

Package ‘crs’

August 19, 2026

Version 0.15-46

Date 2026-08-08

Imports boot, stats, quantreg

Suggests ggplot2, knitr, logspline, MASS, quadprog, rgl, rmarkdown,
testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Title Categorical Regression Splines

Description Regression splines that handle a mix of continuous and categorical (discrete) data often encountered in applied settings. I would like to gratefully acknowledge support from the Natural Sciences and Engineering Research Council of Canada (NSERC, <<https://www.nserc-crsng.gc.ca>>), the Social Sciences and Humanities Research Council of Canada (SSHRC, <<https://www.sshrc-crsh.gc.ca>>), and the Shared Hierarchical Academic Research Computing Network (SHARC-NET, <<https://www.sharcnet.ca>>). We would also like to acknowledge the contributions of the GNU GSL authors. In particular, we adapt the GNU GSL B-spline routine `gsl_bspline.c` adding automated support for quantile knots (in addition to uniform knots), providing missing functionality for derivatives, and for extending the splines beyond their endpoints.

License GPL (>= 3)

URL <https://github.com/JeffreyRacine/R-Package-crs>

BugReports <https://github.com/JeffreyRacine/R-Package-crs/issues>

Repository CRAN

NeedsCompilation yes

Author Jeffrey S. Racine [aut, cre],
Zhenghua Nie [aut],
Brian D. Ripley [ctb] (stepCV.R)

Maintainer Jeffrey S. Racine <racinej@mcmaster.ca>

Date/Publication 2026-08-19 16:40:02 UTC

Contents

crs-package	2
clsd	3
cps71	9
crs	10
crshat	20
crsiv	22
crsivderiv	29
crssigtest	35
crs_nomad_api	37
Engel95	44
frscv	46
frscvNOMAD	50
glp.model.matrix	54
gsl.bs	55
krscv	57
krscvNOMAD	61
plot.crs	67
snomadr	71
tensor.prod.model.matrix	78
uniquecombs	80
wage1	81
Index	83

crs-package	<i>Nonparametric Regression Splines with Continuous and Categorical Predictors</i>
-------------	--

Description

This package provides a method for nonparametric regression that combines the (global) approximation power of regression splines for continuous predictors ('x') with the (local) power of kernel methods for categorical predictors ('z'). The user also has the option of instead using indicator bases for the categorical predictors. When the predictors contain both continuous and categorical (discrete) data types, both approaches offer more efficient estimation than the traditional sample-splitting (i.e. 'frequency') approach where the data is first broken into subsets governed by the categorical z.

To cite the **crs** package type: 'citation("crs")' (without the single quotes).

For a listing of all routines in the **crs** package type: 'library(help="crs")'.

For a listing of all demos in the **crs** package type: 'demo(package="crs")'.

For a 'vignette' that presents a getting-started introduction to the **crs** package type: 'vignette("crs_getting_started", package = "crs")'.

Details

For the continuous predictors the regression spline model employs the B-spline basis matrix using the B-spline routines in the GNU Scientific Library (<https://www.gnu.org/software/gsl/>).

The `tensor.prod.model.matrix` function is used to construct multivariate tensor spline bases when `basis="tensor"` and uses additive B-splines otherwise (i.e. when `basis="additive"`).

For the discrete predictors the product kernel function is of the ‘Li-Racine’ type (see Li and Racine (2007) for details) which is formed by constructing products of one of the following univariate kernels:

(**z is discrete/nominal**) $l(z_i, z, \lambda) = 1$ if $z_i = z$, and λ if $z_i \neq z$. Note that λ must lie between 0 and 1.

(**z is discrete/ordinal**) $l(z_i, z, \lambda) = 1$ if $|z_i - z| = 0$, and $\lambda^{|z_i - z|}$ if $|z_i - z| \geq 1$. Note that λ must lie between 0 and 1.

Alternatively, for the ordinal/nominal predictors the regression spline model will use indicator basis functions.

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca> and Zhenghua Nie <niez@mcmaster.ca>

Maintainer: Jeffrey S. Racine <racinej@mcmaster.ca>

I would like to gratefully acknowledge support from the Natural Sciences and Engineering Research Council of Canada (<https://www.nserc-crsng.gc.ca>), the Social Sciences and Humanities Research Council of Canada (<https://www.sshrc-crsh.gc.ca>), and the Shared Hierarchical Academic Research Computing Network (<https://www.sharcnet.ca>).

References

Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.

Ma, S. and J.S. Racine and L. Yang (2015), “Spline Regression in the Presence of Categorical Predictors,” *Journal of Applied Econometrics*, Volume 30, 705-717.

Ma, S. and J.S. Racine (2013), “Additive Regression Splines with Irrelevant Categorical and Continuous Regressors,” *Statistica Sinica*, Volume 23, 515-541.

cld

Categorical Logspline Density

Description

`cld` computes the logspline density, density derivative, distribution, and smoothed quantiles for a one (1) dimensional continuous variable using the approach of Racine (2013).

Usage

```
cIsd(x = NULL,
     beta = NULL,
     xeval = NULL,
     basis = "tensor",
     degree = NULL,
     degree.max = 25,
     degree.min = 2,
     deriv = 1,
     deriv.index = 1,
     display.nomad.progress = TRUE,
     display.warnings = TRUE,
     do.gradient = TRUE,
     elastic.diff = 3,
     elastic.max = TRUE,
     er = NULL,
     knots = "quantiles",
     lbound = NULL,
     max.attempts = 25,
     maxit = 10^5,
     method = c("L-BFGS-B", "Nelder-Mead", "BFGS", "CG", "SANN"),
     monotone = TRUE,
     monotone.lb = -250,
     n.integrate = 500,
     nmulti = 1,
     NOMAD = FALSE,
     penalty = NULL,
     quantile.seq = seq(.01,.99,by=.01),
     random.seed = 42,
     segments = NULL,
     segments.max = 100,
     segments.min = 1,
     ubound = NULL,
     verbose = FALSE)
```

Arguments

Data, Model Inputs And Formula Interface: These arguments identify training data, evaluation data, support bounds, and optional coefficients.

<code>beta</code>	a numeric vector of coefficients (default NULL)
<code>x</code>	a numeric vector of training data
<code>xeval</code>	a numeric vector of evaluation data

Density Basis Structure: These arguments control basis type and spline complexity for constrained density estimation.

basis	a character string (default basis="tensor") indicating whether the additive or tensor product B-spline basis matrix for a multivariate polynomial spline or generalized B-spline polynomial basis should be used
degree	integer/vector specifying the polynomial degree of the B-spline basis for each dimension of the continuous x (default degree=2)
degree.min, degree.max	when elastic.max=FALSE the minimum/maximum degree of the B-spline basis for each of the continuous predictors (default degree.min=2, degree.max=25)
knots	a character string (default knots="quantiles") specifying where knots are to be placed. 'quantiles' specifies knots placed at equally spaced quantiles (equal number of observations lie in each segment) and 'uniform' specifies knots placed at equally spaced intervals
segments	integer/vector specifying the number of segments of the B-spline basis for each dimension of the continuous x (i.e. number of knots minus one) (default segments=1, i.e. Bezier curve)
segments.min, segments.max	when elastic.max=FALSE, the minimum/maximum segments of the B-spline basis for each of the continuous predictors (default segments.min=1, segments.max=100)

Derivatives, Integration, And Quantiles: These arguments control derivative extraction, numerical integration, and quantile evaluation.

deriv	an integer l (default deriv=1) specifying whether to compute the univariate lth partial derivative for each continuous predictor (and difference in levels for each categorical predictor) or not and if so what order. Note that if deriv is higher than the spline degree of the associated continuous predictor then the derivative will be zero and a warning issued to this effect
deriv.index	an integer l (default deriv.index=1) specifying the index (currently only supports 1) of the variable whose derivative is requested
er	a scalar indicating the fraction of data range to extend the tails (default 1/log(n), see extendrange for further details)
n.integrate	the number of evenly spaced integration points on the extended range specified by er (defaults to 500)
quantile.seq	a sequence of numbers lying in [0, 1] on which quantiles from the logspline distribution are obtained

Optimization Controls: These arguments control optimizer choice, restart behavior, elastic search limits, and penalties.

do.gradient	a logical value indicating whether or not to use the analytical gradient during optimization (defaults to TRUE)
elastic.max	a single, non-missing logical condition indicating whether to use 'elastic' search bounds. Numeric scalar conditions retain their historical R interpretation, although TRUE or FALSE is recommended
elastic.diff	an integer indicating how many units the optimal degree/segment must lie from the respective elastic search bounds

<code>max.attempts</code>	maximum number of attempts to undertake if <code>optim</code> fails for any set of initial parameters for each value of <code>nmulti</code>
<code>maxit</code>	maximum number of iterations used by <code>optim</code>
<code>method</code>	see <code>optim</code> for details
<code>nmulti</code>	integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points (default <code>nmulti=1</code>)
<code>NOMAD</code>	a single, non-missing logical condition which when TRUE calls <code>snomadr</code> to determine the optimal degree and segments. Numeric scalar conditions retain their historical R interpretation, although TRUE or FALSE is recommended
<code>penalty</code>	the parameter to be used in the AIC criterion. The method chooses the number of degrees plus number of segments (knots-1) that maximizes $2 \cdot \log 1 - \text{penalty} \cdot (\text{degree} + \text{segments})$. The default is to use the penalty parameter of $\log(n)/2$ (2 would deliver standard AIC, $\log(n)$ standard BIC)
<code>random.seed</code>	seeds the random number generator for initial parameter values when <code>optim</code> is called

Support And Shape Controls: These arguments control support bounds and optional monotonicity constraints.

<code>lbound, ubound</code>	lower/upper bound for the support of the density. For example, if there is a priori knowledge that the density equals zero to the left of 0, and has a discontinuity at 0, the user could specify <code>lbound = 0</code> . However, if the density is essentially zero near 0, one does not need to specify <code>lbound</code>
<code>monotone</code>	a logical value indicating whether modify the standard B-spline basis function so that it is tailored for density estimation (default TRUE)
<code>monotone.lb</code>	a negative bound specifying the lower bound on the linear segment coefficients used when (<code>monotone=FALSE</code>)

Warnings And Progress: These arguments control warnings, verbosity, and displayed optimizer progress.

