Package 'spscsfa'

November 19, 2026

Title Semiparametric Smooth-Coefficient Stochastic Frontier Analysis

Version 0.2.0

Maintainer Kai Sun <ksun1@shu.edu.cn>

Description Provides semiparametric smooth-coefficient stochastic frontier analysis following Sun and Kumbhakar (2013) <doi:10.1016/j.econlet.2013.05.001> where the coefficients of the parametric part vary smoothly with a set of nonparametric variables. Inefficiency term is allowed to depend on a set of determinants through heteroskedasticity. Smooth coefficients are estimated using nonparametric regression and the remaining frontier parameters are estimated by maximum likelihood. Technical efficiency and inefficiency are computed using the Battese and Coelli (1988) <doi:10.1016/0304-4076(88)90053-X> and Jondrow et al. (1982) <doi:10.1016/0304-4076(82)90004-5> (JLMS) methods, respectively. Confidence intervals for technical efficiency are computed using the approach of Horrace and Schmidt (1996) <doi:10.1007/BF00157044>. Marginal effects of inefficiency determinants on the JLMS inefficiency estimator are computed following Kumbhakar and Sun (2013) <doi:10.1016/j.econlet.2013.04.041>. By-products of the package include least-squares cross-validation bandwidth selection for the local-linear semiparametric smooth-coefficient estimator, local-linear semiparametric smooth-coefficient estimation, and analytic gradients of the local-constant semiparametric smooth-coefficient estimator with respect to the continuous z variables following Geng and Sun (2019) <doi:10.1016/j.econlet.2019.108684>.

License AGPL (>= 3)

Encoding UTF-8

Depends R (>= 3.6)

Imports Formula, graphics, grDevices, np, progress (>= 1.2.1), stats

Config/roxygen2/version 8.1.1

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

LazyData true

NeedsCompilation no

Author Kai Sun [aut, cre]

Repository CRAN

Date/Publication 2026-11-19 11:20:15 UTC

Contents

grad . . . 2
inefficiency . . . 4
lcscoefgrad . . . 4
llscoef . . . 6
llscoefbw . . . 11
plot.spscsfa . . . 15
print.spscsfa . . . 17
print.summary.spscsfa . . . 18
ricephil . . . 19
smoothcoef . . . 20
spscsfa . . . 20
summary.spscsfa . . . 27
te . . . 29

Index **30**

`grad` *Extract gradient estimates*

Description

Extracts the gradients estimated by `spscsfa` with `gradients = TRUE`: the gradients of the smooth coefficients θ and β and of the intercept α with respect to the continuous nonparametric variables `z`, and the gradients of the JLMS inefficiency estimates with respect to the inefficiency determinants. The smooth coefficients are local-constant if the model was estimated with `spscreg = "lc"` and local-linear if it was estimated with `spscreg = "ll"`.

Usage

```
grad(x, zvar = NULL)
```

Arguments

`x`	An object of class `"spscsfa"` estimated with `gradients = TRUE`.
`zvar`	Optional character vector with the names of the variables whose gradients are extracted, as they are written in `formula` (they are the names of `x$model_summary$z_names` for the `z` variables and `x$mle_results$inefficiency_columns` for the inefficiency determinants), or their positions among the `z` variables. If `NULL` (default),

all the gradients are returned. For example, `grad(x, zvar = "z1")` extracts all the gradients with respect to `z1`.

If `z1` is not a continuous `z` variable (it is a factor), there is no gradient of the smooth coefficients with respect to it: only `inefficiency_grad`, for the set of binary variables converted from `z1` in the inefficiency equation, can be extracted. Note that `inefficiency_grad` is not available for a continuous `z` variable that is not an inefficiency determinant.

Value

A list with three elements:

`theta_beta_grad`
: The gradients of the smooth coefficients: a list with one matrix for each continuous `z` variable (named after it). The matrix for `z_l` has the columns `d(Intercept (theta))/d(z_l)`, `d(x1)/d(z_l)`, ..., `d(xp)/d(z_l)`, that is, the gradients of $\theta, \beta_1, \ldots, \beta_p$ with respect to `z_l`, with one row for each evaluation point (`ezdat` if it was supplied, the training data otherwise).

`alpha_grad`
: The gradients of α at the training data: a matrix with one column for each continuous `z` variable, named `d(Intercept (alpha))/d(z_l)`.

`inefficiency_grad`
: The gradients of the JLMS inefficiency estimates with respect to the inefficiency determinants: a matrix with one column for each determinant (the binary variables for a categorical one), or `NULL` if the model has no inefficiency determinant.

With `zvar`, the elements are restricted to the variables chosen. An element that is not available for them is `NULL`.

See Also

`smoothcoef`, `spscsfa`

Examples

```
set.seed(1)
n <- 100
x <- rnorm(n)
z1 <- runif(n)
z2 <- factor(sample(c("A", "B", "C"), n, replace = TRUE))
y <- cos(z1) + sin(z1) * x + rnorm(n, 0, 0.1) -
     exp(0.1 + 0.1 * z1) * abs(rnorm(n, 0, 1))
dat <- data.frame(y, x, z1, z2)
result <- spscsfa(y ~ x | z1 + z2, data = dat, gradients = TRUE,
                  verbose = FALSE)

# all the gradients
g <- grad(result)
names(g)

# the gradients with respect to z1
g1 <- grad(result, zvar = "z1")
```

```
head(g1$theta_beta_grad$z1)

# z2 is not continuous: only inefficiency_grad can be extracted, and a note
# is printed
g2 <- grad(result, zvar = "z2")
head(g2$inefficiency_grad)
```

`inefficiency` *Extract JLMS inefficiency estimates*

Description

Extract JLMS inefficiency estimates

Usage

```
inefficiency(x)
```

Arguments

x	An object of class 'spscsfa'.

Value

A numeric vector of JLMS conditional inefficiency estimates.

`lcscoefgrad` *Gradients of the local-constant smooth coefficient estimator*

Description

Computes the analytic gradients, with respect to the continuous z variables, of the smooth coefficients of the local-constant (LC) semiparametric smooth coefficient (SPSC) model $y_i = x_i'\beta(z_i) + u_i$ following Geng and Sun (2019). It is called by `spscsfa` or can be used independently. An `np::npscoefbw` bandwidth object with its default LC regression type and kernels can be passed into the `bws` argument of this function.

Usage

```
lcscoefgrad(bws, txdat, tydat, tzdat, ezdat = NULL)
```

Arguments

bws	The bandwidths: the bandwidth object returned by np::npscoefbw (as spscsfa holds it), an object with a component bw, or a numeric vector with one bandwidth for each column of tzdat. A numeric vector is used with the default kernels of np::npscoefbw: second-order Gaussian for continuous variables, Aitchison-Aitken for unordered and Li-Racine for ordered factors. If it has names that are those of the columns of tzdat, it is matched to them by name; otherwise it is taken in their order.
txdat	A data frame or matrix with the regressors x. The constant is added.
tydat	A numeric vector with the dependent variable.
tzdat	A data frame with the variables the coefficients depend on: numeric (continuous), factors (unordered) or ordered factors.
ezdat	Optional data frame with the evaluation points, with the same variables as tzdat. If NULL (default), the gradients are evaluated at the rows of tzdat.

Details

The LC estimator is

$$\hat{\beta}(z) = \Big(\sum_i x_i x_i' K_i(z) \Big)^{-1} \sum_i x_i y_i K_i(z).$$

Differentiating the locally weighted moment condition $\sum_i x_i \hat{u}_i(z) K_i(z) = 0$, where $\hat{u}_i(z) = y_i - x_i' \hat{\beta}(z)$, with respect to the lth continuous z variable gives

$$D_l \hat{\beta}(z) = \Big(\sum_i x_i x_i' K_i(z) \Big)^{-1} \sum_i x_i \hat{u}^*_{li}(z) K_i(z),$$

that is, another LC regression, of the transformed residuals $\hat{u}^*_{li}(z) = \hat{u}_i(z)(z_{li} - z_l)/h_l^2$, on the original regressors. This is the result for a Gaussian kernel for the continuous variables: $k\left(\frac{z_{li}-z_l}{h_l}\right) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{z_{li}-z_l}{h_l}\right)^2\right)$. Factors enter the product kernel $K_i(z)$ through kernels that do not depend on the continuous variables, so they do not change the formula.

