Package: EstemPMM (via r-universe)

October 24, 2026

Type Package

Title Polynomial Maximization Method for Non-Gaussian Regression

Version 0.5.0.9000

Date 2026-09-19

Description Implements the Polynomial Maximization Method ('PMM') for parameter estimation in linear and time series models when error distributions deviate from normality. The 'PMM2' variant achieves lower variance parameter estimates compared to ordinary least squares ('OLS') when errors exhibit significant skewness. The 'PMM3' variant (S=3) targets symmetric platykurtic error distributions, reducing variance when excess kurtosis is negative. Includes automatic method selection ('pmm_dispatch'), linear regression, 'AR'/'MA'/'ARMA'/'ARIMA' models, and bootstrap inference. Methodology described in Zabolotnii, Warsza, and Tkachenko (2018) <doi:10.1007/978-3-319-77179-3_75>, Zabolotnii, Tkachenko, and Warsza (2022) <doi:10.1007/978-3-031-03502-9_37>, and Zabolotnii, Tkachenko, and Warsza (2023) <doi:10.1007/978-3-031-25844-2_21>, and Zabolotnii (2025) <doi:10.48550/arXiv.2511.07059>.

License GPL-3

Encoding UTF-8

Depends R (>= 3.5.0)

Imports methods, stats, graphics, utils

Suggests dplyr, ggplot2, gridExtra, testthat (>= 3.0.0), rmarkdown, knitr, MASS, numDeriv

LazyData true

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

URL https://github.com/SZabolotnii/EstemPMM

BugReports https://github.com/SZabolotnii/EstemPMM/issues

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.3

Collate 'data.R' 'optimized_direct_pmm2.R' 'pmm_base_classes.R' 'pmm2_classes.R' 'pmm2_common.R' 'pmm2_inference.R' 'pmm2_ma_estimator.R' 'pmm2_main.R' 'pmm2_monte_carlo.R' 'pmm2_package.R' 'pmm2_ts_design.R' 'pmm2_ts_main.R' 'pmm2_ts_methods.R' 'pmm2_ts_recursive.R' 'pmm2_unified.R' 'pmm2_utils.R' 'pmm3_classes.R' 'pmm3_dispatch.R' 'pmm3_main.R' 'pmm3_solver.R' 'pmm3_ts_classes.R' 'pmm3_ts_main.R' 'pmm3_ts_methods.R' 'pmm3_utils.R' 'pmm_show_methods.R' 'pmm_unified_api.R' 'sarimax_wrapper.R' 'wpmm_main.R'

Repository https://szabolotnii.r-universe.dev

Date/Publication 2026-10-24 17:59:52 UTC

RemoteUrl https://github.com/szabolotnii/estempmm

RemoteRef HEAD

RemoteSha b8e962fe2a2c65e65c47a64de1b8b80a2e6205a6

Contents

ar_pmm2 . . . 5
ar_pmm3 . . . 7
arima_pmm2 . . . 8
arima_pmm3 . . . 10
ARIMAPMM2-class . . . 11
ARIMAPMM3-class . . . 12
arma_pmm2 . . . 12
arma_pmm3 . . . 14
ARMAPMM2-class . . . 15
ARMAPMM3-class . . . 15
ARPMM2-class . . . 15
ARPMM3-class . . . 16
auto_mpg . . . 16
BasePMM2-class . . . 17
BasePMM3-class . . . 18
coef,PMM2fit-method . . . 18
coef,PMM3fit-method . . . 19
coef,SARPMM2-method . . . 19
coef,SMAPMM2-method . . . 20
coef,TS2fit-method . . . 20
coef,TS3fit-method . . . 21
compare_ar_methods . . . 21
compare_arima_methods . . . 22
compare_arma_methods . . . 22
compare_ma_methods . . . 23

compare_sar_methods . . . 24
compare_ts_methods . . . 25
compare_with_ols . . . 25
compute_moments . . . 26
compute_moments_pmm3 . . . 27
compute_pmm2_components . . . 27
confint,PMM2fit-method . . . 28
confint,PMM3fit-method . . . 28
confint,TS2fit-method . . . 29
confint,TS3fit-method . . . 30
create_sar_matrix . . . 30
create_sarma_matrix . . . 31
DCOILWTICO . . . 33
djia2002 . . . 33
fitted,PMM2fit-method . . . 34
fitted,PMM3fit-method . . . 35
fitted,TS2fit-method . . . 35
fitted,TS3fit-method . . . 36
format.PMMdispatch . . . 36
get_sarimax_jacobian . . . 37
get_sarimax_residuals . . . 37
lm_pmm2 . . . 38
lm_pmm3 . . . 39
lm_wpmm2 . . . 41
logLik.PMM2fit . . . 43
logLik.PMM3fit . . . 44
logLik.TS2fit . . . 44
logLik.TS3fit . . . 45
ma_pmm2 . . . 45
ma_pmm3 . . . 47
MAPMM2-class . . . 49
MAPMM3-class . . . 49
nobs,PMM2fit-method . . . 49
nobs,PMM3fit-method . . . 50
nobs,TS2fit-method . . . 50
nobs,TS3fit-method . . . 51
plot,PMM2fit,missing-method . . . 51
plot,PMM3fit,missing-method . . . 52
plot,TS2fit,missing-method . . . 52
plot,TS3fit,missing-method . . . 53
plot_pmm2_bootstrap . . . 53
pmm_ar . . . 54
pmm_arima . . . 54
pmm_arma . . . 55
pmm_dispatch . . . 55
pmm_gamma6 . . . 57
pmm_kurtosis . . . 58
pmm_lm . . . 58

pmm_ma . . . 59
pmm_sarima . . . 60
pmm_skewness . . . 61
pmm2_inference . . . 61
pmm2_monte_carlo_compare . . . 62
pmm2_nonlinear_iterative . . . 64
pmm2_nonlinear_onestep . . . 64
pmm2_variance_factor . . . 65
pmm2_variance_matrices . . . 66
PMM2fit-class . . . 66
pmm3_variance_factor . . . 67
pmm3_variance_matrices . . . 68
PMM3fit-class . . . 68
PMMfit-class . . . 69
PMMtsfit-class . . . 69
predict,PMM2fit-method . . . 70
predict,PMM3fit-method . . . 71
predict,TS2fit-method . . . 71
predict,TS3fit-method . . . 72
print.PMMdispatch . . . 72
residuals,PMM2fit-method . . . 73
residuals,PMM3fit-method . . . 73
residuals,TS2fit-method . . . 74
residuals,TS3fit-method . . . 74
sar_pmm2 . . . 75
sarima_pmm2 . . . 77
SARIMAPMM2-class . . . 79
sarma_pmm2 . . . 80
SARMAPMM2-class . . . 82
SARPMM2-class . . . 83
show,PMM2fit-method . . . 84
show,PMM3fit-method . . . 85
show,TS2fit-method . . . 85
show,TS3fit-method . . . 86
sma_pmm2 . . . 86
SMAPMM2-class . . . 88
solve_pmm2_step . . . 89
summary,PMM2fit-method . . . 90
summary,PMM3fit-method . . . 91
summary,SARIMAPMM2-method . . . 91
summary,SARMAPMM2-method . . . 92
summary,SARPMM2-method . . . 92
summary,SMAPMM2-method . . . 93
summary,TS2fit-method . . . 93
summary,TS3fit-method . . . 94
summary.PMMdispatch . . . 94
test_symmetry . . . 95
ts_pmm2 . . . 95

ts_pmm2_inference 97
ts_pmm3 98
TS2fit-class 99
TS3fit-class 100
vcov,PMM2fit-method 100
vcov,PMM3fit-method 101
vcov,TS2fit-method 102
vcov,TS3fit-method 103
WPMMfit-class 103
WPMMfit-methods 104

Index **106**

`ar_pmm2` *Fit an AR model using PMM2 (wrapper)*

Description

Estimates autoregressive model parameters using the Pearson Moment Method (PMM2). PMM2 exploits third and fourth moment information to achieve more accurate parameter estimates than classical maximum likelihood, particularly for non-Gaussian innovations.

Usage

```
ar_pmm2(
  x,
  order = 1,
  method = "pmm2",
  pmm2_variant = c("unified_global", "unified_iterative", "linearized"),
  max_iter = 50,
  tol = 1e-06,
  include.mean = TRUE,
  initial = NULL,
  na.action = na.fail,
  regularize = TRUE,
  reg_lambda = 1e-08,
  verbose = FALSE
)
```

Arguments

`x`	Numeric vector of time series data
`order`	Model order specification: • For AR models: a single integer (AR order) • For MA models: a single integer (MA order) • For ARMA models: vector c(p, q) (AR and MA orders) • For ARIMA models: vector c(p, d, q) (AR, differencing, and MA orders)

method	String: estimation method, one of "pmm2" (default), "css", "ml", "yw", "ols"
pmm2_variant	Character string specifying PMM2 implementation variant. Options: "unified_global" (default, one-step correction), "unified_iterative" (full Newton-Raphson), or "linearized" (specialized for MA/SMA models).
max_iter	Integer: maximum number of iterations for the algorithm
tol	Numeric: tolerance for convergence
include.mean	Logical: whether to include a mean (intercept) term
initial	List or vector of initial parameter estimates (optional)
na.action	Function for handling missing values, default is na.fail
regularize	Logical, add small values to diagonal for numerical stability
reg_lambda	Regularization parameter (if regularize=TRUE)
verbose	Logical: whether to print progress information

Details

PMM2 Variants:

- "unified_global" (default): One-step correction from MLE/CSS estimates. Fast and stable. Recommended for most AR models. Typical improvement: 3-5\
- "unified_iterative": Full Newton-Raphson optimization starting from classical estimates. More accurate but computationally intensive. Best for complex models with strong non-Gaussian features.
- "linearized": Uses first-order Taylor expansion. Not recommended for AR models; designed for MA/SMA where Jacobian computation is complex.

Variant Selection Guidelines:

- For AR(p) models: Use "unified_global" (default)
- If convergence issues occur: Try "unified_iterative"
- For highly skewed/heavy-tailed innovations: Use "unified_iterative"

Computational Characteristics:

- unified_global: ~2x slower than MLE (single correction step)
- unified_iterative: 5-10x slower than MLE (iterative refinement)
- linearized: ~1.5x slower than MLE (approximation-based)

Value

An S4 object of class ARPMM2 containing fitted autoregressive coefficients, residuals, central moment estimates (m2-m4), model order, intercept, original series, and convergence diagnostics.

References

Monte Carlo validation (R=50, n=200): Unified Iterative showed 2.9\ improvement over MLE for AR(1) models. See NEWS.md (Version 0.2.0) for details.

See Also

ma_pmm2, arma_pmm2, arima_pmm2

Examples

```
# Fit AR(2) model with default variant
x <- arima.sim(n = 200, list(ar = c(0.7, -0.3)))
fit1 <- ar_pmm2(x, order = 2)
coef(fit1)

# Compare variants
fit2 <- ar_pmm2(x, order = 2, pmm2_variant = "unified_iterative")
fit3 <- ar_pmm2(x, order = 2, pmm2_variant = "linearized")
```

---

ar_pmm3 *Fit an AR model using PMM3*

Description

Estimates autoregressive model parameters using the Polynomial Maximization Method of order 3 (PMM3). Designed for symmetric platykurtic innovations.

Usage

```
ar_pmm3(
  x,
  order = 1,
  max_iter = 100,
  tol = 1e-06,
  adaptive = FALSE,
  step_max = 5,
  include.mean = TRUE,
  initial = NULL,
  na.action = na.fail,
  verbose = FALSE
)
```

Arguments

x	Numeric vector of time series data
order	Integer: AR order (default 1)
max_iter	Integer: maximum NR iterations (default 100)
tol	Numeric: convergence tolerance (default 1e-6)
adaptive	Logical: re-estimate kappa each iteration (default FALSE)

step_max	Numeric: maximum NR step (default 5.0); see ts_pmm3
include.mean	Logical: include mean term (default TRUE)
initial	Optional initial parameter estimates
na.action	Function for handling missing values (default na.fail)
verbose	Logical: print progress information (default FALSE)

Value

An S4 object of class ARPMM3

See Also

ar_pmm2, ma_pmm3, lm_pmm3

Examples

```
set.seed(42)
x <- arima.sim(n = 200, list(ar = 0.7),
               innov = runif(200, -sqrt(3), sqrt(3)))
fit <- ar_pmm3(x, order = 1)
coef(fit)
```

---

arima_pmm2 *Fit an ARIMA model using PMM2 (wrapper)*

Description

Estimates autoregressive integrated moving-average model parameters using PMM2. ARIMA models extend ARMA to non-stationary series via differencing. PMM2 provides robust parameter estimates for the stationary ARMA component after differencing is applied.

Usage

```
arima_pmm2(
  x,
  order = c(1, 1, 1),
  method = "pmm2",
  pmm2_variant = c("unified_global", "unified_iterative", "linearized"),
  ma_solver = c("linearized", "recursive"),
  max_iter = 50,
  tol = 1e-06,
  include.mean = TRUE,
  initial = NULL,
  na.action = na.fail,
  regularize = TRUE,
```

```
  reg_lambda = 1e-08,
  verbose = FALSE
)
```

Arguments

x	Numeric vector of time series data
order	Model order specification: • For AR models: a single integer (AR order) • For MA models: a single integer (MA order) • For ARMA models: vector c(p, q) (AR and MA orders) • For ARIMA models: vector c(p, d, q) (AR, differencing, and MA orders)
method	String: estimation method, one of "pmm2" (default), "css", "ml", "yw", "ols"
pmm2_variant	Character string specifying PMM2 implementation variant. Options: "unified_global" (default, one-step correction), "unified_iterative" (full Newton-Raphson), or "linearized" (specialized for MA/SMA models).
ma_solver	String: how the moving-average recursion is treated, "linearized" (default) or "recursive". Has no effect when the MA order is zero. See Details.
max_iter	Integer: maximum number of iterations for the algorithm
tol	Numeric: tolerance for convergence
include.mean	Logical: whether to include a mean (intercept) term
initial	List or vector of initial parameter estimates (optional)
na.action	Function for handling missing values, default is na.fail
regularize	Logical, add small values to diagonal for numerical stability
reg_lambda	Regularization parameter (if regularize=TRUE)
verbose	Logical: whether to print progress information

Details

PMM2 Variants:

- "unified_global" (default): One-step correction from MLE/CSS estimates. Fast and reliable for most ARIMA specifications.
- "unified_iterative": Full Newton-Raphson optimization. Recommended for ARIMA models with complex dynamics or when unified_global shows residual non-Gaussianity.
- "linearized": First-order approximation. Not typically recommended for ARIMA unless MA component dominates.

Variant Selection Guidelines:

- For standard ARIMA(p,d,q): Use "unified_global" (default)
- For models with d >= 2 or high orders: Try "unified_iterative"
- If MA component is large relative to AR: Consider "unified_iterative"

ARIMA Estimation Workflow:

1. Apply differencing of order d to achieve stationarity
2. Estimate ARMA(p,q) model on differenced series using PMM2
3. Return coefficients and diagnostics for the integrated model

Differencing Notes: The d parameter determines how many times the series is differenced. d=0 reduces to ARMA, d=1 handles unit root processes, d=2 is rare but useful for some economic series with trend acceleration.

Value

An S4 object of class ARIMAPMM2 containing fitted AR and MA coefficients, residual series, central moments, differencing order, intercept, original series, and convergence diagnostics.

See Also

arma_pmm2, sarima_pmm2, ar_pmm2

Examples

```
# Fit ARIMA(1,1,1) model to non-stationary series
x <- cumsum(arima.sim(n = 200, list(ar = 0.6, ma = -0.4)))
fit1 <- arima_pmm2(x, order = c(1, 1, 1))
coef(fit1)

# ARIMA(2,1,0) - random walk with AR(2) innovations
x2 <- cumsum(arima.sim(n = 250, list(ar = c(0.7, -0.3))))
fit2 <- arima_pmm2(x2, order = c(2, 1, 0), pmm2_variant = "unified_global")

# ARIMA(0,2,2) - double differencing with MA(2)
x3 <- cumsum(cumsum(arima.sim(n = 300, list(ma = c(0.5, 0.3)))))
fit3 <- arima_pmm2(x3, order = c(0, 2, 2), pmm2_variant = "unified_iterative")
```

---

arima_pmm3 *Fit an ARIMA model using PMM3*

Description

Estimates ARIMA model parameters using PMM3.