<code>display.nomad.progress</code>	a logical value indicating whether to display the progress of the NOMAD solver (default <code>display.nomad.progress=TRUE</code>)
<code>display.warnings</code>	a logical value indicating whether to display warnings (default <code>display.warnings=TRUE</code>)
<code>verbose</code>	a logical value which when TRUE produces verbose output during optimization

Details

Typical usages are (see below for a list of options and also the examples at the end of this help file)

```
model <- clds(x)
```

cldsd computes a logspline density estimate of a one (1) dimensional continuous variable.

The spline model employs the tensor product B-spline basis matrix for a multivariate polynomial spline via the B-spline routines in the GNU Scientific Library (<https://www.gnu.org/software/gsl/>) and the `tensor.prod.model.matrix` function.

When `basis="additive"` the model becomes additive in nature (i.e. no interaction/tensor terms thus semiparametric not fully nonparametric).

When `basis="tensor"` the model uses the multivariate tensor product basis.

Value

cldsd returns a cldsd object. The generic functions `coef`, `fitted`, `plot` and `summary` support objects of this type. The `plot` method supports `er=FALSE` to plot the density on the sample realizations (default is 'extended range' data), `distribution=TRUE` to plot the distribution, `derivative=TRUE` to plot the density derivative, and `output=c("plot","data","plot-data")` or behavior to draw, return the plot data, or both. Bootstrap, interval, surface-rendering, rug, legend, overlay, and regression-gradient plot controls are not supported for this curve route and fail fast when supplied. See `plot.crs` for the shared CRS plot-output conventions. The returned object has the following components:

<code>density</code>	estimates of the density function at the sample points
<code>density.er</code>	the density evaluated on the 'extended range' of the data
<code>density.deriv</code>	estimates of the derivative of the density function at the sample points
<code>density.deriv.er</code>	estimates of the derivative of the density function evaluated on the 'extended range' of the data
<code>distribution</code>	estimates of the distribution function at the sample points
<code>distribution.er</code>	the distribution evaluated on the 'extended range' of the data
<code>xer</code>	the 'extended range' of the data
<code>degree</code>	integer/vector specifying the degree of the B-spline basis for each dimension of the continuous x
<code>segments</code>	integer/vector specifying the number of segments of the B-spline basis for each dimension of the continuous x
<code>xq</code>	vector of quantiles
<code>tau</code>	vector generated by <code>quantile.seq</code> or input by the user (lying in $[0,1]$) from which the quantiles <code>xq</code> are obtained
<code>nomad.summary</code>	summary of NOMAD blackbox evaluations and cache activity, present only when NOMAD search was used

Usage Issues

This function should be considered to be in 'beta' status until further notice.

If smoother estimates are desired and `degree=degree.min`, increase `degree.min` to, say, `degree.min=3`.

The use of 'regression' B-splines can lead to undesirable behavior at the endpoints of the data (i.e. when `monotone=FALSE`). The default 'density' B-splines ought to be well-behaved in these regions.

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

References

Racine, J.S. (2013), “Logspline Mixed Data Density Estimation,” manuscript.

See Also

[logspline](#), [plot.crs](#)

Examples

```
## Not run:
## Old Faithful eruptions data histogram and clds density

library(MASS)
data(faithful)
attach(faithful)

model <- clds(eruptions)

ylim <- c(0,max(model$density,hist(eruptions,breaks=20,plot=FALSE)$density))

plot(model,ylim=ylim)

hist(eruptions,breaks=20,freq=FALSE,add=TRUE,lty=2)

rug(eruptions)

summary(model)

coef(model)

## Simulated data

set.seed(42)
require(logspline)

## Example - simulated data

n <- 250
x <- sort(rnorm(n))
f.dgp <- dnorm(x)

model <- clds(x)

## Standard (cubic) estimate taken from the logspline package
## Compute MSEs

mse.clds <- mean((fitted(model)-f.dgp)^2)
```

```

model.logspline <- logspline(x)

mse.logspline <- mean((dlogspline(x,model.logspline)-f.dgp)^2)

ylim <- c(0,max(fitted(model),dlogspline(x,model.logspline),f.dgp))

plot(model,
      ylim=ylim,
      sub=paste("MSE: logspline = ",format(mse.logspline),", clsd = ",
              format(mse.clsd)),
      lty=3,
      col=3)

xer <- model$xer

lines(xer,dlogspline(xer,model.logspline),col=2,lty=2)
lines(xer,dnorm(xer),col=1,lty=1)

rug(x)

legend("topright",c("DGP",
                    paste("Cubic Logspline Density (package 'logspline', knots = ",
                          model.logspline$nknots,")",sep=""),
                    paste("clsd Density (degree = ", model$degree, ", segments = ",
                          model$segments, ", penalty = ",round(model$penalty,2),")",sep="")),
      lty=1:3,
      col=1:3,
      bty="n",
      cex=0.75)

summary(model)

coef(model)

## Simulate data with known bounds

set.seed(42)
n <- 10000
x <- runif(n,0,1)

model <- clsd(x,lbound=0,ubound=1)

plot(model)

## End(Not run)

```

Description

Canadian cross-section wage data consisting of a random sample taken from the 1971 Canadian Census Public Use Tapes for male individuals having common education (grade 13). There are 205 observations in total.

Usage

```
data("cps71")
```

Format

A data frame with 2 columns, and 205 rows.

logwage the first column, of type numeric

age the second column, of type integer

Source

Aman Ullah

References

Pagan, A. and A. Ullah (1999), *Nonparametric Econometrics*, Cambridge University Press.

Examples

```
## Example - fit a spline model for log wages as a function of age.

data(cps71, package = "crs")

model.crs <- crs(logwage~age, data = cps71, complexity="degree-knots")
with(cps71, plot(age, logwage, cex=0.25, col="grey",
  sub=paste("crs-CV = ", formatC(model.crs$cv.score,format="f",digits=3))))
lines(cps71$age, fitted(model.crs), lty=1, col=1)

crs.txt <- paste("crs (R-squared = ", formatC(model.crs$r.squared,format="f",digits=3),")", sep="")
legend(22.5, 15, crs.txt, lty=1, col=1, bty="n")

summary(model.crs)
```

crs

Categorical Regression Splines

Description

crs computes a regression spline estimate of a one (1) dimensional dependent variable on an r-dimensional vector of continuous and categorical ([factor/ordered](#)) predictors (Ma and Racine (2013), Ma, Racine and Yang (2015)).

Usage

```

crs(...)
## Default S3 method:
crs(xz,
    y,
    basis = c("auto", "additive", "tensor", "glp"),
    complexity = c("degree-knots", "degree", "knots"),
    data.return = FALSE,
    degree = NULL,
    deriv = 0,
    display.nomad.progress = TRUE,
    display.warnings = TRUE,
    include = NULL,
    kernel = TRUE,
    knots = c("quantiles", "uniform", "auto"),
    lambda = NULL,
    model.return = FALSE,
    prune = FALSE,
    segments = NULL,
    tau = NULL,
    weights = NULL,
    ...)

## S3 method for class 'formula'
crs(formula,
    basis = c("auto", "additive", "tensor", "glp"),
    complexity = c("degree-knots", "degree", "knots"),
    cv = c("nomad", "exhaustive", "none"),
    cv.df.min = 1,
    cv.func = c("cv.ls", "cv.gcv", "cv.aic"),
    cv.threshold = 1000,
    data = list(),
    data.return = FALSE,
    degree = NULL,
    degree.max = 10,
    degree.min = 0,
    deriv = 0,
    display.nomad.progress = TRUE,
    display.warnings = TRUE,
    include = NULL,
    initial.mesh.size.integer = "1",
    initial.mesh.size.real = "r1.0e-01",
    kernel = TRUE,
    knots = c("quantiles", "uniform", "auto"),
    lambda = NULL,
    lambda.discrete = FALSE,
    lambda.discrete.num = 100,
    max.bb.eval = NULL,

```

```

max.eval = NULL,
min.mesh.size.integer = 1,
min.mesh.size.real = paste(sqrt(.Machine$double.eps)),
min.frame.size.integer = 1,
min.frame.size.real = 1,
model.return = FALSE,
nmulti = 2,
opts=list(),
prune = FALSE,
random.seed = 42,
restarts = 0,
segments = NULL,
segments.max = 10,
segments.min = 1,
singular.ok = FALSE,
tau = NULL,
weights = NULL,
...)

```

Arguments

Data, Model Inputs And Formula Interface: These arguments identify the model formula/data interface and explicit data inputs.

data	an optional data frame containing the variables in the model
formula	a symbolic description of the model to be fit
xz	numeric (x) and or nominal/ordinal (factor/ordered) predictors (z)
y	a numeric vector of responses.

Basis, Spline, And Kernel Structure: These arguments control basis type, spline complexity, factor inclusion, and optional kernel smoothing.

basis	a character string (default basis="auto") indicating whether the additive or tensor product B-spline basis matrix for a multivariate polynomial spline or generalized B-spline polynomial basis should be used. Note this can be automatically determined by cross-validation if cv="nomad" or cv="exhaustive" and basis="auto", and is an 'all or none' proposition (i.e. interaction terms for all predictors or for no predictors given the nature of 'tensor products'). Note also that if there is only one predictor this defaults to basis="additive" to avoid unnecessary computation as the spline bases are equivalent in this case
complexity	a character string (default complexity="degree-knots") indicating whether model 'complexity' is determined by the degree of the spline or by the number of segments (i.e. number of knots minus one). This option allows the user to use cross-validation to select either the spline degree (number of knots held fixed) or the number of knots (spline degree held fixed) or both the spline degree and number of knots For the continuous predictors the regression spline model employs either the additive or tensor product B-spline basis matrix for a multivariate polynomial

	spline via the B-spline routines in the GNU Scientific Library (https://www.gnu.org/software/gsl/) and the <code>tensor.prod.model.matrix</code> function
<code>degree</code>	integer/vector specifying the polynomial degree of the B-spline basis for each dimension of the continuous <code>x</code> (default <code>degree=3</code> , i.e. cubic spline)
<code>degree.max</code>	the maximum degree of the B-spline basis for each of the continuous predictors (default <code>degree.max=10</code>)
<code>degree.min</code>	the minimum degree of the B-spline basis for each of the continuous predictors (default <code>degree.min=0</code>)
<code>include</code>	integer/vector specifying whether each of the nominal/ordinal (<code>factor/ordered</code>) predictors in <code>x</code> are included or omitted from the resulting estimate
<code>kernel</code>	a logical value (default <code>kernel=TRUE</code>) indicating whether to use kernel smoothing or not
<code>knots</code>	a character string (default <code>knots="quantiles"</code>) specifying where knots are to be placed. 'quantiles' specifies knots placed at equally spaced quantiles (equal number of observations lie in each segment) and 'uniform' specifies knots placed at equally spaced intervals. If <code>knots="auto"</code> , the knot type will be automatically determined by cross-validation
<code>lambda</code>	a vector of bandwidths for each dimension of the categorical <code>z</code>
<code>lambda.discrete</code>	if <code>lambda.discrete=TRUE</code> , the bandwidth will be discretized into <code>lambda.discrete.num+1</code> points and <code>lambda</code> will be chosen from these points
<code>lambda.discrete.num</code>	a positive integer indicating the number of discrete values that <code>lambda</code> can assume - this parameter will only be used when <code>lambda.discrete=TRUE</code>
<code>segments</code>	integer/vector specifying the number of segments of the B-spline basis for each dimension of the continuous <code>x</code> (i.e. number of knots minus one) (default <code>segments=1</code> , i.e. Bezier curve)
<code>segments.max</code>	the maximum segments of the B-spline basis for each of the continuous predictors (default <code>segments.max=10</code>)
<code>segments.min</code>	the minimum segments of the B-spline basis for each of the continuous predictors (default <code>segments.min=1</code>)

Cross-Validation And Search Controls: These arguments control cross-validation objective selection and restart behavior.

