Package 'jumps'

May 8, 2026

Type Package

Title Hodrick-Prescott Filter with Jumps

Version 1.0

Date 2025-03-20

Maintainer Matteo Pelagatti <matteo.pelagatti@unimib.it>

Description A set of functions to compute the Hodrick-Prescott (HP) filter with automatically selected jumps. The original HP filter extracts a smooth trend from a time series, and our version allows for a small number of automatically identified jumps. See Maranzano and Pelagatti (2024) <doi:10.2139/ssrn.4896170> for details.

License GPL-3

Imports Rcpp (>= 1.0.10), stats, nloptr

LinkingTo Rcpp

RoxygenNote 7.3.2

Suggests knitr, rmarkdown, testthat (>= 3.0.0), ggplot2, xts

Config/testthat/edition 3

VignetteBuilder knitr

Encoding UTF-8

LazyData true

NeedsCompilation yes

Author Matteo Pelagatti [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-1860-7535>), Paolo Maranzano [aut, cph] (ORCID: <https://orcid.org/0000-0002-9228-2759>)

Depends R (>= 3.5.0)

Repository CRAN

Date/Publication 2025-03-24 15:00:06 UTC

Contents

`jumps-package` *Hodrick-Prescott Filter with Jumps*

Description

Extraction of a smooth trend with automatically selected jumps.

Details

This package implements a version of the HP filter and of smoothing splines that allow discontinuities (jumps). The jumps are automatically selected using a LASSO-like regularization and the optimal regularization constant can be found providing a grid of alternatives and using information criteria. The user can also add regressors such as deterministic seasonal components.

Author(s)

Matteo Pelagatti, Paolo Maranzano.

Maintainer: Matteo Pelagatti <matteo.pelagatti@unimib.it>, Paolo Maranzano <paolo.maranzano@unimib.it>

References

Maranzano and Pelagatti (2024) A Hodrick-Prescott filter with automatically selected jumps.

auto_hpfj *Automatic selection of the optimal HP filter with jumps*

Description

The regularization constant for the HP filter with jumps is the maximal sum of standard deviations for the level disturbance. This value has to be passed to the hpfj function. The auto_hpfj runs hpfj on a grid of regularization constants and returns the output of hpfj selected by the chosen information criterion.

Usage

```
auto_hpfj(
  y,
  grid = seq(0, sd(y, na.rm = TRUE) * 10, sd(y, na.rm = TRUE)/10),
  ic = c("bic", "hq", "aic", "aicc"),
  edf = TRUE
)
```

Arguments

y	numeric vector cotaining the time series;
grid	numeric vector containing the grid for the argument maxsum of the hpfj function;
ic	string with information criterion for the choice: the default is "bic" (simulations show this is the best choice), but also "hq", "aic" and "aicc" are available;
edf	logical scalar: TRUE (default) if the number of degrees of freedom should be computed as "effective degrees of freedom" (Efron, 1986) as opposed to a more traditional way (although not supported by theory) when FALSE.

Value

The same ouput as the hpjf function corresponding to the best choice according to the selected information criterion.

Examples

```
mod <- auto_hpfj(Nile)
plot(as.numeric(Nile))
lines(mod$smoothed_level)
```

auto_hpfjx *Automatic selection of the optimal HP filter with jumps and regressors*

Description

This function needs more testing since it does not seem to work as expected. For this reasin the wrapper hpj at the moment does not allow regressors. The regularization constant for the HP filter with jumps is the maximal sum of standard deviations for the level disturbance. This value has to be passed to the hpfjx function. The auto_hpfjx runs hpfjx on a grid of regularization constants and returns the output of hpfjx selected by the chosen information criterion.

Usage

```
auto_hpfjx(
  y,
  X,
  grid = seq(0, sd(y) * 10, sd(y)/10),
  ic = c("bic", "hq", "aic", "aicc"),
  edf = TRUE
)
```

Arguments

y	numeric vector cotaining the time series;
X	numeric matrix with regressors in the columns;
grid	numeric vector containing the grid for the argument maxsum of the hpfj function;
ic	string with information criterion for the choice: the default is "bic" (simulations show this is the best choice), but also "hq", "aic" and "aicc" are available;
edf	logical scalar: TRUE (default) if the number of degrees of freedom should be computed as "effective degrees of freedom" (Efron, 1986) as opposed to a more traditional way (although not supported by theory) when FALSE.