Usage

```
arima_pmm3(
  x,
  order = c(1, 1, 1),
  max_iter = 100,
  tol = 1e-06,
```

```
  adaptive = FALSE,
  step_max = 5,
  include.mean = TRUE,
  initial = NULL,
  na.action = na.fail,
  verbose = FALSE
)
```

Arguments

x	Numeric vector of time series data
order	Numeric vector c(p, d, q): AR, differencing, and MA orders
max_iter	Integer: maximum NR iterations (default 100)
tol	Numeric: convergence tolerance (default 1e-6)
adaptive	Logical: re-estimate kappa each iteration (default FALSE)
step_max	Numeric: maximum NR step (default 5.0); see ts_pmm3
include.mean	Logical: include mean term (default TRUE)
initial	Optional initial parameter estimates
na.action	Function for handling missing values (default na.fail)
verbose	Logical: print progress information (default FALSE)

Value

An S4 object of class ARIMAPMM3

See Also

arima_pmm2, arma_pmm3, lm_pmm3

Examples

```
set.seed(42)
x <- cumsum(arima.sim(n = 200, list(ar = 0.6),
            innov = runif(200, -sqrt(3), sqrt(3))))
fit <- arima_pmm3(x, order = c(1, 1, 0))
coef(fit)
```

ARIMAPMM2-class *S4 class for storing PMM2 ARIMA model results*

Description

S4 class for storing PMM2 ARIMA model results

ARIMAPMM3-class *S4 class for PMM3 ARIMA model results*

Description

S4 class for PMM3 ARIMA model results

arma_pmm2 *Fit an ARMA model using PMM2 (wrapper)*

Description

Estimates autoregressive moving-average model parameters using PMM2. ARMA models combine AR and MA components, capturing both direct past value dependencies and innovation structure. PMM2 leverages higher moments to improve parameter estimation accuracy.

Usage

```
arma_pmm2(
  x,
  order = c(1, 1),
  method = "pmm2",
  pmm2_variant = c("unified_global", "unified_iterative", "linearized"),
  ma_solver = c("linearized", "recursive"),
  max_iter = 50,
  tol = 1e-06,
  include.mean = TRUE,
  initial = NULL,
  na.action = na.fail,
  regularize = TRUE,
  reg_lambda = 1e-08,
  verbose = FALSE
)
```

Arguments

x	Numeric vector of time series data
order	Model order specification: • For AR models: a single integer (AR order) • For MA models: a single integer (MA order) • For ARMA models: vector c(p, q) (AR and MA orders) • For ARIMA models: vector c(p, d, q) (AR, differencing, and MA orders)
method	String: estimation method, one of "pmm2" (default), "css", "ml", "yw", "ols"

pmm2_variant	Character string specifying PMM2 implementation variant. Options: "unified_global" (default, one-step correction), "unified_iterative" (full Newton-Raphson), or "linearized" (specialized for MA/SMA models).
ma_solver	String: how the moving-average recursion is treated, "linearized" (default) or "recursive". Has no effect when the MA order is zero. See Details.
max_iter	Integer: maximum number of iterations for the algorithm
tol	Numeric: tolerance for convergence
include.mean	Logical: whether to include a mean (intercept) term
initial	List or vector of initial parameter estimates (optional)
na.action	Function for handling missing values, default is na.fail
regularize	Logical, add small values to diagonal for numerical stability
reg_lambda	Regularization parameter (if regularize=TRUE)
verbose	Logical: whether to print progress information

Details

PMM2 Variants:

- "unified_global" (default): One-step correction from MLE/CSS estimates. Balances speed and accuracy for ARMA models. Recommended for most use cases.
- "unified_iterative": Full Newton-Raphson optimization. Best for models with complex dynamics or strong non-Gaussian features.
- "linearized": First-order approximation. Generally not recommended for ARMA; better suited for pure MA/SMA models.

Variant Selection Guidelines:

- For ARMA(p,q) with p,q <= 2: Use "unified_global" (default)
- For higher-order ARMA or ill-conditioned models: Try "unified_iterative"
- For quick exploration: Start with "unified_global"

ARMA Estimation Challenges: ARMA models have more parameters than pure AR or MA models, making estimation more sensitive to initialization and numerical stability. PMM2 benefits from robust moment-based constraints that help regularize the parameter space.

Value

An S4 object of class ARMAPMM2 containing fitted AR and MA coefficients, residuals, central moments, model specification, intercept, original series, and convergence diagnostics.

See Also

ar_pmm2, ma_pmm2, arima_pmm2

Examples

```
# Fit ARMA(2,1) model
x <- arima.sim(n = 250, list(ar = c(0.7, -0.3), ma = 0.5))
fit1 <- arma_pmm2(x, order = c(2, 1))
coef(fit1)

# Try iterative variant for better accuracy
fit2 <- arma_pmm2(x, order = c(2, 1), pmm2_variant = "unified_iterative")

# Higher-order ARMA
x2 <- arima.sim(n = 300, list(ar = c(0.6, -0.2), ma = c(0.4, 0.3)))
fit3 <- arma_pmm2(x2, order = c(2, 2), pmm2_variant = "unified_global")
```

---

arma_pmm3 *Fit an ARMA model using PMM3*

Description

Estimates ARMA model parameters using PMM3.

Usage

```
arma_pmm3(
  x,
  order = c(1, 1),
  max_iter = 100,
  tol = 1e-06,
  adaptive = FALSE,
  step_max = 5,
  include.mean = TRUE,
  initial = NULL,
  na.action = na.fail,
  verbose = FALSE
)
```

Arguments

x	Numeric vector of time series data
order	Numeric vector c(p, q): AR and MA orders
max_iter	Integer: maximum NR iterations (default 100)
tol	Numeric: convergence tolerance (default 1e-6)
adaptive	Logical: re-estimate kappa each iteration (default FALSE)
step_max	Numeric: maximum NR step (default 5.0); see ts_pmm3
include.mean	Logical: include mean term (default TRUE)

initial	Optional initial parameter estimates
na.action	Function for handling missing values (default na.fail)
verbose	Logical: print progress information (default FALSE)

Value

An S4 object of class ARMAPMM3

See Also

arma_pmm2, arima_pmm3, lm_pmm3

Examples

```
set.seed(42)
x <- arima.sim(n = 250, list(ar = 0.7, ma = -0.3),
               innov = runif(250, -sqrt(3), sqrt(3)))
fit <- arma_pmm3(x, order = c(1, 1))
coef(fit)
```

ARMAPMM2-class *S4 class for storing PMM2 ARMA model results*

Description

S4 class for storing PMM2 ARMA model results

ARMAPMM3-class *S4 class for PMM3 ARMA model results*

Description

S4 class for PMM3 ARMA model results

ARPMM2-class *S4 class for storing PMM2 AR model results*

Description

S4 class for storing PMM2 AR model results

ARPMM3-class *S4 class for PMM3 AR model results*

Description

S4 class for PMM3 AR model results

auto_mpg *Auto MPG Dataset*

Description

Fuel consumption and vehicle characteristics for 398 automobiles from the 1970s and 1980s. This dataset is used in published PMM research to demonstrate PMM2 on asymmetric regression residuals and to illustrate dispatcher diagnostics on practical automotive data.

Usage

```
auto_mpg
```

Format

A data frame with 398 rows and 9 variables:

mpg Miles per gallon (fuel efficiency)

cylinders Number of cylinders (4, 6, or 8)

displacement Engine displacement (cubic inches)

horsepower Engine horsepower (6 missing values)

weight Vehicle weight (pounds)

acceleration Time to accelerate from 0 to 60 mph (seconds)

model_year Model year (70-82, i.e., 1970-1982)

origin Origin (1 = American, 2 = European, 3 = Japanese)

car_name Car model name

Details

Practical regression examples:

- **MPG vs Acceleration** (PMM2, linear): residuals have gamma3 = 0.49, g2 = 0.86 (Zabolotnii et al., 2018)
- **MPG vs Weight** (PMM2, quadratic): residuals have gamma3 = 0.8, g2 = 0.83 (Zabolotnii et al., 2025)
- **MPG vs Horsepower** (diagnostic, quadratic): residuals are nearly symmetric with positive excess kurtosis; `pmm_dispatch()` recommends PMM3 (n = 392 >= `n_leptokurtic`), although the errors are heteroscedastic and a pairs bootstrap shows no variance gain.

Source

UCI Machine Learning Repository https://archive.ics.uci.edu/dataset/9/auto+mpg

References

Quinlan, J.R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Machine Learning, 236-243.

Zabolotnii S., Warsza Z.L., Tkachenko O. (2018) Polynomial Estimation of Linear Regression Parameters for the Asymmetric PDF of Errors. Springer AISC, vol 743. doi:10.1007/9783319771793_75

Examples

```
data(auto_mpg)
# PMM2 example: MPG vs Acceleration (asymmetric residuals)
fit_ols <- lm(mpg ~ acceleration, data = auto_mpg)
pmm_skewness(residuals(fit_ols))  # gamma3 ~ 0.5 -> PMM2
pmm_dispatch(residuals(fit_ols))
fit_pmm2 <- lm_pmm2(mpg ~ acceleration, data = auto_mpg, na.action = na.omit)
coef(fit_pmm2)  # compare with coef(fit_ols)
```

BasePMM2-class *Virtual S4 class for the PMM2 model family*

Description

Virtual intermediate that inherits common slots (coefficients, residuals, convergence, iterations, call) from PMMfit and adds the PMM2-specific central moments m_2, m_3, m_4. Concrete subclasses are PMM2fit (regression) and the time-series classes under TS2fit.

Slots

`m2` numeric second central moment of initial residuals

`m3` numeric third central moment of initial residuals

`m4` numeric fourth central moment of initial residuals

BasePMM3-class *Virtual S4 class for the PMM3 model family*

Description

Virtual intermediate that inherits common slots from PMMfit and adds the PMM3-specific moments and cumulant coefficients used for symmetric platykurtic errors. Concrete subclasses are PMM3fit (regression) and the time-series classes under TS3fit.

Slots

`m2` numeric second central moment of initial residuals

`m4` numeric fourth central moment of initial residuals

`m6` numeric sixth central moment of initial residuals

`gamma4` numeric excess kurtosis coefficient

`gamma6` numeric sixth-order cumulant coefficient

`g_coefficient` numeric theoretical variance reduction factor g3

`kappa` numeric moment ratio used by the Newton-Raphson solver

coef,PMM2fit-method *Extract coefficients from PMM2fit object*

Description

Extract coefficients from PMM2fit object

Usage

```
## S4 method for signature 'PMM2fit'
coef(object, ...)
```

Arguments

`object`	PMM2fit object
`...`	Additional arguments (not used)

Value

Vector of coefficients

coef,PMM3fit-method *Extract coefficients from PMM3fit object*

Description

Extract coefficients from PMM3fit object

Usage

```
## S4 method for signature 'PMM3fit'
coef(object, ...)
```

Arguments

object	PMM3fit object
...	Additional arguments (not used)

Value

Vector of coefficients

coef,SARPMM2-method *Extract coefficients from SARPMM2 object*

Description

Extract coefficients from SARPMM2 object

Usage

```
## S4 method for signature 'SARPMM2'
coef(object, ...)
```

Arguments

object	SARPMM2 object
...	Additional arguments (not used)

Value

Named numeric vector of coefficients

coef,SMAPMM2-method *Extract coefficients from SMAPMM2 object*

Description

Extract coefficients from SMAPMM2 object

Usage

```
## S4 method for signature 'SMAPMM2'
coef(object, ...)
```

Arguments

object	SMAPMM2 object
...	Additional arguments (not used)

Value

Named vector of seasonal MA coefficients

coef,TS2fit-method *Extract coefficients from TS2fit object*

Description

Extract coefficients from TS2fit object

Usage

```
## S4 method for signature 'TS2fit'
coef(object, ...)
```

Arguments

object	TS2fit object
...	Additional arguments (not used)

Value

Named vector of coefficients

coef,TS3fit-method *Extract coefficients from TS3fit object*

Description

Extract coefficients from TS3fit object

Usage

```
## S4 method for signature 'TS3fit'
coef(object, ...)
```

Arguments

object	TS3fit object
...	Additional arguments (not used)

Value

Named vector of coefficients

compare_ar_methods *Compare AR methods*

Description

Compare AR methods

Usage

```
compare_ar_methods(x, order = 1, include.mean = TRUE, pmm2_args = list())
```

Arguments

x	Numeric vector of time series data
order	Model order specification (see ts_pmm2 for format)
include.mean	Logical, whether to include intercept term
pmm2_args	List of additional arguments to pass to ts_pmm2()

Value

A named list containing the fitted objects for each estimation approach (e.g., YW/OLS/MLE or CSS/ML alongside PMM2) plus two data frames: `coefficients` (side-by-side parameter estimates) and `residual_stats` (residual RSS, MAE, skewness, and kurtosis).

compare_arima_methods *Compare ARIMA methods*

Description

Compare ARIMA methods

Usage

```
compare_arima_methods(
  x,
  order = c(1, 1, 1),
  include.mean = TRUE,
  pmm2_args = list()
)
```

Arguments

x	Numeric vector of time series data
order	Model order specification (see ts_pmm2 for format)
include.mean	Logical, whether to include intercept term
pmm2_args	List of additional arguments to pass to ts_pmm2()

Value

A named list containing the fitted objects for each estimation approach (e.g., YW/OLS/MLE or CSS/ML alongside PMM2) plus two data frames: `coefficients` (side-by-side parameter estimates) and `residual_stats` (residual RSS, MAE, skewness, and kurtosis).

compare_arma_methods *Compare ARMA methods*

Description

Compare ARMA methods

Usage

```
compare_arma_methods(
  x,
  order = c(1, 1),
  include.mean = TRUE,
  pmm2_args = list()
)
```

Arguments

x	Numeric vector of time series data
order	Model order specification (see ts_pmm2 for format)
include.mean	Logical, whether to include intercept term
pmm2_args	List of additional arguments to pass to ts_pmm2()

Value

A named list containing the fitted objects for each estimation approach (e.g., YW/OLS/MLE or CSS/ML alongside PMM2) plus two data frames: `coefficients` (side-by-side parameter estimates) and `residual_stats` (residual RSS, MAE, skewness, and kurtosis).

`compare_ma_methods` *Compare MA methods*

Description

Compare MA methods

Usage

```
compare_ma_methods(x, order = 1, include.mean = TRUE, pmm2_args = list())
```

Arguments

x	Numeric vector of time series data
order	Model order specification (see ts_pmm2 for format)
include.mean	Logical, whether to include intercept term
pmm2_args	List of additional arguments to pass to ts_pmm2()

Value

A named list containing the fitted objects for each estimation approach (e.g., YW/OLS/MLE or CSS/ML alongside PMM2) plus two data frames: `coefficients` (side-by-side parameter estimates) and `residual_stats` (residual RSS, MAE, skewness, and kurtosis).

`compare_sar_methods` *Compare SAR model estimation methods*

Description

Compares different estimation methods (OLS, PMM2, CSS, ML) for SAR models on the same data.

Usage

```
compare_sar_methods(
  x,
  order = c(1, 1),
  period = 12,
  methods = c("ols", "pmm2", "css"),
  verbose = TRUE
)
```

Arguments

`x`	Time series data
`order`	Model order c(p, P) for SAR specification
`period`	Seasonal period
`methods`	Character vector of methods to compare (default: c("ols", "pmm2", "css"))
`verbose`	Logical: print results to console (default TRUE)

Value

Data frame with comparison results (invisibly)

Examples

```
set.seed(42)
y <- arima.sim(n = 120,
  model = list(order = c(1, 0, 0), ar = 0.7,
    seasonal = list(order = c(1, 0, 0), ar = 0.5, period = 12)))
compare_sar_methods(y, order = c(1, 1), period = 12)
```

compare_ts_methods *Compare PMM2 with classical time series estimation methods*

Description

Compare PMM2 with classical time series estimation methods

Usage

```
compare_ts_methods(
  x,
  order,
  model_type = c("ar", "ma", "arma", "arima"),
  include.mean = TRUE,
  pmm2_args = list()
)
```

Arguments

x	Numeric vector of time series data
order	Model order specification (see ts_pmm2 for format)
model_type	Model type: "ar", "ma", "arma", or "arima"
include.mean	Logical, whether to include intercept term
pmm2_args	List of additional arguments to pass to ts_pmm2()

Value

A named list containing the fitted objects for each estimation approach (e.g., YW/OLS/MLE or CSS/ML alongside PMM2) plus two data frames: `coefficients` (side-by-side parameter estimates) and `residual_stats` (residual RSS, MAE, skewness, and kurtosis).

compare_with_ols *Compare PMM2 with OLS*

Description

Compare PMM2 with OLS

Usage

```
compare_with_ols(formula, data, pmm2_args = list())
```

Arguments

formula	Model formula
data	Data frame
pmm2_args	List of arguments to pass to lm_pmm2()

Value

List with OLS and PMM2 fit objects

Examples

```
result <- compare_with_ols(mpg ~ wt, data = mtcars)
```

---

compute_moments *Calculate moments and cumulants of error distribution*

Description

Calculate moments and cumulants of error distribution

Usage

```
compute_moments(errors)
```

Arguments

errors	numeric vector of errors

Value

list with moments, cumulants and theoretical variance reduction coefficient

`compute_moments_pmm3` *Compute central moments for PMM3 from residuals*

Description

Computes the second, fourth, and sixth central moments, along with standardised cumulant coefficients gamma3, gamma4, gamma6, the theoretical efficiency factor g3, and the moment ratio kappa used in the PMM3 solver.

