

Package ‘muse’

June 30, 2026

Type Package

Title Multiple Unobserved Sources of Error State Space Models

Version 0.1.0

Date 2026-06-26

URL <https://github.com/config-1/muse>

BugReports <https://github.com/config-1/muse/issues>

Language en-GB

Description Implements the Power / Trend / Seasonal (PTS) model, a unified state-space framework based on the Multiple Source of Error (MSOE) model. It brings the trend, seasonal and irregular component models of Harvey (1989) [doi:10.1017/CBO9781107049994](https://doi.org/10.1017/CBO9781107049994), Durbin and Koopman (2012) [doi:10.1093/acprof:oso/9780199641178.001.0001](https://doi.org/10.1093/acprof:oso/9780199641178.001.0001), Proietti (2000) [doi:10.1016/S0169-2070\(00\)00037-6](https://doi.org/10.1016/S0169-2070(00)00037-6), Sbrana and Silvestrini (2023) [doi:10.1016/j.ijforecast.2022.03.003](https://doi.org/10.1016/j.ijforecast.2022.03.003) and others together under a single estimation, selection and forecasting interface, with an optional Box-Cox power transformation. Models are estimated by maximum likelihood through the Kalman filter and smoother, with automatic component selection by information criteria.

License LGPL-2.1

Depends R (>= 3.0.2), greybox, smooth

Imports Rcpp (>= 0.12.3), generics (>= 0.1.2), zoo

LinkingTo Rcpp, RcppArmadillo (>= 0.8.100.0.0)

Suggests testthat (>= 3.0.0), knitr, rmarkdown

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

ByteCompile true

Config/roxygen2/version 8.0.0

RoxygenNote 8.0.0

NeedsCompilation yes

Author Diego J. Pedregal [aut, ctb] (ORCID:
<https://orcid.org/0000-0003-4958-0969>),
 Ivan Svetunkov [aut, cre] (Senior Lecturer at Centre for Marketing
 Analytics and Forecasting, Lancaster University, UK, ORCID:
<https://orcid.org/0000-0001-7826-0281>)

Maintainer Ivan Svetunkov <ivan@svetunkov.com>

Repository CRAN

Date/Publication 2026-06-30 20:30:17 UTC

Contents

initials	2
muse	3
pts	3
pts-methods	6
Index	11

initials	<i>Initial state values for a fitted pts object.</i>
----------	--

Description

Returns the smoothed structural states at the first observation, with the "Error" and "Fit" columns dropped. For deterministic components – e.g. the slope under the global (G) trend, where the slope is fixed throughout the horizon – this equals the $t = 0$ initial value; for stochastic components it is the smoother's estimate at $t = 1$, a close proxy to the $t = 0$ initial.

Usage

```
initials(object, ...)
```

Arguments

object	A fitted object of class "pts".
...	Unused.

Value

Named numeric vector of initial structural-state values.

muse

muse package

Description

Package contains functions implementing Multiple Source of Error state space models for purposes of time series analysis and forecasting.

Details

Package: muse
Type: Package
Date: 2024-10-18 - Inf
License: LGPL-2.1

The following functions are included in the package:

Author(s)

Diego J. Pedregal, <Diego.Pedregal@uclm.es>

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

[adam](#)

Examples

```
x <- rnorm(100,100,10)
```

pts

pts: Power / Trend / Seasonal state-space model

Description

Estimates a PTS (Power / Trend / Seasonal) state-space model for a univariate time series. This is the user-facing entry point of the **muse** package and mirrors the calling convention used elsewhere in the **smooth** family: `pts()` estimates the model, and `forecast.pts` produces forecasts from the fitted object without re-estimating.

Usage

```
pts(data, model = "ZZZ", lags = stats::frequency(data), orders = list(ar
= 0, ma = 0, select = FALSE), formula = NULL, regressors = c("use"),
outliers = c("ignore", "use", "select"), level = 0.99, ic = c("AICc",
"BICc", "BIC", "AIC"), lambda_estim = c("likelihood", "guerrero",
"decomp-guerrero"), biasadj = FALSE, h = 0, holdout = FALSE,
verbose = FALSE, ...)
```