The kernel weights are those of np::npksum. A bandwidth object from np::npscoefbw must therefore be for regtype = "lc", with fixed bandwidths (not scale factors) and a second-order Gaussian kernel without bounds for the continuous variables; its kernels for factors are used as they are. If the weighted cross-product matrix at an evaluation point is singular, a ridge is added to it and a warning is given.

Value

A named list with one element for each continuous variable in tzdat, in the order of its columns and named after them. The element for the lth continuous variable is a matrix with one row for each evaluation point and one column for each smooth coefficient, in the order of the coefficients (the constant first, then the columns of txdat). Its columns are the gradients of all smooth coefficients with respect to z_l and are named as llscoef names its gradients: d(Intercept)/d(z_l), d(x1)/d(z_l), ... The list is empty if there is no continuous variable in tzdat.

References

Xin Geng and Kai Sun (2019). Gradient estimation of the local-constant semiparametric smooth coefficient model. *Economics Letters*, 185, 108684.

See Also

`llscoef`, whose gradients are those of the local-linear estimator.

Examples

```
set.seed(1)
n <- 200
z <- runif(n)
x <- rnorm(n)
y <- sin(2 * pi * z) - cos(2 * pi * z) * x + rnorm(n, sd = 0.5)
dat <- data.frame(y, x, z)

bw <- np::npscoefbw(y ~ x | z, data = dat)
g <- lcscoefgrad(bw, txdat = dat["x"], tydat = dat$y, tzdat = dat["z"])

# g$z has the columns d(Intercept)/d(z) and d(x)/d(z); the true gradients
# are 2 * pi * cos(2 * pi * z) and 2 * pi * sin(2 * pi * z)
head(g$z)

## Not run:

# gradients on a grid of z, through 'ezdat': plotted against z, with the
# true gradients (dashed)
ez <- data.frame(z = seq(0, 1, length.out = 100))
g.ez <- lcscoefgrad(bw, txdat = dat["x"], tydat = dat$y, tzdat = dat["z"],
                    ezdat = ez)

op <- par(mfrow = c(1, 2))
plot(ez$z, g.ez$z[, "d(Intercept)/d(z)"], type = "l",
     xlab = expression(z),
     ylab = expression(partialdiff * beta[0](z) / partialdiff * z))
lines(ez$z, 2 * pi * cos(2 * pi * ez$z), lty = 2)
plot(ez$z, g.ez$z[, "d(x)/d(z)"], type = "l",
     xlab = expression(z),
     ylab = expression(partialdiff * beta[1](z) / partialdiff * z))
lines(ez$z, 2 * pi * sin(2 * pi * ez$z), lty = 2)
par(op)

## End(Not run)
```

`llscoef` *Local-linear semiparametric smooth coefficient estimation*

Description

Estimates the semiparametric smooth coefficient (SPSC) model by local-linear kernel regression, at the training data or at new evaluation points. The bandwidth object of `llscoefbw` can be passed as `bws`. It is used by `spscsfa` for the estimation of the smooth coefficients.

Usage

```
llscoef(bws, ...)

## S3 method for class 'llsmoothcoefficientbw'
llscoef(
  bws,
  formula = NULL,
  data = NULL,
  newdata = NULL,
  betas = FALSE,
  residuals = FALSE,
  gradients = FALSE,
  ...
)

## S3 method for class 'formula'
llscoef(
  bws,
  data = NULL,
  newdata = NULL,
  betas = FALSE,
  residuals = FALSE,
  gradients = FALSE,
  na.action = stats::na.omit,
  ...
)

## Default S3 method:
llscoef(
  bws,
  formula,
  data = NULL,
  newdata = NULL,
  betas = FALSE,
  residuals = FALSE,
  gradients = FALSE,
  na.action = stats::na.omit,
  ...
)

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

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

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

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

Arguments

bws	A bandwidth specification: an object of class "llsmoothcoefficientbw" returned by llscoefbw, a formula y ~ x | z (the bandwidths are then selected by llscoefbw), or a numeric vector of local-linear bandwidths, one for each variable in the z part of formula. If the bandwidths have names, as in the result of llscoefbw, they are matched to the names of the z variables, in any order: a warning is given if their order differs from that of the z variables in the formula, and an error if the names do not match. Bandwidths without names are taken in the order of the z variables. A bandwidth object of np::npscoefbw made with regtype = "ll", fixed bandwidths and the default kernels is also accepted, and its bandwidths are used; one made with another regtype, bwtype, bwscaling or kernel is not, and a vector taken from it is not suitable, either. See 'Details' for the arguments each kind of bws needs.
...	Further arguments, not used.
formula	A formula y ~ x | z (or y ~ x | z | q), where y is the dependent variable, x are the parametric regressors (the constant is added automatically) and z are the variables that the smooth coefficients depend on. Required if bws is a numeric vector. Must not be given if bws is a bandwidth object from llscoefbw, which already keeps the formula.
data	Optional data frame containing the variables in formula. If omitted, the data of the call to llscoefbw are used for a bandwidth object, and otherwise the variables are taken from the environment of formula.
newdata	Optional data frame with the evaluation points. If omitted, the training data are used. See 'Details'.
betas	Logical; if TRUE, the smooth coefficients are computed and returned. Defaults to FALSE.
residuals	Logical; if TRUE, the residuals are computed and returned. Defaults to FALSE.
gradients	Logical; if TRUE, the gradients of the smooth coefficients are computed and returned. Defaults to FALSE.
na.action	As in llscoefbw: what to do with rows that have missing values. It is used by the methods for a formula and for a vector of bandwidths. A bandwidth object from llscoefbw carries its own, which is used, so na.action is not an argument then. It applies to newdata too.
object, x	An object of class "llsmoothcoefficient".

Details

The model is $y_i = x_i'\beta(z_i) + u_i$, where x_i consists of a constant followed by the variables in the first part of `formula` and z_i holds the variables in the second part. A third part, as accepted by `spscsfa` for the inefficiency determinants, is ignored. Observations with missing values are removed.

For a given z, the local-linear estimator following Geng and Sun (2019) is

$$\hat{\gamma}(z) = \Big(\sum_i S_i S_i' K_i(z) \Big)^{-1} \sum_i S_i y_i K_i(z),$$

where $S_i = (x_i', (x_i \otimes (z_i - z))')'$ involves the continuous variables in z only and $K_i(z)$ is a generalized product kernel (Second-order Gaussian for continuous, Aitchison-Aitken for unordered and Li-Racine for ordered variables, the defaults of `np::npksum`). The first elements of $\hat{\gamma}(z)$ estimate the intercept and slope smooth coefficients and the remaining ones their gradients. Variables of the second part of `formula` are classified by their class: numeric variables are continuous, factors are unordered categorical and ordered factors are ordered categorical.

Formula and bandwidths `bws` is the first argument, and what it is decides which of the other arguments are needed. The formula is given once, to `llscoefbw`:

- `bws` is the object returned by `llscoefbw` with `LSCV = TRUE`: `formula` must not be given, and `data` is not needed. The object keeps the formula and the call to `llscoefbw`, whose `data` are evaluated again in the environment the call was made from. `data` replaces the training data, `newdata` gives the evaluation points.
- `bws` is a formula: the bandwidths are selected first, by `llscoefbw(bws, data = data)`, and used in the same call.
- `bws` is a numeric vector of local-linear bandwidths, such as the result of `llscoefbw` with `LSCV = FALSE`: `formula` is required, and `data` unless the variables are in the environment of `formula`. A bandwidth object of `np::npscoefbw` is accepted too, and its bandwidths are used, if it was made with `regtype = "ll"`, fixed bandwidths (`bwtype = "fixed"`, `bwscaling = FALSE`) and the default kernels (`ckertype`, `ckerorder`, `ckerbound`, `ukertype` and `okertype` arguments). However, one made with `regtype = "lc"`, the default `regtype` argument of `np::npscoefbw`, has bandwidths for the local-constant estimator, and gives an error like the other objects that do not qualify. A user-supplied vector that does not qualify cannot be told apart from other vectors, but such a vector is not suitable, either.

A bandwidth object returned by `llscoefbw` cannot be used together with a `formula`. This is because an object of class `"llsmoothcoefficientbw"` stores the formula and data used to select the bandwidths. Therefore, formula is not allowed to appear again to guarantee internal consistency.

Evaluation points The estimation always uses the training data: those of the call to `llscoefbw` for a bandwidth object, otherwise those given by `formula` and `data`. `newdata` only decides where the smooth coefficients are evaluated. It needs the variables of the `x` and `z` parts of `formula` (the response is not needed and is ignored). The `x` variables matter for the fitted values only, and any values, such as the training means, can be used when only the smooth coefficients are of interest. Factors in `newdata` are matched to the levels of the training data, and rows of `newdata` with missing values are dropped.

Residuals The residuals are those of the training data, that is, y minus the fitted values at the training points, also when `newdata` is supplied.

Gradients The gradients of the smooth coefficients with respect to the continuous variables in z, one column for each coefficient and variable, in X-major order: all the gradients of the first coefficient (the intercept) come first, then those of the second one, and so on, each in the order of the continuous z variables.

Value

`llscoef` returns an object of class `"llsmoothcoefficient"`, a list with components

`mean`	The fitted values at the evaluation points.
`beta`	If `betas = TRUE`, the matrix of smooth coefficients, with one row for each evaluation point and the columns `"Intercept"` and the x variables.
`resid`	If `residuals = TRUE`, the residuals, y minus the fitted values at the training points.
`grad`	If `gradients = TRUE`, the matrix of gradients of the smooth coefficients with respect to the continuous variables in z, in X-major order, with one row for each evaluation point.

Together with `bw`, `x_names`, `z_names`, `n_train`, `n_eval`, `eval_train` (evaluation and training data are the same), `omit` (the rows with missing values that were dropped from the evaluation data: from `newdata` if it is given, and otherwise from the training data) and `call`. With `na.action = na.exclude`, `mean`, `beta` and `grad` are padded with `NA` at the rows of the evaluation data that were dropped (those in `omit`), and `resid` at the dropped rows of the training data, so that they line up with `newdata` or with the data. `n_train` and `n_eval` still count the rows that were used.

Note

For an object of class `"llsmoothcoefficient"`, the smooth coefficients and residuals can be extracted with `coef` and `residuals`, respectively, and the fitted values with `fitted`.

References

Xin Geng and Kai Sun (2019). Gradient estimation of the local-constant semiparametric smooth coefficient model. *Economics Letters*, 185, 108684.

Tristen Hayfield and Jeffrey S. Racine (2008). Nonparametric econometrics: The np package. *Journal of Statistical Software*, 27(5), 1–32.

Qi Li, Cliff J. Huang, Dong Li and Tsu-Tan Fu (2002). Semiparametric smooth coefficient models. *Journal of Business & Economic Statistics*, 20(3), 412–422.

See Also

`llscoefbw`, `lcscoefgrad`, `spscsfa`

Examples