<code>cv</code>	a character string (default <code>cv="nomad"</code>) indicating whether to use nonsmooth mesh adaptive direct search, exhaustive search, or no search (i.e. use user supplied values for <code>degree</code> , <code>segments</code> , and <code>lambda</code>)
<code>cv.df.min</code>	the minimum degrees of freedom to allow when conducting NOMAD-based cross-validation (default <code>cv.df.min=1</code>)
<code>cv.func</code>	a character string (default <code>cv.func="cv.ls"</code>) indicating which method to use to select smoothing parameters. <code>cv.gcv</code> specifies generalized cross-validation (Craven and Wahba (1979)), <code>cv.aic</code> specifies expected Kullback-Leibler cross-validation (Hurvich, Simonoff, and Tsai (1998)), and <code>cv.ls</code> specifies least-squares cross-validation

<code>cv.threshold</code>	an integer (default <code>cv.threshold=1000</code>) that controls the automatic switch from NOMAD to exhaustive search for simple problems with no categorical predictors. If <code>cv="nomad"</code> and the number of degree/segments combinations is less than or equal to <code>cv.threshold</code> , <code>crs</code> quietly uses <code>cv="exhaustive"</code> because enumeration is cheap and deterministic. Set <code>cv.threshold=0</code> to disable this automatic switch and keep <code>cv="nomad"</code> on the NOMAD path. Use <code>cv="exhaustive"</code> to request exhaustive search explicitly.
<code>nmulti</code>	integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points (default <code>nmulti=2</code>)
<code>prune</code>	a logical value (default <code>prune=FALSE</code>) specifying whether the (final) model is to be ‘pruned’ using a stepwise cross-validation criterion based upon a modified version of stepAIC (see below for details)
<code>random.seed</code>	when it is not missing and not equal to 0, the initial points will be generated using this seed when using frscvNOMAD or krscvNOMAD and <code>nmulti > 0</code>
<code>restarts</code>	integer specifying the number of times to restart the process of finding extrema of the cross-validation function (for the bandwidths only) from different (random) initial points
<code>singular.ok</code>	a logical value (default <code>singular.ok=FALSE</code>) that, when <code>FALSE</code> , discards singular bases during cross-validation (a check for ill-conditioned bases is performed).

NOMAD Controls: These arguments control NOMAD mesh settings and optional solver controls.

<code>initial.mesh.size.integer</code>	argument passed to the NOMAD solver (see snomadr for further details)
<code>initial.mesh.size.real</code>	argument passed to the NOMAD solver (see snomadr for further details)
<code>max.bb.eval</code>	argument passed to the NOMAD solver. The default <code>NULL</code> lets <code>crs</code> choose a route-specific evaluation budget: 10000 for continuous-only frscvNOMAD search and 1000 for kernel/categorical krscvNOMAD search. These defaults were set on the basis of simulation evidence and real-world applications. User-supplied values are passed through to the selected NOMAD route; see snomadr for further details.
<code>max.eval</code>	optional NOMAD total point-lookup budget. This is distinct from <code>max.bb.eval</code> : <code>max.bb.eval</code> limits true blackbox objective computations, while <code>max.eval</code> limits total NOMAD point lookups, including cache hits. The default <code>NULL</code> uses the selected route’s default <code>MAX_EVAL</code> : continuous-only NOMAD search uses <code>MAX_EVAL = 1000</code> , while mixed/kernel NOMAD search retains the historical route behavior of using <code>MAX_BB_EVAL</code> as <code>MAX_EVAL</code> . If supplied, <code>max.eval</code> is passed to NOMAD as <code>MAX_EVAL</code> . Supplying both <code>max.eval</code> and <code>opts\$MAX_EVAL</code> with conflicting values is an error.
<code>min.frame.size.integer</code>	arguments passed to the NOMAD solver (see snomadr for further details)
<code>min.frame.size.real</code>	arguments passed to the NOMAD solver (see snomadr for further details)

`min.mesh.size.integer` arguments passed to the NOMAD solver (see [snomadr](#) for further details)

`min.mesh.size.real` argument passed to the NOMAD solver (see [snomadr](#) for further details)

`opts` list of optional arguments to be passed to [snomadr](#)

Quantile, Weights, And Derivatives: These arguments control derivative extraction, quantile level, and observation weights.

`deriv` an integer l (default `deriv=0`) specifying whether to compute the univariate l th partial derivative for each continuous predictor (and difference in levels for each categorical predictor) or not and if so what order. Note that if `deriv` is higher than the spline degree of the associated continuous predictor then the derivative will be zero and a warning issued to this effect. For `predict.crs()`, an explicitly supplied `deriv=` overrides the value stored on the fitted object; when `deriv` is omitted, prediction preserves the fitted object's derivative setting.

`tau` if non-null a number in (0,1) denoting the quantile for which a quantile regression spline is to be estimated rather than estimating the conditional mean (default `tau=NULL`). Criterion function set by `cv.func=` are modified accordingly to admit quantile regression.

`weights` an optional vector of weights to be used in the fitting process. Should be 'NULL' or a numeric vector. If non-NULL, weighted least squares is used with weights 'weights' (that is, minimizing $\sum(w \cdot e^2)$); otherwise ordinary least squares is used.

Returned State And Output Controls: These arguments control whether fitted model state is returned.

`data.return` a logical value indicating whether to return x, z, y or not (default `data.return=FALSE`)

`model.return` a logical value indicating whether to return the list of [lm](#) models or not when `kernel=TRUE` (default `model.return=FALSE`)

Warnings And Progress: These arguments control warnings and displayed optimizer progress.

`display.nomad.progress` a logical value indicating whether to display the progress of the NOMAD solver (default `display.nomad.progress=TRUE`)

`display.warnings` a logical value indicating whether to display warnings (default `display.warnings=TRUE`)

Additional Arguments: Further optional arguments are passed through to lower-level routines.

`...` optional arguments

Details

Typical usages are (see below for a list of options and also the examples at the end of this help file)

```
## Estimate the model and let the basis type be determined by
## cross-validation (i.e. cross-validation will determine whether to
## use the additive, generalized, or tensor product basis)

model <- crs(y~x1+x2)

## Estimate the model for a specified degree/segment/bandwidth
## combination and do not run cross-validation (will use the
## additive basis by default)

model <- crs(y~x1+factor(x2),cv="none",degree=3,segments=1,lambda=.1)

## Plot the mean and (asymptotic) error bounds

plot(model,errors = "asymptotic")

## Plot the first partial derivative and (asymptotic) error bounds

plot(model,gradients = TRUE,errors = "asymptotic")
```

crs computes a regression spline estimate of a one (1) dimensional dependent variable on an r -dimensional vector of continuous and categorical (**factor/ordered**) predictors.

The regression spline model employs the tensor product B-spline basis matrix for a multivariate polynomial spline via the B-spline routines in the GNU Scientific Library (<https://www.gnu.org/software/gsl/>) and the **tensor.prod.model.matrix** function.

When `basis="additive"` the model becomes additive in nature (i.e. no interaction/tensor terms thus semiparametric not fully nonparametric).

When `basis="tensor"` the model uses the multivariate tensor product basis.

When `kernel=FALSE` the model uses indicator basis functions for the nominal/ordinal (**factor/ordered**) predictors rather than kernel weighting.

When `kernel=TRUE` the product kernel function for the discrete predictors is of the ‘Li-Racine’ type (see Li and Racine (2007) for details).

When `cv="nomad"`, numerical search is undertaken using Nonsmooth Optimization by Mesh Adaptive Direct Search (Abramson, Audet, Couture, Dennis, Jr., and Le Digabel (2011)).

When `kernel=TRUE` and `cv="exhaustive"`, numerical search is undertaken using **optim** and the box-constrained L-BFGS-B method (see **optim** for details). The user may restart the algorithm as many times as desired via the `restarts` argument (default `restarts=0`). The approach ascends from `degree=0` (or `segments=0`) through `degree.max` and for each value of `degree` (or `segments`) searches for the optimal bandwidths. After the most complex model has been searched then the optimal `degree/segments/lambda` combination is selected. If any element of the optimal `degree` (or `segments`) vector coincides with `degree.max` (or `segments.max`) a warning is produced and the user ought to restart their search with a larger value of `degree.max` (or `segments.max`).

Note that the default plot method for a `crs` object displays the fitted conditional mean or conditional quantile surface. One-dimensional partial fitted curves are drawn by default; for two continuous predictors, `perspective=TRUE` requests a two-dimensional surface and `renderer="rgl"` requests the `rgl` renderer. Surface displays use transparent viridis coloring, NP-style rotation controls via `view`, and optional data overlays/rugs via `data_overlay` and `data_rug`. Gradient displays are requested with `gradients=TRUE`; higher-order derivatives use `gradient_order=i`. Intervals are controlled by the modern **np**-style errors, band, alpha, bootstrap, and B arguments.

Note that setting `prune=TRUE` produces a final ‘pruning’ of the model via a stepwise cross-validation criterion achieved by modifying `stepAIC` and replacing `extractAIC` with `extractCV` throughout the function. This option may be enabled to remove potentially superfluous bases thereby improving the finite-sample efficiency of the resulting model. Note that if the cross-validation score for the pruned model is no better than that for the original model then the original model is returned with a warning to this effect. Note also that this option can only be used when `kernel=FALSE`.

