Package 'tfarima'

May 8, 2026

Type Package

Title Transfer Function and ARIMA Models

Version 0.4.1

Description Build customized transfer function and ARIMA models with multiple operators and parameter restrictions. Provides tools for model identification, estimation using exact or conditional maximum likelihood, diagnostic checking, automatic outlier detection, calendar effects, forecasting, and seasonal adjustment. The new version also supports unobserved component ARIMA model specification and estimation for structural time series analysis.

License GPL (>= 2)

URL https://github.com/gallegoj/tfarima

Depends R (>= 3.5.0)

Imports Rcpp (>= 1.0.0), stats, MASS, numDeriv, zoo, nnls, quadprog

LinkingTo Rcpp, RcppArmadillo

Suggests knitr, rmarkdown

Encoding UTF-8

RoxygenNote 7.3.3

VignetteBuilder knitr

NeedsCompilation yes

Author Jose L. Gallego [aut, cre]

Maintainer Jose L. Gallego <jose.gallego@unican.es>

Repository CRAN

Date/Publication 2025-11-03 19:40:02 UTC

Contents

`tfarima-package` *Transfer Function and ARIMA Models*

Description

The **tfarima** package provides classes and methods to build customized transfer function and ARIMA models with multiple operators and parameter restrictions. It includes functions for model identification, estimation using exact or conditional maximum likelihood, diagnostic checking, automatic outlier detection, calendar effects, forecasting, and seasonal adjustment.

Details

The current version extends the functionality by incorporating the estimation of unobserved components in ARIMA models through the UCARIMA representation and structural time series models.

Author(s)

Jose Luis Gallego <jose.gallego@unican.es>

References

Bell, W. R. and Hillmer, S. C. (1983). Modeling Time Series with Calendar Variation. *Journal of the American Statistical Association*, 78(383), 526–534.

Box, G. E. P., Jenkins, G. M., Reinsel, G. C., and Ljung, G. M. (2015). *Time Series Analysis: Forecasting and Control*. John Wiley & Sons, Hoboken.

Box, G. E. P., Pierce, D. A., and Newbold, D. A. (1987). Estimating Trend and Growth Rates in Seasonal Time Series. *Journal of the American Statistical Association*, 82(397), 276–282.

Box, G. E. P. and Tiao, G. C. (1975). Intervention Analysis with Applications to Economic and Environmental Problems. *Journal of the American Statistical Association*, 70(349), 70–79.

Chen, C. and Liu, L. (1993). Joint Estimation of Model Parameters and Outlier Effects in Time Series. *Journal of the American Statistical Association*, 88(421), 284–297.

Thompson, H. E. and Tiao, G. C. (1971). Analysis of Telephone Data: A Case Study of Forecasting Seasonal Time Series. *Bell Journal of Economics*, 2(2), 515–541.

add_um *Addition or substraction of univariate (ARIMA) models*

Description

add_um creates a univariate (ARIMA) model from the addition or substraction of two univariate (arima) models.

Usage

```
add_um(um1, um2, add = TRUE, tol = 1e-05)
```

Arguments

um1, um2	Two "um" S3 objects.
add	logical. If FALSE, the second model is substracted from the first one.
tol	tolerance to check if a value is null.

Value

A "um" S3 object.

Note

The + and - operators can also be used to add or substract ARIMA models.

Examples

```
um1 <- um(i = "(1 - B)", ma = "(1 - 0.8B)")
um2 <- um(i = "(1 - B12)", ma = "(1 - 0.8B^12)")
um3 <- add_um(um1, um2)
um4 <- um3 - um2
```

AIC.ssm *AIC for fitted state space models*

Description

AIC for fitted state space models

Usage

```
## S3 method for class 'ssm'
AIC(object, k = 2, ...)
```

Arguments

object	an object of class ssm.
k	numeric, penalty per parameter (default is 2).
...	additional arguments.

Value

The AIC value, or NULL if model is not fitted.

AIC.tfm *AIC and BIC for Transfer Function Models*

Description

Computes Akaike's Information Criterion (AIC) and Bayesian Information Criterion (BIC) for transfer function models.

Usage

```
## S3 method for class 'tfm'
AIC(object, ..., k = 2)

## S3 method for class 'tfm'
BIC(object, ...)
```

Arguments

object	A fitted tfm object.
...	Additional tfm objects for model comparison.
k	Numeric. Penalty per parameter. Default is 2 for AIC. Use k = log(n) for BIC where n is sample size.

Details

AIC = -2*logLik + k*npar, where npar is the number of parameters. Lower values indicate better fit penalized for complexity.

Value

If one model: numeric value of AIC/BIC. If multiple models: data frame with columns df (degrees of freedom) and AIC for each model.

See Also

logLik.tfm, BIC.tfm

Examples

```
## Not run:
model1 <- tfm(output, inputs = tf1, noise = noise1)
model2 <- tfm(output, inputs = tf2, noise = noise2)

# Single model AIC
AIC(model1)

# Compare models
AIC(model1, model2)

# BIC
BIC(model1)

## End(Not run)
```

`airline` *Airline Model (SARIMA(0,1,1)x(0,1,1)s)*

Description

Creates a seasonal ARIMA model with the structure popularized by Box and Jenkins using airline passenger data: (0,1,1)x(0,1,1)s.

Usage

```
airline(z, bc = FALSE, sma = c("standard", "generalized", "factorized"), ...)
```

Arguments

`z`	A `ts` object (must have frequency > 1).
`bc`	Logical. If TRUE, applies Box-Cox (log) transformation.
`sma`	Character. Specification for seasonal MA operator. Options are: standard, generalized or factorized. See manual for more details.
`...`	Additional arguments passed to `um`.

Details

This is a convenience function equivalent to: `um(z, bc = bc, i = list(1, c(1, s)), ma = list(1, c(1, s)), ...)` where s = frequency(z).

Value

A `um` object with airline model specification.

See Also

`um`

as.lagpol *Lag polynomial converter*

Description

as.lagpol converts a numeric vector c(1, -a_1, ..., -a_d) into a lag polynomial $(1 - a_1 B - ... - a_p B^p)$.

Usage

```
as.lagpol(pol, p = 1, coef.name = "a")
```

Arguments

pol	the numeric vector to be converted into an object of class lag polynomial.
p	the exponent of the lag polynomial, positive integer.
coef.name	name prefix for coefficients, character.

Value

An object of class lagpol.

Examples

```
as.lagpol(c(1, -0.8))
as.lagpol(c(1, 0, 0, 0, -0.8))
```

as.ssm.ucarima *Structural form for an ARIMA model*

Description

as.ssm finds the structural form for an ARIMA model from its the eventual forecast function.

Usage

```
## S3 method for class 'ucarima'
as.ssm(object, ...)

as.ssm(object, ...)

## S3 method for class 'um'
as.ssm(
  object,
  z = NULL,
```

```
  msoe = TRUE,
  H = NULL,
  cform = TRUE,
  tol = 1.490116e-08,
  nonadm = c("quadprog", "nnls", "none"),
  envir = NULL,
  ...
)
```

Arguments

object	an object of class um.
...	other arguments.
z	an optional time series.
msoe	logical, TRUE for multiple source of errors and FALSE for single source of error.
H	an optional matrix to reduce the number of variances.
cform	logical. TRUE for contemporaneous form and FALSE for future form.
tol	tolerance to check if the elements of b and C are zero.
nonadm	character, the method to overcome nonadmissibility: non-linear least squares, quadratic programming or none.
envir	environment, see "um".

Value

An object of class ssm

Examples

```
airl <- um(i = list(1, c(1, 12)), ma = "(1 - 0.8B)(1 - 0.8B12)")
ssm1 <- as.ssm(airl, index = c(1, 0, rep(2, 11)))
ssm1
```

---

as.ucarima *Generic function for coercion to class "ucarima"*

Description

Coerce an object to class "ucarima"

Usage

```
as.ucarima(object, ...)
```

Arguments

object	An object to be coerced.
...	Further arguments passed to or from methods.

Value

An object of class "ucarima".

---

as.ucarima.um *Coerce a Univariate Model to UCARIMA form*

Description

Converts an object of class "um" (univariate model) to its equivalent "ucarima" representation, i.e., the ARIMA-model-based decomposition of unobserved components (trend, seasonal, cycle, irregular, etc.) implied by the univariate ARIMA structure, following the approach of Hillmer and Tiao (1982).

Usage

```
## S3 method for class 'um'
as.ucarima(
  object,
  ar = NULL,
  i = NULL,
  single = FALSE,
  canonical = FALSE,
  cwfact = c("roots", "iter", "best"),
  pfrac = c("gcd", "solve"),
  tol = 1e-05,
  envir = parent.frame(),
  ...
)
```

Arguments

object	An object of class "um".
ar	Autoregressive lag polynomial for the signal component.
i	Integration lag polynomial for the signal component.
single	Logical. If TRUE, extracts a single component; if FALSE, extracts multiple components.
canonical	Logical. If TRUE, applies the canonical decomposition constraint.
cwfact	Method for Cramer-Wold factorization: "roots" polynomial (Godolphin 1976), "wilson" iterative algorithm (Wilson 1969), or the "best" of both methods.

pfrac	Method for partial fraction decomposition: "gcd" (extended Euclidean algorithm) or "solve" (linear system solver).
tol	Numerical tolerance for zero and unit values. Default is 1e-5.
envir	Environment for evaluation.
...	Additional arguments.

Details

The UCARIMA decomposition expresses a univariate ARIMA model as the sum of independent component ARIMA models (trend, seasonal, cycle, irregular, etc.) obtained through the factorization of its spectral density. This provides a model-based interpretation of signal extraction and seasonal adjustment.

References

Hillmer, S. C., & Tiao, G. C. (1982). An ARIMA-model-based approach to seasonal adjustment. *Journal of the American Statistical Association*, **77**(377), 63–70.

Burman, J. P. (1980). Seasonal adjustment by signal extraction. *Journal of the Royal Statistical Society: Series A*, **143**(3), 321–337.

Godolphin, E. J. (1976). On the Cramer–Wold factorization. *Biometrika*, **63**(2), 367–372. doi:10.1093/biomet/63.2.367

Tunnicliffe Wilson, G. (1969). Factorization of the covariance generating function of a pure moving average process. *SIAM Journal on Numerical Analysis*, **6**(1), 1–7. doi:10.1137/0706001

See Also

ucarima

---

as.um *Convert* arima *into* um.

Description

as.um converts an object of class arima into an object of class um.

Usage

```
as.um(arima, ...)
```

Arguments

arima	an object of class arima.
...	additional arguments.

Value

An object of class um.

Examples

```
z <- AirPassengers
a <- arima(log(z), order = c(0,1,1),
seasonal = list(order = c(0,1,1), frequency = 12))
um1 <- as.um(a)
```

---

autocorr.ucarima *Theoretical simple/partial autocorrelations of an ARMA model*

Description

autocorr computes the simple/partial autocorrelations of an ARMA model.

Usage

```
## S3 method for class 'ucarima'
autocorr(x, ...)

autocorr(x, ...)

## S3 method for class 'um'
autocorr(x, lag.max = 10, par = FALSE, ...)
```

Arguments

x	an object of class um.
...	additional arguments.
lag.max	maximum lag for autocovariances.
par	logical. If TRUE partial autocorrelations are computed.

Value

A numeric vector.

Note

The I polynomial is ignored.

Examples

```
ar1 <- um(ar = "1-0.8B")
autocorr(ar1, lag.max = 13)
autocorr(ar1, lag.max = 13, par = TRUE)
```

autocov.ssm *Theoretical autocovariances of an ARMA model*

Description

autocov computes the autocovariances of an ARMA model.

Usage

```
## S3 method for class 'ssm'
autocov(mdl, lag.max = NULL, arma = TRUE, varphi = FALSE, tol = 1e-04, ...)

## S3 method for class 'ucarima'
autocov(mdl, ma = FALSE, ...)

autocov(mdl, ...)

## S3 method for class 'um'
autocov(mdl, lag.max = 10, ...)
```

Arguments

mdl	an object of class um or ucm.
lag.max	maximum lag for autocovariances.
arma	logical. If TRUE, the autocovariances for the stationary ARMA model of the reduced form are computed. Otherwise, the autocovariances are only computed for the MA part.
varphi	logical. If TRUE, the varphi polynomial of the reduced form is also returned.
tol	tolerance to check if a root is close to one.
...	additional arguments.
ma	logical; if true, autocovariances are computed for the MA model. By default, ma = FALSE and autocovariances are computed for the ARMA model.

Value

A numeric vector.

Note

The I polynomial is ignored.

Examples

```
# Local level model
ssm1 <- ssm(b = 1, C = 1, S = diag(c(irr = 0.8, lvl = 0.04)))
autocov(ssm1)

ar1 <- um(ar = "1-0.8B")
autocov(ar1, lag.max = 13)
```

`autocov2MA` *Convert autocovariances to MA parameters*

Description

Computes MA polynomial coefficients and error variance from autocovariances.

