[1:sampleT,] - output[1:sampleT,]

sampleT The end of the period (starting from 1) for which output is used for calculating one step ahead predictions.

predictT The end of the period for which the model is simulated. sampleT must be less than or equal predictT. If predictT is greater than sampleT then each step ahead beyond sampleT is based on the prediction of the previous step and not corrected by the prediction error.

The element `estimates` may optionally also contain and element `filter` which may have

state The one step ahead (filter) estimate of the state E[z(t)|y(t-1), u(t)]. Note: In the case where there is no input u this corresponds to what would usually be called the predicted state - not the filtered state.

track The estimated state tracking error P(t|t-1). Again note, this corresponds to the predicted tracking eror not the filtered tracking error. This is NULL for innovations models.

smooth a list of:

state The smoother (two sided filter) estimate of the state E[z(t)| sampleT].

track The smoothed estimate of the state tracking error P(t|sampleT). This is NULL for innovations models.

See Also

`estVARXls`, `TSmodel`, `TSdata`

`TSmodel` *Time Series Models*

Description

Construct or extract a "TSmodel" from objects.

Usage

```
TSmodel(obj, ...)
## S3 method for class 'TSmodel':
TSmodel(obj, ...)
## S3 method for class 'TSestModel':
TSmodel(obj, ...)
is.TSmodel(obj)
```

Arguments

`obj`	An object containing an object of class TSmodel or a list containing the information necessary to build an object of class "TSmodel".
`...`	arguments passed to other methods.

Details

This is a generic method which will extract a "TSmodel" from an object (e.g. a TSestModel. The default method will try to build an ARMA or state-space "TSmodel" from a list, which must contain the necessary information.

This class of objects is returned by estimation methods or can be built according to the description for specific sub-classes (eg "ARMA", "SS").

The `TSmodel` class of objects has methods for the generic functions `print, testEqual, seriesNames, seriesNamesInput, seriesNamesOutput, l, roots, stability, forecast, featherForecasts, horizonForecasts, simulate, MonteCarloSimulations`

Also, the function `is.TSmodel` and the functions `toSS, toARMA, to.troll` are supported. Other model classes inherit from the class `TSmodel`.

This class of objects contains a time series model. It is the class of objects expected by many of the functions in this package.

Sub-class (e.g. ARMA and SS for linear, time-invariant ARMA and state space models.) are documented individually. Many of the functions in this package are designed for estimating and converting among various representations of these types of models.

See Also

`ARMA`, `SS`, `TSestModel`, `TSdata`

Index