Value

`crs` returns a `crs` object. The generic functions `fitted` and `residuals` extract (or generate) estimated values and residuals. Furthermore, the functions `summary`, `predict`, and `plot` support objects of this type. The `plot.crs` method follows the modern **np** plotting interface, including `errors=c("none", "asymptotic", "bootstrap")`, `gradients=TRUE`, `gradient_order`, `output=c("plot", "data", "plot-data", "both")`, `perspective`, `renderer`, `view`, `neval`, `band`, `alpha`, `bootstrap`, `B`, and the control helpers documented in `plot.crs`. `predict.crs()` honors explicit `deriv=` requests for new evaluation data and, when `newdata` is omitted, computes explicit derivative requests at the training data; ordinary `predict(object)` calls retain the stored fitted-value behavior. The returned object has the following components:

<code>fitted.values</code>	estimates of the regression function (conditional mean) at the sample points or evaluation points
<code>lwr, upr</code>	lower/upper bound for a 95% confidence interval for the <code>fitted.values</code> (conditional mean) obtained from <code>predict.lm</code> via the argument <code>interval="confidence"</code> . When plotting with <code>errors = "bootstrap"</code> , bootstrap-based bounds are used instead for mean plots, including one-dimensional fitted curves and two-dimensional fitted surfaces. Mean-regression plots default to the fast fixed-design wild bootstrap (<code>bootstrap="wild"</code> , <code>B=1999</code>) following the modern np interface; the legacy refit bootstrap remains available with <code>bootstrap="inid"</code> , and np -style block refit bootstraps are available with <code>bootstrap="fixed"</code> and <code>bootstrap="geom"</code> . Quantile CRS plots fit with <code>tau</code> follow the npqreg convention and default to the refit <code>bootstrap="inid"</code> selector; <code>fixed</code> and <code>geom</code> are available for quantile fitted curves, while <code>wild</code> remains mean-only. Mean CRS gradient/effect plots requested with <code>gradients=TRUE</code> also support bootstrap derivative/effect intervals; numeric continuous panels use the corresponding <code>crshat()</code> derivative operator, and categorical panels use focal-minus-baseline effect operators.
<code>residuals</code>	residuals computed at the sample points or evaluation points
<code>degree</code>	integer/vector specifying the degree of the B-spline basis for each dimension of the continuous x
<code>segments</code>	integer/vector specifying the number of segments of the B-spline basis for each dimension of the continuous x

<code>include</code>	integer/vector specifying whether each of the nominal/ordinal (factor/ordered) predictors <code>z</code> are included or omitted from the resulting estimate if <code>kernel=FALSE</code> (see below)
<code>kernel</code>	a logical value indicating whether kernel smoothing was used (<code>kernel=TRUE</code>) or not
<code>lambda</code>	vector of bandwidths used if <code>kernel=TRUE</code>
<code>nomad.summary</code>	summary of NOMAD blackbox evaluations, cache activity, and effective NOMAD options, present only when NOMAD search was used
<code>call</code>	a symbolic description of the model
<code>r.squared</code>	coefficient of determination (Doksum and Samarov (1995))
<code>model.lm</code>	an object of 'class' 'lm' if <code>kernel=FALSE</code> or a list of objects of 'class' 'lm' if <code>kernel=TRUE</code> (accessed by <code>model.lm[[1]]</code> , <code>model.lm[[2]]</code> ,...). By way of example, if <code>foo</code> is a <code>crs</code> object and <code>kernel=FALSE</code> , then <code>foo\$model.lm</code> is an object of 'class' 'lm', while objects of 'class' 'lm' return the model.frame in <code>model.lm\$model</code> which can be accessed via <code>foo\$model.lm\$model</code> where <code>foo</code> is the <code>crs</code> object (the model frame <code>foo\$model.lm\$model</code> contains the B-spline bases underlying the estimate which might be of interest). Again by way of example, when <code>kernel=TRUE</code> then <code>foo\$model.lm[[1]]\$model</code> contains the model frame for the first unique combination of categorical predictors, <code>foo\$model.lm[[2]]\$model</code> the second and so forth (the weights will potentially differ for each model depending on the value(s) of <code>lambda</code>)
<code>deriv.mat</code>	a matrix of derivatives (or differences in levels for the categorical <code>z</code>) whose order is determined by <code>deriv=</code> in the <code>crs</code> call
<code>deriv.mat.lwr</code>	a matrix of 95% coverage lower bounds for <code>deriv.mat</code>
<code>deriv.mat.upr</code>	a matrix of 95% coverage upper bounds for <code>deriv.mat</code>
<code>hatvalues</code>	the hatvalues for the estimated model
<code>P.hat</code>	the kernel probability estimates corresponding to the categorical predictors in the estimated model

Usage Issues

Note that when `kernel=FALSE` [summary](#) supports the option `sigtest=TRUE` that conducts an F-test for significance for each predictor.

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

References

- Abramson, M.A. and C. Audet and G. Couture and J.E. Dennis Jr. and S. Le Digabel (2011), "The NOMAD project". Software available at <https://www.gerad.ca/nomad>.
- Craven, P. and G. Wahba (1979), "Smoothing Noisy Data With Spline Functions," *Numerische Mathematik*, 13, 377-403.
- Doksum, K. and A. Samarov (1995), "Nonparametric Estimation of Global Functionals and a Measure of the Explanatory Power of Covariates in Regression," *The Annals of Statistics*, 23 1443-1473.

Hurvich, C.M. and J.S. Simonoff and C.L. Tsai (1998), “Smoothing Parameter Selection in Nonparametric Regression Using an Improved Akaike Information Criterion,” *Journal of the Royal Statistical Society B*, 60, 271-293.

Le Digabel, S. (2011), “Algorithm 909: NOMAD: Nonlinear Optimization With The MADS Algorithm”. *ACM Transactions on Mathematical Software*, 37(4):44:1-44:15.

Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.

Ma, S. and J.S. Racine and L. Yang (2015), “Spline Regression in the Presence of Categorical Predictors,” *Journal of Applied Econometrics*, Volume 30, 705-717.

Ma, S. and J.S. Racine (2013), “Additive Regression Splines with Irrelevant Categorical and Continuous Regressors,” *Statistica Sinica*, Volume 23, 515-541.

Racine, J.S. (2011), “Cross-Validated Quantile Regression Splines,” manuscript.

See Also

[smooth.spline](#), [loess](#), [npreg](#)

Examples

```
set.seed(42)
## Example - simulated data
n <- 1000
num.eval <- 50
x1 <- runif(n)
x2 <- runif(n)
z <- rbinom(n,1,.5)
dgp <- cos(2*pi*x1)+sin(2*pi*x2)+z
z <- factor(z)
y <- dgp + rnorm(n,sd=.5)

## Estimate a model with specified degree, segments, and bandwidth
model <- crs(y~x1+x2+z,degree=c(5,5),
             segments=c(1,1),
             lambda=0.1,
             cv="none",
             kernel=TRUE)

summary(model)

## Perspective plot
x1.seq <- seq(min(x1),max(x1),length=num.eval)
x2.seq <- seq(min(x2),max(x2),length=num.eval)
x.grid <- expand.grid(x1.seq,x2.seq)
newdata <- data.frame(x1=x.grid[,1],x2=x.grid[,2],
                     z=factor(rep(0,num.eval**2),levels=c(0,1)))
z0 <- matrix(predict(model,newdata=newdata),num.eval,num.eval)
newdata <- data.frame(x1=x.grid[,1],x2=x.grid[,2],
                     z=factor(rep(1,num.eval**2),levels=c(0,1)))
z1 <- matrix(predict(model,newdata=newdata),num.eval,num.eval)
zlim=c(min(z0,z1),max(z0,z1))
persp(x=x1.seq,y=x2.seq,z=z0,
```

```

      xlab="x1",ylab="x2",zlab="y",zlim=zlim,
      col=grDevices::adjustcolor("red",alpha.f=0.35),
      border=grDevices::adjustcolor("red",alpha.f=0.60),
      ticktype="detailed",
      theta=45,phi=45)
par(new=TRUE)
persp(x=x1.seq,y=x2.seq,z=z1,
      xlab="x1",ylab="x2",zlab="y",zlim=zlim,
      col=grDevices::adjustcolor("blue",alpha.f=0.35),
      border=grDevices::adjustcolor("blue",alpha.f=0.60),
      theta=45,phi=45,
      ticktype="detailed")

## Partial regression surface plot
plot(model,errors = "asymptotic")
## NP-style data overlay and support rug
plot(model,data_rug=TRUE)
## For two continuous predictors, surface intervals are available through the
## same errors interface
## plot(crs(y~x1+x2,cv="none"),perspective=TRUE,errors="bootstrap",B=99)
## plot(crs(y~x1+x2,cv="none"),perspective=TRUE,renderer="rgl",data_rug=TRUE)
## Not run:
## A plot example where we extract the partial surfaces, confidence
## intervals etc. automatically generated by plot(...) but do
## not plot, rather save for separate use.
pdat <- plot(model,errors = "asymptotic",output ="data")

## Column 1 is the (evaluation) predictor ([,1]), 2-4 ([,-1]) the mean,
## lwr, and upr (note the returned value is a 'list' hence pdat[[1]] is
## data for the first predictor, pdat[[2]] the second etc). Note that
## matplot() can plot this nicely.
matplot(pdat[[1]][,1],pdat[[1]][,-1],
        xlab=names(pdat[[1]][1]),ylab=names(pdat[[1]][2]),
        lty=c(1,2,2),col=c(1,2,2),type="l")

## End(Not run)

```

crshat

CRS Hat Operator

Description

Construct or apply the linear hat operator for fixed-structure CRS mean regression fits. The helper is intended for fixed- X workflows such as fast plot bootstrapping and constrained-estimation diagnostics.