Usage

```
autocov2MA(x, method = c("roots", "acov"), tol = 1e-05)
```

Arguments

`x`	Numeric vector of autocovariances (length q).
`method`	Estimation method: "roots" (Godolphin 1976) or "acov" (Wilson 1969). Default is "roots".
`tol`	Tolerance for zero autocovariance. Default 1e-5.

Value

Named vector: c(s2, ma0=1, ma1=-theta1, ..., maq=-thetaq).

References

Godolphin, E. J. (1976). On the Cramer-Wold factorization. *Biometrika*, **63**(2), 367-372. doi:10.1093/biomet/63.2.367

Tunnicliffe Wilson, G. (1969). Factorization of the covariance generating function of a pure moving average process. *SIAM Journal on Numerical Analysis*, **6**(1), 1-7. doi:10.1137/0706001

Examples

```
ma1 <- um(ma = "1 - 0.8B", sig2 = 0.5)
autocov2MA(autocov(ma1, 1))
autocov2MA(autocov(ma1, 1), method = "acov")
```

`BuildingMat` *Monthly Retail Sales: Building Material and Supplies Dealers (NAICS 4441)*

Description

Monthly U.S. retail sales for building material and supplies dealers (NAICS code 4441), in millions of dollars, seasonally adjusted. Source: U.S. Census Bureau, Monthly Retail Trade Survey (via FRED).

Usage

```
BuildingMat
```

Format

A time series object of class `ts` with frequency 12, starting in January 1992 and ending in December 2024.

References

U.S. Census Bureau, Monthly Retail Trade Survey. FRED series code: MRTSSM4441USS.

`calendar.ssm` *Calendar effects*

Description

`calendar` extends the ARIMA model `um` by including a set of deterministic variables to capture the calendar variation in a monthly time series. Two equivalent representations are available: (i) D0, D1, ..., D6, (ii) L, D1-D0, ..., D6-D0 where D0, D2, ..., D6 are deterministic variables representing the number of Sundays, Mondays, ..., Saturdays, L = D0 + D1 + ... + D6 is the of the month. Alternatively, the Leap Year indicator (LPY) can be included instead of L. The seven trading days can also be compacted into two variables: week days and weekends. Optionally, a deterministic variable to estimate the Easter effect can also be included, see "`easter`".

Usage

```
## S3 method for class 'ssm'
calendar(
  mdl,
  form = c("dif", "td", "td7", "td6", "wd"),
  ref = 0,
  lom = TRUE,
  lpyear = TRUE,
  easter = FALSE,
```

```
  len = 4,
  easter.mon = FALSE,
  n.ahead = 0,
  p.value = 1,
  envir = NULL,
  ...
)

## S3 method for class 'tfm'
calendar(
  mdl,
  y = NULL,
  form = c("dif", "td", "td7", "td6", "wd"),
  ref = 0,
  lom = TRUE,
  lpyear = TRUE,
  easter = FALSE,
  len = 4,
  easter.mon = FALSE,
  n.ahead = 0,
  p.value = 1,
  envir = parent.frame(),
  ...
)

calendar(mdl, ...)

## S3 method for class 'um'
calendar(
  mdl,
  y = NULL,
  form = c("dif", "td", "td7", "td6", "wd"),
  ref = 0,
  lom = TRUE,
  lpyear = TRUE,
  easter = FALSE,
  len = 4,
  easter.mon = FALSE,
  n.ahead = 0,
  p.value = 1,
  envir = parent.frame(),
  ...
)
```

Arguments

mdl	an object of class um or tfm.
form	representation for calendar effects: (1) form = dif, L, D1-D0, ..., D6-D0; (2)

	form = td, LPY, D1-D0, ..., D6-D0; (3) form = td7, D0, D2, ..., D6; (4) form = td6, D1, D2, ..., D6; (5) form = wd, (D1+...+D5) - 2(D6+D0)/5.
ref	a integer indicating the the reference day. By default, ref = 0.
lom, lpyear	a logical value indicating whether or not to include the lom/lead year indicator.
easter	logical. If TRUE an Easter effect is also estimated.
len	the length of the Easter, integer.
easter.mon	logical. TRUE indicates that Easter Monday is a public holiday.
n.ahead	a positive integer to extend the sample period of the deterministic variables with n.ahead observations, which could be necessary to forecast the output.
p.value	estimates with a p-value greater than p.value are omitted.
envir	environment in which the function arguments are evaluated. If NULL the calling environment of this function will be used.
...	other arguments.
y	a time series.

Value

An object of class "tfm".

References

W. R. Bell & S. C. Hillmer (1983) Modeling Time Series with Calendar Variation, Journal of the American Statistical Association, 78:383, 526-534, DOI: 10.1080/01621459.1983.10478005

Examples

```
data(rsales)
um1 <- um(rsales, i = list(1, c(1, 12)), ma = list(1, c(1, 12)), bc = TRUE)
tfm1 <- calendar(um1)
```

CalendarVar *Calendar variables*

Description

'CalendarVar()' creates a set of deterministic regressors to capture calendar effects (trading/working days, length-of-month, leap-year and Easter).

Usage

```
CalendarVar(
  x,
  form = c("dif", "td", "td7", "td6", "wd", "null"),
  ref = 0,
  lom = TRUE,
  lpyear = TRUE,
  easter = FALSE,
  len = 4,
  easter.mon = FALSE,
  n.ahead = 0
)
```

Arguments

x	A ‘ts‘ object used to determine start, length and frequency.
form	Character selecting the set of calendar variables: ‘"dif"‘ (differences wrt reference day), ‘"td"‘ (6 dummies + lom; omits reference), ‘"td7"‘ (7 dummies), ‘"td6"‘ (6 dummies; omits reference), ‘"wd"‘ (weekdays vs weekend), or ‘"null"‘ (no trading-day regressors).
ref	Non-negative integer (0–6) indicating the reference day (0 = Sunday, 1 = Monday, . . . , 6 = Saturday). Ignored unless ‘form‘ needs it.
lom	Logical. If ‘TRUE‘ include a length-of-month regressor.
lpyear	Logical. If ‘TRUE‘ include a leap-year regressor.
easter	Logical. If ‘TRUE‘ include an Easter regressor.
len	Integer duration for Easter effect (days). Typical values: 4–8.
easter.mon	Logical. ‘TRUE‘ if Holy Monday is a public holiday.
n.ahead	Integer. Extra observations to extend the sample (forecast horizon).

Value

An object of class ‘ts‘ or ‘mts‘ with the requested regressors.

References

Bell, W.R. and Hillmer, S.C. (1983) “Modeling time series with calendar variation”, *Journal of the American Statistical Association*, 78, 526–534.

Examples

```
X <- CalendarVar(AirPassengers, form = "wd", easter = TRUE, len = 5)
```

ccf.tfm *Cross-correlation check*

Description

ccf displays ccf between prewhitened inputs and residuals.

Usage

```
ccf.tfm(
  x,
  lag.max = NULL,
  method = c("exact", "cond"),
  envir = parent.frame(),
  ...
)
```

Arguments

x	a tfm object.
lag.max	number of lags.
method	Exact/conditional residuals.
envir	environment in which the function arguments are evaluated.
...	additional arguments.

coef.tfm *Coefficients of a Transfer Function Model*

Description

Extracts the estimated coefficients from a fitted transfer function model of class tfm. This is a method for the generic coef function.

Usage

```
## S3 method for class 'tfm'
coef(object, ...)
```

Arguments

object	An object of class tfm.
...	Further arguments (currently unused).

Value

A named numeric vector with the estimated coefficients of the model, including regression coefficients, transfer function parameters, and noise model parameters.

See Also

tfm

Examples

```
## Not run:
mdl <- tfm(y, xreg = X, noise = um())
coef(mdl)

## End(Not run)
```

---

coef.ucm *Extract coefficients from UCM objects*

Description

Extract coefficients from UCM objects

Usage

```
## S3 method for class 'ucm'
coef(object, ...)
```

Arguments

object	an object of class ucm.
...	currently unused.

Value

A named numeric vector of model coefficients.

coef.um *Coefficients of a univariate model*

Description

coef extracts the "coefficients" from a um object.

Usage

```
## S3 method for class 'um'
coef(object, ...)
```

Arguments

object	a um object.
...	other arguments.

Value

A numeric vector.

cwfact *Cramer-Wold Factorization*

Description

cwfact performs the Cramer-Wold factorization of the generating autocovariance function of a pure moving average (MA) process, expressed as:

$$g(x) = \theta(x)\theta(x^{-1})$$

where

$$g(x) = g_0 + g_1(x + x^{-1}) + \cdots + g_q(x^q + x^{-q})$$

and

$$\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_q x^q$$

Usage

```
cwfact(
  g,
  th = NULL,
  method = c("roots", "wilson", "best"),
  tol = 1e-08,
  iter.max = 500
)
```

Arguments

g	A numeric vector with the autocovariance coefficients c(g0,g1, ..., gq).
th	Optional numeric vector with initial values for the MA coefficients $\theta(x)$.
method	A character string specifying the factorization method to use. Options are "roots" (default) and "wilson".
tol	A numeric tolerance for convergence (only used for method = "wilson"). Default is 1e-8.
iter.max	Maximum number of iterations for the Wilson method. Default is 100.

Details

The factorization can be computed by finding the roots of the polynomial $g(x)$, or using the iterative Wilson (1969) algorithm as implemented by Laurie (1981).

The implementation for method = "laurie" is a custom R adaptation of Algorithm AS 175 from Laurie (1981).

Value

A numeric vector containing the moving average coefficients c(theta_0, ..., theta_q).

References

Wilson, G. T. (1969). Factorization of the covariance generating function of a pure moving average process. *SIAM Journal on Numerical Analysis*, 6(1), 1–7.

Laurie, D. P. (1981). Cramer-Wold Factorization. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 31(1), 86–93.

Examples

```
g <- autocov(um(ma = "1 - 0.8B"), lag.max = 1)
cwfact(g, method = "roots")
cwfact(g, method = "wilson")
```

---

decomp.ssm *Unobserved components decomposition*

Description

Estimates the unobserved components of a time series (trend, seasonal, cycle, stationary, and irregular) based on the structure of an underlying model. The estimation can be carried out through the UCARIMA representation, the state-space model (SSM) form, or via forward/backward forecasts.

Usage

```
## S3 method for class 'ssm'
decomp(mdl, tol = 1e-05, ...)

## S3 method for class 'tfm'
decomp(
  mdl,
  y = NULL,
  method = c("mixed", "forecast", "backcast"),
  envir = NULL,
  ...
)

## S3 method for class 'ucarima'
decomp(mdl, ...)

decomp(mdl, ...)

## S3 method for class 'um'
decomp(
  mdl,
  z = NULL,
 method = c("ucarima", "ucarima0", "ssm", "ssm0", "mixed", "forecast", "backcast"),
  envir = parent.frame(),
  ...
)

## S3 method for class 'um'
decomp(
  mdl,
  z = NULL,
 method = c("ucarima", "ucarima0", "ssm", "ssm0", "mixed", "forecast", "backcast"),
  envir = parent.frame(),
  ...
)
```

Arguments

mdl	An object of class um or tfm, representing a univariate ARIMA or transfer function model.
tol	numeric tolerance for classifying eigenvalues.
...	Additional arguments passed to internal methods.
y	an object of class ts.
method	Character string specifying the decomposition method. Options are: • "ucarima", "ucarima0" – using the UCARIMA representation, without or with the canonical requirement.

- "ssm", "ssm0" – using the state-space model representation, with multiple sources of error (MSOE) or a single source of error (SSOE), respectively.
- "mixed" – combining forward and backward forecasts.
- "forecast", "backcast" – using the forward or backward eventual forecast function. The last three options are deprecated and will be removed in a future release.

envir — environment in which the function arguments are evaluated. If NULL the calling environment of this function will be used.

z — an object of class ts.

Details

The function applies the corresponding internal routines to estimate the components depending on the chosen method. For UCARIMA-based methods, the Wiener–Kolmogorov filter is used. For state-space approaches, a Kalman smoother is applied.

Value

A data.frame with the estimated unobserved components.

Examples

```
Z <- AirPassengers
um1 <- um(Z, i = list(1, c(1, 12)), ma = list(1, c(1, 12)), bc = TRUE)
uc1 <- decomp(um1, method = "ucarima")
```

---

diagchk.ssm *Diagnostic checking*

Description

For objects of class ucarima, this method calls diagchk.um internally to perform diagnostic checking.

diagchk displays tools for diagnostic checking.

Usage

```
## S3 method for class 'ssm'
diagchk(mdl, lag.max = NULL, lags.at = NULL, freq.at = NULL, std = TRUE, ...)

## S3 method for class 'ucarima'
diagchk(mdl, ...)

diagchk(mdl, ...)

## S3 method for class 'um'
```

```
diagchk(
  mdl,
  z = NULL,
  method = c("exact", "cond"),
  lag.max = NULL,
  lags.at = NULL,
  freq.at = NULL,
  std = TRUE,
  envir = NULL,
  ...
)
```

Arguments

mdl	an object of class um.
lag.max	integer; maximum number of lags for ACF/PACF.
lags.at	numeric vector; specific lags in ACF/PACF plots.
freq.at	numeric vector; specific frequencies in (cum) periodogram plot.
std	logical; if TRUE standardized residuals are used.
...	additional arguments.
z	optional, an object of class ts.
method	character; "exact" or "conditional" residuals.
envir	environment in which the function arguments are evaluated. If NULL the calling environment of this function will be used.