Usage

```
compute_moments_pmm3(residuals)
```

Arguments

`residuals`	numeric vector of residuals (typically from OLS)

Value

A list with components:

`m2`	Second central moment
`m4`	Fourth central moment
`m6`	Sixth central moment
`gamma3`	Skewness coefficient (for symmetry check)
`gamma4`	Excess kurtosis
`gamma6`	Sixth-order cumulant coefficient
`g3`	Theoretical variance reduction factor
`kappa`	Moment ratio for NR solver (NA if near-Gaussian)
`improvement_pct`	Expected variance reduction percentage

`compute_pmm2_components` *Compute PMM2 weights and components*

Description

Compute PMM2 weights and components

Usage

```
compute_pmm2_components(residuals)
```

Arguments

`residuals`	Residual vector

Value

List with moments and PMM2 parameters (gamma3, gamma4, weights)

`confint,PMM2fit-method`

Confidence intervals for PMM2fit coefficients

Description

Computes normal-approximation confidence intervals using the asymptotic covariance matrix from `vcov,PMM2fit-method`.

Usage

```
## S4 method for signature 'PMM2fit'
confint(object, parm, level = 0.95, ...)
```

Arguments

`object`	PMM2fit object
`parm`	character or integer vector of parameter names/indices to include
`level`	confidence level (default 0.95)
`...`	Additional arguments (not used)

Value

Matrix with lower and upper confidence limits

`confint,PMM3fit-method`

Confidence intervals for PMM3fit coefficients

Description

Computes normal-approximation confidence intervals using the asymptotic covariance matrix from `vcov,PMM3fit-method`.

Usage

```
## S4 method for signature 'PMM3fit'
confint(object, parm, level = 0.95, ...)
```

Arguments

`object`	PMM3fit object
`parm`	character or integer vector of parameter names/indices to include
`level`	confidence level (default 0.95)
`...`	Additional arguments (not used)

Value

Matrix with lower and upper confidence limits

`confint,TS2fit-method` *Confidence intervals for TS2fit AR model coefficients*

Description

For AR models, computes normal-approximation CIs using `vcov,TS2fit-method`. For MA/ARMA/ARIMA models, use `ts_pmm2_inference` instead.

Usage

```
## S4 method for signature 'TS2fit'
confint(object, parm, level = 0.95, ...)
```

Arguments

`object`	TS2fit (or subclass) object
`parm`	character or integer vector of parameter names/indices
`level`	confidence level (default 0.95)
`...`	Additional arguments (not used)

Value

Matrix with lower and upper confidence limits

confint,TS3fit-method *Confidence intervals for TS3fit AR model coefficients*

Description

For AR models, computes normal-approximation CIs using vcov,TS3fit-method. For MA/ARMA/ARIMA models, refit the analogous PMM2 model and use ts_pmm2_inference.

Usage

```
## S4 method for signature 'TS3fit'
confint(object, parm, level = 0.95, ...)
```

Arguments

object	TS3fit (or subclass) object
parm	character or integer vector of parameter names/indices
level	confidence level (default 0.95)
...	Additional arguments (not used)

Value

Matrix with lower and upper confidence limits

create_sar_matrix *Create design matrix for seasonal AR model*

Description

Constructs a design matrix for Seasonal Autoregressive (SAR) models, optionally including non-seasonal AR lags and multiplicative cross-terms.

Usage

```
create_sar_matrix(x, p = 0, P = 1, s = 12, multiplicative = FALSE)
```

Arguments

x	Numeric vector (centered time series)
p	Non-seasonal AR order (default 0)
P	Seasonal AR order (must be positive)
s	Seasonal period (e.g., 12 for monthly data)
multiplicative	Logical, include multiplicative cross-terms (default FALSE)

Details

For a SAR(P)_s model:

$$y_t = \Phi_1 y_{t-s} + \cdots + \Phi_P y_{t-Ps} + \epsilon_t.$$

For an additive AR(p)+SAR(P)_s model:

$$y_t = \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \Phi_1 y_{t-s} + \cdots + \Phi_P y_{t-Ps} + \epsilon_t.$$

For a multiplicative AR(p) x SAR(P)_s model (multiplicative = TRUE): Includes cross-terms such as y_{t-1-s}, y_{t-1-2s}, etc.

Value

Design matrix with lagged values. Columns are:

- First p columns: non-seasonal lags $(y_{t-1}, \ldots, y_{t-p})$
- Next P columns: seasonal lags $(y_{t-s}, \ldots, y_{t-Ps})$
- If multiplicative = TRUE: additional $p \times P$ columns for cross-terms

Examples

```
# Simple SAR(1)_12 model
x <- rnorm(120)
X <- create_sar_matrix(x, p = 0, P = 1, s = 12)

# AR(1) + SAR(1)_12 additive model
X <- create_sar_matrix(x, p = 1, P = 1, s = 12)

# AR(1) x SAR(1)_12 multiplicative model
X <- create_sar_matrix(x, p = 1, P = 1, s = 12, multiplicative = TRUE)
```

---

create_sarma_matrix *Create design matrix for seasonal ARMA model*

Description

Constructs a design matrix for Seasonal ARMA (SARMA) models, combining non-seasonal and seasonal AR and MA components.

Usage

```
create_sarma_matrix(
  x,
  residuals,
  p = 0,
  P = 0,
  q = 0,
```

```
  Q = 0,
  s = 12,
  multiplicative = FALSE
)
```

Arguments

x	Numeric vector (centered time series)
residuals	Numeric vector (initial residuals/innovations)
p	Non-seasonal AR order (default 0)
P	Seasonal AR order (default 0)
q	Non-seasonal MA order (default 0)
Q	Seasonal MA order (default 0)
s	Seasonal period (e.g., 12 for monthly data)
multiplicative	Logical, include multiplicative cross-terms (default FALSE)

Details

For a SARMA(p,q) x (P,Q)_s model:

$$y_t = \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \Phi_1 y_{t-s} + \cdots + \Phi_P y_{t-Ps} + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q} + \Theta_1 \epsilon_{t-s} + \cdots + \Theta_Q \epsilon_{t-Qs} + \epsilon_t.$$

Where:

- p, P are non-seasonal and seasonal AR orders
- q, Q are non-seasonal and seasonal MA orders
- s is the seasonal period

Value

Design matrix with lagged values. Columns are ordered as:

- First p columns: non-seasonal AR lags $(y_{t-1}, \ldots, y_{t-p})$
- Next P columns: seasonal AR lags $(y_{t-s}, \ldots, y_{t-Ps})$
- Next q columns: non-seasonal MA lags $(\epsilon_{t-1}, \ldots, \epsilon_{t-q})$
- Next Q columns: seasonal MA lags $(\epsilon_{t-s}, \ldots, \epsilon_{t-Qs})$
- If multiplicative=TRUE: AR cross-terms then MA cross-terms

Examples

```
# Simple SARMA(1,0) x (1,0)_12 model (AR+SAR, no MA)
x <- rnorm(120)
residuals <- rnorm(120)
X <- create_sarma_matrix(x, residuals, p = 1, P = 1, q = 0, Q = 0, s = 12)

# Full SARMA(1,1) x (1,1)_12 model
X <- create_sarma_matrix(x, residuals, p = 1, P = 1, q = 1, Q = 1, s = 12)
```

DCOILWTICO *WTI Crude Oil Prices*

Description

Daily spot prices for West Texas Intermediate (WTI) crude oil in U.S. dollars per barrel.

Usage

```
DCOILWTICO
```

Format

A data frame with observations for each trading day:

observation_date Date of observation in YYYY-MM-DD format

DCOILWTICO Crude Oil Price: West Texas Intermediate (WTI) in USD per barrel

Source

Federal Reserve Economic Data (FRED), Federal Reserve Bank of St. Louis https://fred.stlouisfed.org/series/DCOILWTICO

Examples

```
data(DCOILWTICO)
head(DCOILWTICO)
summary(DCOILWTICO$DCOILWTICO)
```

djia2002 *Dow Jones Industrial Average Daily Data (July-December 2002)*

Description

Daily closing prices and changes of the Dow Jones Industrial Average for the second half of 2002. Used in published PMM2 research to demonstrate AR(1) estimation with asymmetric innovations.

Usage

```
djia2002
```

Format

A data frame with 127 rows and 3 variables:

date Trading date

close DJIA closing price (USD)

change Daily change in closing price (first difference; NA for the first observation)

Details

The daily changes exhibit positive skewness, making this dataset suitable for PMM2 estimation. In the original paper, an AR(1) model fitted to the change series yielded PMM2 coefficient a1 = -0.43 versus OLS a1 = -0.49, with g2 = 0.77 (23\

Source

Yahoo Finance via the quantmod R package.

References

Zabolotnii S., Tkachenko O., Warsza Z.L. (2022) Application of the Polynomial Maximization Method for Estimation Parameters of Autoregressive Models with Asymmetric Innovations. Springer AISC, vol 1427. doi:10.1007/9783031035029_37

Examples

```
data(djia2002)
# AR(1) with PMM2
changes <- na.omit(djia2002$change)
pmm_skewness(changes)  # positive skewness -> PMM2
fit <- ar_pmm2(changes, order = 1)
summary(fit)
```

---

fitted,PMM2fit-method *Extract fitted values from PMM2fit object*

Description

Extract fitted values from PMM2fit object

Usage

```
## S4 method for signature 'PMM2fit'
fitted(object, data = NULL, ...)
```

Arguments

object	PMM2fit object
data	Optional data source for model reconstruction, if object does not contain saved data
...	Additional arguments (not used)

Value

Vector of fitted values

fitted,PMM3fit-method *Extract fitted values from PMM3fit object*

Description

Extract fitted values from PMM3fit object

Usage

```
## S4 method for signature 'PMM3fit'
fitted(object, ...)
```

Arguments

object	PMM3fit object
...	Additional arguments (not used)

Value

Vector of fitted values

fitted,TS2fit-method *Extract fitted values from TS2fit object*

Description

Extract fitted values from TS2fit object

Usage

```
## S4 method for signature 'TS2fit'
fitted(object, ...)
```

Arguments

object	TS2fit object
...	Additional arguments (not used)

Value

Vector of fitted values

fitted,TS3fit-method *Extract fitted values from TS3fit object*

Description

Extract fitted values from TS3fit object

Usage

```
## S4 method for signature 'TS3fit'
fitted(object, ...)
```

Arguments

object	TS3fit object
...	Additional arguments (not used)

Value

Vector of fitted values

format.PMMdispatch *Format method for PMMdispatch objects*

Description

Format method for PMMdispatch objects

Usage

```
## S3 method for class 'PMMdispatch'
format(x, ...)
```

Arguments

x	object of class PMMdispatch
...	additional arguments (ignored)

Value

A character string summarising the selected method.

get_sarimax_jacobian *Calculate SARIMAX Jacobian (Numerical)*

Description

Calculate SARIMAX Jacobian (Numerical)

Usage

```
get_sarimax_jacobian(theta, ...)
```

Arguments

theta	Parameters
...	Arguments passed to get_sarimax_residuals

Value

Jacobian matrix (n x p)

get_sarimax_residuals *Calculate SARIMAX Residuals*

Description

Calculate SARIMAX Residuals

Usage

```
get_sarimax_residuals(
  theta,
  y,
  xreg = NULL,
  order = c(0, 0, 0),
  seasonal = list(order = c(0, 0, 0), period = NA),
  include.mean = TRUE
)
```

Arguments

theta	Combined vector of parameters (AR, MA, SAR, SMA, Intercept, Regressors)
y	Time series data
xreg	Exogenous regressors (optional)
order	ARIMA order c(p, d, q)
seasonal	Seasonal order c(P, D, Q)
include.mean	Boolean, whether to include mean/intercept
period	Seasonal period

Value

Vector of residuals

`lm_pmm2` *PMM2: Main function for PMM2 (S=2)*

Description

Fits a linear model using the Polynomial Maximization Method (order 2), which is robust to non-Gaussian errors.

Usage

```
lm_pmm2(
  formula,
  data,
  max_iter = 50,
  tol = 1e-06,
  regularize = TRUE,
  reg_lambda = 1e-08,
  na.action = na.fail,
  weights = NULL,
  verbose = FALSE
)
```

Arguments

`formula`	R formula for the model
`data`	data.frame containing variables in the formula
`max_iter`	integer: maximum number of iterations for the algorithm
`tol`	numeric: tolerance for convergence, compared with the root-mean-square change of the fitted values in a Newton step, in units of the OLS residual standard deviation
`regularize`	logical: add small value to diagonal for numerical stability
`reg_lambda`	numeric: regularization parameter (if regularize=TRUE); scaled down with the Jacobian when its mean absolute diagonal element is below 1, so that it does not dominate when y is measured in small units
`na.action`	function for handling missing values, default is na.fail
`weights`	optional weight vector (not yet implemented)
`verbose`	logical: whether to print progress information

Details

The PMM2 algorithm works as follows:

1. Fits ordinary least squares (OLS) regression to obtain initial estimates
2. Computes central moments (m2, m3, m4) from OLS residuals
3. Iteratively improves parameter estimates using a gradient-based approach

PMM2 is especially useful when error terms are not Gaussian.

Value

S4 object of class `PMM2fit`

Examples

```
set.seed(123)
n <- 80
x <- rnorm(n)
y <- 2 + 3 * x + rt(n, df = 3)
dat <- data.frame(y = y, x = x)

fit <- lm_pmm2(y ~ x, data = dat)
summary(fit, formula = y ~ x, data = dat)
```

`lm_pmm3` *PMM3: Fit linear model using Polynomial Maximization Method (S=3)*

Description

Fits a linear model using PMM3, which is designed for symmetric platykurtic error distributions. Uses a cubic stochastic polynomial with Newton-Raphson solver.

Usage

```
lm_pmm3(
  formula,
  data,
  max_iter = 100,
  tol = 1e-06,
  adaptive = FALSE,
  step_max = 5,
  na.action = na.fail,
  verbose = FALSE,
  symmetric = TRUE
)
```

Arguments

`formula`	R formula for the model
`data`	data.frame containing variables in the formula
`max_iter`	integer: maximum number of NR iterations (default 100)
`tol`	numeric: convergence tolerance (default 1e-6), compared with the Newton step measured like `step_max`
`adaptive`	logical: re-estimate kappa each iteration (default FALSE)
`step_max`	numeric: maximum NR step, measured as the root-mean-square change of the fitted values in units of the OLS residual standard deviation (default 5.0)
`na.action`	function for handling missing values (default na.fail)
`verbose`	logical: print progress information (default FALSE)
`symmetric`	logical: `TRUE` (default) fits the symmetric PMM3 estimator described below; `FALSE` fits the general degree-3 estimator, which also uses the odd moments and is valid for skewed errors.

Details

The PMM3 algorithm works as follows:

1. Fits OLS regression to obtain initial estimates
2. Computes central moments (m2, m4, m6) from OLS residuals
3. Checks symmetry: warns if |gamma3| exceeds both 0.3 and its sampling noise $2\sqrt{6/n}$, the bound `pmm_dispatch` uses (PMM2 may be more appropriate)
4. Computes moment ratio kappa = (m6 - 3*m4*m2) / (m4 - 3*m2^2)
5. Iterates Newton-Raphson with score Z = X'(eps*(kappa - eps^2))

PMM3 achieves lower variance than OLS when errors are symmetric and platykurtic (gamma4 < 0), with theoretical efficiency g3 = 1 - gamma4^2 / (6 + 9*gamma4 + gamma6).

The symmetric estimator assumes $\gamma_3 = \gamma_5 = 0$. On skewed errors it remains consistent for the slopes but loses most of its gain. With chi-squared(3), gamma, exponential and lognormal errors its OLS-to-PMM3 MSE ratio is 1.1 to 1.5, against 1.7 to 2.4 for PMM2 and 2.1 to 3.8 for the general estimator (`dev_scripts/check_pmm3_general.R`). With `symmetric = FALSE` the polynomial is $\psi(e) = \sum_{k=1}^{3} h_k(e^k - \alpha_k)$, where α_k are the central moments of the OLS residuals and $h = F^{-1}b$ solves the full Kunchenko normal system with $F_{pq} = \alpha_{p+q} - \alpha_p\alpha_q$ and $b_k = k\alpha_{k-1}$. Its variance-reduction factor is $g_3 = 1/(\alpha_2\, b^\top F^{-1} b)$. This factor never exceeds the PMM2 factor g_2, and it equals the symmetric g_3 when the odd moments vanish. The Newton iteration uses backtracking on the score norm and falls back to OLS with a warning when it fails. `adaptive = TRUE` re-estimates the moments and h at every iteration.

Value

S4 object of class `PMM3fit`

Examples

```
set.seed(123)
n <- 100
x <- rnorm(n)
y <- 2 + 3 * x + runif(n, -sqrt(3), sqrt(3))
dat <- data.frame(y = y, x = x)

fit <- lm_pmm3(y ~ x, data = dat)
summary(fit)

# Skewed errors: the general degree-3 estimator
dat$y2 <- 2 + 3 * x + rgamma(n, 2, 1) - 2
fit_g <- lm_pmm3(y2 ~ x, data = dat, symmetric = FALSE)
coef(fit_g)
```

`lm_wpmm2` *Weak-moment PMM2 and PMM3 for linear models*

Description

Fit a linear model by the polynomial maximization method with windowed (weak) moments in place of raw residual moments. `lm_wpmm2` is the degree-2 estimator for skewed errors, `lm_wpmm3` the degree-3 estimator for symmetric heavy-tailed errors. Both are meant for residuals whose raw cumulants are unstable: heavy tails, outliers, or moments that do not exist.

