

# Package: muse (via r-universe)

July 1, 2026

**Type** Package

**Title** Multiple Unobserved Sources of Error State Space Models

**Version** 0.1.0

**Date** 2026-06-22

**URL** <https://github.com/config-11/muse>

**BugReports** <https://github.com/config-11/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

**Roxygen** list(old\_usage = TRUE)

**ByteCompile** true

**Config/roxygen2/version** 8.0.0

**RoxygenNote** 8.0.0

**Config/pak/sysreqs** cmake texlive libssl-dev

**Repository** https://config-i1.r-universe.dev

**Date/Publication** 2026-06-22 17:52:57 UTC

**RemoteUrl** https://github.com/config-i1/muse

**RemoteRef** HEAD

**RemoteSha** ec3f6a78243c484bde63045607f3c8f8dd239b59

## Contents

initials . . . . .	2
muse . . . . .	3
pts . . . . .	3
pts-methods . . . . .	6
<b>Index</b>	<b>9</b>

---

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("BICc",
  "AICc", "BIC", "AIC"), h = 0, holdout = FALSE, verbose = FALSE, ...)
```

**Arguments**

data	response series. Either a univariate <code>ts</code> / <code>zoo</code> / numeric vector, OR a matrix / <code>data.frame</code> whose first column is the response and whose remaining columns are external regressors ( <code>xregs</code> ).
model	3-letter PTS specification string. The three positions encode Power / Trend / Seasonal: <ul style="list-style-type: none"> <li>• Power: Z to estimate Box-Cox <math>\lambda</math>, or a numeric value (e.g. "0", "0.5", "1").</li> <li>• Trend: Z (auto), N (none / random walk), L (local linear), D (damped / smooth random walk), G (global / deterministic).</li> <li>• Seasonal: Z (auto), N (none), D (discrete / linear), T (trigonometric / equal).</li> </ul>
lags	seasonal period (default <code>frequency(data)</code> ). Scalar for the structural seasonal; may also be a vector <code>c(1, s)</code> mirroring <code>smooth::adam</code> '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 <code>orders\$lags</code> ).
orders	ARMA / SARMA spec for the irregular component. Three forms: <ul style="list-style-type: none"> <li>• Full list <code>list(ar, ma, select)</code> for a non-seasonal <math>ARMA(p, q)</math> – <code>ar</code>, <code>ma</code> non-negative scalars (default 0); <code>select = TRUE</code> asks the engine to search ARMA orders up to that cap.</li> <li>• Numeric shortcut <code>c(p, q)</code> – equivalent to <code>list(ar = p, ma = q, select = FALSE)</code>; <code>c(p)</code> is treated as <code>c(p, 0)</code>.</li> <li>• Seasonal <code>list(ar = c(p, P), ma = c(q, Q), lags = c(1, s))</code> for <math>SARMA(p, q)(P, Q)_s</math> – <code>ar</code> / <code>ma</code> are length-L vectors paired position-wise with <code>lags</code>, with <code>lags[1] = 1</code> (non-seasonal block) and <code>lags[2] = s</code> (seasonal block). If <code>orders\$lags</code> is omitted the default falls back to the top-level <code>lags</code> argument (or <code>c(1, frequency(data))</code>). The seasonal SARMA polynomial is multiplied internally, so the BFGS only optimises the free <math>\phi_i, \Phi_j, \theta_i, \Theta_j</math> coefficients. <code>select = TRUE</code> runs a grid search over every <math>(p', q', P', Q')</math> tuple with <math>0 \leq p' \leq ar[1]</math> and so on, and picks the candidate with the lowest ic.</li> </ul> <p>PTS has no differencing – any <code>orders\$i</code> supplied is silently ignored.</p>
formula	optional formula <code>response ~ x1 + x2 + ...</code> ; only meaningful when data is a matrix or <code>data.frame</code> . Used to pick the response column + <code>xreg</code> columns explicitly.
regressors	handling of <code>xregs</code> . Currently only "use" (apply all supplied <code>xregs</code> 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 `outliers = "use"` the detected events are returned on the fitted object as `$outliersDetected` (a data frame with `time` and `type` columns) and the corresponding dummy coefficients appear in `coef(m)` as `Beta(...)` 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 -> $\approx 2.576$ , 0.95 -> $\approx 1.960$ . 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 "BICc" (default), "AICc", "BIC", "AIC". Matches the adam option set. BICc is the default because its stiffer complexity penalty guards against runaway-trend candidates whose likelihood beats simpler shapes by only a couple of units.
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"> <li>• 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.</li> </ul>

## 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`
- 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)
```

pts-methods

*S3 methods for objects of class pts***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  $\lambda$  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  $\sigma^2$  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, ...)

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

```

**Arguments**

x, object	A fitted object of class "pts".
digits	number of significant digits used in printed output.
...	further arguments passed to underlying generics.
which	integer vector of plot panels to draw; passed through to plot.smooth (see ?smooth::plot.adam). Defaults to c(1, 2, 4, 6).
all	logical; if TRUE the holdout sample is included in the observation count. Default FALSE.
newdata	unused (reserved for forecast paths with new regressors).

h	forecast horizon (steps ahead).
interval	interval type; one of "prediction", "confidence", "simulated", or "none".
level	confidence level for prediction intervals (default 0.95).
side	one of "both", "upper", "lower" – selects two-sided vs one-sided intervals.
cumulative	if TRUE, return the cumulative h-step total.
nsim	number of simulated paths when interval = "simulated" or under the cumulative fallback (default 10000).
scenarios	if TRUE and interval = "simulated", return the simulated path matrix as \$scenarios.

### 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](https://doi.org/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](https://doi.org/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](https://doi.org/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