Usage

```
crshat(object, ...)
```

```
## S3 method for class 'crs'
crshat(object,
        newdata = NULL,
        y = NULL,
        output = c("matrix", "apply", "constraint"),
        deriv = 0,
        deriv.index = 1,
        rcond.min = 1e-08,
        use.svd.fallback = TRUE,
        ...)
```

Arguments

<code>object</code>	A fitted mean-regression object of class "crs".
<code>newdata</code>	Optional evaluation data. If omitted, the training data are used.
<code>y</code>	Optional response vector or matrix. Required for <code>output="constraint"</code> ; when supplied with <code>output="apply"</code> , the operator is applied directly without returning a dense hat matrix.
<code>output</code>	One of "matrix", "apply", or "constraint".
<code>deriv</code>	Non-negative integer derivative order. <code>deriv=0</code> returns the fitted-mean hat operator. Positive values return the derivative hat operator for continuous predictor <code>deriv.index</code> .
<code>deriv.index</code>	Positive integer identifying the continuous predictor whose derivative operator is requested.
<code>rcond.min</code>	Minimum reciprocal condition number used by the weighted least-squares primitive before falling back to QR/SVD policy.
<code>use.svd.fallback</code>	Logical value controlling the weighted least-squares fallback policy.
<code>...</code>	Additional arguments, currently unused.

Details

For a fitted CRS mean-regression object, `crshat()` returns the linear operator mapping the training response to fitted values at `newdata`. The default `output="matrix"` returns a dense matrix with class `c("crshat", "matrix")`. The `output="apply"` route applies the same operator to `y` and is preferable when many right-hand sides are needed.

The helper preserves the fitted object's fixed spline/kernel structure, including basis family, degree, segments, pruning state, categorical-kernel bandwidths, weights, and the package's weighted least-squares rank policy. It does not run bandwidth or degree selection.

For `deriv > 0`, `crshat()` returns the derivative operator $H^{(s)}$ such that $H^{(s)}y$ matches the corresponding CRS gradient prediction at `newdata`. This is the CRS analogue of `npreghat`'s derivative-selector behavior in **np**. Derivative operators are available for fixed-structure mean CRS fits across additive, tensor, and GLP bases, including categorical-kernel fits. If the requested derivative order exceeds the fitted spline degree for that predictor, the returned operator is zero.

Quantile CRS objects remain intentionally unsupported because quantile estimation is not linear in the response.

Value

For output="matrix", a matrix of dimension nrow(newdata) x nrow(training data). For output="apply", fitted values produced by applying the operator to y. For output="constraint", the transpose operator scaled by the supplied one-column response vector.

See Also

[crs](#), [predict](#)

Examples

```
set.seed(42)
x <- runif(30)
y <- sin(2 * pi * x) + rnorm(30, sd = 0.1)
fit <- crs(y ~ x, cv = "none", degree = 2, segments = 1,
           display.warnings = FALSE, display.nomad.progress = FALSE)
nd <- data.frame(x = seq(0.1, 0.9, length.out = 5))
H <- crshat(fit, newdata = nd)
drop(H %*% fit$y)
crshat(fit, newdata = nd, output = "apply")
Hd <- crshat(fit, newdata = nd, deriv = 1, deriv.index = 1)
drop(Hd %*% fit$y)
```

crsiv

Nonparametric Instrumental Regression

Description

crsiv computes nonparametric estimation of an instrumental regression function φ defined by conditional moment restrictions stemming from a structural econometric model: $E[Y - \varphi(Z, X)|W] = 0$, and involving endogenous variables Y and Z , exogenous variables X , and instruments W . The function φ is the solution of an ill-posed inverse problem.

When method="Tikhonov", crsiv uses the approach of Darolles, Fan, Florens and Renault (2011) modified for regression splines (Darolles et al use local constant kernel weighting). When method="Landweber-Fridman", crsiv uses the approach of Horowitz (2011) using the regression spline methodology implemented in the **crs** package.

Usage

```
crsiv(y, ...)

## S3 method for class 'formula'
crsiv(y, data = NULL, subset, na.action, ...)

## Default S3 method:
crsiv(y,
      z,
```

```

w,
x = NULL,
zeval = NULL,
weval = NULL,
xeval = NULL,
alpha = NULL,
alpha.max = 1e-01,
alpha.min = 1e-10,
alpha.tol = .Machine$double.eps^0.25,
constant = 0.5,
deriv = 0,
display.nomad.progress = TRUE,
display.warnings = TRUE,
iterate.diff.tol = 1.0e-08,
iterate.max = 1000,
method = c("Landweber-Fridman", "Tikhonov"),
opts = list("MAX_BB_EVAL"=10000,
            "EPSILON"=.Machine$double.eps,
            "INITIAL_MESH_SIZE"="r1.0e-01",
            "MIN_MESH_SIZE"=paste("r",sqrt(.Machine$double.eps),sep=""),
            "MIN_FRAME_SIZE"=paste("r",1,sep=""),
            "DISPLAY_DEGREE"=0),
penalize.iteration = TRUE,
smooth.residuals = TRUE,
start.from = c("Eyz", "EEyz"),
starting.values = NULL,
stop.on.increase = TRUE,
...)

```

Arguments

Data, Model Inputs And Formula Interface: These arguments identify the response, endogenous variables, instruments, and exogenous covariates. The formula interface uses $y \sim z \mid w$ or $y \sim z \mid w \mid x$. Each partition may contain multiple additive terms and scalar transformations. Dot notation, interaction operators, offsets, and matrix-valued terms are not supported.

data	an optional data frame, list, or environment containing the variables in the IV formula. When data is supplied, every formula variable must be present in it; ambient variables do not fill missing columns
subset	an optional specification of observations to retain when using the formula interface
na.action	a function specifying the action for missing observations when using the formula interface. With <code>na.exclude</code> , <code>fitted()</code> and <code>residuals()</code> restore omitted rows as NA
w	a q -variate data frame of instruments. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof
x	an r -variate data frame of exogenous predictors. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof

<code>y</code>	either a one-dimensional numeric or integer response vector, with element i corresponding to row i of <code>z</code> , or an IV formula of the form $y \sim z \mid w$ or $y \sim z \mid w \mid x$
<code>z</code>	a p -variate data frame of endogenous predictors. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof

Derivatives: This argument controls derivative extraction.

<code>deriv</code>	an integer l (default <code>deriv=0</code>) specifying whether to compute the univariate l th partial derivative for each continuous predictor (and difference in levels for each categorical predictor) or not and if so what order. Note that if <code>deriv</code> is higher than the spline degree of the associated continuous predictor then the derivative will be zero and a warning issued to this effect (see important note below)
--------------------	--

Evaluation Inputs: These arguments identify evaluation data for the IV fit.

<code>weval</code>	a q -variate data frame of instruments on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by <code>w</code>
<code>xeval</code>	an r -variate data frame of exogenous predictors on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by <code>x</code>
<code>zeval</code>	a p -variate data frame of endogenous predictors on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by <code>z</code>

Regularization And Iteration Controls: These arguments control regularization, Landweber-Fridman iteration, residual smoothing, and stopping behavior.

<code>alpha</code>	a numeric scalar that, if supplied, is used rather than numerically solving for α , when using <code>method="Tikhonov"</code>
<code>alpha.max</code>	maximum of search range for α , the Tikhonov regularization parameter, when using <code>method="Tikhonov"</code>
<code>alpha.min</code>	minimum of search range for α , the Tikhonov regularization parameter, when using <code>method="Tikhonov"</code>
<code>alpha.tol</code>	the search tolerance for optimize when solving for α , the Tikhonov regularization parameter, when using <code>method="Tikhonov"</code>
<code>constant</code>	the constant to use when using <code>method="Landweber-Fridman"</code>
<code>iterate.diff.tol</code>	the search tolerance for the difference in the stopping rule from iteration to iteration when using <code>method="Landweber-Fridman"</code> (disable by setting to zero)
<code>iterate.max</code>	an integer indicating the maximum number of iterations permitted before termination occurs when using <code>method="Landweber-Fridman"</code>
<code>method</code>	the regularization method employed (default "Landweber-Fridman", see Horowitz (2011); see Darolles, Fan, Florens and Renault (2011) for details for "Tikhonov")
<code>penalize.iteration</code>	a logical value indicating whether to penalize the norm by the number of iterations or not (default TRUE)

<code>smooth.residuals</code>	a logical value (defaults to TRUE) indicating whether to optimize bandwidths for the regression of $y - \varphi(z)$ on w or for the regression of $\varphi(z)$ on w during iteration
<code>start.from</code>	a character string indicating whether to start from $E(Y z)$ (default, "Eyz") or from $E(E(Y z) z)$ (this can be overridden by providing <code>starting.values</code> below)
<code>starting.values</code>	a value indicating whether to commence Landweber-Fridman assuming $\varphi_{-1} = \text{starting.values}$ (proper Landweber-Fridman) or instead begin from $E(y z)$ (defaults to NULL, see details below)
<code>stop.on.increase</code>	a logical value (defaults to TRUE) indicating whether to halt iteration if the stopping criterion (see below) increases over the course of one iteration (i.e. it may be above the iteration tolerance but increased)
Warnings And Progress: These arguments control warnings and displayed optimizer progress.	
<code>display.nomad.progress</code>	a logical value indicating whether to display the progress of the NOMAD solver (default <code>display.nomad.progress=TRUE</code>)
<code>display.warnings</code>	a logical value indicating whether to display warnings (default <code>display.warnings=TRUE</code>)
Additional Arguments: Further NOMAD and CRS controls are passed through to lower-level routines.	
<code>...</code>	additional arguments supplied to <code>crs</code> . In formula calls, weights and observation-indexed <code>starting.values</code> are aligned through the same model frame as the response and IV roles. Formula-time evaluation arguments (<code>newdata</code> , <code>zeval</code> , <code>weval</code> , and <code>xeval</code>) are deliberately not supported; use the native vector interface for evaluation-grid fitting
<code>opts</code>	arguments passed to the NOMAD solver (see <code>snomad</code> for further details)

Details

The formula method is a training-grid interface. It constructs one model frame for the response, endogenous variables, instruments, optional exogenous variables, weights, and observation-indexed starting values, then calls the established native estimator once. Thus subset, missing values, and row-indexed controls cannot acquire different row maps. Formula fits support `fitted()`, `residuals()`, `predict()` without `newdata`, `summary()`, and `plot()`. Arbitrary formula `newdata` is deferred because evaluating a retained spline projection is not generally the same operation as resolving the IV problem on a new grid. Native objects retain their established post-fit CRS projection route.

Tikhonov regularization requires computation of weight matrices of dimension $n \times n$ which can be computationally costly in terms of memory requirements and may be unsuitable (i.e. unfeasible) for large datasets. Landweber-Fridman will be preferred in such settings as it does not require

construction and storage of these weight matrices while it also avoids the need for numerical optimization methods to determine α , though it does require iteration that may be equally or even more computationally demanding in terms of total computation time.