```
set.seed(1)
n <- 200
x <- rnorm(n)
z1 <- runif(n)
```

```
z2 <- runif(n)
z3 <- factor(sample(c("A", "B", "C"), n, replace = TRUE))
y <- cos(z1 + z2) + sin(z1 - z2) * x + rnorm(n, 0, 0.1)
dat <- data.frame(y, x, z1, z2, z3)

bw <- llscoefbw(y ~ x | z1 + z2 + z3, data = dat)

# estimation at the training data: the formula and the data
# are those of llscoefbw()
fit <- llscoef(bw, betas = TRUE, residuals = TRUE, gradients = TRUE)
fit
head(coef(fit))
head(residuals(fit))
head(fit$grad)

# smooth coefficients on a grid of z1, with z2 and z3 fixed
grid <- data.frame(x = mean(x), z1 = seq(0, 1, length.out = 50), z2 = 0.5,
                   z3 = factor("A", levels = levels(dat$z3)))
fit.grid <- llscoef(bw, newdata = grid, betas = TRUE)
plot(grid$z1, coef(fit.grid)[, "x"], xlab=expression(z[1]), ylab=expression(beta(z[1])),
      type = "l")

# bandwidths selected inside llscoef(), from a formula
fit.f <- llscoef(y ~ x | z1 + z2 + z3, data = dat, betas = TRUE)

# given a bandwidth vector that is not an object of class
# "llsmoothcoefficientbw": the formula is needed
fit.b <- llscoef(bw$bw, formula = y ~ x | z1 + z2 + z3, data = dat, betas = TRUE)
```

`llscoefbw` *Least-squares cross-validation bandwidths for the local-linear SPSC estimator*

Description

Selects the bandwidths of the local-linear semiparametric smooth-coefficient (SPSC) estimator by least-squares cross-validation (LSCV). The result can be passed to `llscoef` through its `bws` argument or to `spscsfa` through its `bandwidth` argument.

Usage

```
llscoefbw(formula, bws, data, LSCV, na.action, max.search, grid_scan)

## Default S3 method:
llscoefbw(
  formula,
  bws = NULL,
  data = NULL,
```

```
  LSCV = TRUE,
  na.action = stats::na.omit,
  max.search = 2L,
  grid_scan = 2^c(-2:5)
)

## S3 method for class 'formula'
llscoefbw(
  formula,
  bws = NULL,
  data = NULL,
  LSCV = TRUE,
  na.action = stats::na.omit,
  max.search = 2L,
  grid_scan = 2^c(-2:5)
)

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

Arguments

formula	A formula y ~ x | z (or y ~ x | z | q), where y is the dependent variable, x are the parametric regressors (the constant is added automatically) and z are the variables that the smooth coefficients depend on.
bws	Optional numeric vector of local-linear bandwidths, one for each variable in the z part of formula, in that order, or an object returned by llscoefbw. An object returned by np::npscoefbw made with regtype = "ll", fixed bandwidths and the default kernels is also accepted, and its bandwidths are used; one made with another regtype, bwtype, bwscaling or kernel is not, and a vector taken from it is not suitable, either. With LSCV = FALSE these bandwidths are returned without a search. Otherwise they replace the rule-of-thumb starting values.
data	Optional data frame containing the variables in formula. If omitted, the variables are taken from the environment of formula.
LSCV	Logical; if TRUE (default) the bandwidths are selected by LSCV. If FALSE, bws is checked, named and returned.
na.action	A function, or the name of a function, that decides what to do with rows that have missing values in any part of formula, as in model.frame. The default, na.omit, drops them. na.exclude also drops them, but llscoef then pads its results with NA at those rows, so that they line up with the data. na.fail gives an error. na.pass is not allowed: an error is given if missing values remain. It is kept in the returned object, and llscoef uses it.
max.search	Maximum number of Nelder-Mead searches, a positive integer (default 2): the one from the starting values and up to max.search - 1 more, from the most promising points of a coarse scan of the bandwidths (see 'Details'). With max.search = 1 there is no scan and a single search is made from the starting values. It is ignored with LSCV = FALSE.