Value

The ouput of the hpjf function corresponding to the best choice according to the selected information criterion.

Examples

```
y <- log(AirPassengers)
n <- length(y)
mod <- auto_hpfjx(y, trigseas(n, 12))
hpj <- ts(mod$smoothed_level, start(y), frequency = 12)
plot(y)
lines(hpj, col = "red")
```

auto_hpfj_fix *Automatic selection of the optimal HP filter with jumps and fixed smoothing constant*

Description

The regularization constant for the HP filter with jumps is the maximal sum of standard deviations for the level disturbance. This value has to be passed to the hpfj_fix function. The function auto_hpfj_fix runs hpfj_fix on a grid of regularization constants and returns the output of hpfj_fix according to the chosen information criterion.

Usage

```
auto_hpfj_fix(
  y,
  lambda,
  grid = seq(0, sd(y) * 10, sd(y)/10),
  ic = c("bic", "hq", "aic", "aicc"),
  edf = TRUE
)
```

Arguments

y	numeric vector cotaining the time series;
lambda	smoothing constant;
grid	numeric vector containing the grid for the argument maxsum of the hpfj function;
ic	string with information criterion for the choice: the default is "bic" (simulations show this is the best choice), but also "hq", "aic" and "aicc" are available;
edf	logical scalar: TRUE (default) if the number of degrees of freedom should be computed as "effective degrees of freedom" (Efron, 1986) as opposed to a more traditional way (although not supported by theory) when FALSE.

Value

The ouput of the hpjf function corresponding to the best choice according to the selected information criterion.

Examples

```
mod <- auto_hpfj_fix(Nile, "annual")
plot(as.numeric(Nile))
lines(mod$smoothed_level)
```

BIC.hpj *BIC method for the class hpj*

Description

BIC method for the class hpj

Usage

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

Arguments

object	an object of class hpj;
...	additional objects of class hpj.

Value

If just one object is provided, a numeric value with the corresponding BIC. If multiple objects are provided, a data.frame with rows corresponding to the objects and columns representing the number of parameters in the model (df) and the BIC.

da *Internal function for computing scores w/r to regression coefficients*

Description

This function, not intended for end-users, implements the following recursions needed in computing scores with respect to regression coefficients:

$$Da_{t+1}^{(1)} = Da_t^{(1)} + Da_t^{(2)} - k_t^{(1)} x_t - k_t^{(1)} Da_t^{(1)}$$

$$Da_{t+1}^{(2)} = a_t^{(2)} - k_t^{(2)} x_t - k_t^{(2)} Da_t^{(1)}$$

where $a_t^{(1)}$, $a_t^{(2)}$ are the one-step-ahead Kalman filtered state variables, and $k_t^{(1)}$, $k_t^{(2)}$ the respective Kalman gain elements. The symbol D represent the partial derivative with respect to the regression coefficients and x_t is the vector of regressors. All variables are passed by reference and, so, no output is needed.

Usage

```
da(k1, k2, X, A1, A2)
```

Arguments

k1	numeric vector of n elements with the Kalman gain sequence for the first state variable;
k2	numeric vector of n elements with the Kalman gain sequence for the second state variable;
X	numeric matrix of dimension $n \times k$ with the regressors;
A1	numeric matrix of dimension $n \times k$ that, after calling the function will contain the sequence of gradients $Da_t^{(1)}$; the first row must be of zero values;
A2	numeric matrix of dimension $n \times k$ that, after calling the function will contain the sequence of gradients $Da_t^{(2)}$; the first row must be of zero values;

Value

It does not return anything as it writes on the A1 and A2 matrices passed as reference.

---

dummyseas *Seasonal dummy variables*

Description

It produces a matrix with seasonal dummies summing to zero to be used as regressors. If s is the seasonal period, then the j-th dummy equals 1 in season j and $-1/(s-1)$ in the other seasons, so that the variable sums to 0 over one year period.