Arguments

data	response series. Either a univariate ts / zoo / numeric vector, OR a matrix / data.frame whose first column is the response and whose remaining columns are external regressors (xregs).
model	3-letter PTS specification string. The three positions encode Power / Trend / Seasonal: <ul style="list-style-type: none"> • Power: Z to estimate Box-Cox λ, or a numeric value (e.g. "0", "0.5", "1"). • Trend: Z (auto), N (none / random walk), L (local linear), D (damped / smooth random walk), G (global / deterministic). • Seasonal: Z (auto), N (none), D (discrete / linear), T (trigonometric / equal).
lags	seasonal period (default frequency(data)). Scalar for the structural seasonal; may also be a vector c(1, s) mirroring smooth::adam's convention – the last entry is the structural period, the full vector becomes the default lag set for the irregular's ARMA blocks (overridable per-fit via orders\$lags).
orders	ARMA / SARMA spec for the irregular component. Three forms: <ul style="list-style-type: none"> • Full list list(ar, ma, select) for a non-seasonal $ARMA(p, q)$ – ar, ma non-negative scalars (default 0); select = TRUE asks the engine to search ARMA orders up to that cap. • Numeric shortcut c(p, q) – equivalent to list(ar = p, ma = q, select = FALSE); c(p) is treated as c(p, 0). • Seasonal list(ar = c(p, P), ma = c(q, Q), lags = c(1, s)) for $SARMA(p, q)(P, Q)_s$ – ar / ma are length-L vectors paired position-wise with lags, with lags[1] = 1 (non-seasonal block) and lags[2] = s (seasonal block). If orders\$lags is omitted the default falls back to the top-level lags argument (or c(1, frequency(data))). The seasonal SARMA polynomial is multiplied internally, so the BFGS only optimises the free $\phi_i, \Phi_j, \theta_i, \Theta_j$ coefficients. select = TRUE runs a grid search over every (p', q', P', Q') tuple with $0 \leq p' \leq ar[1]$ and so on, and picks the candidate with the lowest ic. <p>PTS has no differencing – any orders\$i supplied is silently ignored.</p>
formula	optional formula response ~ x1 + x2 + ...; only meaningful when data is a matrix or data.frame. Used to pick the response column + xreg columns explicitly.
regressors	handling of xregs. Currently only "use" (apply all supplied xregs as fixed-coefficient covariates). Adam's "select" and "adapt" modes are not yet implemented.

outliers	what to do about outliers, mirroring <code>adam()</code> 's interface: "ignore" (default – fit ignores the possibility of outliers) or "use" (run the engine's outlier detector once after the structural fit, classify each event as AO / LS / SC, and refit with the detected events as fixed regressor dummies). Adam's "select" mode (IC-pruning of detected dummies) is not yet supported – passing it errors with a clear message. When <code>outliers = "use"</code> the detected events are returned on the fitted object as <code>\$outliersDetected</code> (a data frame with <code>time</code> and <code>type</code> columns) and the corresponding dummy coefficients appear in <code>coef(m)</code> as <code>Beta(...)</code> entries.
level	confidence level driving the outlier z-score threshold (default 0.99). Translated to a two-sided z via <code>qnorm((1 + level) / 2)</code> : 0.99 -> ≈ 2.576 , 0.95 -> ≈ 1.960 . The z drives the AO threshold; LS / SC scale proportionally to preserve the engine's relative stiffness (LS $\approx 1.087 * z$, SC $\approx 1.304 * z$). Ignored when <code>outliers = "ignore"</code> .
ic	information criterion for automatic model selection; one of "AICc" (default), "BICc", "BIC", "AIC". Matches the adam option set; AICc is the default, as in adam.
lambda_estim	how the Box-Cox power λ is chosen when the power slot of model is "Z"; one of: <ul style="list-style-type: none"> "likelihood" (default) – estimate λ jointly with the structural parameters by maximum likelihood in the engine. "guerrero" – the classical Guerrero (1993) variance- stabilisation screen on raw season-length blocks. "decomp-guerrero" – Guerrero on an <code>msdecompose</code>-smoothed trend (the former default). Ignored when a numeric power is supplied (e.g. " <code>0.5ZZ</code> ").
biasadj	logical (default FALSE). Point forecasts are the back-transformed conditional <i>median</i> $g^{-1}(\mu)$. When TRUE, a second-order bias correction is applied so the point forecast approximates the conditional <i>mean</i> (as in <code>forecast : InvBoxCox(biasadj = TRUE)</code>). Prediction-interval quantiles are unaffected. Has no effect at $\lambda = 1$.
h	forecast horizon. If $h > 0$ a forecast is computed at fit time and cached on the object; <code>forecast(object, h)</code> can later recompute for a different horizon cheaply.
holdout	logical. If TRUE and $h > 0$, the last h observations of data are withheld from estimation and returned in <code>\$holdout</code> for later accuracy assessment.
verbose	logical: print intermediate optimisation output.
...	advanced / undocumented passthroughs. Supported keys: <ul style="list-style-type: none"> B - numeric vector of starting values for the optimiser (natural-scale variances, in the order returned in <code>\$B</code> by a default fit). Mirrors the same hatch in <code>smooth : adam()</code>. The optimised vector is returned in the <code>\$B</code> slot of the output regardless of whether the user supplied one.