`grid_scan` A strictly increasing numeric vector of finite positive multipliers for the coarse scan (default `2^c(-2:5)`). Each multiplier is applied to all continuous starting bandwidths, while categorical bandwidths are kept at their starting values. It affects the search only when `LSCV = TRUE`, `max.search > 1`, and at least one continuous variable is present in the `z` part of `formula`.

`x` An object of class `"llsmoothcoefficientbw"`.

`...` Further arguments, ignored.

Details

The model is $y_i = x_i'\beta(z_i) + u_i$, where x_i consists of a constant followed by the variables in the first part of `formula` and z_i holds the variables in the second part. A third part, as accepted by `spscsfa` for the inefficiency determinants, is ignored. Observations with missing values are removed.

For a given z, the local-linear estimator following Geng and Sun (2019) is

$$\hat{\gamma}(z) = \Big(\sum_i S_i S_i' K_i(z) \Big)^{-1} \sum_i S_i y_i K_i(z),$$

where $S_i = (x_i', (x_i \otimes (z_i - z))')'$ involves the continuous variables in z only and $K_i(z)$ is a generalized product kernel (Second-order Gaussian for continuous, Aitchison-Aitken for unordered and Li-Racine for ordered variables, the defaults of `np::npksum`). The first elements of $\hat{\gamma}(z)$ estimate the smooth coefficients. The LSCV criterion is the mean of the squared leave-one-out residuals.

Variables of the second part of `formula` are classified by their class: numeric variables are continuous, factors are unordered categorical and ordered factors are ordered categorical.

Starting values Rule-of-thumb bandwidths: $1.059224\, sd(z)\, n^{-1/(4+p)}$ for a continuous variable and $1.059224\, n^{-2/(4+p)}$ for a categorical variable, where n is the number of observations and p the number of continuous variables in z. A categorical starting value outside the admissible range (see below) is replaced by half of the upper limit.

Admissible bandwidths The criterion takes a very large value for inadmissible bandwidths. Continuous bandwidths must be positive, unordered categorical bandwidths must lie in $[0, (c-1)/c]$ with c the number of levels, and ordered categorical bandwidths in $[0, 1]$. Singular local systems are ridged.

Search Nelder-Mead, that is `optim` with `method = "Nelder-Mead"` and its default settings, started from the starting values.

Scan and further searches The LSCV criterion is not a smooth function of the bandwidths: with few observations it can have local minima, for example at small bandwidths, where a Nelder-Mead search that starts nearby stops. Unless `max.search = 1`, the criterion is therefore also evaluated on a coarse scan: the starting values with every continuous bandwidth multiplied by each value in `grid_scan` (categorical bandwidths are kept at their starting values). By default, `grid_scan = 2^c(-2:5)`, which takes 8 evaluations. The scan moves all the continuous bandwidths together, and is skipped if no variable in the `z` part is continuous.

Progress Progress reporting during bandwidth selection is provided by the **progress** package.

Value

With LSCV = FALSE, a numeric vector of bandwidths, one for each variable in the z part of formula, in that order and named after those variables.

With LSCV = TRUE, an object of class "llsmoothcoefficientbw": a list with components

bw the bandwidths, a numeric vector named after the variables in the z part of formula.

formula the formula that was supplied.

n_obs the number of observations used.

n_dim the number of variables in the z part.

lscv the value of the LSCV criterion at bw.

n_feval the number of evaluations of the criterion, those of the scan and of every search made.

scan_factor the multiplier used to initialize the Nelder-Mead search that produced the selected bandwidths. It is 1 when the search from the original starting values gives the best result; otherwise it is the corresponding value in grid_scan.

n_conti, n_unord, n_ord the numbers of continuous, unordered categorical and ordered categorical variables in the z part.

call the call to llscoefbw, with the environment it was made from.

obs_omit, n_obs_omit the rows of the training data that were dropped because of missing values in any part of formula, and their number. The row numbers refer to data, or to the variables in the environment of formula.

na_action the na.action function used.

References

Xin Geng and Kai Sun (2019). Gradient estimation of the local-constant semiparametric smooth coefficient model. *Economics Letters*, 185, 108684.

Tristen Hayfield and Jeffrey S. Racine (2008). Nonparametric econometrics: The np package. *Journal of Statistical Software*, 27(5), 1–32.

Qi Li, Cliff J. Huang, Dong Li and Tsu-Tan Fu (2002). Semiparametric smooth coefficient models. *Journal of Business & Economic Statistics*, 20(3), 412–422.

See Also

llscoef, spscsfa

Examples

```
## Not run:
set.seed(1)
n <- 200
x <- rnorm(n)
z1 <- runif(n)
z2 <- runif(n)
z3 <- factor(sample(c("A", "B", "C"), n, replace = TRUE))
y <- cos(z1 + z2) + sin(z1 - z2) * x + rnorm(n, 0, 0.1)
dat <- data.frame(y, x, z1, z2, z3)
```

```
bw <- llscoefbw(y ~ x | z1 + z2 + z3, data = dat)
bw
bw$bw

# the formula and the data are those of llscoefbw()
fit <- llscoef(bw, betas = TRUE)

# a single search from the starting values, without the scan
llscoefbw(y ~ x | z1 + z2 + z3, data = dat, max.search = 1)

# check and return given bandwidths without a search
llscoefbw(y ~ x | z1 + z2 + z3, bws = bw$bw, data = dat, LSCV = FALSE)

## End(Not run)
```

`plot.spscsfa` *Plot smooth coefficients of an SPSC-SFA model*

Description

Plot method for objects of class `"spscsfa"`. Each estimated smooth coefficient, i.e. the intercept $\theta(Z)$ and the slope coefficients $\beta(Z)$, is plotted against one continuous nonparametric variable Z at a time, over a grid running from `min(Z)` to `max(Z)`. The smooth coefficients are those of the estimator used to fit the model: local-constant if `spscreg = "lc"` and local-linear if `spscreg = "ll"` in `spscsfa`.

Usage

```
## S3 method for class 'spscsfa'
plot(
  x,
  which = NULL,
  zvar = NULL,
  n_grid = 100L,
  rug = TRUE,
  mfrow = NULL,
  ask = NULL,
  ...
)
```

Arguments

`x`	An object of class `"spscsfa"` returned by `spscsfa`.
`which`	Smooth coefficients to plot, given as positions or names of the columns of `smoothcoef(x)$theta_beta`. The default (`NULL`) plots all of them: the intercept followed by one slope coefficient for each variable in the parametric part of the formula.

`zvar`	Continuous Z variables to plot against, given as names or as positions in the nonparametric part of the formula. The default (`NULL`) uses all continuous Z variables.
`n_grid`	Number of equally spaced grid points between `min(Z)` and `max(Z)` (default 100).
`rug`	Logical; if `TRUE` (default), the observed values of Z are marked by a rug along the horizontal axis.
`mfrow`	Integer vector of length 2 giving the number of rows and columns of panels per page. The default (`NULL`) arranges all smooth coefficients against one Z variable on a single, roughly square page. The graphical parameters are restored on exit.
`ask`	Logical; if `TRUE`, the user is asked before each new page is drawn. The default asks only if more than one page is drawn on an interactive device.
`...`	Further graphical parameters passed to `plot`, for example `col`, `lty`, `lwd`, `ylim` or `main`. By default lines are drawn (`type = "l"`) and the axes are labelled with the name of the Z variable and of the smooth coefficient.

Details

For every continuous variable in the nonparametric part Z of the model (the second part of the formula), the smooth coefficients are evaluated on `n_grid` equally spaced points between the sample minimum and maximum of that variable. The grid is supplied to `spscsfa` through its `ezdat` argument, re-using the bandwidth of the fitted model, so no new cross-validation is carried out. The same estimator (`spscreg`) as in the fitted model is used for the grid: the local-constant estimator for `spscreg = "lc"` and the local-linear estimator for `spscreg = "ll"`.