Usage

```
dummyseas(n, s, remove = s)
```

Arguments

n	length of time series;
s	seasonal period;
remove	to avoid collinearity with the constant remove a column, by default it is column s; if NULL no column is removed.

Value

It returns a matrix with n rows and $s-1$ column unless `remove = NULL`.

Examples

```
y <- log(AirPassengers)
X <- dummyseas(length(y), 12)
X <- cbind(X, t = 1:length(y))
reg <- lm(y~X)
```

employed_IT *Dataset: employed_IT*

Description

A dataset containing the quarterly time series of employed people by age class in Italy in the time span 1992Q4-2024Q1.

Usage

```
employed_IT
```

Format

A multivariate time series with 126 rows and 10 columns:

Y15.24 Thousand of employed in the age class 15-24

Y15.64 Thousand of employed in the age class 15-64

Y20.64 Thousand of employed in the age class 20-64

Y25.54 Thousand of employed in the age class 25-54

Y55.64 Thousand of employed in the age class 55-64

Y15.29 Thousand of employed in the age class 15-29

Y15.74 Thousand of employed in the age class 15-74

Y25.29 Thousand of employed in the age class 25-29

Y29.54 Thousand of employed in the age class 29-54

Y29.74 Thousand of employed in the age class 29-74

Source

Eurostat

hpfj *HP filter with automatic jumps detection.*

Description

This is the lower-level function for the HP filter with jumps. The user should use the `hpj` function instead, unless in need of more control and speed. The function estimates the HP filter with jumps. Jumps happen contextually in the level and in the slope: the standard deviation of the slope disturbance is γ times the standard deviation of the level disturbance at time t. The HP smoothing parameter λ is estimated via MLE (assuming normally distributed disturbances) as in Wahba (1978): $\lambda = \sigma_{\varepsilon}^2/\sigma_{\zeta}^2$.

Usage

```
hpfj(y, maxsum = sd(y), edf = TRUE, parinit = NULL)
```

Arguments

y	vector with the time series;
maxsum	maximum sum of additional level standard deviations;
edf	boolean if TRUE computes effective degrees of freedom otherwise computes the number of degrees of freedom in the LASSO-regression way;
parinit	either NULL or vector of 3+n parameters with starting values for the optimizer; the order of the parameters is sd(slope disturbnce), sd(observatio noise), square root of gamma, n additional std deviations for the slope.

Value

list with the following slots:

- opt: the output of the optimization function (nloptr)
- nobs: number of observations
- df: number of estimated parameters (model's degrees of freedom)
- loglik: value of the log-likelihood at maximum
- ic: vector of information criteria (aic, aicc, bic, hq)
- smoothed_level: vector with smoothed level with jumps (hp filter with jumps)
- var_smoothed_level: variance of the smoothed level

References

Whaba (1978) "Improper priors, spline smoothing and the problem of guarding against model errors in regression", *Journal of the Royal Statistical Society. Series B*, Vol. 40(3), pp. 364-372. DOI:10.1111/j.2517-6161.1978.tb01050.x

Examples

```
set.seed(202311)
n <- 100
mu <- 100*cos(3*pi/n*(1:n)) - ((1:n) > 50)*n - c(rep(0, 50), 1:50)*10
y <- mu + rnorm(n, sd = 20)
plot(y, type = "l")
lines(mu, col = "blue")
hp <- hpfj(y, 60)
lines(hp$smoothed_level, col = "red")
```

hpfjx *HP filter with jumps and regressors (still experimental)*

Description

This function needs more testing since it does not seem to work as expected. For this reasin the wrapper hpj at the moment does not allow regressors. This is the same as hpfj but with the possibility of including regressors. The regressors should be zero-mean so that the HP filter can be interpreted as a mean value of the time series. Jumps happen contextually in the level and in the slope: the standard deviation of the slope disturbance is γ times the standard deviation of the level disturbance at time t. The HP smoothing parameter λ is estimated via MLE (assuming normally distributed disturbances) as in Wahba (1978): $\lambda = \sigma_\varepsilon^2/\sigma_\zeta^2$.