Usage

```
lm_wpmm2(
  formula,
  data,
  sigma_mult = 2.5,
  max_iter = 50,
  tol = 1e-07,
  na.action = na.fail
)

lm_wpmm3(
  formula,
  data,
  sigma_mult = 2.5,
  max_iter = 50,
  tol = 1e-07,
  na.action = na.fail
)
```

Arguments

formula	R formula for the model.
data	data.frame containing the variables in formula.
sigma_mult	window width as a multiple of the robust residual scale (default 2.5). A wider window moves the estimator towards classical PMM2/PMM3 and OLS; a narrower one makes it more robust.
max_iter	maximum number of IRLS iterations (default 50).
tol	convergence tolerance on the relative change of the coefficient vector (default 1e-7); the change is divided by the larger of the coefficient norm and min(1, norm of the OLS coefficients).
na.action	function for handling missing values (default na.fail).

Details

Starting from OLS, each iteration

1. centres a Gaussian window $w_i = \exp(-\frac{1}{2}((r_i - \mathrm{med}(r))/(\sigma\, s))^2)$ on the residual median, with s the MAD of the residuals and σ = sigma_mult;
2. computes windowed central moments of the residuals around the windowed mean, $r_i^c = r_i - \sum w_i r_i / \sum w_i$;
3. solves the weighted least-squares problem with weights $W_i = w_i(1+a_2 r_i^c)$, $a_2 = -c_3^w/(2(c_2^w)^2 + c_4^w)$ for WPMM2, or $W_i = w_i(\kappa^w - (r_i^c)^2)/|\kappa^w|$, $\kappa^w = (m_6^w - 3m_4^w m_2^w)/(m_4^w - 3(m_2^w)^2)$ for WPMM3. Negative weights are set to zero, so both estimators are redescending.

The estimates agree with the inner estimators fit_kg2_weak_pmm2 and fit_kg2_weak_pmm3 of the gmdhpmm package (EstemPMM-GMDH), which add a ridge penalty of 1e-8.

Two limitations follow from the construction.

- Under skewed errors the intercept estimates the location at which the windowed score vanishes. That location depends on sigma_mult, does not shrink as n grows, and is not the error mean. The slopes are unaffected. lm_wpmm2 does not remove this offset.
- κ^w is the coefficient of the windowed law. It ignores the derivative of the window, so it is not the variance-minimising coefficient, and the gap grows as the window narrows.

lm_wpmm3 targets symmetric leptokurtic errors. For symmetric platykurtic errors use lm_pmm3. When the windowed law is mesokurtic ($m_4^w \approx 3(m_2^w)^2$), lm_wpmm3 reduces to the Welsch M-estimator.

Standard errors (vcov) come from the M-estimator sandwich $A^{-1}BA^{-1}$ with $A = \sum \psi'(r_i) x_i x_i^\top$, $B = \sum \psi(r_i)^2 x_i x_i^\top$ and $\psi(r) = W(r)\, r$. The window centre, scale and weak moments are treated as fixed. For errors independent of the regressors this leaves the slope standard errors asymptotically correct. It can understate the intercept standard error.

Value

S4 object of class WPMMfit.

See Also

lm_pmm2, lm_pmm3, pmm_lm

Examples

```
set.seed(1)
n <- 200
x <- rnorm(n)
dat <- data.frame(x = x, y = 1 + 2 * x + rt(n, df = 3))
fit3 <- lm_wpmm3(y ~ x, data = dat)
summary(fit3)

dat$y2 <- 1 + 2 * x + (rexp(n) - 1) + 3 * rbinom(n, 1, 0.05)
fit2 <- lm_wpmm2(y2 ~ x, data = dat)
coef(fit2)
```

---

logLik.PMM2fit *Extract log-likelihood from PMM2fit object*

Description

Returns a Gaussian approximate log-likelihood. Defined as an S3 method so that `stats::AIC` and `stats::BIC` (which dispatch `logLik` via `UseMethod`) work without explicit S4 methods.

Usage

```
## S3 method for class 'PMM2fit'
logLik(object, ...)
```

Arguments

`object`	PMM2fit object
`...`	Additional arguments (not used)

Value

Object of class `logLik`

logLik.PMM3fit *Extract log-likelihood from PMM3fit object*

Description

Returns a Gaussian approximate log-likelihood. Defined as an S3 method so that `stats::AIC` and `stats::BIC` (which dispatch `logLik` via `UseMethod`) work without explicit S4 methods.

Usage

```
## S3 method for class 'PMM3fit'
logLik(object, ...)
```

Arguments

`object`	PMM3fit object
`...`	Additional arguments (not used)

Value

Object of class `logLik`

logLik.TS2fit *Extract log-likelihood from TS2fit object*

Description

Returns a Gaussian approximate log-likelihood. Non-finite residuals (which can arise from differencing or burn-in) are dropped so that AIC/BIC dispatched through this method match the convention of `compare_ts_methods()`. Defined as an S3 method so that `stats::AIC` and `stats::BIC` dispatch correctly via `UseMethod("logLik")`.

Usage

```
## S3 method for class 'TS2fit'
logLik(object, ...)
```

Arguments

`object`	TS2fit (or subclass) object
`...`	Additional arguments (not used)

Value

Object of class `logLik`

logLik.TS3fit *Extract log-likelihood from TS3fit object*

Description

Returns a Gaussian approximate log-likelihood. Defined as an S3 method so that `stats::AIC` and `stats::BIC` dispatch correctly via `UseMethod("logLik")`.

Usage

```
## S3 method for class 'TS3fit'
logLik(object, ...)
```

Arguments

`object`	TS3fit (or subclass) object
`...`	Additional arguments (not used)

Value

Object of class `logLik`

ma_pmm2 *Fit an MA model using PMM2 (wrapper)*

Description

Estimates moving-average model parameters using the Pearson Moment Method (PMM2). For MA models, PMM2 can achieve substantial improvements over MLE, particularly when innovations are non-Gaussian. Monte Carlo experiments showed up to 23\ reduction for MA(1) models.

Usage

```
ma_pmm2(
  x,
  order = 1,
  method = "pmm2",
  pmm2_variant = c("unified_global", "unified_iterative", "linearized"),
  ma_solver = c("linearized", "recursive"),
  max_iter = 50,
  tol = 1e-06,
  include.mean = TRUE,
  initial = NULL,
  na.action = na.fail,
  regularize = TRUE,
```

```
  reg_lambda = 1e-08,
  verbose = FALSE
)
```

Arguments

x	Numeric vector of time series data
order	Model order specification: • For AR models: a single integer (AR order) • For MA models: a single integer (MA order) • For ARMA models: vector c(p, q) (AR and MA orders) • For ARIMA models: vector c(p, d, q) (AR, differencing, and MA orders)
method	String: estimation method, one of "pmm2" (default), "css", "ml", "yw", "ols"
pmm2_variant	Character string specifying PMM2 implementation variant. Options: "unified_global" (default, one-step correction), "unified_iterative" (full Newton-Raphson), or "linearized" (specialized for MA/SMA models, **recommended for MA**).
ma_solver	String: how the moving-average recursion is treated, "linearized" (default) or "recursive". Has no effect when the MA order is zero. See Details.
max_iter	Integer: maximum number of iterations for the algorithm
tol	Numeric: tolerance for convergence
include.mean	Logical: whether to include a mean (intercept) term
initial	List or vector of initial parameter estimates (optional)
na.action	Function for handling missing values, default is na.fail
regularize	Logical, add small values to diagonal for numerical stability
reg_lambda	Regularization parameter (if regularize=TRUE)
verbose	Logical: whether to print progress information

Details

PMM2 Variants:

- "unified_global" (default): One-step correction from MLE/CSS estimates. Fast and reliable. Typical improvement: 15-23\
- "unified_iterative": Full Newton-Raphson optimization. Slower but can achieve better accuracy for complex MA models.
- "linearized": Uses first-order Taylor expansion around MLE. **Recommended for MA/SMA models** as it avoids complex Jacobian computation while maintaining accuracy. Fastest option for MA models.

Variant Selection Guidelines:

- For MA(q) models: Use "linearized" (fastest, MA-optimized)
- If you need maximum accuracy: Try "unified_global" (best MSE)
- For exploration: Compare all three variants

Computational Characteristics:

- `linearized`: ~1.5x slower than MLE (recommended)
- `unified_global`: ~2x slower than MLE (high accuracy)
- `unified_iterative`: 5-10x slower than MLE (iterative)

Why MA Models Benefit Most: MA parameter estimation from MLE has known numerical difficulties due to non-identifiability and flat likelihood regions. PMM2 uses moment constraints that better resolve these issues, leading to larger improvements than for AR models.

Value

An S4 object of class `MAPMM2` containing moving-average coefficients, residual innovations, central moments, model order, intercept, original series, and convergence diagnostics.

References

Monte Carlo validation (R=50, n=200): Unified Global showed 23.0\ improvement over MLE for MA(1) models - the largest improvement among all model types. See NEWS.md (Version 0.2.0) for full comparison.

See Also

`ar_pmm2`, `arma_pmm2`, `sma_pmm2`

Examples

```
# Fit MA(1) model with linearized variant (recommended)
x <- arima.sim(n = 200, list(ma = 0.6))
fit1 <- ma_pmm2(x, order = 1, pmm2_variant = "linearized")
coef(fit1)

# Compare with unified_global (best accuracy)
fit2 <- ma_pmm2(x, order = 1, pmm2_variant = "unified_global")

# Higher-order MA
x2 <- arima.sim(n = 300, list(ma = c(0.7, -0.4, 0.2)))
fit3 <- ma_pmm2(x2, order = 3, pmm2_variant = "linearized")
```

`ma_pmm3` *Fit an MA model using PMM3*

Description

Estimates moving-average model parameters using PMM3.

Usage

```
ma_pmm3(
  x,
  order = 1,
  max_iter = 100,
  tol = 1e-06,
  adaptive = FALSE,
  step_max = 5,
  include.mean = TRUE,
  initial = NULL,
  na.action = na.fail,
  verbose = FALSE
)
```

Arguments

x	Numeric vector of time series data
order	Integer: AR order (default 1)
max_iter	Integer: maximum NR iterations (default 100)
tol	Numeric: convergence tolerance (default 1e-6)
adaptive	Logical: re-estimate kappa each iteration (default FALSE)
step_max	Numeric: maximum NR step (default 5.0); see ts_pmm3
include.mean	Logical: include mean term (default TRUE)
initial	Optional initial parameter estimates
na.action	Function for handling missing values (default na.fail)
verbose	Logical: print progress information (default FALSE)

Value

An S4 object of class MAPMM3

See Also

ma_pmm2, ar_pmm3, lm_pmm3

Examples

```
set.seed(42)
x <- arima.sim(n = 200, list(ma = 0.6),
               innov = runif(200, -sqrt(3), sqrt(3)))
fit <- ma_pmm3(x, order = 1)
coef(fit)
```

MAPMM2-class *S4 class for storing PMM2 MA model results*

Description

S4 class for storing PMM2 MA model results

MAPMM3-class *S4 class for PMM3 MA model results*

Description

S4 class for PMM3 MA model results

nobs,PMM2fit-method *Number of observations in PMM2fit object*

Description

Number of observations in PMM2fit object

Usage

```
## S4 method for signature 'PMM2fit'
nobs(object, ...)
```

Arguments

object	PMM2fit object
...	Additional arguments (not used)

Value

Integer number of observations

nobs,PMM3fit-method *Number of observations in PMM3fit object*

Description

Number of observations in PMM3fit object

Usage

```
## S4 method for signature 'PMM3fit'
nobs(object, ...)
```

Arguments

object	PMM3fit object
...	Additional arguments (not used)

Value

Integer number of observations

nobs,TS2fit-method *Number of observations in TS2fit object*

Description

Returns the effective sample size (length of the residual vector).

Usage

```
## S4 method for signature 'TS2fit'
nobs(object, ...)
```

Arguments

object	TS2fit (or subclass) object
...	Additional arguments (not used)

Value

Integer effective sample size

nobs,TS3fit-method *Number of observations in TS3fit object*

Description

Number of observations in TS3fit object

Usage

```
## S4 method for signature 'TS3fit'
nobs(object, ...)
```

Arguments

object	TS3fit (or subclass) object
...	Additional arguments (not used)

Value

Integer effective sample size

plot,PMM2fit,missing-method *Plot diagnostic plots for PMM2fit object*

Description

Plot diagnostic plots for PMM2fit object

Usage

```
## S4 method for signature 'PMM2fit,missing'
plot(x, y, which = 1:4, ...)
```

Arguments

x	PMM2fit object
y	Not used (compatibility with generic)
which	Set of plots to display (values 1-4)
...	Additional arguments passed to plotting functions

Value

Invisibly returns the input object

`plot,PMM3fit,missing-method`

Plot diagnostic plots for PMM3fit object

Description

Plot diagnostic plots for PMM3fit object

Usage

```
## S4 method for signature 'PMM3fit,missing'
plot(x, y, which = 1:4, ...)
```

Arguments

`x`	PMM3fit object
`y`	Not used (compatibility with generic)
`which`	Set of plots to display (values 1-4)
`...`	Additional arguments passed to plotting functions

Value

Invisibly returns the input object

`plot,TS2fit,missing-method`

Build diagnostic plots for TS2fit objects

Description

Build diagnostic plots for TS2fit objects

Usage

```
## S4 method for signature 'TS2fit,missing'
plot(x, y, which = c(1:4), ...)
```

Arguments

`x`	TS2fit object
`y`	Not used (for S4 method compatibility)
`which`	Integer vector indicating which plots to produce
`...`	additional arguments passed to plot functions

Value

Invisibly returns x

`plot,TS3fit,missing-method` *Plot diagnostic plots for TS3fit object*

Description

Plot diagnostic plots for TS3fit object

Usage

```
## S4 method for signature 'TS3fit,missing'
plot(x, y, which = 1:4, ...)
```

Arguments

`x`	TS3fit object
`y`	Not used
`which`	Set of plots to display (values 1-4)
`...`	Additional arguments

Value

Invisibly returns the input object

`plot_pmm2_bootstrap` *Plot bootstrap distributions for PMM2 fit*

Description

Plot bootstrap distributions for PMM2 fit

Usage

```
plot_pmm2_bootstrap(object, coefficients = NULL)
```

Arguments

`object`	Result from pmm2_inference
`coefficients`	Which coefficients to plot, defaults to all

Value

Invisibly returns histogram information

pmm_ar — *Fit an AR model with the polynomial maximization method*

Description

Unified entry point for PMM2 and PMM3 autoregressive estimation.

Usage

```
pmm_ar(x, order, method = c("pmm2", "pmm3"), ...)
```

Arguments

x	univariate time series or numeric vector.
order	non-negative integer giving the autoregressive order.
method	One of "pmm2" or "pmm3".
...	Additional arguments forwarded to ar_pmm2 or ar_pmm3.

Value

An S4 object of class ARPMM2 (for method = "pmm2") or ARPMM3.

pmm_arima — *Fit an ARIMA model with the polynomial maximization method*

Description

Unified entry point for PMM2 and PMM3 ARIMA estimation. This is the recommended primary API for non-Gaussian ARIMA modelling and the interface featured in the JSS submission accompanying the package.

Usage

```
pmm_arima(x, order, method = c("pmm2", "pmm3"), ...)
```

Arguments

x	univariate time series or numeric vector.
order	integer vector c(p, d, q) giving the AR order, degree of differencing, and MA order.
method	One of "pmm2" or "pmm3".
...	Additional arguments forwarded to arima_pmm2 or arima_pmm3.

Value

An S4 object of class ARIMAPMM2 or ARIMAPMM3.

Examples

```
set.seed(2026)
x <- arima.sim(list(ar = 0.6), n = 200,
               rand.gen = function(k) rgamma(k, 2, 1) - 2)
pmm_arima(x, order = c(1, 0, 0), method = "pmm2")
```

---

pmm_arma *Fit an ARMA model with the polynomial maximization method*

Description

Unified entry point for PMM2 and PMM3 ARMA estimation.

Usage

```
pmm_arma(x, order, method = c("pmm2", "pmm3"), ...)
```

Arguments

x	univariate time series or numeric vector.
order	integer vector c(p, q) giving the AR and MA orders.
method	One of "pmm2" or "pmm3".
...	Additional arguments forwarded to arma_pmm2 or arma_pmm3.

Value

An S4 object of class ARMAPMM2 or ARMAPMM3.

---

pmm_dispatch *Automatic PMM method selection*

Description

Analyses OLS residual cumulants to recommend the best estimation method: OLS (Gaussian errors), PMM2 (asymmetric errors), or PMM3 (symmetric errors with non-zero excess kurtosis).