When using `method="Landweber-Fridman"`, an optimal stopping rule based upon $\|E(y|w) - E(\varphi_k(z, x)|w)\|^2$ is used to terminate iteration. However, if local rather than global optima are encountered the resulting estimates can be overly noisy. To best guard against this eventuality set `nmulti` to a larger number than the default `nmulti=2` for `crs` when using `cv="nomad"` or instead use `cv="exhaustive"` if possible (this may not be feasible for non-trivial problems).

Note that for subsequent Landweber-Fridman iterations, a “warm start” strategy is employed. The optimal parameters (spline degree, number of segments, and bandwidths or inclusion indicators) from the previous iteration are used as starting values for the current iteration. The user-supplied `nmulti` is respected for all iterations. For iterations after the first successful one, these optimal parameters serve as the first of the multiple initial points (a warm start), while any remaining restarts are cold starts. If `nmulti` is not explicitly supplied by the user, it defaults to the `crs` default (2) for the first iteration and to 1 for all subsequent iterations. This strategy provides a balance between computational efficiency and robustness, allowing the NOMAD solver to refine the structural parameters as the residuals evolve incrementally while still guarding against local optima.

When using `method="Landweber-Fridman"`, iteration will terminate when either the change in the value of $\|(E(y|w) - E(\varphi_k(z, x)|w))/E(y|w)\|^2$ from iteration to iteration is less than `iterate.diff.tol` or we hit `iterate.max` or $\|(E(y|w) - E(\varphi_k(z, x)|w))/E(y|w)\|^2$ stops falling in value and starts rising.

When your problem is a simple one (e.g. univariate Z , W , and X) you might want to avoid `cv="nomad"` and instead use `cv="exhaustive"` since exhaustive search may be feasible (for `degree.max` and `segments.max` not overly large). This will guarantee an exact solution for each iteration (i.e. there will be no errors arising due to numerical search).

`demo(crsiv)`, `demo(crsiv_exog)`, and `demo(crsiv_exog_persp)` provide flexible interactive demonstrations similar to the example below that allow you to modify and experiment with parameters such as the sample size, method, and so forth in an interactive session.

Value

`crsiv` returns a `crsiv` object (which inherits from the `crs` class). The generic functions `print`, `summary`, `fitted`, `residuals`, `predict`, and `plot` support objects of this type.

For the `plot` function, the options include `plot.data=FALSE` or `data_overlay=FALSE` (logical values indicating whether to plot the data as a scatter plot), `deriv=FALSE` (a logical value indicating whether to plot the derivative of the structural function rather than the function itself), `errors=c("none", "asymptotic")` (asymptotic derivative confidence bounds when available), `output=c("plot", "data", "plot-data")` or `behavior` (whether to draw, return the plot data, or both), and `xtrim=0.0` (a scalar in $[0, 0.5]$ used to trim the plotted support of z by quantiles). Bootstrap, surface-rendering, rug, legend, and regression-gradient plot controls are not supported for this curve route and fail fast when supplied. See `plot.crs` for the shared CRS plot-output and data-overlay conventions. Note that the `plot` method for `crsiv` objects currently only supports univariate endogenous predictors z .

See `crs` for details on the return object components.

In addition to the standard `crs` components, `crsiv` returns components `phi` and either `alpha` when `method="Tikhonov"` or `phi`, `phi.mat`, `num.iterations`, `norm.stop`, `norm.value` and `convergence` when `method="Landweber-Fridman"`.

`fitted()` and `predict()` without `newdata` return the selected structural state `phi`. On a training-grid fit, `residuals()` returns the original response minus that same selected state. `summary()` returns a structured `summary.crsiv` object. New objects also contain a namespaced `iv` component recording the public IV call, role labels, row map, original response, evaluation grid, and selected-state metadata; inherited CRS fields are retained unchanged.

Note

Using the option `deriv=` computes (effectively) the analytical derivative of the estimated $\varphi(Z, X)$ and not that using `crsivderiv`, which instead uses the method of Florens and Racine (2012). Though both are statistically consistent, practitioners may desire one over the other hence we provide both.

Note

This function should be considered to be in ‘beta test’ status until further notice.

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>, Samuele Centorrino <samuele.centorrino@univ-tlse1.fr>

References

- Carrasco, M. and J.P. Florens and E. Renault (2007), “Linear Inverse Problems in Structural Econometrics Estimation Based on Spectral Decomposition and Regularization,” In: James J. Heckman and Edward E. Leamer, Editor(s), *Handbook of Econometrics*, Elsevier, 2007, Volume 6, Part 2, Chapter 77, Pages 5633-5751
- Darolles, S. and Y. Fan and J.P. Florens and E. Renault (2011), “Nonparametric Instrumental Regression,” *Econometrica*, 79, 1541-1565.
- Fève, F. and J.P. Florens (2010), “The Practice of Non-parametric Estimation by Solving Inverse Problems: The Example of Transformation Models,” *Econometrics Journal*, 13, S1-S27.
- Florens, J.P. and J.S. Racine (2012), “Nonparametric Instrumental Derivatives,” Working Paper.
- Fridman, V. M. (1956), “A Method of Successive Approximations for Fredholm Integral Equations of the First Kind,” *Uspeskhi, Math. Nauk.*, 11, 233-334, in Russian.
- Horowitz, J.L. (2011), “Applied Nonparametric Instrumental Variables Estimation,” *Econometrica*, 79, 347-394.
- Landweber, L. (1951), “An Iterative Formula for Fredholm Integral Equations of the First Kind,” *American Journal of Mathematics*, 73, 615-24.
- Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.

See Also

`npreg`, `crs`, `plot.crs`

Examples

```
## Not run:
## This illustration was made possible by Samuele Centorrino
## <samuele.centorrino@univ-tlse1.fr>

set.seed(42)
n <- 500

## For trimming the plot (trim .5% from each tail)

trim <- 0.005

## The DGP is as follows:

## 1)  $y = \phi(z) + u$ 

## 2)  $E(u|z) \neq 0$  (endogeneity present)

## 3) Suppose there exists an instrument  $w$  such that  $z = f(w) + v$  and
##  $E(u|w) = 0$ 

## 4) We generate  $v$ ,  $w$ , and generate  $u$  such that  $u$  and  $z$  are
## correlated. To achieve this we express  $u$  as a function of  $v$  (i.e.  $u =$ 
##  $\gamma v + \epsilon$ )

v <- rnorm(n, mean=0, sd=0.27)
eps <- rnorm(n, mean=0, sd=0.05)
u <- -0.5*v + eps
w <- rnorm(n, mean=0, sd=1)

## In Darolles et al (2011) there exist two DGPs. The first is
##  $\phi(z)=z^2$  and the second is  $\phi(z)=\exp(-\text{abs}(z))$  (which is
## discontinuous and has a kink at zero).

fun1 <- function(z) { z^2 }
fun2 <- function(z) { exp(-abs(z)) }

z <- 0.2*w + v

## Generate two  $y$  vectors for each function.

y1 <- fun1(z) + u
y2 <- fun2(z) + u

## You set  $y$  to be either  $y1$  or  $y2$  (ditto for  $\phi$ ) depending on which
## DGP you are considering:

y <- y1
phi <- fun1

## Sort on  $z$  for plotting and fit through the formula/data interface.
```

```

ivdata <- data.frame(y,z,w,u,v)
ivdata <- ivdata[order(ivdata$z),]
rm(y,z,w,u,v)

model.iv <- crsiv(y ~ z | w, data=ivdata)

plot(ivdata$z,fitted(model.iv),
     xlim=quantile(ivdata$z,c(trim,1-trim)),
     main="",
     xlab="Z",
     ylab="Structural function",
     type="l",
     lwd=2)

lines(ivdata$z,phi(ivdata$z),lty=2)
rug(ivdata$z)

## End(Not run)

```

crsivderiv

Nonparametric Instrumental Derivatives

Description

crsivderiv uses the approach of Florens and Racine (2012) to compute the partial derivative of a nonparametric estimation of an instrumental regression function φ defined by conditional moment restrictions stemming from a structural econometric model: $E[Y - \varphi(Z, X)|W] = 0$, and involving endogenous variables Y and Z and exogenous variables X and instruments W . The derivative function φ' is the solution of an ill-posed inverse problem, and is computed using Landweber-Fridman regularization.

Usage

```

crsivderiv(y, ...)

## S3 method for class 'formula'
crsivderiv(y, data = NULL, subset, na.action, ...)

## Default S3 method:
crsivderiv(y,
           z,
           w,
           x = NULL,
           zeval = NULL,
           weval = NULL,
           xeval = NULL,
           constant = 0.5,
           display.nomad.progress = TRUE,

```

```

display.warnings = TRUE,
iterate.diff.tol = 1.0e-08,
iterate.max = 1000,
opts = list("MAX_BB_EVAL"=10000,
            "EPSILON"=.Machine$double.eps,
            "INITIAL_MESH_SIZE"="r1.0e-01",
            "MIN_MESH_SIZE"=paste("r",sqrt(.Machine$double.eps),sep=""),
            "MIN_FRAME_SIZE"=paste("r",1,sep=""),
            "DISPLAY_DEGREE"=0),
penalize.iteration = TRUE,
smooth.residuals = TRUE,
start.from = c("Eyz","EEyz"),
starting.values = NULL,
stop.on.increase = TRUE,
...)

```

Arguments

Data, Model Inputs And Formula Interface: These arguments identify the response, endogenous variable, instruments, and exogenous covariates. The formula interface uses $y \sim z \mid w$ or $y \sim z \mid w \mid x$. Each partition may contain multiple additive terms and scalar transformations. Dot notation, interaction operators, offsets, and matrix-valued terms are not supported.

<code>data</code>	an optional data frame, list, or environment containing the variables in the IV formula. When data is supplied, every formula variable must be present in it; ambient variables do not fill missing columns
<code>subset</code>	an optional specification of observations to retain when using the formula interface
<code>na.action</code>	a function specifying the action for missing observations when using the formula interface. With <code>na.exclude</code> , <code>fitted()</code> and <code>residuals()</code> restore omitted rows as NA
<code>w</code>	a q -variate data frame of instruments. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof
<code>x</code>	an r -variate data frame of exogenous predictors. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof
<code>y</code>	either a one-dimensional numeric or integer response vector, with element i corresponding to row i of z , or an IV formula of the form $y \sim z \mid w$ or $y \sim z \mid w \mid x$
<code>z</code>	a one-column data frame of continuous endogenous predictors. The current implementation of <code>crsivderiv</code> supports univariate continuous z only

Evaluation Inputs: These arguments identify evaluation data for the derivative fit.