Value

An object of class `c("pts", "smooth")`. Slot names mirror `smooth : adam()`'s return list where the concept is shared; pts-only extensions are flagged below.

- Inputs / spec: `y`, `model`, `modelUC*`, `lags`, `lambda*`

- Parameters: B (estimated parameter vector), covp* (parameter covariance), nParam – an adam-style 2 x 5 matrix (rows Estimated / Provided; columns nParamInternal, nParamXreg, nParamOccurrence, nParamScale, nParamAll). nparam() returns the [Estimated, nParamAll] cell. Estimated structural initials (level/slope, cycle, seasonal states) are folded into nParamInternal, exactly as smooth::adam does
- In-sample fit: fitted, residuals, states plus pts-specific comp* (additive BC-scale decomposition with Error/Fit columns)
- Cached forecast: forecast (original scale, if h > 0) and forecast_args* for cheap re-forecasting
- Likelihood + diagnostics: logLik, table* (C++ validation text)
- Scalars read by plot.smooth/diagnostics: distribution = "dnorm", loss = "likelihood", occurrence = NULL, holdout
- Bookkeeping: call, timeElapsed

AIC / AICc / BIC / BICc are derived on demand via the methods, not stored on the object. (* = pts-specific extension.)

Author(s)

Diego J. Pedregal, <Diego.Pedregal@uclm.es>

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

[forecast.pts](#)

Examples

```
# Automatic model selection (Power / Trend / Seasonal) on monthly data
model <- pts(AirPassengers, model = "ZZZ", h = 12, holdout = TRUE)
model

# A fixed specification: no transform, local-linear trend, trigonometric
# seasonality, with a 12-step forecast
fixedModel <- pts(AirPassengers, model = "1LT", h = 12)
forecast(fixedModel, h = 12)
```

Description

Standard accessors and printing for fitted pts objects. `forecast(object, h)` generates forecasts from the fitted parameters without re-running the optimiser; `predict(object)` returns the in-sample fitted values.

`print.pts` mirrors `smooth::print.adam` (`smooth/R/adam.R:5862`) section by section, substituting the PTS-specific MSOE concepts where ETS-specific ones do not apply: the "initialisation" line becomes the Box-Cox λ block, and the "Persistence vector g" block becomes the MSOE innovation variances (the same variances are otherwise the structural pieces of the B parameter vector). The C++ validation diagnostics live on `$cppOutput` should the user want them.

`forecast.pts` uses the C++ `forecastOnly` entry point: it skips re-estimation and just propagates the Kalman filter h steps forward from the fitted state, so changing h is cheap. `interval` selects the variance source:

"prediction" (default) state propagation + future shocks (the engine's `yForV`). Bands the next observation.

"confidence" conditional-mean variance. In a state-space model $\text{var}(E[y_{t+h} | \text{obs}]) = \text{var}(y_{t+h} | \text{obs}) - \sigma^2$, where σ^2 is the BC-scale residual variance: we read it straight off the fitted scale, no re-forecast needed.

"simulated" empirical quantiles of `nsim` forward paths from `simulate.pts()`. Returns the path matrix in `$scenarios` when `scenarios = TRUE`.

"none" no bands; lower = upper = mean.

`level` accepts a vector; lower / upper are then $h \times nLevels$ matrices (or $1 \times nLevels$ when `cumulative = TRUE`). `side` produces two-sided / upper-only / lower-only intervals – the absent side is set to $\mp\infty$ on the BC scale and back-transformed to the original-scale support boundary (0 for $\lambda > 0$, $-\infty$ for the identity transform). `cumulative = TRUE` collapses the horizon into one total – exact for `interval = "simulated"` (sum within each path); the engine does not expose cross-step state covariance, so the other intervals fall back to simulation totals.

Usage

```
## S3 method for class 'pts'
print(x, digits = 4, ...)

## S3 method for class 'pts'
plot(x, which = c(1, 2, 4, 6), ...)

## S3 method for class 'pts'
fitted(object, ...)

## S3 method for class 'pts'
residuals(object, ...)

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

## S3 method for class 'pts'
```

```

vcov(object, ...)

## S3 method for class 'pts'
nobs(object, all = FALSE, ...)

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

## S3 method for class 'pts'
predict(object, newdata = NULL, ...)

## S3 method for class 'pts'
forecast(object, h = 10, newdata = NULL,
  interval = c("prediction", "confidence", "simulated", "none"),
  level = 0.95, side = c("both", "upper", "lower"), cumulative = FALSE,
  nsim = NULL, scenarios = FALSE, biasadj = NULL, ...)

## S3 method for class 'pts'
initials(object, ...)