Examples

```
# Local level model
b <- 1
C <- as.matrix(1)
ssm1 <- ssm(Nile, b, C, S = diag(c(irr = 15127.7, lvl = 1453.2)))
diagchk(ssm1)
z <- AirPassengers
airl <- um(z, i = list(1, c(1,12)), ma = list(1, c(1,12)), bc = TRUE)
diagchk(airl)
```

---

diagchk.tfm *Diagnostic Checking for Transfer Function Models*

Description

Produces diagnostic plots for residuals of a fitted transfer function model.

Usage

```
## S3 method for class 'tfm'
diagchk(
  mdl,
  y = NULL,
  method = c("exact", "cond"),
  lag.max = NULL,
  lags.at = NULL,
  freq.at = NULL,
  std = TRUE,
  envir = NULL,
  ...
)
```

Arguments

mdl	A fitted tfm object.
y	Optional ts object for alternative output series.
method	Character: "exact" or "cond" for residual calculation.
lag.max	Maximum lag for ACF/PACF plots.
lags.at	Specific lags to display in ACF/PACF.
freq.at	Specific frequencies for cumulative periodogram.
std	Logical. If TRUE, standardizes residuals.
envir	Environment for evaluation. NULL uses calling environment.
...	Additional arguments passed to ide.

Details

Generates five diagnostic plots: time series plot of residuals, histogram, ACF, PACF, and cumulative periodogram. Uses the ide function for plotting.

See Also

tfm, tsdiag.tfm

---

display *Graphs for ARMA models*

Description

display shows graphs characterizing one or a list of ARMA models.

Usage

```
display(um, ...)

## S3 method for class 'um'
display(
  um,
  lag.max = 25,
  n.freq = 501,
  log.spec = FALSE,
  lags.at = NULL,
  graphs = c("acf", "pacf", "spec"),
  byrow = FALSE,
  eq = TRUE,
  cex = 1.25,
  ...
)

## Default S3 method:
display(um, ...)
```

Arguments

um	an object of class um or a list of these objects.
...	additional arguments.
lag.max	number of lags for ACF/PACF.
n.freq	number of frequencies for the spectrum.
log.spec	logical. If TRUE log spectrum is computed.
lags.at	the lags of the ACF/PACF at which tick-marks are to be drawn.
graphs	vector of graphs.
byrow	orientation of the graphs.
eq	logical. If TRUE the model equation is used as title.
cex	double. Font size for equation text.

Examples

```
um1 <- um(ar = "(1 - 0.8B)(1 - 0.8B^12)")
um2 <- um(ma = "(1 - 0.8B)(1 - 0.8B^12)")
display(list(um1, um2))
```

easter *Easter effect*

Description

easter extends the ARIMA model um by including a regression variable to capture the Easter effect.

Usage

```
easter(um, ...)

## S3 method for class 'um'
easter(
  um,
  z = NULL,
  len = 4,
  easter.mon = FALSE,
  n.ahead = 0,
  envir = NULL,
  ...
)
```

Arguments

um	an object of class um.
...	other arguments.
z	a time series.
len	a positive integer specifying the duration of the Easter.
easter.mon	logical. If TRUE Easter Monday is also taken into account.
n.ahead	a positive integer to extend the sample period of the Easter regression variable with n.ahead observations, which could be necessary to forecast the output.
envir	environment in which the function arguments are evaluated. If NULL the calling environment of this function will be used.

Value

An object of class "tfm".

Examples

```
data(rsales)
um1 <- um(rsales, i = list(1, c(1, 12)), ma = list(1, c(1, 12)), bc = TRUE)
tfm1 <- easter(um1)
```

`equation.ucarima` *Equation of ucarima model*

Description

`equation` prints the equation of an object of class um.

Usage

```
## S3 method for class 'ucarima'
equation(x, ...)

equation(x, ...)

## S3 method for class 'um'
equation(
  x,
  unscramble = FALSE,
  digits = 4,
  z = "z",
  a = "a",
  width = NULL,
  ...
)
```

Arguments

`x`	an object of class `um`, a list of these objects or an object of class `ucarima`.
`...`	additional arguments.
`unscramble`	logical. If TRUE, AR, I and MA polynomials are unscrambled.
`digits`	integer. Number of significant digits.
`z`	character. Symbol for time series.
`a`	character. Symbol for error.
`width`	integer. Maximum width for line wrapping. If NULL, uses console width.

Examples

```
equation(um(ar = "(1 - 0.8B)"))
```

factorize *Factorized form of a univariate ARIMA model*

Description

factorize.

Usage

```
factorize(um, ...)

## S3 method for class 'um'
factorize(um, full = TRUE, ...)
```

Arguments

um	an object of class um or a list of these objects.
...	additional arguments.
full	logical value. If TRUE, lag polynomials are completely factorized. Otherwise, they are factored isolating positive real roots and grouping the remaining roots.

Examples

```
factorize(um(ar = "(1 - 0.8B)"))
```

factors *Lag polynomial factorization*

Description

factors extracts the simplifying factors of a polynomial in the lag operator by replacing, if needed, its approximate unit or real roots to exact unit or real roots.

Usage

```
factors(lp, ...)

## S3 method for class 'lagpol'
factors(lp, full = TRUE, tol = 1e-05, expand = FALSE, ...)
```

Arguments

lp	an object of class lagpol.
...	additional arguments.
full	logical value. If TRUE, the lag polynomial is completely factored. Otherwise, it is factored separating positive real roots from the others.
tol	tolerance for nonzero coefficients.
expand	logical value to indicate whether or not the factored lag polynomial must be expanded.

Value

factors returns a list with the simplifying factors of the lag polynomial or the expanded polynomial.

Examples

```
factors( as.lagpol(c(1, rep(0, 11), -1)) )
```

---

fit.ssm *Estimation of the ARIMA model*

Description

fit fits the univariate model to the time series z.

Usage

```
## S3 method for class 'ssm'
fit(
  mdl,
  z = NULL,
  updateSSM,
  param,
  show.iter = FALSE,
  tol = 1e-04,
  method = "BFGS",
  ...
)

fit(mdl, ...)

## S3 method for class 'um'
fit(
  mdl,
  z = NULL,
```

```
  method = c("exact", "cond"),
  optim.method = "BFGS",
  show.iter = FALSE,
  envir = NULL,
  ...
)
```

Arguments

mdl	an object of class um or tfm.
z	a time series.
updateSSM	user function to update the parameters of the SS model. The function must take a model object and a parameter vector as inputs and return an updated model object.
param	a numeric vector of named parameters passed to the updateSSM function.
show.iter	logical value to show or hide the estimates at the different iterations.
tol	numeric. Tolerance to check if a root is close to one.
method	Exact/conditional maximum likelihood.
...	additional arguments for the optim function.
optim.method	the method argument of the optim function.
envir	environment in which the function arguments are evaluated. If NULL the calling environment of this function will be used.

Value

An object of class "ssm" with the estimated parameters.

An object of class "um" with the estimated parameters.

Note

The um function estimates the corresponding ARIMA model when a time series is provided. The fit function is useful to fit a model to several time series, for example, in a Monte Carlo study.

Examples

```
# Predefined local level model
ucm1 <- ucm(Nile, uc = "llm", fit = FALSE)
ucm1 <- fit(ucm1)
ucm1

# User defined local level model
ssm1 <- ssm(Nile, b = 1, C = 1, S = diag(c(1, 0.5)) )
param <- c(irr = var(Nile), lvl = var(diff(Nile)))
updateSSM <- function(mdl, param) {
mdl$S[1,1] <- param[1]
mdl$S[2,2] <- param[2]
mdl
```

```
}
fit(ssm1, updateSSM = updateSSM, param = param)

z <- AirPassengers
airl <- um(i = list(1, c(1, 12)), ma = list(1, c(1, 12)), bc = TRUE)
airl <- fit(airl, z)
```

`fit.tfm` *Fit a Transfer Function Model*

Description

Estimates the parameters of a transfer function model of class `tfm` by (conditional or exact) maximum likelihood.

Usage

```
## S3 method for class 'tfm'
fit(
  mdl,
  y = NULL,
  method = c("exact", "cond"),
  optim.method = "BFGS",
  show.iter = FALSE,
  fit.noise = TRUE,
  envir = NULL,
  ...
)
```

Arguments

`mdl`	An object of class `tfm` created with `tfm`.
`y`	Optional `ts` object containing the output series. If `NULL`, the output stored in `noise` is used.
`method`	Character string specifying likelihood method: "exact" for exact maximum likelihood or "cond" for conditional maximum likelihood. Default is "exact".
`optim.method`	Character. Optimization method passed to `optim`. Default is `"BFGS"`. Other options: "Nelder-Mead", "CG", "L-BFGS-B".
`show.iter`	Logical. If `TRUE`, prints iteration progress of the likelihood optimization.
`fit.noise`	Logical. If `TRUE` (default), the parameters of the noise model are estimated. If `FALSE`, noise parameters are fixed at their current values.
`envir`	Environment in which the function arguments are evaluated. If `NULL`, the calling environment is used.
`...`	Additional arguments.

Value

An updated object of class `tfm` containing fitted parameters, estimated innovation variance, and optimization details.

See Also

`tfm`

Examples

```
## Not run:
data(seriesJ)
Y <- seriesJ$Y - mean(seriesJ$Y)
X <- seriesJ$X - mean(seriesJ$X)
umx <- um(X, ar = 3)
umy <- fit(umx, Y)
tfx <- tfest(Y, X, delay = 3, p = 2, q = 2, um.x = umx, um.y = umy)
tfmy <- tfm(Y, inputs = tfx, noise = um(ar = 2), fit = FALSE)
tfmy_fit <- fit(tfmy)

## End(Not run)
```

`fit.ucarima` *Estimation of UCARIMA models*

Description

Estimates the parameters of a UCARIMA model using maximum likelihood.

Usage

```
## S3 method for class 'ucarima'
fit(mdl, z = NULL, method = "BFGS", show.iter = FALSE, envir = NULL, ...)
```

Arguments

`mdl`	An object of class `ucarima`.
`z`	Optional. A time series object (`ts`) used for estimation if not already included in `mdl`.
`method`	Character string specifying the optimization method passed to `optim`. Default is `"BFGS"`.
`show.iter`	Logical. If `TRUE`, displays parameter estimates during each iteration of the optimization process. Default is `FALSE`.
`envir`	Environment where the estimation is evaluated. If `NULL` (default), uses the parent environment.
`...`	Additional arguments passed to `optim`.

Value

An updated ucarima object with estimated parameters, log-likelihood, and convergence information.

Examples

```
# Define a local level model with trend and irregular components
trend <- um(i = 1, sig2 = c(s2t = 0.5))
irreg <- um(sig2 = c(s2i = 1))

# Create and estimate the UCARIMA model
uca <- ucarima(ucm = list(trend = trend, irreg = irreg))
uca_fitted <- fit(uca, Nile)
```

---

ide *Identification plots*

Description

ide displays graphs useful to identify a tentative ARIMA model for a time series.

Usage

```
ide(
  Y,
  transf = list(),
  order.polreg = 0,
  lag.max = NULL,
  lags.at = NULL,
  freq.at = NULL,
  wn.bands = TRUE,
  graphs = c("plot", "acf", "pacf"),
  set.layout = TRUE,
  byrow = TRUE,
  main = "",
  plot.abline.args = NULL,
  plot.points.args = NULL,
  envir = NULL,
  ...
)
```

Arguments

Y	Univariate or multivariate time series.
transf	Data transformations, list(bc = F, d = 0, D = 0, S = F), where bc is the Box-Cox logarithmic transformation, d and D are the number of nonseasonal and seasonal differences, and S is the annual sum operator.

order.polreg	an integer indicating the order of a polynomial trend.
lag.max	number of autocorrelations.
lags.at	the lags of the ACF/PACF at which tick-marks are to be drawn.
freq.at	the frequencies of the (cum) periodogram at at which tick-marks are to be drawn.
wn.bands	logical. If TRUE confidence intervals for sample autocorrelations are computed assuming a white noise series.
graphs	graphs to be shown: plot, hist, acf, pacf, pgram, cpgram (cummulative periodogram), rm (range-median).
set.layout	logical. If TRUE the layout is set by the function, otherwise it is set by the user.
byrow	logical. If TRUE the layout is filled by rows, otherwise it is filled by columns.
main	title of the graph.
plot.abline.args	Add straight lines to time series plot.
plot.points.args	Add points to time series plot.
envir	environment in which the function arguments are evaluated. If NULL the calling environment of this function will be used.
...	additional arguments.

Examples

```
Y <- AirPassengers
ide(Y, graphs = c("plot", "rm"))
ide(Y, transf = list(list(bc = TRUE, S = TRUE), list(bc = TRUE, d = 1, D = 1)))
```

---

init_kf *Initialization of Kalman filter*

Description

init_kf computes the starting values x0 and P0 by generalized least squares using the first n observations.