Usage

```
pmm_dispatch(
  residuals,
  symmetry_threshold = 0.3,
  kurtosis_threshold = -0.7,
  g2_threshold = 0.95,
  verbose = TRUE,
  g3_threshold = 0.95,
  n_leptokurtic = 100,
  clear_asymmetry_z = 2
)
```

Arguments

`residuals`	numeric vector of OLS residuals
`symmetry_threshold`	numeric: |gamma3| threshold for symmetry (default 0.3)
`kurtosis_threshold`	numeric: gamma4 threshold for PMM3 (default -0.7)
`g2_threshold`	numeric: minimum g2 improvement to justify PMM2 (default 0.95)
`verbose`	logical: print decision reasoning (default TRUE)
`g3_threshold`	numeric: symmetric residuals with g3 below this value go to PMM3 once `n >= n_leptokurtic` (default 0.95)
`n_leptokurtic`	integer: smallest sample size for that PMM3 branch (default 100; below it PMM3 is unstable on heavy-tailed errors)
`clear_asymmetry_z`	numeric: multiple of $\sqrt{6/n}$ that $\lvert\gamma_3\rvert$ must exceed for skewed platykurtic residuals to go to PMM2 rather than PMM3 (default 2)

Details

The rule, in order:

1. $|\gamma_3| > 1$: PMM2.
2. $|\gamma_3|$ above `symmetry_threshold` and γ_4 below `kurtosis_threshold`: PMM2 when the skewness is also clear of its sampling noise, $|\gamma_3| > z\sqrt{6/n}$ with `z = clear_asymmetry_z`; otherwise PMM3. In small samples platykurtic residuals often show a spurious skewness above 0.3.
3. $|\gamma_3|$ above `symmetry_threshold` and g_2 below `g2_threshold`: PMM2.
4. Symmetric and γ_4 below `kurtosis_threshold`: PMM3.
5. Symmetric, g_3 below `g3_threshold` and $n \geq$ `n_leptokurtic`: PMM3. This covers symmetric heavy-tailed errors (logistic, Laplace, Student t).
6. Otherwise OLS.

The defaults of `clear_asymmetry_z`, `g3_threshold` and `n_leptokurtic` come from a Monte Carlo calibration over 20 error laws, n from 30 to 1000 and 1, 5 or 10 regressors (`dev_scripts/calibrate_dispatch.R` and `calibrate_dispatch_p.R` in the source repository). Against the 0.5.0 rule they halve the mean efficiency loss relative to the best of OLS, PMM2 and PMM3 and cut the worst case from 2.7 to 1.2. Neither $n - p$ instead of n nor a minimum n/p improved the rule, so it uses the residuals alone. Steps 2 and 5 can be switched off with `clear_asymmetry_z = 0` and `g3_threshold = 0`, which reproduces the 0.5.0 rule.

Value

An object of S3 class `"PMMdispatch"`: a list with components accessible via `$` and a dedicated `print` method. Components:

`method`	Character: "OLS", "PMM2", or "PMM3"
`gamma3`	Sample skewness
`gamma4`	Sample excess kurtosis
`gamma6`	Sample 6th cumulant coefficient
`g2`	PMM2 efficiency factor
`g3`	PMM3 efficiency factor
`g_selected`	Efficiency factor for chosen method
`improvement_pct`	Expected variance reduction percentage
`reasoning`	Human-readable decision explanation
`n`	Sample size

Examples

```
set.seed(42)
x <- rnorm(200); eps <- runif(200, -1, 1)
y <- 1 + 2 * x + eps
fit_ols <- lm(y ~ x)
pmm_dispatch(residuals(fit_ols))
```

---

pmm_gamma6 *Compute sixth-order cumulant coefficient gamma6*

Description

Calculates $\gamma_6 = m_6/m_2^3 - 15m_4/m_2^2 + 30$ from a numeric vector. For a Gaussian distribution gamma6 equals zero.

Usage

```
pmm_gamma6(x)
```

Arguments

x	numeric vector

Value

Numeric scalar: the sixth-order cumulant coefficient

---

pmm_kurtosis *Calculate kurtosis from data*

Description

Calculate kurtosis from data

Usage

```
pmm_kurtosis(x, excess = TRUE)
```

Arguments

x	numeric vector
excess	logical, whether to return excess kurtosis (kurtosis - 3)

Value

Kurtosis value

---

pmm_lm *Fit a linear model with the polynomial maximization method*

Description

Unified entry point for PMM2 and PMM3 linear regression. Dispatches to `lm_pmm2` or `lm_pmm3` based on the `method` argument. With `method = "auto"`, the `pmm_dispatch` rule is applied to the residuals of an initial OLS fit to select the recommended method.

Usage

```
pmm_lm(
  formula,
  data,
  method = c("auto", "pmm2", "pmm3", "wpmm2", "wpmm3"),
  ...
)
```

Arguments

formula	R formula for the model.
data	data.frame containing the variables in formula.
method	One of "auto", "pmm2", "pmm3", "wpmm2" or "wpmm3". "auto" consults pmm_dispatch and falls back to PMM2 when the dispatcher recommends OLS, so the return value is always a PMMfit object. In that case a message says so. The dispatcher's decision is stored in the attribute "dispatch" of the fit. "auto" never selects the weak-moment estimators.
...	Additional arguments forwarded to the underlying fitter (lm_pmm2, lm_pmm3, lm_wpmm2 or lm_wpmm3). With method = "auto", named arguments the selected fitter does not take (for example reg_lambda when PMM3 is selected) are dropped with a message.

Value

An S4 object of class PMM2fit, PMM3fit or WPMMfit, all inheriting from PMMfit.

See Also

pmm_dispatch, pmm_arima, lm_pmm2, lm_pmm3.

Examples

```
set.seed(123)
n <- 80
x <- rnorm(n)
y <- 2 + 3 * x + rgamma(n, 2, 1) - 2
dat <- data.frame(y = y, x = x)
fit <- pmm_lm(y ~ x, data = dat, method = "pmm2")
fit
```

---

pmm_ma *Fit an MA model with the polynomial maximization method*

Description

Unified entry point for PMM2 and PMM3 moving-average estimation.

Usage

```
pmm_ma(x, order, method = c("pmm2", "pmm3"), ...)
```

Arguments

x	univariate time series or numeric vector.
order	non-negative integer giving the moving-average order.
method	One of "pmm2" or "pmm3".
...	Additional arguments forwarded to ma_pmm2 or ma_pmm3.

Value

An S4 object of class MAPMM2 or MAPMM3.

---

pmm_sarima *Fit a seasonal ARIMA model with the polynomial maximization method*

Description

Unified entry point for SARIMA-PMM2 estimation. PMM3 seasonal estimation is not yet implemented; method = "pmm3" therefore raises an error. All arguments other than method are forwarded verbatim to sarima_pmm2 – see that function for the seasonal order specification.

Usage

```
pmm_sarima(x, ..., method = c("pmm2", "pmm3"))
```

Arguments

x	univariate time series or numeric vector.
...	Arguments forwarded to sarima_pmm2 (notably order and seasonal).
method	Currently only "pmm2" is supported.

Value

An S4 object of class SARIMAPMM2.

pmm_skewness *Calculate skewness from data*

Description

Calculate skewness from data

Usage

```
pmm_skewness(x)
```

Arguments

x	numeric vector

Value

Skewness value

pmm2_inference *Bootstrap inference for PMM2 fit*

Description

Residual-bootstrap standard errors, p-values, and confidence intervals for a PMM2 regression fit. Three CI methods are available (see `ci_method`).

Usage

```
pmm2_inference(
  object,
  formula,
  data,
  B = 200,
  seed = NULL,
  parallel = FALSE,
  cores = NULL,
  ci_method = c("normal", "percentile", "basic")
)
```

Arguments

`object`	object of class PMM2fit
`formula`	the same formula that was used initially
`data`	data frame that was used initially
`B`	number of bootstrap replications
`seed`	(optional) for reproducibility
`parallel`	logical, whether to use parallel computing
`cores`	number of cores to use for parallel computing, defaults to auto-detect
`ci_method`	character: confidence-interval method.

"normal" (default) Symmetric Wald interval $\hat{\beta} \pm z_{1-\alpha/2}\,\widehat{\mathrm{SE}}$ using the bootstrap standard deviation as $\widehat{\mathrm{SE}}$. Always contains the point estimate and is robust to mild finite-sample bias of the bootstrap distribution.

"percentile" Empirical 2.5/97.5 percentiles of the bootstrap re-estimates (Efron's original percentile method). Not centred on the point estimate when the bootstrap distribution is biased.

"basic" Davison & Hinkley (1997) basic/pivotal interval $[2\hat{\beta} - q_{1-\alpha/2},\ 2\hat{\beta} - q_{\alpha/2}]$, reflecting the percentile bracket about the point estimate.

The returned object includes a `bias` column ($\mathrm{mean}(\text{boot}) - \hat{\beta}$) so users can detect finite-sample bias and switch CI methods if desired.

Value

data.frame with columns: Estimate, Std.Error, bias, t.value, p.value, conf.low, conf.high.

`pmm2_monte_carlo_compare` *Monte Carlo comparison of PMM2 estimation methods*

Description

Function generates time series for given models, repeatedly estimates parameters using different methods and compares their accuracy by MSE criterion. Additionally outputs theoretical and empirical characteristics of the innovation distribution (skewness, excess kurtosis, theoretical gain of PMM2).

Usage

```
pmm2_monte_carlo_compare(
  model_specs,
  methods = c("css", "pmm2"),
  n,
  n_sim,
  innovations = list(type = "gaussian"),
```

```
  seed = NULL,
  include.mean = TRUE,
  progress = interactive(),
  verbose = FALSE
)
```

Arguments

model_specs	List of model specifications. Each element must contain: **model** "ar", "ma" or "arma" **order** order (for AR/MA) or vector c(p, q) for ARMA **theta** numeric vector of true parameters; for ARMA a list `list(ar = ..., ma = ...)` **label** (optional) model name in report **innovations** (optional) description of innovation distribution: `list(type = "gamma", shape = 2)`, `list(type = "student_t", df = 5)`, etc. Can also pass an arbitrary generation function via `generator`.
methods	Vector of estimation methods (e.g., `c("css","pmm2")`). The first method is considered baseline for relative MSE calculation.
n	Sample size for simulation.
n_sim	Number of Monte Carlo experiments.
innovations	Function or distribution description, used by default for all models (if not specified in spec).
seed	Initial seed for random number generator (optional).
include.mean	Logical flag: whether to include intercept during estimation.
progress	Logical flag: print Monte Carlo progress.
verbose	Whether to print diagnostic messages on failures.

Value

List with three components:

parameter_results MSE and relative MSE for each parameter

summary Averaged MSE over parameters for each model/method

gain Comparison of theoretical and empirical PMM2 gain

pmm2_nonlinear_iterative

Universal PMM2 estimator (Iterative)

Description

Universal PMM2 estimator (Iterative)

Usage

```
pmm2_nonlinear_iterative(
  theta_init,
  fn_residuals,
  fn_jacobian = NULL,
  max_iter = 100,
  tol = 1e-06,
  verbose = FALSE
)
```

Arguments

theta_init	Initial parameter values
fn_residuals	Function(theta) returning the residual vector
fn_jacobian	Function(theta) returning the Jacobian matrix (n x p). J_ij = d(y_hat_i)/d(theta_j) = -d(epsilon_i)/d(theta_j). If NULL, numerical Jacobian via numDeriv is used
max_iter	Maximum number of iterations
tol	Convergence tolerance
verbose	Print progress

Value

List with results (theta, residuals, convergence, etc.)

pmm2_nonlinear_onestep

Universal PMM2 estimator (One-step Global)

Description

Applies a single PMM2 correction to classical estimation results.

Usage

```
pmm2_nonlinear_onestep(
  theta_classical,
  fn_residuals,
  fn_jacobian = NULL,
  verbose = FALSE
)
```

Arguments

theta_classical	Parameter estimates from a classical method (e.g., MLE)
fn_residuals	Function(theta) returning the residual vector
fn_jacobian	Function(theta) returning the Jacobian matrix. If NULL, numerical Jacobian via numDeriv is used
verbose	Print progress

Value

List with results (theta, residuals, etc.)

---

pmm2_variance_factor *Calculate theoretical skewness, kurtosis coefficients and variance reduction factor*

Description

Calculate theoretical skewness, kurtosis coefficients and variance reduction factor

Usage

```
pmm2_variance_factor(m2, m3, m4)
```

Arguments

m2, m3, m4	central moments of second, third and fourth orders

Value

List with fields c3, c4 and g

pmm2_variance_matrices *Calculate theoretical variance matrices for OLS and PMM2*

Description

Calculate theoretical variance matrices for OLS and PMM2

Usage

```
pmm2_variance_matrices(X, m2, m3, m4)
```

Arguments

X	Design matrix with column of ones
m2, m3, m4	central moments of OLS residuals

Value

List with fields ols, pmm2, c3, c4, g

PMM2fit-class *S4 class for storing PMM2 regression model results*

Description

Class for storing results of linear model estimation using PMM2

Slots

coefficients numeric vector of estimated parameters

residuals numeric vector of final residuals

m2 numeric second central moment of initial residuals

m3 numeric third central moment of initial residuals

m4 numeric fourth central moment of initial residuals

convergence logical or integer code indicating whether algorithm converged

iterations numeric number of iterations performed

call original function call

Slots

coefficients Estimated coefficients

residuals Final residuals

m2 Second central moment

m3 Third central moment

m4 Fourth central moment

convergence Convergence status

iterations Number of iterations performed

call Original call

---

pmm3_variance_factor *Calculate PMM3 theoretical variance reduction factor*

Description

Computes the standardised cumulant coefficients gamma4 and gamma6 from the raw central moments, and derives the PMM3 efficiency factor $g_3 = 1 - \gamma_4^2/(6 + 9\gamma_4 + \gamma_6)$.

Usage

```
pmm3_variance_factor(m2, m4, m6)
```

Arguments

m2	Second central moment
m4	Fourth central moment
m6	Sixth central moment

Value

A list with components:

gamma4	Excess kurtosis
gamma6	Sixth-order cumulant coefficient
g3	Theoretical variance reduction factor

pmm3_variance_matrices *Calculate theoretical variance matrices for OLS and PMM3*

Description

Returns the asymptotic covariance matrices $V_{\mathrm{OLS}} = m_2 (X^\top X)^{-1}$ and $V_{\mathrm{PMM3}} = g_3 \, m_2 (X^\top X)^{-1}$ where $g_3 = 1 - \gamma_4^2 / (6 + 9\gamma_4 + \gamma_6)$.

Usage

```
pmm3_variance_matrices(X, m2, m4, m6)
```

Arguments

`X`	Design matrix with column of ones for the intercept.
`m2`, `m4`, `m6`	Second, fourth, and sixth central moments computed from the initial OLS residuals.

Details

The PMM3 formula is the symmetric-platykurtic specialisation of Kunchenko's polynomial-maximisation framework (s = 3); see `notes/pmm3_vcov_derivation.md` for the derivation and Zabolotnii et al. (2018, 2022, 2023) for the general method. Under Gaussian errors $g_3 = 1$ and the matrix coincides with the OLS covariance.

Value

A list with components `ols` (OLS covariance matrix), `pmm3` (PMM3 covariance matrix), and the scalar efficiency diagnostics `gamma4`, `gamma6`, `g3`.

PMM3fit-class *S4 class for PMM3 regression fit results*

Description

Concrete class returned by `lm_pmm3`. Inherits all slots from `BasePMM3`, which in turn inherits common slots from `PMMfit`.

Slots

`symmetric` logical: `TRUE` for the symmetric estimator (cubic polynomial with even-moment coefficients), `FALSE` for the general degree-3 estimator of `lm_pmm3(symmetric = FALSE)`

`poly_coef` numeric: coefficients h_1, h_2, h_3 of the general estimator's polynomial; empty for the symmetric estimator

PMMfit-class *Virtual root S4 class for PMM fit objects*

Description

Common parent of every concrete fit class returned by the package (PMM2fit, PMM3fit and the time-series subclasses). Holds the slots that are meaningful regardless of polynomial order or model family.

Details

This class is virtual and cannot be instantiated directly; it exists so that S4 methods (e.g. `show`) can be defined once and inherited by every concrete subclass.

Slots

`coefficients` numeric vector of estimated parameters

`residuals` numeric vector of final residuals/innovations

`convergence` logical or integer indicating algorithm convergence

`iterations` numeric number of iterations performed

`call` original function call

PMMtsfit-class *Virtual S4 class for PMM time-series fit objects*

Description

Common parent of every concrete time-series fit class (TS2fit, TS3fit and their AR/MA/ARMA/ARIMA/seasonal subclasses). Provides the time-series-specific slots so that methods such as `predict`, `plot`, and `forecast` can be defined once and inherited.

Details

This class is virtual and inherits from `PMMfit`.

Slots

`model_type` character string identifying the model family (one of `"ar"`, `"ma"`, `"arma"`, `"arima"`, `"sar"`, `"sma"`, `"sarma"`, `"sarima"`)

`intercept` numeric scalar intercept/mean term

`original_series` numeric vector of the original time series

`order` list of order parameters (entries depend on `model_type`; see the relevant fitting function)

predict,PMM2fit-method

Prediction method for PMM2fit objects

Description

Computes predictions for new data using a fitted PMM2 model. The method extracts the formula from the fitted model, constructs a design matrix from the new data, and computes predictions via matrix multiplication. This approach works with arbitrary variable names and model specifications.