```

Arguments

<code>x, object</code>	A fitted object of class "pts".
<code>digits</code>	number of significant digits used in printed output.
<code>...</code>	further arguments passed to underlying generics.
<code>which</code>	integer vector of plot panels to draw; passed through to <code>plot.smooth</code> (see <code>?smooth::plot.adam</code>). Defaults to <code>c(1, 2, 4, 6)</code> .
<code>all</code>	logical; if TRUE the holdout sample is included in the observation count. Default FALSE.
<code>newdata</code>	unused (reserved for forecast paths with new regressors).
<code>h</code>	forecast horizon (steps ahead).
<code>interval</code>	interval type; one of "prediction", "confidence", "simulated", or "none".
<code>level</code>	confidence level for prediction intervals (default 0.95).
<code>side</code>	one of "both", "upper", "lower" – selects two-sided vs one-sided intervals.
<code>cumulative</code>	if TRUE, return the cumulative h-step total.
<code>nsim</code>	number of simulated paths when <code>interval = "simulated"</code> or under the cumulative fallback (default 10000).
<code>scenarios</code>	if TRUE and <code>interval = "simulated"</code> , return the simulated path matrix as <code>\$scenarios</code> .
<code>biasadj</code>	point-forecast back-transform: FALSE (median, the default) or TRUE (bias-corrected mean). Defaults to the value the model was fitted with (<code>object\$biasadj</code>).

Value

The methods return the following:

- `fitted`, `predict` – the in-sample fitted values as a `ts` / numeric vector on the original (back-transformed) scale.
- `residuals` – the model residuals as a `ts` / numeric vector on the Box-Cox scale.
- `coef` – a named numeric vector of the estimated parameters (structural variances plus any ARMA / damping coefficients).
- `vcov` – the parameter variance-covariance matrix.
- `nobs` – an integer, the number of in-sample observations (plus the holdout when `all = TRUE`).
- `nparam` – an integer, the number of estimated parameters (the degrees of freedom used by the information criteria).
- `logLik` – an object of class "logLik": the maximised log-likelihood, with `df` and `nobs` attributes.
- `sigma`, `extractSigma` – a numeric scalar, the residual standard deviation; `extractScale` – the maximum-likelihood scale of the assumed distribution.
- `actuals` – the original response series.
- `modelType` – a character string with the PTS model code (e.g. "PTS(0,N,T)"); `modelName` – a human-readable description; `errorType` – the character error type ("A", additive on the Box-Cox scale).
- `lags` – the integer seasonal period; `orders` – a list of the irregular ARMA orders (`ar`, `ma`, `lags`); `initials` – a named numeric vector of the estimated initial structural-state values.
- `forecast` – an object of class "pts.forecast": a list with the point forecast `$mean` and the interval bounds `$lower` / `$upper` (original scale), together with the forecast variance, level, and interval metadata.
- `print` and `plot` are called for their side effects (console output and a diagnostic plot, respectively) and invisibly return their first argument.

Author(s)

Diego J. Pedregal, <Diego.Pedregal@uclm.es>

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Granger, C. W. J., & Newbold, P. (1976). Forecasting transformed series. *Journal of the Royal Statistical Society: Series B (Methodological)*, 38(2), 189-203. doi:10.1111/j.2517-6161.1976.tb01585.x
- Pankratz, A., & Dudley, U. (1987). Forecasts of power-transformed series. *Journal of Forecasting*, 6(4), 239-248. doi:10.1002/for.3980060403
- Guerrero, V. M. (1993). Time-series analysis supported by power transformations. *Journal of Forecasting*, 12(1), 37-48. doi:10.1002/for.3980120104

Examples

```
model <- pts(AirPassengers, model = "1LT", h = 12, holdout = TRUE)
print(model)
fitted(model)
residuals(model)
# forecast 12 steps ahead with 95% prediction intervals
forecast(model, h = 12, interval = "prediction", level = 0.95)
```

Index

- * **models**
 - muse, 3
- * **nonlinear**
 - muse, 3
- * **regression**
 - muse, 3
- * **smooth**
 - muse, 3
- * **ts**
 - muse, 3
- * **univar**
 - muse, 3

adam, 3

coef.pts (pts-methods), 6

fitted.pts (pts-methods), 6

forecast.pts, 3, 6

forecast.pts (pts-methods), 6

initials, 2

initials.pts (pts-methods), 6

logLik.pts (pts-methods), 6

muse, 3

muse-package (muse), 3

nobs.pts (pts-methods), 6

plot.pts (pts-methods), 6

predict.pts (pts-methods), 6

print.pts (pts-methods), 6

pts, 3

pts-methods, 6

residuals.pts (pts-methods), 6

vcov.pts (pts-methods), 6