Usage

```
init_kf(mdl, z = NULL, n = 0)
```

Arguments

mdl	an object of class ssm.
z	optional time series if it differs from model series.
n	integer, number of observations used to estimate the initial conditions. If n < d (dimension of state vector), it defaults to length(z).

Value

A list with components:

x0	initial state vector estimate
P0	covariance matrix of the initial state estimate

`intervention.tfm` *Intervention analysis/Outlier treatment*

Description

`intervention` estimates the effect of a intervention at a known time.

Usage

```
## S3 method for class 'tfm'
intervention(
  mdl,
  y = NULL,
  type,
  time,
  n.ahead = 0,
  envir = parent.frame(),
  ...
)

intervention(mdl, ...)

## S3 method for class 'um'
intervention(
  mdl,
  y = NULL,
  type,
  time,
  n.ahead = 0,
  envir = parent.frame(),
  ...
)
```

Arguments

mdl	an object of class `um` or `tfm`.
y	a "ts" object, optional.
type	the type intervention (pulse, step, ramp) or the type of outlier (AO, LS, TC, IO).
time	the date of the intervention, in format c(year, season).

n.ahead	a positive integer to extend the sample period of the intervention variable with n.ahead observations, which could be necessary to forecast the output.
envir	the environment in which to look for the time series z when it is passed as a character string.
...	additional arguments.

Value

an object of class "tfm" or a table.

`InterventionVar` *Intervention variables*

Description

'InterventionVar()' creates pulse, step, or ramp variables at a given date.

Usage

```
InterventionVar(Y, date, type = c("P", "S", "R"), n.ahead = 0)
```

Arguments

Y	A 'ts' object used to determine start, length and frequency.
date	Either a single positive index within the sample, or a vector 'c(year, month)' for monthly series. For non-seasonal series, 'c(year)' is also accepted.
type	One of '"P"' (pulse), '"S"' (step), or '"R"' (ramp = cumulative step).
n.ahead	Integer. Extra observations to extend the sample.

Value

A 'ts' intervention variable.

References

Box, G.E.P. and Tiao, G.C. (1975) "Intervention analysis with applications to economic and environmental problems", *JASA*, 70(349), 70–79.

Examples

```
# Pulse at March 1958:
P <- InterventionVar(AirPassengers, date = c(1958, 3), type = "P")
# Or by index within the extended sample (here no extension):
P2 <- InterventionVar(AirPassengers, date = 123, type = "P")
```

inv *Inverse of a lag polynomial*

Description

inv inverts a lag polynomial until the indicated lag.

Usage

```
inv(lp, ...)

## S3 method for class 'lagpol'
inv(lp, lag.max = 10, ...)
```

Arguments

lp	an object of class lagpol.
...	additional arguments.
lag.max	largest order of the inverse lag polynomial.

Value

inv returns a numeric vector with the coefficients of the inverse lag polynomial truncated at lag.max.

Examples

```
inv(as.lagpol(c(1, 1.2, -0.8)))
```

irf *Impulse response function*

Description

Computes and plots the impulse response function (IRF) or step response function (SRF) of a transfer function.

Usage

```
irf(tf, lag.max = 10, cum = FALSE, plot = TRUE)
```

Arguments

tf	An object of class "tf".
lag.max	Integer. Maximum number of lags to compute (default 10).
cum	Logical. If TRUE computes step response function (cumulative), if FALSE computes impulse response function (default FALSE).
plot	Logical. If TRUE creates a plot, if FALSE returns values only (default TRUE).

Value

If plot = FALSE, a named numeric vector with IRF/SRF values. If plot = TRUE, creates a plot and returns nothing (invisibly).

Examples

```
# Create transfer function
x <- rep(0, 100); x[50] <- 1
tfx <- tf(x, w0 = 0.8, ar = "(1 - 0.5B)")

# Plot impulse response function
irf(tfx, lag.max = 15)

# Get step response values without plot
srf_values <- irf(tfx, lag.max = 10, cum = TRUE, plot = FALSE)
```

---

kf *Kalman filter for SS models*

Description

kf computes the innovations and the conditional states with the Kalman filter algorithm.

Usage

```
kf(mdl, z = NULL, x0 = NULL, P0 = NULL, filtered = FALSE, ...)
```

Arguments

mdl	an object of class ssm.
z	time series to be filtered when it differs from the model series.
x0	initial state vector.
P0	covariance matrix of x1.
filtered	logical. If TRUE, the filtered states x_{t|t} and their covariance matrices P_{t|t} are returned. Otherwise, the forecasted states x_{t|t-1} and thier covariance matrices P_{t|t-1} are returned.
...	additional arguments.

Value

A list with the innovations, the conditional states and their covariance matrices.

---

ks *Kalman smoother for SS models*

Description

`ks` computes smoothed states and their covariance matrices.

Usage

```
ks(mdl, x0 = NULL, P0 = NULL)
```

Arguments

`mdl`	an object of class `ssm`.
`x0`	initial state vector.
`P0`	covariance matrix of x0.

---

lagpol *Lag polynomials*

Description

`lagpol` creates a lag polynomial of the form:

$$(1 - coef_1 B^s - ... - coef_d B^{sd})^p$$

This class of lag polynomials is defined by:

- the base lag polynomial $1 - coef_1 B^s - ... - coef_d B^{sd}$,
- the exponent ‘p‘ of the base lag polynomial (default is ‘p = 1‘),
- the spacing parameter ‘s‘ in sparse lag polynomials (default is ‘s = 1‘),
- the vector of ‘d‘ coefficients ‘c(coef_1, ..., coef_d)‘, which can be mathematical expresions dependent on ‘k‘ parameters ‘c(param_1, ..., param_k)‘.

Usage

```
lagpol(param = NULL, s = 1, p = 1, lags = NULL, coef = NULL)
```

Arguments

param	a vector/list of named parameters. These parameters can be used within the coefficient expressions.
s	an integer specifying the lag spacing or seasonal period.
p	an integer specifying the exponent applied to the base lag polynomial.
lags	an optional vector of lags for sparse polynomials. If NULL, lags are determined by s.
coef	an optional vector of mathematical expressions defining the coefficients of the lag polynomial. If NULL, the names in param are used.

Value

lagpol An object of class 'lagpol' with the following components:

coef vector of coefficients c(coef_1, ..., coef_p) provided to create the lag polynomial.

pol base lag polynomial vector: $1 - \text{coef}_1 B^s - \cdots - \text{coef}_d B^{sd}$.

Pol lag polynomial raised to the power 'p'. If 'p = 1', this equals 'pol'.

Examples

```
# Simple AR(1) lag polynomial: 1 - 0.8B
lagpol(param = c(phi = 0.8))

# AR(2) lag polynomial with seasonal lag s = 4: 1 - 1.2B^4 + 0.6B^8
lagpol(param = c(phi1 = 1.2, phi2 = -0.6), s = 4)

# Integration operator squared: (1 - B)^2 = 1 - 2B + B^2
lagpol(param = c(delta = 1), p = 2)

# Lag polynomial using explicit coefficients
lagpol(coef = c("1"), p = 2)  # (1 - B)^2 = 1 - 2B + B^2

# Custom coefficients defined by mathematical expressions
lagpol(param = c(theta = 0.8), coef = c("2*cos(pi/6)*sqrt(theta)", "-theta"))
```

---

lagpol0 *Create lag polynomial objects*

Description

'lagpol0' is a flexible constructor for lagpol objects. It accepts multiple input formats: polynomial orders (d, s, p), literal equations (e.g., $1 - 0.8B$), or seasonal specifications for special lag polynomials like $AR/I/MA(s-1)$.

Usage

```
lagpol0(op, type, envir = parent.frame())
```

Arguments

op	Polynomial specification in one of these formats: • Numeric vector $c(d, s, p)$ specifying polynomial degree, lag step (default 1), and exponent (default 1) • Character equation (e.g., "1 - 0.8B") • Seasonal period or frequency (e.g., "12" or "1/12") • List combining multiple specifications
type	Operator type: "ar" (autoregressive), "ma" (moving average), or "i" (integration).
envir	Environment for argument evaluation. Defaults to parent frame.

Value

List of lagpol objects.

See Also

lagpol

Examples

```
# AR(1) polynomial
lagpol0(op = 1, type = "ar")

# From literal equation
lagpol0(op = "1 - 0.8B", type = "ar")

# Multiple polynomials at once
lagpol0(op = list(1, "1 - 0.5B"), type = "ma")

# Seasonal polynomial
lagpol0(op = "12", type = "ar")

# Custom orders with seasonal component
lagpol0(op = c(2, 12, 1), type = "ar")
```

logLik.ssm *Log-likelihood of a SS model*

Description

logLik.ssm computes the exact or conditional log-likelihood of a state space model.

Usage

```
## S3 method for class 'ssm'
logLik(object, method = c("exact", "cond"), ...)
```

Arguments

object	an object of class ssm.
method	character. Either "exact" or "conditional" maximum likelihood.
...	additional parameters.

Value

The log-likelihood value.

Examples

```
# Local level model
b <- 1
C <- as.matrix(1)
ssm1 <- ssm(Nile, b, C, S = diag(c(irr = 15127.7, lvl = 1453.2)))
logLik(ssm1)
```

logLik.tfm *Log-Likelihood of Transfer Function Model*

Description

Computes the log-likelihood for a fitted transfer function model.

Usage

```
## S3 method for class 'tfm'
logLik(
  object,
  y = NULL,
  method = c("exact", "cond"),
  envir = parent.frame(),
  ...
)
```

Arguments

object	A fitted tfm object.
y	Optional ts object for alternative output series.
method	Character: "exact" estimates presample values; "cond" fixes presample values at zero.
envir	Environment for evaluation.
...	Additional arguments (currently unused).

Value

Numeric value of the log-likelihood.

See Also

tfm, fit.tfm, AIC.tfm

---

logLik.um *Log-likelihood of an ARIMA model*

Description

logLik computes the exact or conditional log-likelihood of object of the class um.

Usage

```
## S3 method for class 'um'
logLik(object, z = NULL, method = c("exact", "cond"), ...)
```

Arguments

object	an object of class um.
z	an object of class ts.
method	exact or conditional.
...	additional arguments.

Value

The exact or conditional log-likelihood.

`modify.tfm` *Modifying a TF or an ARIMA model*

Description

`modify` modifies an object of class `um` or `tfm` by adding and/or removing lag polynomials.

Usage

```
## S3 method for class 'tfm'
modify(
  mdl,
  ar = NULL,
  i = NULL,
  ma = NULL,
  mu = NULL,
  sig2 = NULL,
  bc = NULL,
  ...
)

modify(mdl, ...)

## S3 method for class 'um'
modify(
  mdl,
  ar = NULL,
  i = NULL,
  ma = NULL,
  mu = NULL,
  sig2 = NULL,
  bc = NULL,
  ...
)
```

Arguments

`mdl`	an object of class `um` or `tfm`.
`ar`	list of stationary AR lag polynomials.
`i`	list of nonstationary AR (I) polynomials.
`ma`	list of MA polynomials.
`mu`	mean of the stationary time series.
`sig2`	variance of the error.
`bc`	logical. If TRUE logs are taken.
`...`	additional arguments.

Value

An object of class `um` or `um`.

Examples

```
um1 <- um(ar = "(1 - 0.8B)")
um2 <- modify(um1, ar = list(0, "(1 - 0.9B)"), ma = "(1 - 0.5B)")
```

`nabla.ucarima` *Unscramble I polynomial*

Description

`nabla` multiplies the I polynomials of an object of the `um` class.

Usage

```
## S3 method for class 'ucarima'
nabla(x, i = NULL, lp = TRUE, ...)

nabla(x, ...)

## S3 method for class 'um'
nabla(x, ...)
```

Arguments

`x`	an object of class `um`.
`i`	integer. Omit this component model.
`lp`	logical indicating the type of return: `lagpol` object or numeric vector.
`...`	additional arguments.

Value

A numeric vector `c(1, a1, ..., ad)`

Note

This function returns the member variable `um$nabla`.

Examples

```
um1 <- um(i = "(1 - B)(1 - B^12)")
nabla(um1)
```

noise.ssm *Extract Noise Component from Transfer Function Model*

Description

Computes the noise series (output minus fitted signal) from a transfer function model.

Usage

```
## S3 method for class 'ssm'
noise(mdl, diff = TRUE, exp = FALSE, ...)

noise(mdl, ...)

## S3 method for class 'tfm'
noise(mdl, y = NULL, diff = TRUE, exp = FALSE, envir = parent.frame(), ...)
```

Arguments

mdl	A tfm object.
diff	Logical. If TRUE (default), returns differenced noise series (stationary). If FALSE, returns noise in original scale.
exp	Logical. If TRUE, applies exponential transformation (inverse of log). Only relevant when diff = FALSE and Box-Cox transformation was used (bc = TRUE).
...	Additional arguments.
y	Optional ts object for alternative output series.
envir	Environment for evaluation. NULL uses calling environment.

Details

The noise represents the component of the output not explained by the transfer functions and exogenous regressors. When diff = TRUE, the differencing operator from the noise model is applied, resulting in a stationary series suitable for ARMA modeling.