Usage

```
## S4 method for signature 'PMM2fit'
predict(object, newdata = NULL, debug = FALSE, ...)
```

Arguments

object	PMM2fit object returned by lm_pmm2()
newdata	Data frame containing the predictor variables with the same names as those used in the original model fit. Required parameter.
debug	Logical value indicating whether to output debug information about the prediction process. Default is FALSE.
...	Additional arguments (currently not used)

Details

The prediction is computed as X %*% beta where X is the design matrix constructed from newdata using the model formula, and beta are the estimated coefficients. The method automatically handles:

- Models with intercepts and without
- Arbitrary variable names (not limited to "x1", "x2", etc.)
- Interaction terms and transformations specified in the formula
- Automatic coefficient reordering to match design matrix columns

Value

Numeric vector of predicted values for the observations in newdata

Examples

```
# Fit model
fit <- lm_pmm2(mpg ~ wt + hp, data = mtcars)

# Predict on new data
newdata <- data.frame(wt = c(2.5, 3.0), hp = c(100, 150))
predictions <- predict(fit, newdata = newdata)
```

predict,PMM3fit-method *Predict method for PMM3fit objects*

Description

Predict method for PMM3fit objects

Usage

```
## S4 method for signature 'PMM3fit'
predict(object, newdata = NULL, ...)
```

Arguments

object	PMM3fit object
newdata	Data frame with predictor variables
...	Additional arguments (not used)

Value

Numeric vector of predicted values

predict,TS2fit-method *Prediction method for TS2fit objects*

Description

Prediction method for TS2fit objects

Usage

```
## S4 method for signature 'TS2fit'
predict(object, n.ahead = 1, ...)
```

Arguments

object	TS2fit object
n.ahead	Number of steps ahead for prediction
...	additional arguments (not used)

Value

Vector or list of predictions, depending on model type

predict,TS3fit-method *Predict method for TS3fit objects*

Description

Predict method for TS3fit objects

Usage

```
## S4 method for signature 'TS3fit'
predict(object, n.ahead = 1, ...)
```

Arguments

object	TS3fit object
n.ahead	Integer: number of steps ahead to forecast
...	Additional arguments (not used)

Value

Numeric vector of predicted values

print.PMMdispatch *Print method for PMMdispatch objects*

Description

Print method for PMMdispatch objects

Usage

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

Arguments

x	object of class PMMdispatch returned by pmm_dispatch
...	additional arguments (ignored)

Value

The object x, invisibly.

`residuals,PMM2fit-method`

Extract residuals from PMM2fit object

Description

Extract residuals from PMM2fit object

Usage

```
## S4 method for signature 'PMM2fit'
residuals(object, ...)
```

Arguments

`object`	PMM2fit object
`...`	Additional arguments (not used)

Value

Vector of residuals

`residuals,PMM3fit-method`

Extract residuals from PMM3fit object

Description

Extract residuals from PMM3fit object

Usage

```
## S4 method for signature 'PMM3fit'
residuals(object, ...)
```

Arguments

`object`	PMM3fit object
`...`	Additional arguments (not used)

Value

Vector of residuals

`residuals,TS2fit-method`

Extract residuals from TS2fit object

Description

Extract residuals from TS2fit object

Usage

```
## S4 method for signature 'TS2fit'
residuals(object, ...)
```

Arguments

`object`	TS2fit object
`...`	Additional arguments (not used)

Value

Vector of residuals (innovations)

`residuals,TS3fit-method`

Extract residuals from TS3fit object

Description

Extract residuals from TS3fit object

Usage

```
## S4 method for signature 'TS3fit'
residuals(object, ...)
```

Arguments

`object`	TS3fit object
`...`	Additional arguments (not used)

Value

Vector of residuals

sar_pmm2 *Fit Seasonal AR model using PMM2 method*

Description

Fits a Seasonal Autoregressive (SAR) model using the Polynomial Maximization Method (PMM2). The model can include both non-seasonal and seasonal AR components.

Usage

```
sar_pmm2(
  x,
  order = c(0, 1),
  season = list(period = 12),
  method = "pmm2",
  pmm2_variant = c("unified_global", "unified_iterative", "linearized"),
  include.mean = TRUE,
  multiplicative = FALSE,
  max_iter = 50,
  tol = 1e-06,
  regularize = TRUE,
  reg_lambda = 1e-08,
  verbose = FALSE
)
```

Arguments

x	Numeric vector of time series data
order	Vector of length 2: c(p, P) where: • p: Non-seasonal AR order • P: Seasonal AR order
season	List with seasonal specification: list(period = s) where s is the seasonal period (e.g., 12 for monthly data with annual seasonality)
method	Estimation method: "pmm2" (default), "ols", "css"
pmm2_variant	Character string specifying PMM2 implementation variant. Options: "unified_global" (default, one-step correction from MLE/CSS), "unified_iterative" (full Newton-Raphson), or "linearized" (not recommended for SAR models).
include.mean	Logical, include intercept/mean term (default TRUE)
multiplicative	Logical, use multiplicative form with cross-terms (default FALSE)
max_iter	Maximum iterations for PMM2 algorithm (default 50)
tol	Convergence tolerance (default 1e-6)
regularize	Logical, use regularization for numerical stability (default TRUE)
reg_lambda	Regularization parameter (default 1e-8)
verbose	Logical, print progress information (default FALSE)

Details

The SAR model has the form

$$y_t = \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \Phi_1 y_{t-s} + \cdots + \Phi_P y_{t-Ps} + \mu + \epsilon_t.$$

Where:

- p is the non-seasonal AR order
- P is the seasonal AR order
- s is the seasonal period
- epsilon_t are innovations (errors)

The PMM2 method provides more efficient parameter estimates than OLS when the innovation distribution is asymmetric (non-Gaussian). The expected variance reduction is given by g = 1 - c3^2 / (2 + c4), where c3 and c4 are the skewness and excess kurtosis coefficients.

Variant Recommendations for SAR:

- `"unified_global"` (default): Fast one-step correction, suitable for most SAR models
- `"unified_iterative"`: Best accuracy for complex seasonal patterns
- `"linearized"`: Not recommended for SAR (designed for MA/SMA)

Value

S4 object of class SARPMM2 containing:

- coefficients: Estimated AR and SAR parameters
- residuals: Model residuals/innovations (padded with zeros to match original length)
- m2, m3, m4: Central moments of residuals
- convergence: Convergence status
- iterations: Number of iterations performed

See Also

`ar_pmm2`, `ts_pmm2`, `compare_sar_methods`

Examples

```
# Generate synthetic seasonal data
n <- 120
y <- arima.sim(n = n, list(ar = 0.7, seasonal = list(sar = 0.5, period = 12)))

# Fit SAR(1,1)_12 model with PMM2
fit <- sar_pmm2(y, order = c(1, 1), season = list(period = 12))
summary(fit)

# Simple seasonal model (no non-seasonal component)
fit_pure_sar <- sar_pmm2(y, order = c(0, 1), season = list(period = 12))
```

```
# Compare with OLS
fit_ols <- sar_pmm2(y, order = c(1, 1), season = list(period = 12), method = "ols")
```

`sarima_pmm2` *Fit a Seasonal ARIMA model using PMM2 method*

Description

Fits a Seasonal Autoregressive Integrated Moving Average (SARIMA) model with both non-seasonal and seasonal differencing operators.

Usage

```
sarima_pmm2(
  x,
  order = c(0, 1, 0, 1),
  seasonal = list(order = c(1, 1), period = 12),
  method = "pmm2",
  pmm2_variant = c("unified_global", "unified_iterative", "linearized"),
  include.mean = NULL,
  max_iter = 50,
  tol = 1e-06,
  regularize = TRUE,
  reg_lambda = 1e-08,
  ma_method = c("mle", "pmm2"),
  verbose = FALSE,
  multiplicative = TRUE
)
```

Arguments

`x`	Numeric vector of time series data
`order`	Vector of length 4: c(p, P, q, Q) where: • p: Non-seasonal AR order • P: Seasonal AR order • q: Non-seasonal MA order • Q: Seasonal MA order
`seasonal`	List with seasonal specification: list(order = c(d, D), period = s) • d: Non-seasonal differencing order • D: Seasonal differencing order • period: Seasonal period (s)
`method`	Estimation method: "pmm2" (default), "css-ml", "css"

`pmm2_variant`	Character string specifying PMM2 implementation variant. Options: `"unified_global"` (default, one-step correction), `"unified_iterative"` (full Newton-Raphson, **recommended for SARIMA**), or `"linearized"` (specialized for MA/SMA models).
`include.mean`	Logical, include drift term (default TRUE for d+D=0, FALSE otherwise)
`max_iter`	Maximum iterations for PMM2 algorithm (default 50)
`tol`	Convergence tolerance (default 1e-6)
`regularize`	Logical, use regularization for numerical stability (default TRUE)
`reg_lambda`	Regularization parameter (default 1e-8)
`ma_method`	Method for MA/SMA estimation: "mle" (default) or "pmm2"
`verbose`	Logical, print progress information (default FALSE)
`multiplicative`	Logical, use multiplicative seasonal model form with cross-terms between non-seasonal and seasonal components (default TRUE). If FALSE, uses additive form.

Details

The SARIMA(p,d,q) x (P,D,Q)_s model satisfies

$$(1-\phi_1 B-\cdots-\phi_p B^p)(1-\Phi_1 B^s-\cdots-\Phi_P B^{Ps})(1-B)^d(1-B^s)^D y_t = (1+\theta_1 B+\cdots+\theta_q B^q)(1+\Theta_1 B^s+\cdots+\Theta_Q B^{Qs})\epsilon_t.$$

Where B is the backshift operator. The model combines:

- Non-seasonal differencing: (1-B)^d
- Seasonal differencing: (1-B^s)^D
- Non-seasonal ARMA components
- Seasonal ARMA components

Variant Recommendations for SARIMA:

- `"unified_iterative"` (recommended): Monte Carlo experiments showed 16.4\ seasonal dynamics
- `"unified_global"` (default): Faster alternative with good accuracy
- `"linearized"`: Not recommended for SARIMA (designed for MA/SMA)

Value

S4 object of class SARIMAPMM2 containing:

- coefficients: Estimated parameters
- residuals: Model residuals/innovations (padded with zeros to match original length)
- m2, m3, m4: Central moments of residuals
- convergence: Convergence status
- iterations: Number of iterations performed

References

Monte Carlo validation (R=50, n=200): Unified Iterative achieved 16.4\ improvement for SARIMA models. See NEWS.md (Version 0.2.0).

See Also

sarma_pmm2, arima_pmm2

Examples

```
set.seed(123)
n <- 200
y <- arima.sim(n = n,
  model = list(order = c(1, 0, 1), ar = 0.5, ma = 0.3,
    seasonal = list(order = c(1, 0, 0), ar = 0.4, period = 12)))
fit <- sarima_pmm2(y,
  order = c(1, 0, 1, 0),
  seasonal = list(order = c(1, 0), period = 12))
summary(fit)
```

SARIMAPMM2-class *S4 class for Seasonal ARIMA model results with PMM2*

Description

This class stores the results of fitting a Seasonal Autoregressive Integrated Moving Average (SARIMA) model using the PMM2 method. It extends SARMA with differencing operators.

Details

The SARIMAPMM2 class represents fitted SARIMA(p,d,q) x (P,D,Q)_s models:

$$(1-\phi_1 B-\cdots-\phi_p B^p)(1-\Phi_1 B^s-\cdots-\Phi_P B^{Ps})(1-B)^d(1-B^s)^D y_t = (1+\theta_1 B+\cdots+\theta_q B^q)(1+\Theta_1 B^s+\cdots+\Theta_Q B^{Qs})\epsilon_t.$$

Where B is the backshift operator.

Slots

`coefficients` Numeric vector of estimated parameters

`residuals` Numeric vector of residuals/innovations

`m2` Second central moment of residuals

`m3` Third central moment of residuals

`m4` Fourth central moment of residuals

`convergence` Logical, whether PMM2 algorithm converged

`iterations` Integer, number of iterations performed

`call` Original function call

`model_type` Character, model type identifier ("sarima")

`intercept` Numeric, intercept/mean term

`original_series` Numeric vector, original time series data

`order` List with model specification: list(ar, sar, ma, sma, d, D, period)

- ar: Non-seasonal AR order (p)
- sar: Seasonal AR order (P)
- ma: Non-seasonal MA order (q)
- sma: Seasonal MA order (Q)
- d: Non-seasonal differencing order
- D: Seasonal differencing order
- period: Seasonal period (s)

See Also

`sarima_pmm2` for fitting SARIMA models

`sarma_pmm2` *Fit a Seasonal ARMA model using PMM2 method*

Description

Fits a Seasonal Autoregressive Moving Average (SARMA) model that combines both seasonal AR and seasonal MA components, optionally with non-seasonal AR and MA terms as well.

Usage

```
sarma_pmm2(
  x,
  order = c(0, 1, 0, 1),
  season = list(period = 12),
  method = "pmm2",
  include.mean = TRUE,
  max_iter = 50,
  tol = 1e-06,
  regularize = TRUE,
  reg_lambda = 1e-08,
  verbose = FALSE
)
```

Arguments

x	Numeric vector of time series data
order	Vector of length 4: c(p, P, q, Q) where: • p: Non-seasonal AR order • P: Seasonal AR order • q: Non-seasonal MA order • Q: Seasonal MA order
season	List with seasonal specification: list(period = s) where s is the seasonal period (e.g., 12 for monthly data)
method	Estimation method: "pmm2" (default), "css-ml", "css"
include.mean	Logical, include intercept/mean term (default TRUE)
max_iter	Maximum iterations for PMM2 algorithm (default 50)
tol	Convergence tolerance (default 1e-6)
regularize	Logical, use regularization for numerical stability (default TRUE)
reg_lambda	Regularization parameter (default 1e-8)
verbose	Logical, print progress information (default FALSE)

Details

The SARMA model combines four components:

- AR(p): $\phi_1 y_{t-1} + \cdots + \phi_p y_{t-p}$
- Seasonal AR: $\Phi_1 y_{t-s} + \cdots + \Phi_P y_{t-Ps}$
- MA(q): $\theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q}$
- Seasonal MA: $\Theta_1 \epsilon_{t-s} + \cdots + \Theta_Q \epsilon_{t-Qs}$

The PMM2 method provides more efficient parameter estimates than ML/CSS when the innovation distribution is asymmetric (non-Gaussian).

Value

S4 object of class SARMAPMM2 containing:

- coefficients: Estimated parameters
- residuals: Model residuals/innovations (padded with zeros to match original length)
- m2, m3, m4: Central moments of residuals
- convergence: Convergence status
- iterations: Number of iterations performed

See Also

sar_pmm2, sma_pmm2, sarima_pmm2

Examples

```
# Generate synthetic seasonal data with SARMA structure
set.seed(123)
n <- 200
y <- arima.sim(n = n, list(
  ar = 0.5, ma = 0.3,
  seasonal = list(sar = 0.6, sma = 0.4, period = 12)
))

# Fit SARMA(1,1,1,1)_12 model with PMM2
fit <- sarma_pmm2(y, order = c(1, 1, 1, 1), season = list(period = 12))
summary(fit)

# Pure seasonal model (no non-seasonal components)
fit_pure <- sarma_pmm2(y, order = c(0, 1, 0, 1), season = list(period = 12))
```

SARMAPMM2-class *S4 class for Seasonal ARMA model results with PMM2*

Description

This class stores the results of fitting a Seasonal Autoregressive Moving Average (SARMA) model using the PMM2 method. It combines both seasonal AR and seasonal MA components.

Details

The SARMAPMM2 class represents fitted SARMA models combining:

- AR(p): $\phi_1 y_{t-1} + \cdots + \phi_p y_{t-p}$
- Seasonal AR component: $\Phi_1 y_{t-s} + \cdots + \Phi_P y_{t-Ps}$
- MA(q): $\theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q}$
- Seasonal MA component: $\Theta_1 \epsilon_{t-s} + \cdots + \Theta_Q \epsilon_{t-Qs}$

Slots

`coefficients` Numeric vector of estimated parameters (SAR and SMA coefficients)

`residuals` Numeric vector of residuals/innovations

`m2` Second central moment of residuals

`m3` Third central moment of residuals

`m4` Fourth central moment of residuals

`convergence` Logical, whether PMM2 algorithm converged

`iterations` Integer, number of iterations performed

`call` Original function call

`model_type` Character, model type identifier ("sarma")

`intercept` Numeric, intercept/mean term

`original_series` Numeric vector, original time series data

`order` List with model specification: list(ar, sar, ma, sma, period)

- ar: Non-seasonal AR order (p)
- sar: Seasonal AR order (P)
- ma: Non-seasonal MA order (q)
- sma: Seasonal MA order (Q)
- period: Seasonal period (s)

See Also

`sarma_pmm2` for fitting SARMA models

`SARPMM2-class` *S4 class for Seasonal AR model results with PMM2*

Description

This class stores the results of fitting a Seasonal Autoregressive (SAR) model using the PMM2 method. It extends the TS2fit class with additional slots specific to seasonal models.