If the model has more than one Z variable, the other continuous Z variables are held fixed at their sample averages while the variable on the horizontal axis runs over its grid. One page is drawn for each continuous Z variable, with one panel for each smooth coefficient.

Discrete Z variables (factors and ordered factors) are ignored by the plot method: no plots are drawn against them. The smooth coefficients still depend on them, however, so they have to take some value when the coefficients are evaluated; each discrete Z variable is held at its most frequent category.

A term in the second part of the formula is a variable as it enters the model: for `log(z)` the grid runs over the values of `log(z)`, which also labels the axis, and `factor(z2)` is a discrete variable.

Only the smooth coefficients of the first estimation step, i.e. the columns of `smoothcoef(x)$theta_beta`, are plotted. In particular, the intercept is $\theta(Z)$ (`Intercept (theta)`) and not the recovered intercept $\alpha(Z)$.

Value

Invisibly, a named list with one data frame for each Z variable plotted against. The first column of each data frame is the grid of that Z variable and the remaining columns are the smooth coefficients evaluated on the grid, so that the curves can be redrawn or customized.

See Also

`spscsfa`, `smoothcoef`

Examples

```
set.seed(1)
n <- 100
x <- rnorm(n)
z <- runif(n)
y <- cos(z) + sin(z) * x + rnorm(n, 0, 0.1) -
     exp(0.1 + 0.1 * z) * abs(rnorm(n, 0, 1))
dat <- data.frame(output = y, input = x, covariate = z)
result <- spscsfa(output ~ input | covariate, data = dat, verbose = FALSE)

# the intercept and the slope coefficient plotted against 'covariate'
plot(result)

# the plotted values are returned invisibly
curves <- plot(result, n_grid = 50, col = "blue")
head(curves$covariate)

# Two continuous z variables and one discrete z variable: one page per
# continuous z variable; the other continuous z variable is held at its
# average and the factor at its most frequent category
z1 <- runif(n)
z2 <- runif(n)
z3 <- factor(rep(c("A", "B"), length.out = n))
y <- cos(z1 + z2) + sin(z1 - z2) * x + rnorm(n, 0, 0.1) -
     exp(0.1 + 0.1 * z1) * abs(rnorm(n, 0, 1))
dat <- data.frame(y, x, z1, z2, z3)
result <- spscsfa(y ~ x | z1 + z2 + z3, data = dat, verbose = FALSE)
plot(result)

# the slope coefficient only, against z2 only
plot(result, which = 2, zvar = "z2")

# local-linear smooth coefficients (spscreg = "ll"), plotted the same way
result_ll <- spscsfa(y ~ x | z1 + z2 + z3, data = dat, spscreg = "ll", verbose = FALSE)
plot(result_ll)
```

`print.spscsfa` *Print SPSC-SFA results*

Description

`print` method for objects of class `"spscsfa"`. It formats the fit through `summary.spscsfa`, so the display is identical to that of `summary(x)` and responds to the same standard arguments of `print`. As required for `print` methods, the object itself is returned invisibly.

Usage

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

Arguments

x	An object of class "spscsfa", as returned by spscsfa.
digits	Integer; the minimum number of significant digits to use.
...	Further arguments passed to print.summary.spscsfa, and from there to printCoefmat and print.default (for example signif.stars, na.print or right).

Value

The original object of class "spscsfa", returned invisibly. It is a list containing the components smooth_coefficients, mle_results, technical_efficiency, inefficiency, model_summary, formula_parsed and data_info. These components contain the estimated smooth coefficients and recovered intercept, maximum likelihood results, technical efficiency estimates and confidence intervals, JLMS inefficiency estimates, model specification information, parsed formula components, and data cleaning information. If the model was estimated with gradients = TRUE, smooth_coefficients_grad and inefficiency_grad are also returned.

See Also

summary.spscsfa, print.summary.spscsfa, spscsfa

---

print.summary.spscsfa *Print an SPSC-SFA summary*

Description

print method for objects of class "summary.spscsfa" created by summary.spscsfa. Numeric output is controlled by the standard arguments of print: digits and signif.stars are handled explicitly, and any further arguments in ... (for example na.print, print.gap, quote or right) are passed down to printCoefmat and print.default.

Usage

```
## S3 method for class 'summary.spscsfa'
print(
  x,
  digits = max(3L, getOption("digits") - 3L),
  signif.stars = getOption("show.signif.stars"),
  signif.legend = signif.stars,
  ...
)
```

Arguments

x	An object of class "summary.spscsfa".
digits	Integer; the minimum number of significant digits. Defaults to the value stored by summary.spscsfa.

`signif.stars`	Logical; if `TRUE`, significance stars are printed next to the p-values of the inefficiency determinants. Defaults to `getOption("show.signif.stars")`.
`signif.legend`	Logical; if `TRUE`, the legend for the significance stars is printed. Defaults to `signif.stars`.
`...`	Further arguments passed to `printCoefmat` and `print.default`.

Value

The argument `x`, invisibly.

See Also

`summary.spscsfa`, `print.spscsfa`

`ricephil` *Philippine Rice Farm Data*

Description

Data used to illustrate the application of the semiparametric smooth-coefficient stochastic frontier model.

Usage

```
data(ricephil)
```

Format

A data frame with 344 observations and 17 variables.

References

Timothy J. Coelli, D.S. Prasada Rao, Christopher J. O'Donnell, and George E. Battese (2005). *An Introduction to Efficiency and Productivity Analysis*. Springer, New York.

S. Pandey, P. Masciat, L. Velasco, and R. Villano (1999). Risk analysis of a rainfed rice production system in Tarlac, Central Luzon, Philippines. *Experimental Agriculture*, 35(2), 225–237.

smoothcoef *Extract smooth coefficients*

Description

Extracts the smooth coefficients of a fitted model. `smooth_coef` is an alias of `smoothcoef`.

Usage

```
smoothcoef(x)
```

Arguments

`x`	An object of class 'spscsfa'.

Value

A list with two components:

`theta_beta`	The matrix of the smooth coefficients of the first step, $\theta(z)$ and $\beta(z)$: one row for each evaluation point (those of `ezdat` if it was supplied to `spscsfa`, the training data otherwise), and the columns `Intercept (theta)` and the names of the `x` variables. They are local-constant or local-linear estimates, as chosen by `spscreg`.
`alpha`	A matrix with one column: the estimates of the intercept $\alpha(z) = \theta(z) + E(u\|z)$ at the training data.

spscsfa *Semiparametric Smooth-Coefficient Stochastic Frontier Analysis*

Description

Estimates a semiparametric smooth-coefficient stochastic frontier model $y_i = \alpha(Z_i) + X_i'\beta(Z_i) + v_i - u(Z_i)$ following Sun and Kumbhakar (2013). The model allows the coefficients of the parametric part to vary smoothly with a set of nonparametric variables. Technical efficiency and inefficiency are computed using the Battese and Coelli (1988) (BC) and Jondrow et al. (1982) (JLMS) methods, respectively. Confidence intervals for technical efficiency are computed using the approach of Horrace and Schmidt (1996). Marginal effects of the inefficiency determinants on the JLMS inefficiency are computed following Kumbhakar and Sun (2013).