Value

A ts object containing the noise series, computed as output minus all transfer function and regressor effects.

See Also

signal.tfm, residuals.tfm, tfm

outlierDates *Outlier dates*

Description

`outlierDates` shows the indeces and dates of outliers.

Usage

```
outlierDates(x, c = 3)
```

Arguments

`x`	an `ts` object.
`c`	critical value to determine whether or not an observation is an outlier.

Value

A table with the indices, dates and z-scores of the outliers.

outliers.ssm *Outliers detection at known/unknown dates*

Description

`outliers` performs a detection of four types of anomalies (AO, TC, LS and IO) in a time series described by an ARIMA model. If the dates of the outliers are unknown, an iterative detection process like that proposed by Chen and Liu (1993) is conducted.

Usage

```
## S3 method for class 'ssm'
outliers(
  mdl,
  types = c("AO", "LS", "TC"),
  dates = NULL,
  c = 3,
  calendar = FALSE,
  easter = FALSE,
  resid = c("exact", "cond"),
  n.ahead = 0,
  p.value = 1,
  ...
)

## S3 method for class 'tfm'
```

```
outliers(
  mdl,
  y = NULL,
  types = c("AO", "LS", "TC", "IO"),
  dates = NULL,
  c = 3,
  calendar = FALSE,
  easter = FALSE,
  resid = c("exact", "cond"),
  n.ahead = 0,
  p.value = 1,
  tc.fix = TRUE,
  envir = parent.frame(),
  ...
)

## S3 method for class 'ucarima'
outliers(
  mdl,
  types = c("AO", "LS", "TC"),
  dates = NULL,
  c = 3,
  calendar = FALSE,
  easter = FALSE,
  resid = c("exact", "cond"),
  n.ahead = 0,
  p.value = 1,
  ...
)

outliers(mdl, ...)

## S3 method for class 'um'
outliers(
  mdl,
  y = NULL,
  types = c("AO", "LS", "TC", "IO"),
  dates = NULL,
  c = 3,
  calendar = FALSE,
  easter = FALSE,
  resid = c("exact", "cond"),
  n.ahead = 0,
  p.value = 1,
  tc.fix = TRUE,
  envir = NULL,
  ...
)
```

Arguments

mdl	an object of class um or tfm.
types	a vector with the initials of the outliers to be detected, c("AO", "LS", "TC", "IO").
dates	a list of dates c(year, season). If dates = NULL, an iterative detection process is conducted.
c	a positive constant to compare the z-ratio of the effect of an observation and decide whether or not it is an outlier. This argument is only used when dates = NULL.
calendar	logical; if true, calendar effects are also estimated.
easter	logical; if true, Easter effect is also estimated.
resid	type of residuals (exact or conditional) used to identify outliers.
n.ahead	a positive integer to extend the sample period of the intervention variables with n.ahead observations, which could be necessary to forecast the output.
p.value	estimates with a p-value greater than p.value are omitted.
...	other arguments.
y	an object of class ts, optional.
tc.fix	a logical value indicating if the AR coefficient in the transfer function of the TC is estimated or fix.
envir	environment in which the function arguments are evaluated. If NULL the calling environment of this function will be used.

Value

an object of class "tfm" or a table.

Examples

```
data(rsales)
um1 <- um(rsales, i = list(1, c(1, 12)), ma = list(1, c(1, 12)), bc = TRUE)
outliers(um1)
```

---

output *Output of a transfer function or a transfer function model*

Description

output filters the input using the transfer function.

Usage

```
output(object, ...)

## S3 method for class 'tf'
output(object, ...)
```

Arguments

object	an object of the S3 class "tf".
...	additional arguments.

Value

A "ts" object

---

pccf *Prewhitened cross correlation function*

Description

pccf displays cross correlation function between input and output after prewhitening both through a univariate model.

Usage

```
pccf(
  x,
  y,
  um.x = NULL,
  um.y = NULL,
  lag.max = NULL,
  plot = TRUE,
  envir = NULL,
  main = NULL,
  nu.weights = FALSE,
  ...
)
```

Arguments

x	input, a 'ts' object or a numeric vector.
y	output, a 'ts' object or a numeric vector.
um.x	univariate model for input.
um.y	univariate model for output.
lag.max	number of lags, integer.
plot	logical value to indicate if the ccf graph must be graphed or computed.
envir	environment in which the function arguments are evaluated. If NULL the calling environment of this function will be used.
main	title of the graph.
nu.weights	logical. If TRUE the coefficients of the IRF are computed instead of the cross-correlations.
...	additional arguments.

Value

The estimated cross correlations are displayed in a graph or returned into a numeric vector.

`phi.ucarima` *Unscramble AR polynomial*

Description

`phi` multiplies the AR polynomials of an object of the `um` class.

Usage

```
## S3 method for class 'ucarima'
phi(x, i = NULL, ar = TRUE, lp = TRUE, ...)

phi(x, ...)

## S3 method for class 'um'
phi(x, ...)
```

Arguments

`x`	an object of class `um`.
`i`	integer. Omit this component model.
`ar`	logical. If TRUE, the common AR polynomial is included.
`lp`	logical indicating the type of return: `lagpol` object or numeric vector.
`...`	additional arguments.

Value

A numeric vector `c(1, a1, ..., ad)`

Note

This function returns the member variable `um$phi`.

Examples

```
um1 <- um(ar = "(1 - 0.8B)(1 - 0.5B)")
phi(um1)
```

pi.weights *Pi weights of an AR(I)MA model*

Description

`pi.weights` computes the pi-weights of an AR(I)MA model.

Usage

```
pi.weights(um, ...)

## S3 method for class 'um'
pi.weights(um, lag.max = 10, var.pi = FALSE, ...)
```

Arguments

`um`	an object of class `um`.
`...`	additional arguments.
`lag.max`	largest AR(Inf) coefficient required.
`var.pi`	logical. If TRUE (FALSE), the I polynomials is considered (ignored).

Value

A numeric vector.

Examples

```
um1 <- um(i = "(1 - B)(1 - B^12)", ma = "(1 - 0.8B)(1 - 0.8B^12)")
pi.weights(um1, var.pi = TRUE)
```

polyderivEvalR *Evaluate the k-th derivative of a polynomial at point z*

Description

Evaluate the k-th derivative of a polynomial at point z

Usage

```
polyderivEvalR(pol, z, k = 0)
```

Arguments

pol	Numeric vector of polynomial coefficients in ascending order (pol[1] = constant term, pol[2] = coefficient of z, etc.)
z	Numeric value where the polynomial (or its derivative) is evaluated.
k	Integer. Derivative order (0 = original polynomial).

Value

Numeric value of the k-th derivative of P(z).

Examples

```
pol <- c(1, 2, 3, 4)  # P(z) = 1 + 2z + 3z² + 4z³
polyderivEvalR(pol, 2, 0)  # 49
polyderivEvalR(pol, 2, 1)  # 62
```

---

predict.ssm *Predict method for state space models*

Description

`predict.ssm` generates forecasts from fitted state space models by converting to univariate ARIMA or transfer function models and using their prediction methods.

Usage

```
## S3 method for class 'ssm'
predict(object, ...)
```

Arguments

object	an object of class `ssm`.
...	additional arguments passed to `predict.um` (for models without regressors) or `predict.tfm` (for models with regressors). See their documentation for available options such as forecast horizon, confidence levels, etc.

Value

An object of class `predict.um` or `predict.tfm`.

See Also

`predict.um`, `predict.tfm`

predict.tf *Predict transfer function*

Description

Predict transfer function

Usage

```
## S3 method for class 'tf'
predict(object, n.ahead, ...)
```

Arguments

object	Transfer function object.
n.ahead	Periods to predict.
...	Unused.

Value

Point forecast for input.

predict.tfm *Forecast Transfer Function Model*

Description

Computes point forecasts and prediction intervals for transfer function models.

Usage

```
## S3 method for class 'tfm'
predict(
  object,
  newdata = NULL,
  y = NULL,
  ori = NULL,
  n.ahead = NULL,
  level = 0.95,
  i = NULL,
  envir = NULL,
  ...
)
```

Arguments

object	A fitted tfm object.
newdata	Optional matrix or vector of future values for exogenous regressors and inputs. Rows correspond to forecast horizon, columns to predictors.
y	Optional ts object for alternative output series.
ori	Forecast origin (observation index). Default is last observation.
n.ahead	Number of steps ahead to forecast. Default is series frequency.
level	Confidence level(s) for prediction intervals (0-1). Default is 0.95. Can be a vector for multiple intervals.
i	Optional differencing operator (lagpol) to apply before forecasting.
envir	Environment for evaluation. NULL uses calling environment.
...	Additional arguments (currently unused).

Details

Future values for transfer function inputs can be provided in three ways: (1) extending input series beyond output length, (2) automatic forecasting from associated um models, or (3) via the newdata argument.

If Box-Cox transformation was used, forecasts are back-transformed and intervals adjusted accordingly.

Value

Object of class predict.tfm containing:

z	Complete series including forecasts
rmse	Root mean square error for each forecast
low, upp	Lower and upper prediction interval bounds (matrices)
level	Confidence level(s) used
dates	Time points for all observations
ori, ori.date	Forecast origin (index and date)
n.ahead	Number of forecasts

See Also

tfm, fit.tfm

predict.um *Forecasts from an ARIMA model*

Description

predict computes point and interval predictions for a time series from models of class um.

Usage

```
## S3 method for class 'um'
predict(
  object,
  z = NULL,
  ori = NULL,
  n.ahead = 1,
  level = 0.95,
  i = NULL,
  envir = NULL,
  ...
)
```

Arguments

object	an object of class um.
z	an object of class ts.
ori	the origin of prediction. By default, it is the last observation.
n.ahead	number of steps ahead.
level	confidence level.
i	transformation of the series z to be forecasted. It is a lagpol as those of a um object.
envir	environment in which the function arguments are evaluated. If NULL the calling environment of this function will be used.
...	additional arguments.

Value

An object of class "predict.um".

Examples

```
Z <- AirPassengers
um1 <- um(Z, i = list(1, c(1, 12)), ma = list(1, c(1, 12)), bc = TRUE)
p <- predict(um1, n.ahead = 12)
p
plot(p, n.back = 60)
```

print.lagpol *Print Method for Lag Polynomial Objects*

Description

Prints objects of class `lagpol`.

Usage

```
## S3 method for class 'lagpol'
print(x, digits = 2, width = getOption("width"), ...)
```

Arguments

`x`	An object of class `lagpol`.
`digits`	Integer. Number of significant digits to display. Default is 2.
`width`	Integer. Maximum number of characters per line. Default is the console width.
`...`	Additional arguments (currently unused).

Value

Invisibly returns the `lagpol` object.

print.ssm *Print method for* `ssm` *objects*

Description

Print method for `ssm` objects

Usage

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

Arguments

`x`	an object of class `ssm`.
`...`	further arguments passed to or from other methods.

Value

Invisibly returns `NULL`.

print.summary.ssm *Print summary of fitted state space model*

Description

print.summary.ssm prints a detailed summary of the results of the estimation of a ssm object.

Usage

```
## S3 method for class 'summary.ssm'
print(x, stats = TRUE, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

x	an object of class summary.ssm.
stats	logical. If TRUE, additional diagnostic statistics are printed.
digits	integer. Number of significant digits to display for numeric values.
...	additional arguments.

Value

Invisible NULL.

print.summary.tfm *Print Summary of Transfer Function Model*

Description

Print method for objects of class summary.tfm.

Usage

```
## S3 method for class 'summary.tfm'
print(
  x,
  stats = TRUE,
  short = FALSE,
  digits = max(3L, getOption("digits") - 3L),
  ...
)
```

Arguments

x	A summary.tfm object.
stats	Logical. If TRUE, prints diagnostic statistics.
short	Logical. If TRUE, prints abbreviated output.
digits	Number of significant digits.
...	Additional arguments.

See Also

summary.tfm

---

print.summary.um *Print Summary of Univariate Model*

Description

Print method for objects of class summary.um.

Usage

```
## S3 method for class 'summary.um'
print(
  x,
  stats = TRUE,
  short = FALSE,
  digits = max(3L, getOption("digits") - 3L),
  ...
)
```

Arguments

x	A summary.um object.
stats	Logical. If TRUE, prints diagnostic statistics.
short	Logical. If TRUE, prints abbreviated output.
digits	Number of significant digits.
...	Additional arguments.

See Also

summary.tfm

print.tf *Print method for transfer function objects*

Description

Print method for transfer function objects

Print ucarima models

Print univariate models

Print univariate models

Usage

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

## S3 method for class 'ucarima'
print(x, ...)

## S3 method for class 'predict.um'
print(x, rows = NULL, ...)

## S3 method for class 'um'
print(x, arima = FALSE, ...)
```

Arguments

x	An object of class um.
...	Additional arguments.
rows	integer. Number of rows printed.
arima	logical. If TRUE, lag ARIMA polynomials are printed.

print.tfm *Print Transfer Function Model*

Description

Print method for objects of class tfm.