<code>weval</code>	a q -variate data frame of instruments on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by <code>w</code>
<code>xeval</code>	an r -variate data frame of exogenous predictors on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by <code>x</code>

`zeval` a one-column data frame of continuous endogenous predictors on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by `z`

Landweber-Fridman Iteration Controls: These arguments control iteration, residual smoothing, starting values, and stopping behavior.

`constant` the constant to use when using Landweber-Fridman iteration

`iterate.diff.tol` the search tolerance for the difference in the stopping rule from iteration to iteration when using Landweber-Fridman (disable by setting to zero)

`iterate.max` an integer indicating the maximum number of complete positive-index Landweber-Fridman states that may be evaluated. The initialization state $N = 0$ is not counted

`penalize.iteration` a logical value indicating whether to penalize the norm by the number of iterations or not (default TRUE)

`smooth.residuals` a logical value (defaults to TRUE) indicating whether to optimize bandwidths for the regression of $y - \varphi(z)$ on w or for the regression of $\varphi(z)$ on w during iteration

`start.from` a character string indicating whether to start from $E(Y|z)$ (default, "Eyz") or from $E(E(Y|z)|z)$ (this can be overridden by providing `starting.values` below)

`starting.values` optional derivative values for the initialization state φ'_0 . They are used directly before the first reported update; no hidden -1 to 0 update is performed. When NULL, the initialization is obtained from $E(y|z)$ according to `start.from` (see details below)

`stop.on.increase` a logical value (defaults to TRUE) indicating whether to halt iteration if the stopping criterion (see below) increases over the course of one iteration (i.e. it may be above the iteration tolerance but increased)

Warnings And Progress: These arguments control warnings and displayed optimizer progress.

`display.nomad.progress` a logical value indicating whether to display the progress of the NOMAD solver (default `display.nomad.progress=TRUE`)

`display.warnings` a logical value indicating whether to display warnings (default `display.warnings=TRUE`)

Additional Arguments: Further NOMAD and CRS controls are passed through to lower-level routines.

... additional arguments supplied to `crs`. In formula calls, `weights` and observation-indexed `starting.values` are aligned through the same model frame as the response and IV roles. Formula-time evaluation arguments (`newdata`, `zeval`, `weval`, and `xeval`) are deliberately not supported; use the native vector interface for evaluation-grid fitting

`opts` arguments passed to the NOMAD solver (see [snomadr](#) for further details)

Details

The formula method is a training-grid interface. It constructs one model frame for the response, endogenous variable, instruments, optional exogenous variables, weights, and observation-indexed starting values, then calls the established native estimator once. Thus subset, missing values, and row-indexed controls cannot acquire different row maps. Formula fits support `fitted()`, `residuals()`, `predict()` without `newdata`, `predict(..., deriv=1)` for the selected derivative, `summary()`, and `plot()`. Arbitrary formula `newdata` is deferred because evaluating a retained spline projection is not generally the same operation as resolving the IV problem on a new grid. Native objects retain their established post-fit CRS projection route.

The iteration index has a single meaning throughout the returned object. The initialization consists of φ'_0 and its centered integral φ_0 . For $N = 1, 2, \dots$, one adjoint update produces φ'_N , integration and centering produce φ_N , and the residual at that same curve produces the stopping value for state N . Thus column N of `phi.prime.mat` and `phi.mat`, and element N of `norm.stop`, always describe one coherent state. With `penalize.iteration=TRUE`, that element is Nq_N , where q_N is the normalized squared residual criterion at φ_N .

For Landweber-Fridman iteration, an optimal stopping rule based upon $\|E(y|w) - E(\varphi_k(z, x)|w)\|^2$ is used to terminate iteration. However, if local rather than global optima are encountered the resulting estimates can be overly noisy. To best guard against this eventuality set `nmulti` to a larger number than the default `nmulti=2` for [crs](#) when using `cv="nomad"` or instead use `cv="exhaustive"` if possible (this may not be feasible for non-trivial problems).

Note that for subsequent Landweber-Fridman iterations, a “warm start” strategy is employed. The optimal parameters (spline degree, number of segments, and bandwidths or inclusion indicators) from the previous iteration are used as starting values for the current iteration. The user-supplied `nmulti` is respected for all iterations. For iterations after the first successful one, these optimal parameters serve as the first of the multiple initial points (a warm start), while any remaining restarts are cold starts. If `nmulti` is not explicitly supplied by the user, it defaults to the [crs](#) default (2) for the first iteration and to 1 for all subsequent iterations. This strategy provides a balance between computational efficiency and robustness, allowing the NOMAD solver to refine the structural parameters as the residuals evolve incrementally while still guarding against local optima.

When using Landweber-Fridman iteration, iteration will terminate when either the change in the value of $\|(E(y|w) - E(\varphi_k(z, x)|w))/E(y|w)\|^2$ from iteration to iteration is less than `iterate.diff.tol` or we hit `iterate.max` or $\|(E(y|w) - E(\varphi_k(z, x)|w))/E(y|w)\|^2$ stops falling in value and starts rising.

When your problem is a simple one (e.g. univariate Z , W , and X) you might want to avoid `cv="nomad"` and instead use `cv="exhaustive"` since exhaustive search may be feasible (for `degree.max` and `segments.max` not overly large). This will guarantee an exact solution for each iteration (i.e. there will be no errors arising due to numerical search).

The current implementation supports a single continuous endogenous regressor only. Instrument and exogenous regressor data may still be mixed continuous and categorical.

Value

`crsivderiv` returns a `crsivderiv` object (which inherits from the [crs](#) class). The generic functions [print](#), [summary](#), [fitted](#), [residuals](#), [predict](#), and [plot](#) support objects of this type.

For the `plot` function, the options are `plot.data=FALSE` or `data_overlay=FALSE` (logical values indicating whether to plot the data as a scatter plot), `phi=FALSE` (a logical value indicating whether to plot the reconstructed structural function rather than its derivative), and `output=c("plot", "data", "plot-data")` or behavior (whether to draw, return the plot data, or both). Bootstrap, asymptotic interval, surface-rendering, rug, legend, and regression gradient plot controls are not supported for this curve route and fail fast when supplied. See `plot.crs` for the shared CRS plot-output and data-overlay conventions. Note that the `plot` method for `crsivderiv` objects currently only supports a univariate continuous endogenous predictor z .

See `crs` for details on the return object components.

In addition to the standard `crs` components, `crsivderiv` returns components `phi.prime`, `phi`, `phi.prime.mat`, `phi.mat`, `num.iterations`, `norm.stop`, `norm.value` and `convergence`. `num.iterations` is the selected positive-index state, and `phi.prime` and `phi` are taken from that same column of their respective matrices. The matrices and `norm.stop` contain all evaluated states, so their common length can exceed `num.iterations` when a later evaluated state triggers stopping or the stopping-rule selector chooses an earlier state.

`fitted()` and `predict()` without `newdata` return the selected structural state `phi`; `predict(..., deriv=1)` without `newdata` returns the matching selected `phi.prime`. On a training-grid fit, `residuals()` returns the original response minus that same selected structural state. `summary()` returns a structured `summary.crsivderiv` object. New objects also contain a namespaced `iv` component recording the public IV call, role labels, row map, original response, evaluation grid, and selected-state metadata; inherited CRS fields are retained unchanged.

Note

This function currently supports univariate z only. This function should be considered to be in ‘beta test’ status until further notice.

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

References

- Carrasco, M. and J.P. Florens and E. Renault (2007), “Linear Inverse Problems in Structural Econometrics Estimation Based on Spectral Decomposition and Regularization,” In: James J. Heckman and Edward E. Leamer, Editor(s), *Handbook of Econometrics*, Elsevier, 2007, Volume 6, Part 2, Chapter 77, Pages 5633-5751
- Darolles, S. and Y. Fan and J.P. Florens and E. Renault (2011), “Nonparametric Instrumental Regression,” *Econometrica*, 79, 1541-1565.
- Fève, F. and J.P. Florens (2010), “The Practice of Non-parametric Estimation by Solving Inverse Problems: The Example of Transformation Models,” *Econometrics Journal*, 13, S1-S27.
- Florens, J.P. and J.S. Racine (2012), “Nonparametric Instrumental Derivatives,” Working Paper.
- Fridman, V. M. (1956), “A Method of Successive Approximations for Fredholm Integral Equations of the First Kind,” *Uspekhi, Math. Nauk.*, 11, 233-334, in Russian.
- Horowitz, J.L. (2011), “Applied Nonparametric Instrumental Variables Estimation,” *Econometrica*, 79, 347-394.

Landweber, L. (1951), “An Iterative Formula for Fredholm Integral Equations of the First Kind,” American Journal of Mathematics, 73, 615-24.

Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.

See Also

npreg, [crsiv](#), [crs](#), [plot.crs](#)

Examples

```
## Not run:
## This illustration was made possible by Samuele Centorrino
## <samuele.centorrino@univ-tlse1.fr>

set.seed(42)
n <- 500

## For trimming the plot (trim .5% from each tail)

trim <- 0.005

## The DGP is as follows:

## 1)  $y = \phi(z) + u$ 

## 2)  $E(u|z) \neq 0$  (endogeneity present)

## 3) Suppose there exists an instrument  $w$  such that  $z = f(w) + v$  and
##  $E(u|w) = 0$ 

## 4) We generate  $v$ ,  $w$ , and generate  $u$  such that  $u$  and  $z$  are
## correlated. To achieve this we express  $u$  as a function of  $v$  (i.e.  $u =$ 
##  $\gamma v + \epsilon$ )

v <- rnorm(n, mean=0, sd=0.27)
eps <- rnorm(n, mean=0, sd=0.05)
u <- -0.5*v + eps
w <- rnorm(n, mean=0, sd=1)

## In Darolles et al (2011) there exist two DGPs. The first is
##  $\phi(z)=z^2$  and the second is  $\phi(z)=\exp(-\text{abs}(z))$  (which is
## discontinuous and has a kink at zero).

fun1 <- function(z) { z^2 }
fun2 <- function(z) { exp(-abs(z)) }

z <- 0.2*w + v

## Generate two y vectors for each function.

y1 <- fun1(z) + u
```

```

y2 <- fun2(z) + u

## You set y to be either y1 or y2 (ditto for phi) depending on which
## DGP you are considering:

y <- y1
phi <- fun1

## Sort on z (for plotting)

ivdata <- data.frame(y,z,w,u,v)
ivdata <- ivdata[order(ivdata$z),]
rm(y,z,w,u,v)

model.ivderiv <- crsivderiv(y ~ z | w, data=ivdata)

plot(ivdata$z,model.ivderiv$phi.prime,
     xlim=quantile(ivdata$z,c(trim,1-trim)),
     main="",
     xlab="Z",
     ylab="Derivative",
     type="l",
     lwd=2)

rug(ivdata$z)

## End(Not run)

```

crssigtest

Regression Spline Significance Test with Mixed Data Types

Description

crssigtest implements a consistent test of significance of an explanatory variable in a nonparametric regression setting that is analogous to a simple t -test in a parametric regression setting. The test is based on Ma and Racine (2011).