Details

The SARPMM2 class represents fitted SAR models of the form

$$y_t = \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \Phi_1 y_{t-s} + \cdots + \Phi_P y_{t-Ps} + \epsilon_t.$$

Where:

- p is the non-seasonal AR order
- P is the seasonal AR order
- s is the seasonal period

Slots

`coefficients` Numeric vector of estimated parameters (AR and SAR coefficients)

`residuals` Numeric vector of residuals/innovations

`m2` Second central moment of residuals

`m3` Third central moment of residuals

`m4` Fourth central moment of residuals

`convergence` Logical, whether PMM2 algorithm converged

`iterations` Integer, number of iterations performed

`call` Original function call

`model_type` Character, model type identifier ("sar")

`intercept` Numeric, intercept/mean term

`original_series` Numeric vector, original time series data

`order` List with model specification: list(ar, sar, period)

- ar: Non-seasonal AR order (p)
- sar: Seasonal AR order (P)
- period: Seasonal period (s)

See Also

`sar_pmm2` for fitting SAR models

`show,PMM2fit-method` *Show method for PMM2fit objects*

Description

Prints a concise lm-style display: the original call, coefficients, and a one-line algorithm footer. Use `summary` for full diagnostics including moments, the g2 efficiency factor, and bootstrap inference.

Usage

```
## S4 method for signature 'PMM2fit'
show(object)
```

Arguments

`object`	object of class `PMM2fit`

Value

The object (invisibly).

show,PMM3fit-method *Show method for PMM3fit objects*

Description

Prints a concise lm-style display: the original call, coefficients, and a one-line algorithm footer. Use summary for full diagnostics including moments, gamma4/gamma6 cumulants, and the g3 efficiency factor.

Usage

```
## S4 method for signature 'PMM3fit'
show(object)
```

Arguments

object	object of class PMM3fit

Value

The object (invisibly).

show,TS2fit-method *Show method for TS2fit objects (and subclasses)*

Description

Inherited by ARPMM2, MAPMM2, ARMAPMM2, ARIMAPMM2 and the seasonal SARPMM2/SMAPMM2/SARMAPMM subclasses. Use summary for full diagnostics.

Usage

```
## S4 method for signature 'TS2fit'
show(object)
```

Arguments

object	object inheriting from TS2fit

Value

The object (invisibly).

show,TS3fit-method *Show method for TS3fit objects (and subclasses)*

Description

Inherited by ARPMM3, MAPMM3, ARMAPMM3, ARIMAPMM3. Use `summary` for full diagnostics.

Usage

```
## S4 method for signature 'TS3fit'
show(object)
```

Arguments

`object`	object inheriting from `TS3fit`

Value

The object (invisibly).

sma_pmm2 *Fit a Seasonal MA model using PMM2*

Description

Fits a Seasonal Moving Average (SMA) model using the Polynomial Maximization Method (PMM2). This is particularly effective when the innovations have asymmetric or non-Gaussian distributions.

Usage

```
sma_pmm2(
  x,
  order = 1,
  season = list(period = 12),
  method = "pmm2",
  pmm2_variant = c("unified_global", "unified_iterative", "linearized"),
  max_iter = 50,
  tol = 1e-06,
  include.mean = TRUE,
  na.action = na.fail,
  regularize = TRUE,
  reg_lambda = 1e-08,
  verbose = FALSE
)
```

Arguments

x	Numeric vector (time series data)
order	Seasonal MA order (Q)
season	List with seasonal specification: list(period = s)
method	Estimation method: "pmm2" or "css"
pmm2_variant	Character string specifying PMM2 implementation variant. Options: "unified_global" (default, one-step correction), "unified_iterative" (full Newton-Raphson), or "linearized" (specialized for MA/SMA models, **recommended for SMA**).
max_iter	Maximum iterations for PMM2 algorithm
tol	Convergence tolerance
include.mean	Include intercept in the model
na.action	Function to handle missing values
regularize	Add regularization to Jacobian matrix
reg_lambda	Regularization parameter
verbose	Print diagnostic information

Details

The SMA(Q)_s model has the form

$$y_t = \mu + \epsilon_t + \Theta_1 \epsilon_{t-s} + \Theta_2 \epsilon_{t-2s} + \cdots + \Theta_Q \epsilon_{t-Qs}.$$

Where:

- Q is the seasonal MA order
- s is the seasonal period (12 for monthly, 4 for quarterly)
- epsilon_t are innovations (errors)

The PMM2 method provides more efficient parameter estimates than ML/CSS when the innovation distribution is asymmetric (non-Gaussian). The expected variance reduction is given by g = 1 - c3^2 / (2 + c4), where c3 and c4 are the skewness and excess kurtosis coefficients.

Variant Recommendations for SMA:

- "linearized" (recommended): Fastest and most stable for SMA models, avoids complex Jacobian computation while maintaining accuracy
- "unified_global" (default): Good balance of speed and accuracy
- "unified_iterative": Best accuracy but slower, use for complex SMA patterns

Value

An S4 object of class SMAPMM2 containing:

- coefficients: Seasonal MA coefficients (Theta_1, Theta_2, ..., Theta_Q)
- innovations: Estimated innovations (residuals)

- m2, m3, m4: Central moments of innovations
- convergence: Convergence status
- iterations: Number of iterations performed
- intercept: Model intercept
- original_series: Original time series
- order: Model order list(Q, s)

See Also

ma_pmm2, sar_pmm2, arima_pmm2

Examples

```
# Generate synthetic seasonal data
set.seed(123)
n <- 120
s <- 12
theta <- 0.6

# Gamma innovations (asymmetric)
innov <- rgamma(n, shape = 2, scale = 1) - 2
y <- numeric(n)
for (t in 1:n) {
  ma_term <- if (t > s) theta * innov[t - s] else 0
  y[t] <- innov[t] + ma_term
}

# Fit SMA(1)_12 model with PMM2
fit <- sma_pmm2(y, order = 1, season = list(period = 12))
summary(fit)

# Compare with CSS
fit_css <- sma_pmm2(y, order = 1, season = list(period = 12), method = "css")
```

SMAPMM2-class *S4 class for Seasonal MA PMM2 results*

Description

This class stores the results of fitting a Seasonal Moving Average (SMA) model using the Polynomial Maximization Method (PMM2).

Details

The SMA(Q)_s model is expressed as

$$y_t = \mu + \epsilon_t + \Theta_1\epsilon_{t-s} + \Theta_2\epsilon_{t-2s} + \cdots + \Theta_Q\epsilon_{t-Qs}.$$

Where:

- Q is the seasonal MA order
- s is the seasonal period
- epsilon_t are innovations

Slots

`coefficients` Estimated seasonal MA coefficients (Theta_1, Theta_2, ..., Theta_Q)

`innovations` Estimated innovations (residuals epsilon_t)

`m2` Second central moment (variance) of innovations

`m3` Third central moment (skewness indicator) of innovations

`m4` Fourth central moment (kurtosis indicator) of innovations

`convergence` Logical indicating whether PMM2 algorithm converged

`iterations` Number of iterations required for convergence

`call` The function call that created this object

`model_type` Character string "sma"

`intercept` Model intercept (mean)

`original_series` Original time series data

`order` List with Q (seasonal MA order) and s (seasonal period)

See Also

`sma_pmm2` for fitting SMA models

`solve_pmm2_step` *PMM2 step solver*

Description

Solves the linearized system to find the parameter update. Based on the Taylor expansion: e(theta) ~ e(theta_k) - J * delta

Usage

```
solve_pmm2_step(residuals, J, pmm_stats)
```

Arguments

residuals	Current residuals
J	Jacobian matrix (n x p), where J_ij = d(y_hat_i)/d(theta_j). We assume the standard regression definition: y = f(theta) + e, so e = y - f(theta) and d(e)/d(theta) = -d(f)/d(theta). We expect J = d(f)/d(theta) (gradient of the regression function).
pmm_stats	Statistics from compute_pmm2_components

Value

Update vector delta

---

summary,PMM2fit-method

Generic summary method for PMM2fit objects

Description

Generic summary method for PMM2fit objects

Usage

```
## S4 method for signature 'PMM2fit'
summary(object, formula = NULL, data = NULL, B = 100, ...)
```

Arguments

object	object of class "PMM2fit"
formula	(optional) formula used for the model
data	(optional) data used
B	number of bootstrap replications for statistical inference
...	additional arguments (not used)

Value

Prints summary to console; returns object (invisibly).

summary,PMM3fit-method

Summary method for PMM3fit objects

Description

Summary method for PMM3fit objects

Usage

```
## S4 method for signature 'PMM3fit'
summary(object, ...)
```

Arguments

object	PMM3fit object
...	Additional arguments (not used)

Value

Prints summary to console; returns object (invisibly)

summary,SARIMAPMM2-method

Generic summary method for SARIMAPMM2 objects

Description

Generic summary method for SARIMAPMM2 objects

Usage

```
## S4 method for signature 'SARIMAPMM2'
summary(object, ...)
```

Arguments

object	object of class "SARIMAPMM2"
...	additional arguments (not used)

Value

Prints summary to console; returns object (invisibly).

summary,SARMAPMM2-method

Generic summary method for SARMAPMM2 objects

Description

Generic summary method for SARMAPMM2 objects

Usage

```
## S4 method for signature 'SARMAPMM2'
summary(object, ...)
```

Arguments

object	object of class "SARMAPMM2"
...	additional arguments (not used)

Value

Prints summary to console; returns object (invisibly).

summary,SARPMM2-method

Summary method for SARPMM2 objects

Description

Summary method for SARPMM2 objects

Usage

```
## S4 method for signature 'SARPMM2'
summary(object, ...)
```

Arguments

object	SARPMM2 object
...	Additional arguments (not used)

Value

Invisibly returns the object

summary,SMAPMM2-method *Summary method for SMAPMM2 objects*

Description

Summary method for SMAPMM2 objects

Usage

```
## S4 method for signature 'SMAPMM2'
summary(object, ...)
```

Arguments

object	SMAPMM2 object
...	Additional arguments (not used)

Value

Invisibly returns the object

summary,TS2fit-method *Generic summary method for TS2fit objects*

Description

Generic summary method for TS2fit objects

Usage

```
## S4 method for signature 'TS2fit'
summary(object, ...)
```

Arguments

object	object of class "TS2fit" or subclass
...	additional arguments (not used)

Value

Prints summary to console; returns object (invisibly).

summary,TS3fit-method *Summary method for TS3fit objects*

Description

Summary method for TS3fit objects

Usage

```
## S4 method for signature 'TS3fit'
summary(object, ...)
```

Arguments

object	TS3fit object
...	Additional arguments (not used)

Value

Prints summary to console; returns object (invisibly)

summary.PMMdispatch *Summary method for PMMdispatch objects*

Description

Returns a one-row data frame suitable for inclusion in reports or comparison tables.

Usage

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

Arguments

object	object of class PMMdispatch
...	additional arguments (ignored)

Value

A data frame with columns method, n, gamma3, gamma4, gamma6, g2, g3, g_selected, improvement_pct.

test_symmetry *Test whether residuals are sufficiently symmetric for PMM3*

Description

Computes the skewness coefficient gamma3 and checks whether its absolute value falls below a given threshold. This helps decide between PMM2 (asymmetric) and PMM3 (symmetric platykurtic) estimation.

Usage

```
test_symmetry(x, threshold = 0.3)
```

Arguments

x	numeric vector of residuals
threshold	numeric threshold for |gamma3| (default 0.3)

Value

A list with components:

gamma3	Sample skewness coefficient
is_symmetric	Logical: TRUE if |gamma3| <= threshold
message	Human-readable verdict

ts_pmm2 *Fit a time series model using the PMM2 method*

Description

Fit a time series model using the PMM2 method

Usage

```
ts_pmm2(
  x,
  order,
  model_type = c("ar", "ma", "arma", "arima"),
  method = "pmm2",
  ma_solver = c("linearized", "recursive"),
  max_iter = 50,
  tol = 1e-06,
  include.mean = TRUE,
  initial = NULL,
```

```
  na.action = na.fail,
  regularize = TRUE,
  reg_lambda = 1e-08,
  verbose = FALSE
)
```

Arguments

x	Numeric vector of time series data
order	Model order specification: • For AR models: a single integer (AR order) • For MA models: a single integer (MA order) • For ARMA models: vector c(p, q) (AR and MA orders) • For ARIMA models: vector c(p, d, q) (AR, differencing, and MA orders)
model_type	String specifying the model type: "ar", "ma", "arma", or "arima"
method	String: estimation method, one of "pmm2" (default), "css", "ml", "yw", "ols"
ma_solver	String: how the moving-average recursion is treated, `"linearized"` (default) or `"recursive"`. Has no effect when the MA order is zero. See Details.
max_iter	Integer: maximum number of iterations for the algorithm
tol	Numeric: tolerance for convergence
include.mean	Logical: whether to include a mean (intercept) term
initial	List or vector of initial parameter estimates (optional)
na.action	Function for handling missing values, default is na.fail
regularize	Logical, add small values to diagonal for numerical stability
reg_lambda	Regularization parameter (if regularize=TRUE)
verbose	Logical: whether to print progress information

Details

The PMM2 algorithm works as follows:

1. Fits an initial model using a standard method (OLS, Yule-Walker, CSS or ML)
2. Computes central moments (m2, m3, m4) from initial residuals/innovations
3. Uses these moments with a specialized solver (pmm2_algorithm) to find robust parameter estimates

Treatment of the MA recursion (`ma_solver`).

- `"linearized"` (default, historical behaviour): the design matrix is built once from the CSS residuals and held fixed. For AR columns this is exact, because the regressors are observed data. For MA columns it is a one-step linearisation of the innovation recursion and costs a factor of exactly $1 - \theta^2$ in asymptotic variance relative to CSS. The penalty does *not* vanish as n grows.

- `"recursive"` (recommended whenever $q > 0$): the innovations $\varepsilon_t(\beta)$ and the score regressors $x_t(\beta) = -\partial\varepsilon_t/\partial\beta$ are recomputed by the exact recursion at every candidate parameter. When $\mu_3 = 0$ the estimating equation then reduces identically to the CSS first-order condition, so the $1 - \theta^2$ penalty disappears exactly rather than asymptotically. Measured on ARIMA(0,1,1) with $\theta = -0.5$, $T = 500$: relative efficiency against CSS rises from 0.77 to 0.98 (Gaussian), 1.30 to 1.72 (Gamma), 1.33 to 1.81 (lognormal) and 1.35 to 1.73 (chi-squared).

The default is `"linearized"` for backward compatibility only. It will change to `"recursive"` in EstemPMM 1.0.0. Until then, the first fit per session with $q \geq 1$ and `ma_solver` unset prints a message saying so. `ma_solver` is supported for `model_type` `"ma"`, `"arma"` and `"arima"` with $q \geq 1$ and any $p \geq 0$. Seasonal models (`sarima_pmm2` and friends) always use the linearised path.

Not to be confused with the unrelated `ma_method` argument of `sarima_pmm2`, which takes `"mle"`/`"pmm2"` and selects how the seasonal MA terms are estimated. The two control different things and are not interchangeable.

Value

An S4 object `TS2fit` of the corresponding subclass

`ts_pmm2_inference` *Bootstrap inference for PMM2 time series models*

Description

Residual- or block-bootstrap standard errors and confidence intervals for PMM2 time-series fits. Block bootstrap can exhibit substantial finite-sample bias for AR coefficients near the stationarity boundary; see `ci_method` and the returned `bias` column for diagnostics.

Usage

```
ts_pmm2_inference(
  object,
  x = NULL,
  B = 200,
  seed = NULL,
  block_length = NULL,
  method = c("residual", "block"),
  parallel = FALSE,
  cores = NULL,
  debug = FALSE,
  ci_method = c("normal", "percentile", "basic")
)
```

Arguments

object	object of class TS2fit
x	(optional) original time series; if NULL, uses object@original_series
B	number of bootstrap replications
seed	(optional) for reproducibility
block_length	block length for block bootstrap; if NULL, uses heuristic value
method	bootstrap type: "residual" or "block"
parallel	logical, whether to use parallel computing
cores	number of cores for parallel computing
debug	logical, whether to output additional diagnostic information
ci_method	character: confidence-interval method. See pmm2_inference for the full description. The default "normal" returns a Wald interval centred on the point estimate, which is robust to the moderate downward bias that block bootstrap induces for AR coefficients of strongly persistent series. Use "percentile" for Efron's classical percentile interval or "basic" for the Davison & Hinkley (1997) pivotal interval.

Value

data.frame with columns: Estimate, Std.Error, bias, t.value, p.value, conf.low, conf.high.

---

ts_pmm3 *Fit a time series model using PMM3*

Description

Core function that fits AR, MA, ARMA, or ARIMA models using the Polynomial Maximization Method of order 3 (PMM3). Designed for symmetric platykurtic innovations.