Usage

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

Arguments

x	A tfm object.
...	Additional arguments passed to summary.tfm and print.summary.tfm.

Details

Prints a summary of the transfer function model by calling summary(x) with short = TRUE.

See Also

tfm, summary.tfm

---

print.uc *Print method for unobserved components*

Description

Print method for unobserved components

Usage

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

Arguments

x	an object of class uc.
...	currently unused.

Value

Invisibly returns NULL.

---

printLagpol *Print non-normalized polynomial as a lag polynomial*

Description

Prints a non-normalized polynomial as a lagpol object, preserving the original coefficients.

Usage

```
printLagpol(pol, digits = 2, width = getOption("width"))
```

Arguments

pol	Numeric vector with the coefficients of a non-normalized polynomial. The first element can be any numeric value, not necessarily 1.
digits	Integer. Number of significant digits to display. Default is 2.
width	Integer. Maximum number of characters per line. Default is the console width.

Value

Invisibly returns the input vector.

---

printLagpolList *Prints a list of* lagpol *objects.*

Description

Prints a list of lagpol objects.

Usage

```
printLagpolList(llp, digits = 2, width = getOption("width"))
```

Arguments

llp	A list of lagpol objects.
digits	Integer. Number of significant digits to display. Default is 2.
width	Integer. Maximum number of characters per line. Default is the console width.

Value

Invisibly returns the list of lagpol objects.

---

psi.weights *Psi weights of an AR(I)MA model*

Description

psi computes the psi-weights of an AR(I)MA model.

Usage

```
psi.weights(um, ...)

## S3 method for class 'um'
psi.weights(um, lag.max = 10, var.psi = FALSE, ...)
```

Arguments

um	an object of class um.
...	additional arguments.
lag.max	Largest MA(Inf) coefficient required.
var.psi	logical. If TRUE the I polynomials is also inverted. If FALSE it is ignored.

Value

A numeric vector.

Examples

```
um1 <- um(i = "(1 - B)(1 - B^12)", ma = "(1 - 0.8B)(1 - 0.8B^12)")
psi.weights(um1)
psi.weights(um1, var.psi = TRUE)
```

---

residuals.ssm *Residuals of fitted state space models*

Description

residuals.ssm generates the residuals of a fitted ssm object.

Usage

```
## S3 method for class 'ssm'
residuals(object, method = c("exact", "cond"), ...)
```

Arguments

object	an object of class ssm.
method	character. Either "exact" or "conditional" residuals.
...	additional arguments.

Details

These residuals are calculated by first converting the ssm object to a um object and then using the residuals.um function.

Value

A ts object containing the residuals of the SSM.

Examples

```
# Local level model
b <- 1
C <- as.matrix(1)
ssm1 <- ssm(Nile, b, C, S = diag(c(irr = 15127.7, lvl = 1453.2)))
u <-residuals(ssm1)
```

`residuals.tfm` *Extract Residuals from Transfer Function Model*

Description

Computes exact or conditional residuals from a fitted transfer function model.

Usage

```
## S3 method for class 'tfm'
residuals(
  object,
  y = NULL,
  method = c("exact", "cond"),
  envir = parent.frame(),
  ...
)
```

Arguments

`object`	A fitted `tfm` object.
`y`	Optional `ts` object for alternative output series.
`method`	Character: "exact" estimates presample values to compute residuals; "cond" fixes presample values at zero.
`envir`	Environment for evaluation. NULL uses calling environment.
`...`	Currently unused.

Value

A `ts` object containing model residuals with the same time series attributes as the output series.

residuals.ucarima *Residuals of fitted UCARIMA models*

Description

`residuals.ucarima` generates the residuals of a fitted `ucarima` object.

Usage

```
## S3 method for class 'ucarima'
residuals(object, method = c("exact", "cond"), ...)
```

Arguments

`object`	an object of class `ucarima`.
`method`	character. Either "exact" or "conditional" residuals.
`...`	additional arguments.

Details

These residuals are calculated by first converting the `ucarima` object to a `um` object and then using the `residuals.um` function.

Value

A `ts` object containing the residuals of the SSM.

residuals.um *Residuals of the ARIMA model*

Description

`residuals` computes the exact or conditional residuals.

Usage

```
## S3 method for class 'um'
residuals(object, z = NULL, method = c("exact", "cond"), envir = NULL, ...)
```

Arguments

`object`	an object of class `um`.
`z`	an object of class `ts`.
`method`	exact/conditional residuals.
`envir`	environment in which the function arguments are evaluated. If NULL the calling environment of this function will be used.
`...`	additional arguments.

Value

An object of class um.

Examples

```
z <- AirPassengers
airl <- um(z, i = list(1, c(1, 12)), ma = list(1, c(1, 12)), bc = TRUE)
r <- residuals(airl)
summary(r)
```

---

roots *Roots of lag polynomials*

Description

roots computes the roots of lag polynomials from polynomial objects and time series models that contain lag polynomials as components.

Usage

```
roots(x, ...)

## Default S3 method:
roots(x, ...)

## S3 method for class 'lagpol'
roots(x, table = TRUE, tol = 1e-05, ...)

## S3 method for class 'tf'
roots(x, opr = c("arma", "ar", "ma"), ...)

## S3 method for class 'um'
roots(x, opr = c("arma", "ar", "ma", "i", "arima"), ...)
```

Arguments

x	A model object containing lag polynomials ("um", "tfm") or a lag polynomial object ("lagpol").
...	Additional arguments passed to methods.
table	Logical. If TRUE returns detailed table, if FALSE complex vector.
tol	Tolerance for identifying distinct roots.
opr	character. Operators for which roots are computed. Options: "arma", "arma", "ar", "ma", "i" or "arima".

Value

Returns a summary table with the roots of each lagpol.

Examples

```
roots(c(1, 1.2, -0.8))
um1 <- um(ar = "(1 - 0.8B)(1 - 0.8B^12)")
roots(um1)
```

---

roots2lagpol *Lag polynomial from roots*

Description

`roots2lagpol` creates a lag polynomial from its roots.

Usage

```
roots2lagpol(x, lp = TRUE, ...)
```

Arguments

`x`	a vector of real and/or complex roots.
`lp`	logical. If TRUE, a object of class `lagpol` is returned; otherwise a numeric vector is returned.
`...`	additional arguments.

Value

A lag polynomial or a numeric vector.

Examples

```
roots2lagpol(polyroot(c(1, -1)))
roots2lagpol(polyroot(c(1, -1, 1)))
```

---

rsales *Retail Sales of Variety Stores (U.S. Bureau of the Census)*

Description

156 monthly observations from January 1967 to December 1979.

Usage

```
rsales
```

Format

An object of class ts of length 156.

References

Chen, C. and Liu, L. (1993) Joint Estimation of Model Parameters and Outlier Effects in Time Series, Journal of the American Statistical Association, Vol. 88, No. 421, pp. 284-297

S *Annual (rolling) sum*

Description

Computes rolling sum over one year for monthly/quarterly data.

Usage

```
S(x, extend = TRUE)
```

Arguments

x	A ts object.
extend	If TRUE, pads result with NAs to match original length. Default is TRUE.

Value

Time series of annual sums with same frequency as input.

---

sdummies *Seasonal dummies*

Description

'sdummies()' creates a full set of seasonal dummies (reference-coded).

Usage

```
sdummies(Y, ref = 1, constant = FALSE, n.ahead = 0)
```

Arguments

Y	A seasonal 'ts' object.
ref	Reference season (1..frequency).
constant	Logical. If 'TRUE', include an intercept column.
n.ahead	Integer. Extra observations to extend the sample.

Value

A ‘ts‘ matrix with seasonal dummies (and intercept if requested).

Examples

```
D <- sdummies(AirPassengers, ref = 1, constant = TRUE, n.ahead = 24)
```

`seasadj` *Seasonal adjustment*

Description

`seasadj` removes the seasonal component of time series.

Usage

```
seasadj(mdl, ...)

## S3 method for class 'um'
seasadj(
  mdl,
  z = NULL,
 method = c("ucarima", "ucarima0", "ssm", "ssm0", "mixed", "forecast", "backcast"),
  envir = parent.frame(),
  ...
)
```

Arguments

`mdl`	An object of class `um` or `tfm`, representing a univariate ARIMA or transfer function model.
`...`	Additional arguments passed to internal methods.
`z`	an object of class `ts`.
`method`	Character string specifying the decomposition method. Options are: • `"ucarima"`, `"ucarima0"` – using the UCARIMA representation, without or with the canonical requirement. • `"ssm"`, `"ssm0"` – using the state-space model representation, with multiple sources of error (MSOE) or a single source of error (SSOE), respectively. • `"mixed"` – combining forward and backward forecasts. • `"forecast"`, `"backcast"` – using the forward or backward eventual forecast function. The last three options are deprecated and will be removed in a future release.
`envir`	environment in which the function arguments are evaluated. If NULL the calling environment of this function will be used.

Value

seasadj returns a seasonal adjusted time series.

Examples

```
Y <- AirPassengers
um1 <- um(Y, bc = TRUE, i = list(1, c(1,12)), ma = list(1, c(1,12)))
Y <- seasadj(um1)
ide(Y)
```

---

seriesC *Series C Chemical Process Temperature Readings: Every Minute.*

Description

226 observations.

Usage

```
seriesC
```

Format

An object of class numeric of length 226.

References

Box, G.E., Jenkins, G.M., Reinsel, G.C. and Ljung, G.M. (2015) Time Series Analysis: Forecasting and Control. John Wiley & Sons, Hoboken.

---

seriesJ *Gas furnace data*

Description

Sampling interval 9 seconds; observations for 296 pairs of data points.

Usage

```
seriesJ
```

Format

A object of class data.frame with 296 rows and 2 columns:

X 0.60-0.04 (input gas rate in cubir feet per minute.)

Y % CO2 in outlet gas.

References

Box, G.E., Jenkins, G.M., Reinsel, G.C. and Ljung, G.M. (2015) Time Series Analysis: Forecasting and Control. John Wiley & Sons, Hoboken.

`setinputs.tfm` *Add or Replace Inputs in Models*

Description

Adds new inputs to transfer function or univariate models.

Usage

```
## S3 method for class 'tfm'
setinputs(
  mdl,
  xreg = NULL,
  inputs = NULL,
  y = NULL,
  envir = parent.frame(),
  ...
)

setinputs(mdl, ...)

## S3 method for class 'um'
setinputs(mdl, xreg = NULL, inputs = NULL, y = NULL, envir = NULL, ...)
```

Arguments

`mdl`	A `um` or `tfm` object.
`xreg`	Optional matrix of exogenous regressors.
`inputs`	Optional list of `tf` objects (only for `tfm`).
`y`	Optional `ts` object for output series.
`envir`	Environment for evaluation. Default is calling environment.
`...`	Additional arguments passed to model constructor.

Details

For `tfm` objects: If the model already has inputs of the same type, new ones are appended (combined). The model is re-fitted by default unless `fit = FALSE`.

Value

A `tfm` object.

See Also

um, tfm

---

signal *Signal component of a TF model*

Description

signal extracts the signal of a TF model.

Usage

```
signal(mdl, ...)

## S3 method for class 'tfm'
signal(
  mdl,
  y = NULL,
  diff = FALSE,
  type = c("xreg", "inputs"),
  envir = parent.frame(),
  ...
)
```

Arguments

mdl	an object of the class tfm.
...	additional arguments.
y	output of the TF model if it is different to that of the tfm object.
diff	logical. If TRUE, the signal is differenced with the "i" operator of the univariate model of the noise.
type	Character vector specifying signal components to extract: "xreg" for exogenous regressors only,"inputs" for transfer function inputs only. If both provided (default), includes all components.
envir	Environment in which the function arguments are evaluated. By default, the calling environment is used.

Value

A "ts" object.

`sim.tfm` *Simulate Time Series from ARIMA or Transfer Function Models*

Description

Generates random time series from ARIMA (`um`) or transfer function (`tfm`) models.

Usage

```
## S3 method for class 'tfm'
sim(
  mdl,
  n = 100,
  z0 = NULL,
  n0 = 0,
  a = NULL,
  seed = NULL,
  envir = parent.frame(),
  ...
)

sim(mdl, ...)

## S3 method for class 'um'
sim(
  mdl,
  n = 100,
  z0 = NULL,
  n0 = 0,
  a = NULL,
  seed = NULL,
  envir = parent.frame(),
  ...
)
```

Arguments

`mdl`	An object of class `um` or `tfm`.
`n`	Number of observations to simulate.
`z0`	Initial conditions for nonstationary series. Default is `NULL` (zero initial conditions).
`n0`	Number of initial observations to discard as burn-in. Default is `0`.
`a`	Optional vector of innovations with length `n + n0`. If `NULL`, innovations are drawn from $N(0, \sigma^2)$.
`seed`	Random seed for reproducibility.

envir	Environment for argument evaluation. Default is parent.frame().
...	Additional arguments.

Value

A ts object with the simulated time series.