Usage

```
hpfjx(y, X, maxsum = sd(y), edf = TRUE, parinit = NULL)
```

Arguments

y	vector with the time series
X	matrix with regressors in the columns
maxsum	maximum sum of additional level standard deviations;
edf	boolean if TRUE computes effective degrees of freedom otherwise computes the number of degrees of freedom in the LASSO-regression way.
parinit	either NULL or vector of 3+n parameters with starting values for the optimizer; the order of the parameters is sd(slope disturbnce), sd(observatio noise), square root of gamma, n additional std deviations for the slope

Value

list with the following slots:

- opt: the output of the optimization function (nloptr)
- nobs: number of observations
- df: number of estimated parameters (model's degrees of freedom)
- loglik: value of the log-likelihood at maximum
- ic: vector of information criteria (aic, aicc, bic, hq)
- smoothed_level: vector with smoothed level with jumps (hp filter with jumps)
- var_smoothed_level: variance of the smoothed level

References

Whaba (1978) "Improper priors, spline smoothing and the problem of guarding against model errors in regression", *Journal of the Royal Statistical Society. Series B*, Vol. 40(3), pp. 364-372. DOI:10.1111/j.2517-6161.1978.tb01050.x

Examples

```
y <- log(AirPassengers)
n <- length(y)
mod <- hpfjx(y, trigseas(n, 12))
hpj <- ts(mod$smoothed_level, start(y), frequency = 12)
plot(y)
lines(hpj, col = "red")
```

---

hpfj_fix *HP filter with automatic jumps detection and fixed smoothing constant*

Description

This is the lower-level function for the HP filter with jumps with fixed smoothing parameter. The user should use the hpj function instead, unless in need of more control and speed. The function estimates the HP filter with jumps. Jumps happen contextually in the level and in the slope: the standard deviation of the slope disturbance is γ times the standard deviation of the level disturbance at time t. In this case the HP smoothing parameter λ is fixed by the user and so that $\sigma_\varepsilon^2 = \lambda\sigma_\zeta^2$.

Usage

```
hpfj_fix(y, lambda, maxsum = sd(y), edf = TRUE, parinit = NULL)
```

Arguments

y	vector with the time series
lambda	either a numeric scalar with the smoothing constant or a string with the frequency of the time series to be selected among c("daily", "weekly", "monthly", "quarterly", "annual"); in this case the values of the smoothing constant are computed according to Ravn and Uhlig (2002), that is, $6.25s^4$, where s is the number of observations per year.
maxsum	maximum sum of additional level standard deviations;
edf	boolean if TRUE computes effective degrees of freedom otherwise computes the number of degrees of freedom in the LASSO-regression way.
parinit	either NULL or vector of 3+n parameters with starting values for the optimizer; the order of the parameters is sd(slope disturbnce), sd(observatio noise), square root of gamma, n additional std deviations for the slope

Value

list with the following slots:

- opt: the output of the optimization function (nloptr)
- nobs: number of observations
- df: number of estimated parameters (model's degrees of freedom)

- loglik: value of the log-likelihood at maximum
- ic: vector of information criteria (aic, aicc, bic, hq)
- smoothed_level: vector with smoothed level with jumps (hp filter with jumps)
- var_smoothed_level: variance of the smoothed level

Examples

```
set.seed(202311)
n <- 100
mu <- 100*cos(3*pi/n*(1:n)) - ((1:n) > 50)*n - c(rep(0, 50), 1:50)*10
y <- mu + rnorm(n, sd = 20)
plot(y, type = "l")
lines(mu, col = "blue")
hp <- hpfj_fix(y, 60, 60)
lines(hp$smoothed_level, col = "red")
```

---

hpj *Hodrick-Prescott filter with jumps*

Description

This function is a wrapper for the various Hodrick-Prescott filters with jumps functions. The end user should use this: it is simpler and more flexible than the other functions.