Usage

```
spscsfa(
  formula,
  data = NULL,
  ezdat = NULL,
  spscreg = c("lc", "ll"),
  bandwidth = NULL,
  gradients = FALSE,
  verbose = TRUE,
  sig_level = 0.05,
  optim_method = "optim_BFGS",
  optim_control = list(maxit = 5000),
  stval = "automatic",
  penalty = 1e+08
)
```

Arguments

formula	A formula of the form y ~ x | z or y ~ x | z | q, where y is the dependent variable, x are parametric inputs, z are nonparametric variables determining smooth coefficients, and q (optional) are inefficiency determinants. When the third formula part q is omitted, the z variables also serve as inefficiency determinants.
data	Optional data frame containing all variables used in formula. If supplied, variables are extracted from this data frame. If not supplied, variables are taken from the environment of formula.
ezdat	Optional data frame containing evaluation points for the nonparametric variables z. If provided, smooth coefficients are estimated at these points; otherwise, the training data points (from z) are used.
spscreg	Character string; the estimator of the smooth coefficients in the first step. "lc" (default) is the local-constant estimator (np::npscoefbw and np::npscoef both being made with default regtype = "lc" and default kernels, along with their default bwtype = "fixed" and bwscaling = FALSE) and "ll" is the local-linear estimator (llscoefbw and llscoef).
bandwidth	Optional bandwidths for the smooth-coefficient estimation. If NULL (default and recommended), bandwidths are selected by least-squares cross-validation (LSCV): for the local-constant estimator if spscreg = "lc", and for the local-linear estimator if spscreg = "ll". Or a numeric vector with one bandwidth for each variable in z, in the order of the formula, of the kind that suits spscreg (local-constant bandwidths for "lc", local-linear ones for "ll"). A named vector, such as the one returned by llscoefbw with LSCV = FALSE, is matched to the z variables by name if its names are those of the z variables, and is taken in the order of the formula otherwise. Or a bandwidth object: for spscreg = "lc" from np::npscoefbw with regtype = "lc" (its default), and for spscreg = "ll" from np::npscoefbw with regtype = "ll", with both using fixed bandwidths (bwtype = "fixed", bwscaling = FALSE) and the default kernels; or from llscoefbw. An object of the other kind is an error, as the bandwidths of the two

estimators are not interchangeable; a vector of the other kind cannot be told apart and is not suitable.

`gradients` Logical; if `TRUE`, the gradients of the smooth coefficients with respect to the continuous `z` variables (local-constant or local-linear, as chosen by `spscreg`) and the gradients of the JLMS inefficiency estimates are computed and returned. If `FALSE` (default), no gradient is computed or returned.

`verbose` Logical; if `TRUE`, progress and intermediate results are printed.

`sig_level` Significance level for confidence intervals of technical efficiency (default 0.05).

`optim_method` Character string specifying the optimizer for MLE in the second step: `"optim"` (uses Nelder-Mead), `"optim_BFGS"` (uses BFGS, the default), or `"nlm"`.

`optim_control` List of control parameters passed to the optimizer for MLE in the second step.

`stval` Starting values for the MLE in the second step. Use `"automatic"` (default) for internally computed starting values, or supply a numeric vector in which the order of its elements is δ_0, δ_1 (if any), and σ_v^2. See Details.

`penalty` To ensure positivity of estimated σ_v^2, penalty applied when σ_v^2 becomes negative during a numerical search (default 1e8).

Details

The stochastic production frontier model is specified as

$$y_i = \alpha(Z_i) + X_i'\beta(Z_i) + v_i - u(Z_i),$$

where y_i can be viewed as the log of output, X_i can be viewed as a vector of the log of inputs, and Z_i is a vector of covariates. $\alpha(\cdot)$ is the intercept, and $\beta(\cdot)$ is the slope vector. Both $\alpha(\cdot)$ and $\beta(\cdot)$ are unknown functions of Z_i. The noise term $v_i \sim iidN(0, \sigma_v^2)$. The inefficiency term is $u(Z_i) = \sigma_u(Z_i)\eta_i$, where $\eta_i \sim iidN^+(0,1)$. The random variables η_i and v_i are assumed to be independent of each other and independent of X and Z.

To guarantee positivity of $\sigma_u(Z_i)$,

$$\sigma_u(Z_i) = \exp(\delta_0 + \delta_1' Z_i).$$

These imply that

$$E(u_i \mid Z_i) = \sigma_u(Z_i)E(\eta_i \mid Z_i) = \sqrt{2/\pi}\,\sigma_u(Z_i) = \sqrt{2/\pi}\,\exp(\delta_0 + \delta_1' Z_i)$$

and $\sigma_u^2(Z_i) = \exp[2(\delta_0 + \delta_1' Z_i)]$.

The estimation involves two steps. In the first step, estimate $\theta(Z_i)$ (`Intercept (theta)`), $\beta(Z_i)$, and ε_i from

$$y_i = \theta(Z_i) + X_i'\beta(Z_i) + \varepsilon_i,$$

where $\theta(Z_i) = \alpha(Z_i) - E(u_i \mid Z_i)$ and $\varepsilon_i = v_i - [u(Z_i) - E(u_i \mid Z_i)] = E(u_i \mid Z_i) + v_i - u(Z_i)$.

Then in the second step, use the estimated ε_i to estimate δ_0, δ_1 and σ_v^2 from

$$\varepsilon_i = \sqrt{2/\pi}\,\exp(\delta_0 + \delta_1' Z_i) + v_i - \exp(\delta_0 + \delta_1' Z_i)\eta_i.$$

Finally, compute $E(u_i \mid Z_i) = \sqrt{2/\pi}\,\exp(\delta_0 + \delta_1' Z_i)$ from estimated δ_0 and δ_1. The intercept in the original stochastic production frontier $\alpha(Z_i)$ (`Intercept (alpha)`) is recovered as

$$\alpha(Z_i) = \theta(Z_i) + E(u_i \mid Z_i).$$

See Sun and Kumbhakar (2013, page 306) for more details.

The smooth coefficients of the first step can be estimated by either a local-constant (LC) or a local-linear (LL) kernel estimator, as chosen by `spscreg`. With `spscreg = "lc"` (default), the bandwidths are selected by `np::npscoefbw` and the smooth coefficients are estimated by `np::npscoef` with default arguments. With `spscreg = "ll"`, the bandwidths are selected by `llscoefbw` and the smooth coefficients are estimated by `llscoef`. If `bandwidth` is `NULL` (default), the bandwidths are selected by least-squares cross-validation (LSCV) for the chosen estimator. The residuals of the chosen estimator are used in the second step, the maximum likelihood estimation.

If `gradients = TRUE`, the gradients of the first-step smooth coefficients with respect to the continuous variables in `z` are estimated as well: by `lcscoefgrad` for the local-constant estimator and by `llscoef` for the local-linear estimator. Since $\alpha(Z_i) = \theta(Z_i) + E(u_i \mid Z_i)$, the gradient of $\alpha(Z_i)$ with respect to the lth continuous Z variable is

$$\partial\alpha(Z_i)/\partial z_l = \partial\theta(Z_i)/\partial z_l + \sqrt{2/\pi}\,\exp(\delta_0 + \delta_1' Z_i)\,\delta_{1l},$$

where δ_{1l} is the coefficient of that variable in the inefficiency equation (zero if the variable is not an inefficiency determinant) and $\partial\theta(Z_i)/\partial z_l$ is evaluated at the training data.

The analytic gradients of the local-constant smooth coefficient estimator with respect to the continuous `z` variables, returned in `smooth_coefficients_grad` if `gradients = TRUE` and `spscreg = "lc"`, are computed following Geng and Sun (2019); see `lcscoefgrad`. The marginal effects of the inefficiency determinants on the JLMS inefficiency estimates, returned as `inefficiency_grad` if `gradients = TRUE`, are computed following Kumbhakar and Sun (2013).