See Also

sim.um, sim.tfm

Examples

```
# AR(1) model
mdl1 <- um(ar = "1 - 0.8B", sig2 = 1)
z1 <- sim(mdl1, n = 100, seed = 123)

# ARIMA(0,1,1) with burn-in
mdl2 <- um(i = 1, ma = "1 - 0.5B", sig2 = 1)
z2 <- sim(mdl2, n = 100, n0 = 50, seed = 456)
```

---

sincos *Trigonometric variables*

Description

'sincos()' creates a full set of seasonal trigonometric regressors (cos/sin harmonics) for a seasonal frequency.

Usage

```
sincos(Y, n.ahead = 0, constant = FALSE)
```

Arguments

Y	A seasonal 'ts' object.
n.ahead	Integer. Extra observations to extend the sample.
constant	Logical. If 'TRUE', include an intercept column.

Value

A 'ts' matrix with trigonometric variables (and intercept if requested).

Examples

```
X <- sincos(AirPassengers, constant = TRUE)
```

spec *Spectrum of an ARMA model*

Description

spec computes the spectrum of an ARMA model.

Usage

```
spec(um, ...)

## S3 method for class 'um'
spec(um, nabla = FALSE, n.freq = 501, ...)
```

Arguments

um	an object of class um.
...	additional parameters.
nabla	logical. If TRUE, the pseudospectrum for a non stationary ARIMA model is calculated. By default, the spectrum is computed for the stationary ARMA model.
n.freq	number of frequencies.

Value

A matrix with the frequencies and the power spectral densities.

Note

The I polynomial is ignored.

Examples

```
um1 <- um(i = "(1 - B)(1 - B^12)", ma = "(1 - 0.8B)(1 - 0.8B^12)")
s <- spec(um1, lag.max = 13)
```

ssm *Time Invariant State Space Model*

Description

ssm creates an S3 object representing a time-invariant state space model:

Usage

```
ssm(z, b, C, S, xreg = NULL, bc = FALSE, cform = TRUE, tol = 1e-05)
```

Arguments

z	an object of class ts.
b	vector of coefficients (m-dimensional).
C	matrix of coefficients (m x m).
S	covariance matrix of the error vector [u(t), v(t)], (m+1 x m+1).
xreg	design matrix of regressors.
bc	logical. If TRUE logs are taken.
cform	logical. If TRUE state equation is given in contemporaneous form, otherwise it is written in lagged form.
tol	tolerance to check if a value is zero or one.

Details

z(t) = b'x(t-j) + u(t) (observation equation), x(t) = Cx(t-1) + v(t) (state equation), j = 0 for contemporaneous form or j = 1 for lagged form. Note: the lagged form (j=1) is equivalent to the future form x(t+1) = Cx(t) + v(t+1).

Value

An object of class ssm containing:

z	the input time series
b	observation coefficients
C	state transition matrix
S	error covariance matrix
xreg	regressor matrix (if provided)
a	regression coefficients for xreg (if computed)
z.name	name of the input series
bc	Box-Cox transformation indicator
m	number of state variables
cform	form indicator (contemporaneous vs lagged)
is.adm	admissibility flag

References

Durbin, J. and Koopman, S.J. (2012) Time Series Analysis by State Space Methods, 2nd ed., Oxford University Press, Oxford.

Harvey, A.C. (1989) Forecasting, Structural Time Series Models and the Kalman Filter. Cambridge University Press, Cambridge.

Examples

```
# Local level model
b <- 1
C <- as.matrix(1)
ssm1 <- ssm(Nile, b, C, S = diag(c(irr = 1, lvl = 0.5)) )
ssm1
```

`std` *Standardize time series*

Description

Centers and scales a time series to zero mean and unit variance.

Usage

```
std(x)
```

Arguments

`x`	A `ts` object or numeric vector.

Value

Standardized time series with same class as input.

`summary.ssm` *Summary of fitted state space model*

Description

`summary.ssm` generates a detailed summary of the results of the estimation of a `ssm` object.

Usage

```
## S3 method for class 'ssm'
summary(object, ...)
```

Arguments

`object`	An object of class `ssm`.
`...`	additional arguments.

Value

An object of class `summary.ssm`.

summary.tfm *Summarize Transfer Function Model*

Description

Produces summary statistics for a fitted transfer function model including parameter estimates, standard errors, and diagnostic tests.

Usage

```
## S3 method for class 'tfm'
summary(
  object,
  y = NULL,
  method = c("exact", "cond"),
  digits = max(3L, getOption("digits") - 3L),
  envir = parent.frame(),
  ...
)
```

Arguments

object	A fitted tfm object.
y	Optional ts object for alternative output series.
method	Character: "exact" or "cond" for residual calculation.
digits	Number of significant digits for printing.
envir	Environment for evaluation. NULL uses calling environment.
...	Additional arguments: p.values (logical) returns only p-values; table (logical) returns only coefficient table.

Details

Computes parameter estimates with standard errors (from Jacobian), z-statistics, p-values, AIC, BIC, log-likelihood, Ljung-Box tests (at lags p+q+1 and n/4+p+q), and Bartlett heteroscedasticity test.

Value

Object of class summary.tfm containing: call, coefficient table, variance-covariance matrix, residuals, diagnostic statistics, information criteria, and time series attributes.

See Also

print.summary.tfm

Examples

```
## Not run:
data(seriesJ)
Y <- seriesJ$Y - mean(seriesJ$Y)
X <- seriesJ$X - mean(seriesJ$X)
umx <- um(X, ar = 3)
umy <- fit(umx, Y)
tfx <- tfest(Y, X, delay = 3, p = 2, q = 2, um.x = umx, um.y = umy)
tfmy <- tfm(Y, inputs = tfx, noise = um(ar = 2))
sm <- summary(tfmy)
print(sm)

## End(Not run)
```

---

summary.um *Summary of um model*

Description

summary prints a summary of the estimation and diagnosis.

Usage

```
## S3 method for class 'um'
summary(
  object,
  z = NULL,
  method = c("exact", "cond"),
  digits = max(3L, getOption("digits") - 3L),
  envir = NULL,
  ...
)
```

Arguments

object	an object of class um.
z	an object of class ts.
method	exact/conditional maximum likelihood.
digits	number of significant digits to use when printing.
envir	environment in which the function arguments are evaluated. If NULL the calling environment of this function will be used.
...	additional arguments.

Value

A list with the summary of the estimation and diagonosis.

Examples

```
z <- AirPassengers
airl <- um(z, i = list(1, c(1,12)), ma = list(1, c(1,12)), bc = TRUE)
summary(airl)
```

`tf` *Transfer function for input*

Description

`tf` creates a rational transfer function for an input, V(B) = w0(1 - w_1B - ... - w_qB^q)/(1-d_1B - ... - d_pB^p)B^dX_t. Note that in this specification the constant term of the MA polynomial is factored out so that both polynomials in the numerator and denominator are normalized and can be specified with the `lagpol` function in the same way as the operators of univariate models.

Usage

```
tf(
  x = NULL,
  delay = 0,
  w0 = 0.01,
  ar = NULL,
  ma = NULL,
  um = NULL,
  n.back = NULL,
  par.prefix = "",
  envir = parent.frame()
)
```

Arguments

`x`	Input time series. A ts object or numeric vector. If NULL, input should be provided inside the `um` object.
`delay`	Integer. Number of periods to delay the input (d in the transfer function).
`w0`	Numeric. Constant term of the transfer function polynomial V(B).
`ar`	Character string or list. Specification of autoregressive polynomials in the denominator.
`ma`	Character string or list. Specification of moving average polynomials in the numerator.
`um`	Univariate model object. Model for the stochastic component of the input series.
`n.back`	Integer. Number of backcasts to compute for extending the input series backward.
`par.prefix`	Character. Prefix for parameter names in transfer function.
`envir`	Environment. Environment for evaluating function arguments. If NULL, uses the calling environment.

Value

An object of class "tf" containing the transfer function specification. Key components include:

x Input time series data (possibly extended with backcasts).

delay Number of periods delay in the transfer function.

w0 Constant gain parameter.

phi AR polynomial coefficients (denominator).

theta MA polynomial coefficients (numerator).

p, q Orders of AR and MA polynomials respectively.

param Named list of all model parameters.

um Univariate model for the input series.

References

Box, G.E., Jenkins, G.M., Reinsel, G.C. and Ljung, G.M. (2015) Time Series Analysis: Forecasting and Control. John Wiley & Sons, Hoboken.

Wei, W.W.S. (2006) Time Series Analysis Univariate and Multivariate Methods. 2nd Edition, Addison Wesley, New York, 33-59.

See Also

um.

Examples

```
x <- rep(0, 100)
x[50] <- 1
tfx <- tf(x, w0 = 0.8, ar = "(1 - 0.5B)(1 - 0.7B^12)")
```

---

tfest *Helper function to create a tf object*

Description

`tfest` estimates the transfer function

$$V(B) = w_0 * (theta(B)/phi(B)) * B^d$$

that relates the input X_t to the output Y_t.

Usage

```
tfest(
  y,
  x,
  delay = 0,
  p = 1,
  q = 2,
  um.y = NULL,
  um.x = NULL,
  n.back = NULL,
  par.prefix = "",
  envir = envir <- parent.frame()
)
```

Arguments

y	Output series, a ts object or numeric vector.
x	Input series, a ts object or numeric vector.
delay	Integer. Number of periods delay (default 0).
p	Integer. Order of the AR polynomial (default 1).
q	Integer. Order of the MA polynomial (default 2).
um.y	Univariate model for output series, um object or NULL.
um.x	Univariate model for input series, um object or NULL.
n.back	Integer. Number of backcasts to compute.
par.prefix	Character. Prefix for parameter names.
envir	Environment for evaluating arguments. If NULL, uses calling environment.

Details

Uses prewhitening to estimate initial parameter values.

Value

An object of class `tf` containing preestimated transfer function parameters.

Examples

```
data(seriesJ)
Y <- seriesJ$Y - mean(seriesJ$Y)
X <- seriesJ$X - mean(seriesJ$X)
umx <- um(X, ar = 3)
umy <- fit(umx, Y)
tfx <- tfest(Y, X, delay = 3, p = 2, q = 2, um.x = umx, um.y = umy)
```

tfm *Transfer Function Model Constructor*

Description

Creates and optionally fits a multiple-input transfer function model. A transfer function model relates an output time series to one or more input series (transfer functions), exogenous regressors, and a noise model.

Usage

```
tfm(
  output = NULL,
  xreg = NULL,
  inputs = NULL,
  noise,
  fit = TRUE,
  new.name = TRUE,
  envir = parent.frame(),
  ...
)
```

Arguments

output	A numeric vector or ts object representing the dependent (output) time series. If NULL, it is taken from noise$z.
xreg	A numeric matrix or ts object of exogenous regressors. Columns correspond to different regressors. Defaults to NULL.
inputs	A list of transfer function objects of class tf. Each element represents one stochastic input. Can also be a single tf object. Defaults to NULL.
noise	An object of class um describing the univariate noise model. This defines the ARIMA-type structure for the residuals.
fit	Logical. If TRUE (default), the model parameters are estimated by maximum likelihood after construction.
new.name	Logical. Internal use. If TRUE (default), a new name is assigned to the output series. Otherwise, the name stored in noise$z is preserved.
envir	Environment in which the function arguments are evaluated. If NULL, the calling environment is used.
...	Additional arguments passed to fit.tfm when fit = TRUE.

Details

All series must have the same frequency. Input series must span at least the same period as output. The function applies differencing and Box-Cox transformation as specified in noise.

Value

Object of class `tfm` with components: output, xreg, inputs, noise, param, kx, k, optim, method, and call.

References

Box, G. E., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). *Time Series Analysis: Forecasting and Control* (5th ed.). Wiley.

See Also

`tf`, `um`, `fit.tfm`

Examples

```
## Not run:
data(seriesJ)
Y <- seriesJ$Y - mean(seriesJ$Y)
X <- seriesJ$X - mean(seriesJ$X)
umx <- um(X, ar = 3)
umy <- fit(umx, Y)
tfx <- tfest(Y, X, delay = 3, p = 2, q = 2, um.x = umx, um.y = umy)
tfmy <- tfm(Y, inputs = tfx, noise = um(ar = 2))

## End(Not run)
```

---

theta *Unscramble MA polynomial*

Description

Unscramble MA polynomial

Usage

```
theta(um)

## S3 method for class 'um'
theta(um)
```

Arguments

`um`	an object of class `um`.

Value

A numeric vector `c(1, a1, ..., ad)`

Note

This function returns the member variable `um$theta`.

Examples

```
um1 <- um(ma = "(1 - 0.8B)(1 - 0.5B)")
theta(um1)
```

---

tsdiag.tfm *Diagnostic Plots for Time-Series Fits Description*

Description

`tsdiag.tfm` is a wrap of the stats::tsdiag function.

Usage

```
## S3 method for class 'tfm'
tsdiag(object, gof.lag = 10, ...)
```

Arguments

`object`	a fitted `um` object.
`gof.lag`	the maximum number of lags for a Portmanteau goodness-of-fit test
`...`	additional arguments.

See Also

stats::tsdiag.

---

tsdiag.um *Diagnostic Plots for Time-Series Fits Description*

Description

`tsdiag.um` is a wrap of the stats::tsdiag function.