Usage

```
hpj(
  y,
  maxsum = NULL,
  lambda = NULL,
  xreg = NULL,
  ic = c("bic", "hq", "aic", "aicc"),
  scl = 1000
)
```

Arguments

y	either a numeric vector or a time series object containing the time series to filter;
maxsum	maximum sum of additional level standard deviations, if NULL the value is selected automatically on a grid using the specified information criterion (ic);
lambda	smoothing constant of the HP filter, if NULL the value is estimated by maximum likelihood;
xreg	matrix of regressors;
ic	string with information criterion for the automatic choice of maxsum: the default is "bic" (simulations show this is the best choice), but also "hq", "aic" and "aicc" are available.

`scl` scaling factor for the time series (default is 1000): the time series is rescaled as (y-min(y))/(max(y)-min(y))*scl. This is done since the default starting values for the optimization seem to work well in this scale); If ‘scl‘ is set equal to the string ‘"original"‘ the time series is not rescaled.

Value

S3 object of class hpj with the following slots:

- y: the input time series;
- maxsum: the maximum sum of additional standard deviations;
- lambda: the smoothing constant of the HP filter;
- pars: vector of estimated parameters (sigma_slope, sigma_noise, gamma);
- hpj: the time series of the HP filter with jumps;
- hpj_std: the time series of the HP filter with jumps standard deviations;
- std_devs: vector of additional standard deviations of the level disturbance;
- breaks: vector of indices of the breaks;
- xreg: matrix of regressors;
- df: model's degrees of freedom;
- loglik: value of the log-likelihood at maximum;
- ic: vector of information criteria (aic, aicc, bic, hq);
- opt: the output of the optimization function (nloptr);
- call: the call to the function.

Examples

```
set.seed(202311)
n <- 100
mu <- 100*cos(3*pi/n*(1:n)) - ((1:n) > 50)*n - c(rep(0, 50), 1:50)*10
y <- mu + rnorm(n, sd = 20)
hp <- hpj(y, 50)
plot(hp)
```

`llt` *Kalman filtering and smoothing for local linear trend plus noise*

Description

It uses the power of C++, scalar computation and pointers to run the Kalman filter, the smoother and compute the log-likelihood. The R user has to supply many vectors that in most cases will be overwritten by the llt() function since they are passed by reference. All passed parameters must be numerical (floating point) vectors: any other kind of variable may cause serious problems to the stability of your system. Passing vectors of integers will make the computations fail.

Usage

```
llt(
  y,
  var_eps,
  var_eta,
  var_zeta,
  cov_eta_zeta,
  a1,
  a2,
  p11,
  p12,
  p22,
  k1,
  k2,
  i,
  f,
  r1,
  r2,
  n11,
  n12,
  n22,
  e,
  d,
  w_ = NULL
)
```

Arguments

y	vector of n observations
var_eps	vector of n variances for the observation noises
var_eta	vector of n variances for the level disturbances
var_zeta	vector of n variances for the slope disturbances
cov_eta_zeta	vector of n covariances between level and slope disturbances
a1	vector of n+1 one-step-ahead prediction of the level; the first element is the initial condition for the level at time t=1, the other elements are arbitrary and will be overwritten
a2	vector of n+1 one-step-ahead prediction of the slope; the first element is the initial condition for the slope at time t=1, the other elements are arbitrary and will be overwritten
p11	vector of n+1 one-step-ahead prediction error variance of the level; the first element is the initial condition for the level at time t=1, the other elements are arbitrary and will be overwritten
p12	vector of n+1 one-step-ahead prediction covariances for level and slope; the first element is the initial condition for the slope at time t=1, the other elements are arbitrary and will be overwritten