The JLMS inefficiency estimator is the conditional mean $E(u_i \mid \varepsilon_i) = \tilde{\mu}_i + \sigma_{*i}\, g_i$, where $\tilde{\mu}_i = -\varepsilon_i\sigma_{ui}^2/\sigma_i^2$, $\sigma_{*i} = \sigma_{ui}\sigma_v/\sigma_i$, $\sigma_i^2 = \sigma_{ui}^2 + \sigma_v^2$, $m_i = \tilde{\mu}_i/\sigma_{*i}$ and $g_i = \phi(m_i)/\Phi(m_i)$. Since $\sigma_{ui} = \exp(\delta_0 + \delta_1' Z_i)$ depends on the inefficiency determinants, the first-order analytic derivative of the JLMS estimator with respect to the l-th inefficiency determinant (holding ε_i fixed) is

$$\frac{\partial E(u_i \mid \varepsilon_i)}{\partial z_{li}} = \frac{\delta_{1l}}{\sigma_i^2}\Big[\sigma_v^2\sigma_{*i}\big(g_i(1+m_i^2) + m_i g_i^2\big) - 2\sigma_{*i}^2\varepsilon_i(1 - g_i^2 - m_i g_i)\Big].$$

This is the σ_{ui} channel of Equation (9) in Kumbhakar and Sun (2013), where the coefficient on z_{li} in σ_{ui} was called γ_l (it is δ_{1l} here, as in Sun and Kumbhakar, 2013); the other two channels vanish because $\mu_i = 0$ and σ_v is constant. If `gradients = TRUE`, the derivatives are returned in `inefficiency_grad`. A column of `inefficiency_grad` that belongs to a dummy variable created from a categorical inefficiency determinant is the analytic partial derivative with the dummy treated as continuous.

Value

A list of class `"spscsfa"` containing:

`smooth_coefficients`
: Smooth coefficient estimates (local-constant if `spscreg = "lc"`, local-linear if `spscreg = "ll"`), residuals, bandwidth, and metadata.

`smooth_coefficients_grad`
: If `gradients = TRUE`. A list whose first element, `theta_beta_grad`, is itself a list with one matrix for each continuous `z` variable (named after it): the gradients of $\theta(z)$ and $\beta(z)$ with respect to that variable, at the evaluation points, with columns `d(Intercept (theta))/d(z)`, `d(x1)/d(z)`, ... Like `x1` used in

a regression table, the `x1` here refers to the smooth coefficient of `x1`. Its second element, `alpha_grad`, is a matrix with one column for each continuous `z` variable, named `d(Intercept (alpha))/d(z)`: the gradients of $\alpha(Z_i)$ at the training data.

`mle_results` MLE parameter estimates, standard errors, and log-likelihood.

`technical_efficiency`
Technical efficiency scores (BC method) and confidence intervals.

`inefficiency` Inefficiency estimates (JLMS method).

`inefficiency_grad`
If `gradients = TRUE`. A matrix of the first-order analytic derivatives of the JLMS inefficiency estimates with respect to the inefficiency determinants (Kumbhakar and Sun, 2013). It has one row per training observation and one column per inefficiency determinant used in the maximum likelihood step, named after the columns of that matrix (categorical variables appear as their binary variables). It is `NULL` when there are no inefficiency determinants (intercept only).

`model_summary` Summary information about the model specification.

`formula_parsed` Parsed formula components.

`data_info` Information about data cleaning and observations used.

`model_frame` The estimation sample: the model frame, without missing values, of the variables used in `formula`. It is kept so that `plot.spscsfa` can re-evaluate the smooth coefficients on a grid of points.

Note

Results can be summarized using `summary.spscsfa`. Extraction functions for smooth coefficients, technical efficiency and inefficiency are available through `smoothcoef`, `te`, and `inefficiency`, respectively. If `gradients = TRUE`, the gradients are extracted with `grad`. The smooth coefficients can be plotted against the continuous nonparametric variables using `plot.spscsfa`.

`spsc_sfa` is an alias of `spscsfa`, `smooth_coef` is an alias of `smoothcoef`, and the methods of the class `"spscsfa"` also handle fitted models and summaries of the class `"spsc_sfa"`.

Author(s)

Kai Sun <ksun1@shu.edu.cn>

References

George E. Battese and Tim J. Coelli (1988). Prediction of firm-level technical efficiencies with a generalized frontier production function and panel data. *Journal of Econometrics*, 38(3), 387–399.

Xin Geng and Kai Sun (2019). Gradient estimation of the local-constant semiparametric smooth coefficient model. *Economics Letters*, 185, 108684.

William C. Horrace and Peter Schmidt (1996). Confidence statements for efficiency estimates from stochastic frontier models. *Journal of Productivity Analysis*, 7(2), 257–282.

James Jondrow, C.A. Knox Lovell, Ivan S. Materov, and Peter Schmidt (1982). On the estimation of technical inefficiency in the stochastic frontier production function model. *Journal of Econometrics*, 19(2–3), 233–238.

Subal C. Kumbhakar and Kai Sun (2013). Derivation of marginal effects of determinants of technical inefficiency. *Economics Letters*, 120(2), 249–253.

Kai Sun and Subal C. Kumbhakar (2013). Semiparametric smooth-coefficient stochastic frontier model. *Economics Letters*, 120(2), 305–309.

Examples

```
set.seed(1)
n <- 100
x <- rnorm(n)
z <- runif(n)
ez <- seq(0, 1, length = 1000)
q <- rnorm(n)
y <- cos(z) + sin(z) * x + rnorm(n, 0, 0.1) -
     exp(0.1 + 0.1 * z) * abs(rnorm(n, 0, 1))
dat <- data.frame(output = y, input = x, covariate = z, ineffvar = q)

# covariate enters the nonparametric part and is also treated as inefficiency determinant
result <- spscsfa(output ~ input | covariate, data = dat,
                  optim_method = "optim_BFGS", verbose = TRUE)
result$smooth_coefficients
result$mle_results
result$technical_efficiency
result$inefficiency

# This gives the same results: the inefficiency variables default to the nonparametric part
result <- spscsfa(output ~ input | covariate | NULL, data = dat,
                  optim_method = "optim_BFGS", verbose = TRUE)

# Use evaluation data (ezdat) for estimating the smooth coefficients
result <- spscsfa(output ~ input | covariate, data = dat,
                  ezdat = data.frame(covariate = ez),
                  optim_method = "optim_BFGS", verbose = TRUE)
nrow(result$smooth_coefficients$theta_beta)

# Gradients of the smooth coefficients, of alpha, and of the inefficiency
result <- spscsfa(output ~ input | covariate, data = dat, gradients = TRUE, verbose = TRUE)
result$smooth_coefficients_grad$theta_beta_grad
head(result$smooth_coefficients_grad$alpha_grad)
head(result$inefficiency_grad)
grad(result, zvar = "covariate")

# Three-part formula: nonparametric part differs from inefficiency determinants
result <- spscsfa(output ~ input | covariate | ineffvar, data = dat,
                  optim_method = "optim_BFGS", verbose = TRUE)

# Customized bandwidth for the nonparametric part
result <- spscsfa(output ~ input | covariate | ineffvar, data = dat,
                  optim_method = "optim_BFGS", bandwidth = c(0.1), verbose = TRUE)

# Restrict no inefficiency determinants (intercept only)
result <- spscsfa(output ~ input | covariate | 1, data = dat,
```

```
                    optim_method = "optim_BFGS", verbose = TRUE)

# Local-linear smooth coefficients (the default is spscreg = "lc")
result <- spscsfa(output ~ input | covariate, data = dat, spscreg = "ll", verbose = TRUE)
result$smooth_coefficients$theta_beta

# Gradients with local-linear smooth coefficients
result <- spscsfa(output ~ input | covariate, data = dat,
                    spscreg = "ll", gradients = TRUE, verbose = TRUE)
result

# User-provided bandwidth for the local-linear estimator: llscoefbw() with
# LSCV = FALSE returns a named vector
bw <- llscoefbw(output ~ input | covariate, bws = 0.2, data = dat, LSCV = FALSE)
result <- spscsfa(output ~ input | covariate, data = dat,
                    spscreg = "ll", bandwidth = bw, verbose = TRUE)

# More variables
set.seed(1)
n <- 100
x1 <- rnorm(n)
x2 <- rnorm(n)
z1 <- runif(n)
z2 <- runif(n)
y <- cos(z1 + z2) + sin(z1 - z2) * x1 + sin(z1 + z2) * x2 +
     rnorm(n, 0, 0.1) - exp(0.1 + 0.1 * z1 + 0.1 * z2) * abs(rnorm(n, 0, 1))
dat <- data.frame(y, x1, x2, z1, z2)

result <- spscsfa(y ~ x1 + x2 | z1 + z2, data = dat, spscreg = "ll", gradients = TRUE)

# z3 is an unordered categorical variable
# spscsfa() detects if z3 is a factor
dat$z3 <- factor(rep(sample(c("A", "B", "C", "D", "E")), 20))

# z1, z2, z3 enter the nonparametric part and are also inefficiency determinants
result <- spscsfa(y ~ x1 + x2 | z1 + z2 + z3, data = dat)

# z3 is preserved as an unordered categorical variable in the nonparametric part
result$smooth_coefficients$bandwidth

# z3 is converted to dummy variables as inefficiency determinants
result$mle_results$delta_table

# Same as above (inefficiency defaults to nonparametric part)
result <- spscsfa(y ~ x1 + x2 | z1 + z2 + z3 | NULL, data = dat)

# Three-part formula: use z3 only as the inefficiency determinant
result <- spscsfa(y ~ x1 + x2 | z1 + z2 + z3 | z3, data = dat)

# Use external variables as inefficiency determinants
dat$q1 <- rnorm(n)
dat$q2 <- rnorm(n)
result <- spscsfa(y ~ x1 + x2 | z1 + z2 + z3 | q1 + q2, data = dat)
```

```
# No inefficiency determinants (intercept only)
result <- spscsfa(y ~ x1 + x2 | z1 + z2 + z3 | 1, data = dat, verbose = TRUE)

# Application with real data (ricephil)
## Not run:

data(ricephil)

ricephil$FARMERCODE <- as.factor(ricephil$FARMERCODE)

# this example shows categorical variable handling
result <- spscsfa(PROD ~ AREA + LABOR + NPK | EDYRS + FARMERCODE | FARMERCODE,
                  data = ricephil, optim_method = "optim_BFGS", verbose = TRUE)

# FARMERCODE is preserved as an unordered categorical variable in the nonparametric part
result$smooth_coefficients$bandwidth

# FARMERCODE is converted to dummy variables as inefficiency determinants
result$mle_results$delta_table

## End(Not run)
```