Usage

```
ts_pmm3(
  x,
  order,
  model_type = c("ar", "ma", "arma", "arima"),
  max_iter = 100,
  tol = 1e-06,
  adaptive = FALSE,
  step_max = 5,
  include.mean = TRUE,
  initial = NULL,
  na.action = na.fail,
  verbose = FALSE
)
```

Arguments

x	Numeric vector of time series data
order	Model order specification (see ts_pmm2 for format)
model_type	Character: "ar", "ma", "arma", or "arima"
max_iter	Integer: maximum NR iterations (default 100)
tol	Numeric: convergence tolerance (default 1e-6)
adaptive	Logical: re-estimate kappa each iteration (default FALSE)
step_max	Numeric: maximum NR step (default 5.0). For AR, MA and ARMA models it is the root-mean-square change of the fitted values in units of the initial residual standard deviation; for ARIMA models it bounds the norm of the coefficient step.
include.mean	Logical: include mean/intercept term (default TRUE)
initial	Optional initial parameter estimates
na.action	Function for handling missing values (default na.fail)
verbose	Logical: print progress information (default FALSE)

Details

The PMM3 time series algorithm:

1. Obtains initial estimates via classical methods (OLS/YW for AR, CSS for MA/ARMA/ARIMA)
2. Computes moments m2, m4, m6 and kappa from initial residuals
3. Checks symmetry: warns if |gamma3| exceeds both 0.3 and $2\sqrt{6/n}$
4. Applies Newton-Raphson with PMM3 score equations

PMM3 is beneficial when innovations are symmetric and platykurtic (gamma4 < 0), e.g. uniform, truncated normal.

Value

An S4 object of the appropriate TS3fit subclass

TS2fit-class *S4 class for PMM2 time-series fit results*

Description

Common parent of every PMM2 time-series fit class (AR, MA, ARMA, ARIMA, and the seasonal SAR/SMA/SARMA/SARIMA subclasses). Inherits the PMM2-specific central moments from BasePMM2 and the time-series slots (model_type, intercept, original_series, order) from PMMtsfit. The diamond inheritance resolves at PMMfit, which both branches share as their virtual root.

Base class for storing results of time series model estimation using PMM2

Slots

`ma_solver` character, which treatment of the moving-average recursion produced the fit: `"linearized"` (design matrix frozen at the CSS residuals) or `"recursive"` (innovation recursion recomputed at every candidate parameter). Pure AR fits are unaffected by the distinction and carry the default `"linearized"`.

Slots

coefficients Estimated coefficients

residuals Final residuals

m2 Second central moment

m3 Third central moment

m4 Fourth central moment

convergence Convergence status

iterations Number of iterations performed

call Original call

model_type Model type

intercept Intercept

original_series Original time series

order Model orders

`TS3fit-class` *S4 class for PMM3 time-series fit results*

Description

Common parent of every PMM3 time-series fit class (ARPMM3, MAPMM3, ARMAPMM3, ARIMAPMM3). Inherits the PMM3-specific moments and cumulant coefficients from `BasePMM3` and the time-series slots from `PMMtsfit`. The diamond inheritance resolves at `PMMfit`, which both branches share as their virtual root.

`vcov,PMM2fit-method` *Variance-covariance matrix for PMM2fit object*

Description

Returns the asymptotic covariance matrix $\sigma^2 g_2 (X^\top X)^{-1} + (1-g_2)\sigma^2 n^{-1} a a^\top$, $a = (X^\top X)^{-1} X^\top 1$, where $g_2 = 1 - c_3^2/(2+c_4)$ is the PMM2 variance reduction factor. The second term comes from estimating the moments from the OLS residuals: with an intercept it affects the intercept only, which therefore has close to the OLS variance, while the slopes keep the factor g_2. Under Gaussian errors $g_2 = 1$ and the result equals the OLS covariance matrix. When the PMM2 iteration failed and `lm_pmm2` returned the OLS estimate, the OLS covariance $\sigma^2 (X^\top X)^{-1}$ is returned instead, since the reduction by g_2 does not apply to that estimate.

Usage

```
## S4 method for signature 'PMM2fit'
vcov(object, ...)
```

Arguments

object	PMM2fit object
...	Additional arguments (not used)

Value

Numeric matrix of the same dimension as the number of coefficients

---

vcov,PMM3fit-method *Variance-covariance matrix for PMM3fit objects*

Description

Returns the asymptotic covariance matrix $V_{\mathrm{PMM3}} = g_3\,\sigma^2 (X^\top X)^{-1}$ where $g_3 = 1 - \gamma_4^2/(6 + 9\gamma_4 + \gamma_6)$ is the PMM3 variance reduction factor (Kunchenko's polynomial maximization method, s = 3 specialised to symmetric platykurtic errors; see pmm3_variance_matrices). Under Gaussian errors $g_3 = 1$ and the result equals the OLS covariance. For `lm_pmm3(symmetric = FALSE)`, g_3 is the general factor $1/(\alpha_2\, b^\top F^{-1} b)$, and the matrix gains the term $(1 - g_3)\sigma^2 n^{-1} a a^\top$, $a = (X^\top X)^{-1} X^\top 1$, for the moments estimated from the OLS residuals (see vcov,PMM2fit-method); with an intercept only the intercept variance changes. The symmetric estimator needs no such term. When the iteration failed and the OLS estimate was returned, the OLS covariance is returned.

Usage

```
## S4 method for signature 'PMM3fit'
vcov(object, ...)
```

Arguments

object	PMM3fit object
...	Additional arguments (not used)

Value

Numeric matrix of the same dimension as the number of coefficients

`vcov,TS2fit-method` *Variance-covariance matrix for TS2fit models*

Description

For AR models, returns $\sigma^2 g_2 (X^\top X)^{-1}$ where X is the lagged design matrix.

Usage

```
## S4 method for signature 'TS2fit'
vcov(object, ...)
```

Arguments

`object`	TS2fit (or subclass) object
`...`	Additional arguments (not used)

Details

For non-seasonal MA/ARMA/ARIMA models fitted with `ma_solver = "recursive"` the same expression applies with X replaced by the exact score regressors $x_t = -\partial\varepsilon_t/\partial\beta$, rebuilt by `pmm2_recursive_design()`. This is the closed-form sandwich $\Sigma = (\Delta/a)\,\mathbf{M}^{-1}$ with $a = \mu_4 - \mu_2^2$, $\Delta = \mu_2 a - \mu_3^2$ and $\mathbf{M} = E[x_t x_t^\top]$: the estimating function is a martingale difference sequence at the true parameter, so no long-run (HAC) correction is required and no numerical Jacobian is needed. Note $\Delta/a = \mu_2 g_2$, i.e. the AR and MA branches share one formula.

For fits produced with the (default) `ma_solver = "linearized"` the frozen CSS design matrix is not the derivative of the criterion, the sandwich above does not apply, and `vcov()` still refuses; use `ts_pmm2_inference` for bootstrap standard errors.

When the fit holds the classical start instead of a PMM2 root (`method = "css"` or another non-PMM method, or a PMM2 iteration that failed, `attr(object, "fallback")`), the factor g_2 is dropped and $\sigma^2 (X^\top X)^{-1}$ is returned.

Value

Numeric covariance matrix of the slope coefficients

Known limitation (intercept)

Neyman orthogonality with respect to the plug-in moments holds if and only if $\mu_3 E[x_t] = 0$. All slope regressors have $E[x_t] = 0$, so estimating $\hat\mu_2, \hat\mu_3, \hat\mu_4$ does not affect the limiting distribution of the AR/MA coefficients. The intercept regressor has $E[x_t] = 1$, so under skewed innovations ($\mu_3 \neq 0$) estimating $\hat\mu_2$ contributes at first order to the intercept and its reported standard error is too small. This affects the linearised branch as well. `vcov()` therefore covers the slope parameters only and warns when an intercept was estimated under visible skewness.

vcov,TS3fit-method *Variance-covariance matrix for TS3fit AR models*

Description

For AR models, returns $\sigma^2 g_3 (X^\top X)^{-1}$ where X is the lagged design matrix and $g_3 = 1 - \gamma_4^2/(6 + 9\gamma_4 + \gamma_6)$ is the PMM3 efficiency factor (Kunchenko, s = 3 specialised to symmetric platykurtic innovations). For MA/ARMA/ARIMA models the asymptotic covariance requires the full Fisher information matrix; use bootstrap standard errors via `ts_pmm2_inference` on an analogous PMM2 fit, or refit with `ar_pmm3` on the residual series.

Usage

```
## S4 method for signature 'TS3fit'
vcov(object, ...)
```

Arguments

`object`	TS3fit (or subclass) object
`...`	Additional arguments (not used)

Details

When the PMM3 iteration fell back to the initial estimate, or was skipped because the innovations are near-Gaussian (`attr(object, "fallback")`), the factor g_3 is dropped and $\sigma^2 (X^\top X)^{-1}$ is returned.

Value

Numeric covariance matrix (AR models only)

WPMMfit-class *S4 class for weak-moment PMM regression fits*

Description

Concrete class returned by `lm_wpmm2` and `lm_wpmm3`. Inherits the common slots from `PMMfit`.

Slots

`degree` numeric: 2 for WPMM2, 3 for WPMM3

`sigma_mult` numeric: window width as a multiple of the robust residual scale

`weights` numeric: final IRLS weights W_i

`window_center` numeric: median of the final residuals

`window_scale` numeric: window standard deviation (`sigma_mult` times the MAD of the final residuals)

`weak_moments` numeric: windowed central moments of the final residuals (`c2`, `c3`, `c4` cumulants for WPMM2; `m2`, `m4`, `m6` moments for WPMM3)

`correction` numeric: the polynomial coefficient, a_2 for WPMM2 or κ^w for WPMM3; `NA` when the windowed law is mesokurtic and WPMM3 reduces to the Welsch estimator

---

`WPMMfit-methods` *Methods for WPMMfit objects*

Description

Accessors, prediction and inference for fits returned by `lm_wpmm2` and `lm_wpmm3`.

Usage

```
## S4 method for signature 'WPMMfit'
coef(object, ...)

## S4 method for signature 'WPMMfit'
residuals(object, ...)

## S4 method for signature 'WPMMfit'
fitted(object, ...)

## S4 method for signature 'WPMMfit'
predict(object, newdata = NULL, ...)

## S4 method for signature 'WPMMfit'
nobs(object, ...)

## S4 method for signature 'WPMMfit'
vcov(object, ...)

## S4 method for signature 'WPMMfit'
confint(object, parm, level = 0.95, ...)

## S4 method for signature 'WPMMfit'
summary(object, ...)

## S4 method for signature 'WPMMfit'
show(object)
```

Arguments

`object`	WPMMfit object.

`...`	additional arguments (not used).
`newdata`	optional data frame with the predictor variables.
`parm`	character or integer vector of parameters to include.
`level`	confidence level (default 0.95).

Value

`coef`, `residuals`, `fitted` and `predict` return numeric vectors; `vcov` the sandwich covariance matrix described in `lm_wpmm2`; `confint` a matrix of normal-approximation limits; `nobs` an integer; `summary` and `show` print and return the object invisibly.

Index

∗ **datasets**
auto_mpg, 16
DCOILWTICO, 33
djia2002, 33

ar_pmm2, 5, *8*, *10*, *13*, *47*, *54*, *76*
ar_pmm3, 7, *48*, *54*, *103*
arima_pmm2, *7*, 8, *11*, *13*, *54*, *79*, *88*
arima_pmm3, 10, *15*, *54*
ARIMAPMM2, *55*
ARIMAPMM2-class, 11
ARIMAPMM3, *55*
ARIMAPMM3-class, 12
arma_pmm2, *7*, *10*, 12, *15*, *47*, *55*
arma_pmm3, *11*, 14, *55*
ARMAPMM2, *55*
ARMAPMM2-class, 15
ARMAPMM3, *55*
ARMAPMM3-class, 15
ARPMM2, *54*
ARPMM2-class, 15
ARPMM3, *54*
ARPMM3-class, 16
auto_mpg, 16

BasePMM2, *99*
BasePMM2-class, 17
BasePMM3, *68*, *100*
BasePMM3-class, 18

coef,PMM2fit-method, 18
coef,PMM3fit-method, 19
coef,SARPMM2-method, 19
coef,SMAPMM2-method, 20
coef,TS2fit-method, 20
coef,TS3fit-method, 21
coef,WPMMfit-method *(*WPMMfit-methods*)*, 104
compare_ar_methods, 21
compare_arima_methods, 22
compare_arma_methods, 22
compare_ma_methods, 23
compare_sar_methods, 24, *76*
compare_ts_methods, 25
compare_with_ols, 25
compute_moments, 26
compute_moments_pmm3, 27
compute_pmm2_components, 27
confint,PMM2fit-method, 28
confint,PMM3fit-method, 28
confint,TS2fit-method, 29
confint,TS3fit-method, 30
confint,WPMMfit-method *(*WPMMfit-methods*)*, 104
create_sar_matrix, 30
create_sarma_matrix, 31

DCOILWTICO, 33
djia2002, 33

fitted,PMM2fit-method, 34
fitted,PMM3fit-method, 35
fitted,TS2fit-method, 35
fitted,TS3fit-method, 36
fitted,WPMMfit-method *(*WPMMfit-methods*)*, 104
format.PMMdispatch, 36

get_sarimax_jacobian, 37
get_sarimax_residuals, 37

lm_pmm2, 38, *43*, *58, 59*
lm_pmm3, *8*, *11*, *15*, 39, *42*, *43*, *48*, *58, 59*, *68*
lm_wpmm2, 41, *59*, *103–105*
lm_wpmm3, *59*, *103, 104*
lm_wpmm3 *(*lm_wpmm2*)*, 41
logLik.PMM2fit, 43
logLik.PMM3fit, 44
logLik.TS2fit, 44
logLik.TS3fit, 45

ma_pmm2, *7*, *13*, 45, *48*, *60*, *88*
ma_pmm3, *8*, 47, *60*
MAPMM2, *60*
MAPMM2-class, 49
MAPMM3, *60*
MAPMM3-class, 49

nobs,PMM2fit-method, 49
nobs,PMM3fit-method, 50
nobs,TS2fit-method, 50
nobs,TS3fit-method, 51
nobs,WPMMfit-method *(*WPMMfit-methods*)*, 104

plot,PMM2fit,missing-method, 51
plot,PMM3fit,missing-method, 52
plot,TS2fit,missing-method, 52
plot,TS3fit,missing-method, 53
plot_pmm2_bootstrap, 53
pmm2_inference, 61, *98*
pmm2_monte_carlo_compare, 62
pmm2_nonlinear_iterative, 64
pmm2_nonlinear_onestep, 64
pmm2_variance_factor, 65
pmm2_variance_matrices, 66
PMM2fit, *17*, *59*
PMM2fit-class, 66
pmm3_variance_factor, 67
pmm3_variance_matrices, 68, *101*
PMM3fit, *18*, *59*
PMM3fit-class, 68
pmm_ar, 54
pmm_arima, 54, *59*
pmm_arma, 55
pmm_dispatch, *40*, 55, *58*, *59*, *72*
pmm_gamma6, 57
pmm_kurtosis, 58
pmm_lm, *43*, 58
pmm_ma, 59
pmm_sarima, 60
pmm_skewness, 61
PMMfit, *17*, *18*, *59*, *68*, *69*, *99*, *100*, *103*
PMMfit-class, 69
PMMtsfit, *99*, *100*
PMMtsfit-class, 69
predict,PMM2fit-method, 70
predict,PMM3fit-method, 71
predict,TS2fit-method, 71
predict,TS3fit-method, 72
predict,WPMMfit-method *(*WPMMfit-methods*)*, 104
print.PMMdispatch, 72

residuals,PMM2fit-method, 73
residuals,PMM3fit-method, 73
residuals,TS2fit-method, 74
residuals,TS3fit-method, 74
residuals,WPMMfit-method *(*WPMMfit-methods*)*, 104

sar_pmm2, 75, *81*, *84*, *88*
sarima_pmm2, *10*, *60*, 77, *80*, *81*, *97*
SARIMAPMM2, *60*
SARIMAPMM2-class, 79
sarma_pmm2, *79*, 80, *83*
SARMAPMM2-class, 82
SARPMM2-class, 83
show,PMM2fit-method, 84
show,PMM3fit-method, 85
show,TS2fit-method, 85
show,TS3fit-method, 86
show,WPMMfit-method *(*WPMMfit-methods*)*, 104
sma_pmm2, *47*, *81*, 86, *89*
SMAPMM2-class, 88
solve_pmm2_step, 89
summary, *84–86*
summary,PMM2fit-method, 90
summary,PMM3fit-method, 91
summary,SARIMAPMM2-method, 91
summary,SARMAPMM2-method, 92
summary,SARPMM2-method, 92
summary,SMAPMM2-method, 93
summary,TS2fit-method, 93
summary,TS3fit-method, 94
summary,WPMMfit-method *(*WPMMfit-methods*)*, 104
summary.PMMdispatch, 94

test_symmetry, 95
TS2fit, *17*
TS2fit-class, 99
TS3fit, *18*
TS3fit-class, 100
ts_pmm2, *76*, 95, *99*
ts_pmm2_inference, *29*, *30*, 97, *102*, *103*
ts_pmm3, *8*, *11*, *14*, *48*, 98

vcov,PMM2fit-method, 100

vcov,PMM3fit-method, 101
vcov,TS2fit-method, 102
vcov,TS3fit-method, 103
vcov,WPMMfit-method *(*WPMMfit-methods*)*, 104

WPMMfit, *42*, *59*
WPMMfit-class, 103
WPMMfit-methods, 104