p22	vector of n+1 one-step-ahead prediction error variance of the slope; the first element is the initial condition for the level at time t=1, the other elements are arbitrary and will be overwritten
k1	vector of the n Kalman gains for the level equation; values are arbitrary and will be overwritten;
k2	vector of the n Kalman gains for the slope equation; values are arbitrary and will be overwritten;
i	vector of the n innovations; values are arbitrary and will be overwritten;
f	vector of the n innovatoin variances;values are arbitrary and will be overwritten;
r1	vector of the n+1 smoothers (Th.5.4 in Pelagatti, 2015) for the level equation; values are arbitrary and will be overwritten;
r2	vector of the n+1 smoothers (Th.5.4 in Pelagatti, 2015) for the slope equation; values are arbitrary and will be overwritten;
n11	vector of the n+1 variance smoothers (Th.5.4 in Pelagatti, 2015) for the level equation; values are arbitrary and will be overwritten;
n12	vector of the n+1 covariance smoothers (Th.5.4 in Pelagatti, 2015) for the level and slope; values are arbitrary and will be overwritten;
n22	vector of the n+1 variance smoothers (Th.5.4 in Pelagatti, 2015) for the slope equation; values are arbitrary and will be overwritten;
e	vector of the n+1 observation error smoothers (Th.5.4 in Pelagatti, 2015); values are arbitrary and will be overwritten;
d	vector of the n+1 observation error variance smoothers (Th.5.4 in Pelagatti, 2015); values are arbitrary and will be overwritten;
w_	NULL (default) or vector of n weights for the effect of observation y_t on the estimation of the hp filter (with jumps) at time t; values are arbitrary and will be overwritten;

Value

The value of the Gaussian log-likelihood net of the -log(2*pi)*n/2 part that can be added if needed.

`logLik.hpj` *logLik method for the class hpj*

Description

logLik method for the class hpj

Usage

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

Arguments

object	an object of class hpj;
...	not used: for consistency with generic function.

Value

An object of logLik class with the log-likelihood value and two attributes: df, the number of degrees of freedom, and nobs, the number of observations.

---

mse *Mean squared error*

Description

It computes the mean squared difference between the elements of the vectors x and y.

Usage

```
mse(y, x)
```

Arguments

y	vector
x	vector

Value

The scalar mean squared difference/error.

---

nobs.hpj *nobs method for the class hpj*

Description

nobs method for the class hpj

Usage

```
## S3 method for class 'hpj'
nobs(object, ...)
```

Arguments

object	an object of class hpj;
...	not used: for consistency with generic function.

Value

The number of (non missing) observations of the time series on which the HP filter with jumps has been applied.

`plot.hpj` *Plot method for the class hpj*

Description

Plot method for the class hpj

Usage

```
## S3 method for class 'hpj'
plot(
  x,
  prob = NULL,
  show_breaks = TRUE,
  main = "original + filter",
  use_ggplot = TRUE,
  ...
)
```

Arguments

`x`	an object of class hpj;
`prob`	coverage probability for the confidence interval of the filter;
`show_breaks`	logical, if `TRUE` vertical lines are drawn at the jumps;
`main`	title of the plot;
`use_ggplot`	logical, if `TRUE` the plot is done with ggplot2;
`...`	additional arguments passed to the plot function (if no ggplot2 is used).

Value

If `use_ggplot` is `TRUE` a ggplot object is returned, otherwise a base plot is shown and nothing is returned.

print.hpj *Print method for the class hpj*

Description

Print method for the class hpj

Usage

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

Arguments

x	an object of class hpj;
...	not used: for consistency with generic function.

Value

No return value, called for side effects

trigseas *Trigonometric seasonal variables*

Description

It produces a matrix with seasonal sinusoids to be used as regressors. By default, for $t = 1, \ldots, n$ and $j = 1, \ldots, \lfloor s/2 \rfloor$, it computes

$$\cos(2\pi j/s), \qquad \sin(2\pi j/s)$$

. Notice that if s is even the sine function at highest frequency is omitted because it equals zero for all t. The used can select a different set of harmonics.

Usage

```
trigseas(n, s, harmonics = NULL)
```

Arguments

n	length of time series;
s	seasonal period;
harmonics	vector of harmonics to be used: the cosine and sine functions are computed at frequencies `2*pi*harmonics/s`, if there is an element of `harmonics` for which `2*pi*harmonics/s` is equal to π the corresponding sine is omitted.

Value

It returns a matrix with n rows and so many columns as the harmonics (by default these are $s - 1$).

Examples

```
y <- log(AirPassengers)
X <- trigseas(length(y), 12)
X <- cbind(X, t = 1:length(y))
reg <- lm(y~X)
```

Index