`summary.spscsfa` *Summarize SPSC-SFA results*

Description

`summary` method for objects of class `"spscsfa"`. Following the convention of `summary`, it *computes and returns* a summary object rather than printing one, so that the result can be assigned, subsetted and reused programmatically. The summary is rendered by `print.summary.spscsfa`, which is dispatched automatically when the object is auto-printed at the console.

Usage

```
## S3 method for class 'spscsfa'
summary(object, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

`object`	An object of class `"spscsfa"`, as returned by `spscsfa`.
`digits`	Integer; the minimum number of significant digits to use when the summary is printed. Stored in the returned object and used as the default by `print.summary.spscsfa`, where it can be overridden.
`...`	Further arguments passed to or from other methods. Accepted for compatibility with the generic `summary` and otherwise unused.

Value

An object of class `"summary.spscsfa"`: a list with components

formula, n_obs, data_supplied, data_info Model formula, number of observations used, whether a data frame was supplied, and the data cleaning information.

n_params_x, x_names Number and names of the parametric variables.

n_params_z, z_names, z_is_factor, z_is_ordered Number, names and types of the nonparametric variables.

categorical_conversion_warning Note about categorical inefficiency determinants converted to binary variables, or `NULL`.

spscreg The estimator of the smooth coefficients: `"lc"` (local-constant) or `"ll"` (local-linear).

bandwidth, bandwidth_user_supplied, n_training, n_evaluation, used_evaluation_data Step 1 smooth-coefficient information.

smooth_coefficients Numeric matrix with one row for each smooth coefficient ($\theta(z)$, the intercept, and then each $\beta(z)$) and the columns `Min.`, `1st Qu.`, `Median`, `Mean`, `3rd Qu.` and `Max.`: the summary statistics of the Step 1 estimates over the evaluation points (the rows of `ezdat` if it was supplied to `spscsfa`, the training observations otherwise), or `NULL` if the estimates are not available.

inefficiency_vars_null, n_inefficiency_vars, sigma_u_formula Specification of the inefficiency equation.

alpha Numeric matrix with one row, `Intercept (alpha)`, and the columns `Min.`, `1st Qu.`, `Median`, `Mean`, `3rd Qu.` and `Max.`: the summary statistics of the intercept $\alpha(z) = \theta(z) + E(u|z)$ over the training observations, or `NULL` if it is not available.

coefficients Numeric matrix of the `delta` coefficients with columns `Estimate`, `Std. Error`, `z value` and `Pr(>|z|)`.

variance Numeric matrix, in the same layout, for `sigma2_v`.

logLik, optim_method Maximized log-likelihood and optimizer used.

te, inefficiency Named numeric vectors of summary statistics (`Mean`, `SD`, `Min`, `Max`, `Median`) for the BC technical efficiency and JLMS inefficiency estimates.

sig_level Significance level used for the technical efficiency confidence intervals.

gradients `NULL` unless the model was estimated with `gradients = TRUE`. Otherwise a list with the elements `theta_beta_grad` (a list with one matrix for each continuous `z` variable), `alpha_grad` and `inefficiency_grad` (matrices), each with one row for each gradient and the columns `Min.`, `1st Qu.`, `Median`, `Mean`, `3rd Qu.` and `Max.`: the summary statistics of that gradient. For `theta_beta_grad` they are computed over the evaluation points: the rows of `ezdat` if it was supplied to `spscsfa`, the training observations otherwise. For `alpha_grad` and `inefficiency_grad` they are always computed over the training observations.

digits Default number of significant digits for printing.

See Also

`print.summary.spscsfa`, `print.spscsfa`, `spscsfa`

`te` *Extract technical efficiency scores*

Description

Extract technical efficiency scores

Usage

```
te(x)
```

Arguments

`x`	An object of class `"spscsfa"`.

Value

A numeric vector of technical efficiency estimates.

Index

∗ **datasets**
    ricephil, 19

coef, *10*
coef.llsmoothcoefficient *(*llscoef*)*, 6

fitted, *10*
fitted.llsmoothcoefficient *(*llscoef*)*, 6

grad, 2, *24*

inefficiency, 4, *24*

lcscoefgrad, 4, *10*, *23*
llscoef, *6*, 6, *11, 12*, *14*, *21*, *23*
llscoefbw, *7, 8*, *10*, 11, *21*, *23*

model.frame, *12*

optim, *13*

plot, *16*
plot.spscsfa, 15, *24*
print, *17, 18*
print.default, *18, 19*
print.llsmoothcoefficient *(*llscoef*)*, 6
print.llsmoothcoefficientbw
        *(*llscoefbw*)*, 11
print.spscsfa, 17, *19*, *28*
print.summary.spscsfa, *18*, 18, *27, 28*
printCoefmat, *18, 19*

residuals, *10*
residuals.llsmoothcoefficient
        *(*llscoef*)*, 6
ricephil, 19

smooth_coef *(*smoothcoef*)*, 20
smoothcoef, *3*, *16*, 20, *24*
spsc_sfa *(*spscsfa*)*, 20
spscsfa, *2–4*, *7*, *9–11*, *13–16*, *18*, *20*, 20, *24*, *27, 28*
summary, *27*
summary.spscsfa, *17–19*, *24*, 27

te, *24*, 29