Usage

```
## S3 method for class 'um'
tsdiag(object, gof.lag = 10, ...)
```

Arguments

object	a fitted um object.
gof.lag	the maximum number of lags for a Portmanteau goodness-of-fit test
...	additional arguments.

See Also

stats::tsdiag.

---

tsvalue *Extract time series value by date*

Description

Retrieves the value of a time series at a specific date.

Usage

```
tsvalue(x, date)
```

Arguments

x	A ts object.
date	Numeric vector: c(year) for annual data or c(year, period) for seasonal data.

Value

Numeric value at the specified date.

---

uc *Unobservable components*

Description

uc creates an S3 object representing a predefined UC (trend, seasonal, cycle, autoregressive and irregular component).

Usage

```
uc(
  type,
  s2 = NULL,
  s = 12,
  form = c("trig", "dummy"),
  per = 5,
  phi = 0.9,
  counter = 1
)
```

Arguments

type	character. The type of component: trend, seasonal, cycle, AR or irregular.
s2	vector with the variances of the errors driving the UC.
s	integer. Seasonal period for seasonal components.
form	character. The form of the seasonal component (trigonometric or dummy seasonality).
per	numeric. The period of the cycle component.
phi	numeric. The damping factor of the cycle component.
counter	integer. Counter for numbering phi cycle parameters.

Value

An object of class uc.

References

Durbin, J. and Koopman, S.J. (2012) Time Series Analysis by State Space Methods, 2nd ed., Oxford University Press, Oxford.

Harvey, A.C. (1989) Forecasting, Structural Time Series Models and the Kalman Filter. Cambridge University Press, Cambridge.

Examples

```
# Trend component
trend <- uc("trend", s2 = c(s2_lvl = 0.5, s2_slp = 0.025))
# Seasonal component
seas <- uc("seasonal", s2 = c(s2_seas = 0.5), s = 12, form = "trig")
# Cycle component
cycle1 <- uc("cycle", s2 = c(s2_c1 = 0.5), per = 5, phi = 0.8)
cycle2 <- uc("cycle", s2 = c(s2_c2 = 0.5), per = 10, phi = 0.8, counter = 2)
```

---

uc0 *Unobservable components (UC) for structural time series models*

Description

uc0 constructs unobservable components by specifying their state-space representation matrices. This is the helper function used internally by uc to create predefined components like trends, seasonals, and cycles.

Usage

```
uc0(name, param, b, C = NULL, S = NULL)
```

Arguments

name	character, name of the UC.
param	a vector or list with the parameters of the UC.
b	vector of the UC in the observation equation.
C	matrix of the UC in the transition matrix.
S	covariance matrix of the error vector driving the UC.

Value

An S3 object of class uc.

Note

Matrices b, C and S can include symbolic expressions in terms of the parameters of the model. These parameters are defined in the param argument with their corresponding names. See the ssm function for more information about b, C and S.

References

Durbin, J. and Koopman, S.J. (2012) Time Series Analysis by State Space Methods, 2nd ed., Oxford University Press, Oxford.

Harvey, A.C. (1989) Forecasting, Structural Time Series Models and the Kalman Filter. Cambridge University Press, Cambridge.

Examples

```
# Local linear trend component
param <- c(s2_lvl = 0.05, s2_slp = 0.025)
b <- c(1, 0)
C <- matrix(c(1, 1, 0, 1), 2, 2, byrow = TRUE)
S <- matrix(c("s2_lvl", "0", "0", "s2_slp"), 2, 2)
trend <- uc0("trend", param, b, C, S)

# Cycle component
param <- c(phi = 0.8, per = 5, s2c = 0.01)
b <- c(1, 0)
C <- matrix(c("phi*cos(2*pi/per)", "phi*sin(2*pi/per)",
"-phi*sin(2*pi/per)", "phi*cos(2*pi/per)"), 2, 2, byrow = TRUE)
S <- matrix(c("s2c*(1 - phi^2)", "0", "0", "s2c*(1 - phi^2)"))
cycle <- uc0("cycle", param, b, C, S)
```

ucarima *Unobserved components ARIMA models*

Description

ucarima creates an S3 object that combines two or more ARIMA models (objects of class um).

Usage

```
ucarima(
  z = NULL,
  bc = FALSE,
  ucm = NULL,
  ar = NULL,
  xreg = NULL,
  fit = TRUE,
  envir = parent.frame(),
  ...
)
```

Arguments

z	an object of class ts or NULL to specify theoretical models.
bc	logical. If TRUE logs are taken.
ucm	a list of um objects specifying the ARIMA models for components such as trend, seasonality, and irregular terms. Alternatively, can be a character string: "ht" for Harvey-Todd seasonal decomposition, or NULL/missing for Harvey-Durbin seasonal decomposition (default when only time series z is provided).
ar	list of stationary AR lag polynomials (lagpol objects).
xreg	matrix of explanatory variables.
fit	logical. If TRUE, model is fitted.
envir	environment.
...	additional arguments.

Value

An object of class ucarima.

References

Harvey, A.C. (1989) Forecasting, Structural Time Series Models and the Kalman Filter. Cambridge University Press, Cambridge.

Examples

```
trend <- um(i = "(1 - B)", sig2 = c(s2t = 1))
seas <- um(i = "(1+B)", sig2 = c(s2s = 0.05))
irreg <- um(sig2 = c(s2i = 0.75))
uca1 <- ucarima(ucm = list(trend = trend, seas = seas, irreg = irreg))
uca1

# Trigonometric seasonality
uca2 <- ucarima(AirPassengers, bc = TRUE)
uca2

# Dummy seasonality
uca3 <- ucarima(AirPassengers, bc = TRUE, ucm = "ht")
uca3
```

---

ucm *Unobserved Components Time Series Models*

Description

ucm creates an S3 object representing an UC model:

Usage

```
ucm(
  z,
  bc = FALSE,
  uc = NULL,
  xreg = NULL,
  cform = TRUE,
  fit = TRUE,
  s = 12,
  ...
)
```

Arguments

z	an object of class ts.
bc	logical. If TRUE, logs are taken.
uc	list of objects of class uc or a character with the name of a predefined model: "llm" local linear model, "lltm" local linear trend model, "bsm" basic structural model, "bsmd" basic structural model with dummy seasonality.
xreg	optional design matrix of regressors.
cform	logical. If TRUE, observation equation is given in contemporaneous form (j = 0); otherwise it is written in lagged form (j = 1).

fit	logical. If TRUE, model is fitted.
s	integer, seasonal period. Optional argument to create a UC model without providing a time series.
...	additional parameters for the fit.ssm function.

Details

z(t) = b'*x(t) + T(t - j) + S(t - j) + C(t - j) + AR(t - j) + I(t),

where z(t) is a time series; x(t) is a set of regressors; T(t - j), S(t - j), C(t - j), AR(t - j) and I(t) are the trend, seasonal, cycle, autoregressive and irregular unobserved components; and i indicates whether the model is written in lagged form (j = 1) or contemporaneous form (j = 0). See uc and ssm for more details.

Value

An object of class ucm and class ssm.

References

Durbin, J. and Koopman, S.J. (2012) Time Series Analysis by State Space Methods, 2nd ed., Oxford University Press, Oxford.

Harvey, A.C. (1989) Forecasting, Structural Time Series Models and the Kalman Filter. Cambridge University Press, Cambridge.

Examples

```
# Local level model
ucm1 <- ucm(Nile, uc = "llm")
ucm1
```

---

um *Univariate (ARIMA) model*

Description

um creates an S3 object representing a univariate ARIMA model, which can contain multiple AR, I and MA polynomials, as well as parameter restrictions.

Usage

```
um(
  z = NULL,
  ar = NULL,
  i = NULL,
  ma = NULL,
  mu = NULL,
```

```
  sig2 = 1,
  bc = FALSE,
  fit = TRUE,
  envir = parent.frame(),
  warn = TRUE,
  ...
)
```

Arguments

z	an object of class ts.
ar	list of stationary AR lag polynomials.
i	list of nonstationary AR (I) polynomials.
ma	list of MA polynomials.
mu	mean of the stationary time series.
sig2	variance of the error.
bc	logical. If TRUE logs are taken.
fit	logical. If TRUE, model is fitted.
envir	the environment in which to look for the time series z when it is passed as a character string.
warn	logical. If TRUE, a warning is displayed for non-admissible models.
...	additional arguments.

Value

An object of class um.

References

Box, G.E.P., Jenkins, G.M., Reinsel, G.C. and Ljung, G.M. (2015) Time Series Analysis: Forecasting and Control. John Wiley & Sons, Hoboken.

Examples

```
ar1 <- um(ar = "(1 - 0.8B)")
ar2 <- um(ar = "(1 - 1.4B + 0.8B^2)")
ma1 <- um(ma = "(1 - 0.8B)")
ma2 <- um(ma = "(1 - 1.4B + 0.8B^2)")
arma11 <- um(ar = "(1 - 1.4B + 0.8B^2)", ma = "(1 - 0.8B)")
```

unitcircle *Unit circle*

Description

unitcircle plots the inverse roots of a lag polynomial together the unit circle.

Usage

```
unitcircle(lp, ...)

## Default S3 method:
unitcircle(lp, ...)

## S3 method for class 'lagpol'
unitcircle(lp, s = 12, ...)
```

Arguments

lp	an object of class lagpol.
...	additional arguments.
s	integer, seasonal period.

Value

unitcircle returns a NULL value.

Examples

```
unitcircle(as.lagpol(c(1, rep(0, 11), -1)))
```

varsel *Variable selection*

Description

varsel omits non-significant inputs from a transfer function model.

Usage

```
varsel(tfm, ...)

## S3 method for class 'tfm'
varsel(tfm, y = NULL, p.value = 0.1, envir = parent.frame(), ...)
```

Arguments

tfm	a tfm object.
...	other arguments.
y	a "ts" object.
p.value	probability value to decide whether or not to omit an input.
envir	environment in which the function arguments are evaluated. By default, the calling environment of this function will be used.

Value

A tfm object or a "um" if no input is significant at that level.

---

wkfilter.as_ucarima *Wiener-Kolmogorov filter*

Description

wkfilter extracts a signal for a time series described by an ARIMA model given the ARIMA model for the signal.

Usage

```
## S3 method for class 'as_ucarima'
wkfilter(object, z = NULL, tol = 1e-05, envir = parent.frame(), ...)

wkfilter(object, ...)

## S3 method for class 'um'
wkfilter(
  object,
  um.uc,
  z = NULL,
  output = c("series", "filter"),
  tol = 1e-05,
  envir = parent.frame(),
  ...
)
```

Arguments

object	an object of class um.
z	an optional ts object. If NULL, the time series to be filtered is contained in the um.z object.
tol	numeric tolerance used in polynomial divisions. Default is 1e-5.
envir	environment to get z when not provided.

...	additional arguments.
um.uc	ARIMA models for the observed time series and the unobserved component (signal).
output	character, output of the function: ‘"series"‘ (default) returns the filtered time series, or ‘"filter"‘ returns the filter coefficients.

Value

An object of class ts containing the estimated signal.

Examples

```
um1 <- airline(AirPassengers, bc = TRUE)
uca1 <- as.ucarima(um1, i = "(1-B)2")
trend <- wkfilter(um1, uca1$ucm$signal1)
seas <- wkfilter(um1, uca1$ucm$signal2)
```

---

wold.pol *Wold polynomial*

Description

Transforming a palindromic polymonial into a Wold polynomial/ Computing the Cramer-Wold factorization

Usage

```
wold.pol(x, type = c("wold", "palindromic", "cramer-wold"), tol = 1e-05)
```

Arguments

x	numeric vector, coefficients of a palindromic or a Wold polynomial.
type	character indicating the type of polynomial: (1) Wold polynomial, (2) Palindromic polynomial and (3) Cramer-Wold factor.
tol	tolerance to check if an autocovariance is zero.

Details

wold.pol can be used with three purposes:

(1) to transform a self-reciprocal or palindromic polynomial a_0 + a_1(B+F) + a_2(B^2+F^2) + ... + a_p(B^p+F^p) into a Wold polynomial b_0 + b_1(B+F) + b_2(B+F)^2 + ... + b_p(B+F)^p;

(2) to revert the previous transformation to obtain the palindromic polynominal from a Wold polynomial and

(3) to compute the Cramer-Wold factorization: b(B+F) = c(B)c(F).

Value

Numeric vector.

Examples

```
wold.pol(c(6, -4, 1))
```

`Wtelephone` *Wisconsin Telephone Company*

Description

Monthly data from January 1951 to October 1966.

Usage

```
Wtelephone
```

Format

A object of class data.frame with 215 rows and 2 columns:

X Monthly outward station movements.

Y Montly inward station movements.

Source

https://drive.google.com/file/d/1LP8aMIQewMrxgOlrg9rN3eWHhZuUsY8K/view?usp=sharing

References

Thompson, H. E. and Tiao, G. C. (1971) "Analysis of Telephone Data: A Case Study of Forecasting Seasonal Time Series," Bell Journal of Economics, The RAND Corporation, vol. 2(2), pages 515-541, Autumn.

Index