Usage

```

crssigtest(model = NULL,
           index = NULL,
           boot = TRUE,
           boot.num = 399,
           boot.type = c("residual","reorder"),
           random.seed = 42)

```

Arguments

Data, Model Inputs And Formula Interface: These arguments identify the fitted model and tested indices.

<code>index</code>	a vector of indices for the columns of <code>model\$xz</code> for which the test of significance is to be conducted. Defaults to $(1, 2, \dots, p)$ where p is the number of columns in <code>model\$xz</code> .
<code>model</code>	a <code>crs</code> model object.

Bootstrap Controls: These arguments control bootstrap execution and reproducibility settings.

<code>boot</code>	a logical value (default TRUE) indicating whether to compute the bootstrap P-value or simply return the asymptotic P-value.
<code>boot.num</code>	an integer value specifying the number of bootstrap replications to use. Defaults to 399.
<code>boot.type</code>	whether to conduct ‘residual’ bootstrapping (<code>iid</code>) or permute (<code>reorder</code>) in place the predictor being tested when imposing the null.
<code>random.seed</code>	an integer used to seed R’s random number generator. This is to ensure replicability. Defaults to 42.

Value

`crssigtest` returns an object of type `sigtest`. [summary](#) supports `sigtest` objects. It has the following components:

<code>index</code>	the vector of indices input
<code>P</code>	the vector of bootstrap P-values for each statistic in <code>F</code>
<code>P.asy</code>	the vector of asymptotic P-values for each statistic in <code>index</code>
<code>F</code>	the vector of pseudo F-statistics <code>F</code>
<code>F.boot</code>	the matrix of bootstrapped pseudo F-statistics generated under the null (one column for each statistic in <code>F</code>)
<code>df1</code>	the vector of numerator degrees of freedom for each statistic in <code>F</code> (based on the smoother matrix)
<code>df2</code>	the vector of denominator degrees of freedom for each statistic in <code>F</code> (based on the smoother matrix)
<code>rss</code>	the vector of restricted sums of squared residuals for each statistic in <code>F</code>
<code>uss</code>	the vector of unrestricted sums of squared residuals for each statistic in <code>F</code>
<code>boot.num</code>	the number of bootstrap replications
<code>boot.type</code>	the <code>boot.type</code>
<code>xnames</code>	the names of the variables in <code>model\$xz</code>

Usage Issues

This function should be considered to be in ‘beta status’ until further notice.

Caution: bootstrap methods are, by their nature, *computationally intensive*. This can be frustrating for users possessing large datasets. For exploratory purposes, you may wish to override the default number of bootstrap replications, say, setting them to `boot.num=99`.

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

References

- Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.
- Ma, S. and J.S. Racine, (2011), “Inference for Regression Splines with Categorical and Continuous Predictors,” Working Paper.

Examples

```
## Not run:
options(crs.messages=FALSE)
set.seed(42)

n <- 1000
z <- rbinom(n,1,.5)
x1 <- rnorm(n)
x2 <- runif(n,-2,2)
z <- factor(z)
## z is irrelevant
y <- x1 + x2 + rnorm(n)

model <- crs(y~x1+x2+z,complexity="degree",segments=c(1,1))
summary(model)

model.sigtest <- crssigtest(model)
summary(model.sigtest)

## End(Not run)
```

crs_nomad_api

Native NOMAD C API for package authors

Description

crs exposes a native C interface to the embedded NOMAD solver for package authors who need to call NOMAD from compiled code without routing each black-box evaluation through the snomadr() R interface.

Details

The installed header ‘crs_nomad_native.h’ defines the final API surface: crs_nomad_problem, crs_nomad_result, crs_nomad_option, crs_nomad_eval_fn, and crs_nomad_solve. The callable is registered with R_RegisterCCallable() under the package name “crs” and symbol name “crs_nomad_solve”.

The final API supports two callback modes. `CRS_NOMAD_CALLBACK_C` calls a package-supplied C callback and is the preferred mode when the objective is available in compiled code. `CRS_NOMAD_CALLBACK_R` is an explicit bridge for routes whose objective still lives in R; callers pass a protected R function and environment through `crs_nomad_r_callback`.

This is the only native NOMAD C callable exported by **crs**; legacy v1/v2 development callables are intentionally not registered.

The callable itself must be invoked from the main R interpreter thread of the current process. It uses R's protection stack, unwind protection, and user interrupt machinery, and must not be called from native worker threads, signal handlers, or contexts that are not executing through R or a registered native call on that thread.

Quick Start For Package Authors

A package that calls the native API typically needs:

1. LinkingTo: `crs` in 'DESCRIPTION', so 'crs_nomad_native.h' is available at compile time.
2. A runtime dependency path that loads the **crs** namespace before calling `R_GetCCallable("crs", "crs_nomad_solve")`. If the route is mandatory, use `Imports: crs`; if it is optional, guard the route with an explicit `requireNamespace("crs")` check before entering the compiled solver path.
3. A callback that fills `m` black-box outputs and returns 0 on success.
4. A zero-initialized `crs_nomad_problem` and `crs_nomad_result`. Always set `api_version` and `struct_size`; this lets future **crs** versions reject incompatible structures cleanly.
5. Caller-owned buffers for `result.solution` and `result.outputs`. **crs** does not allocate result memory for the caller.

In ordinary package code, prefer explicit options over 'nomad.opt' files, and leave `read_nomad_opt_file = 0` unless reading a working directory file is a deliberate part of your user contract.

Minimal C Callback Skeleton

The following abridged skeleton shows the direct C-callback route used by packages whose objective is available in compiled code.

```
#include <R.h>
#include <Rinternals.h>
#include <R_ext/Rdynload.h>
#include <string.h>
#include <crs_nomad_native.h>

typedef struct {
    double target;
} my_eval_data;

static int my_eval(int n, const double *x, int m, double *bb, void *user_data)
{
    my_eval_data *data = (my_eval_data *) user_data;
```

```

    double value = 0.0;
    int i;

    if (m < 1)
        return 1;

    for (i = 0; i < n; ++i) {
        const double d = x[i] - data->target;
        value += d * d;
    }

    bb[0] = value;      /* one CRS_NOMAD_OUTPUT_OBJ output */
    return 0;
}

SEXP my_nomad_solve(SEXP x0_s)
{
    SEXP x0;
    crs_nomad_solve_fn solve;
    crs_nomad_problem problem;
    crs_nomad_result result;
    crs_nomad_option options[2];
    int input_type[2] = {CRS_NOMAD_INPUT_REAL, CRS_NOMAD_INPUT_REAL};
    int output_type[1] = {CRS_NOMAD_OUTPUT_OBJ};
    double lower[2] = {-1.0, -1.0};
    double upper[2] = { 1.0,  1.0};
    double solution[2] = {R_NaN, R_NaN};
    double outputs[1] = {R_NaN};
    my_eval_data data;
    int status;

    PROTECT(x0 = coerceVector(x0_s, REALSXP));
    if (XLENGTH(x0) != 2) {
        UNPROTECT(1);
        error("x0 must have length 2");
    }

    /*
     * Ensure the crs namespace is loaded before R_GetCCallable().
     * Many packages do this in an R wrapper via requireNamespace("crs").
     */
    solve = (crs_nomad_solve_fn)
        R_GetCCallable("crs", "crs_nomad_solve");
    if (solve == NULL) {
        UNPROTECT(1);
        error("failed to resolve crs_nomad_solve");
    }
}

```

```

options[0].name = "MAX_BB_EVAL";
options[0].value = "100";
options[1].name = "DISPLAY_DEGREE";
options[1].value = "0";

memset(&problem, 0, sizeof(problem));
problem.api_version = CRS_NOMAD_API_VERSION;
problem.struct_size = sizeof(problem);
problem.callback_mode = CRS_NOMAD_CALLBACK_C;
problem.n = 2;
problem.m = 1;
problem.x0 = REAL(x0);
problem.bb_input_type = input_type;
problem.bb_output_type = output_type;
problem.lower = lower;
problem.upper = upper;
problem.random_seed = 42;
problem.quiet = 1;
problem.option_count = 2;
problem.options = options;
problem.start_count = 1;
problem.read_nomad_opt_file = 0;

memset(&result, 0, sizeof(result));
result.api_version = CRS_NOMAD_API_VERSION;
result.struct_size = sizeof(result);
result.solution = solution;
result.solution_len = 2;
result.outputs = outputs;
result.outputs_len = 1;

data.target = 0.25;
status = solve(&problem, my_eval, &data, &result);

if (status != CRS_NOMAD_OK || result.status != CRS_NOMAD_OK) {
  UNPROTECT(1);
  error("NOMAD failed:");
}

UNPROTECT(1);
return ScalarReal(result.objective);
}

```

Real package code should validate all R inputs before constructing the borrowed pointer fields, should protect any R objects whose memory is used during the call, and should return the solution, objective, evaluation counts, and diagnostic message in an object shape appropriate for the package.

R Callback Bridge Skeleton

The R-callback bridge is useful when the search geometry is managed in compiled code but the black-box objective still lives in R. This is the pattern used by the **np** and **npRmpi** NOMAD routes while their objective functions remain in R.

```
SEXP my_nomad_solve_r_callback(SEXP eval_f, SEXP eval_env, SEXP x0_s)
{
  crs_nomad_solve_fn solve;
  crs_nomad_problem problem;
  crs_nomad_result result;
  crs_nomad_r_callback callback;
  crs_nomad_option options[1];
  int input_type[2] = {CRS_NOMAD_INPUT_REAL, CRS_NOMAD_INPUT_REAL};
  int output_type[1] = {CRS_NOMAD_OUTPUT_OBJ};
  double lower[2] = {-1.0, -1.0};
  double upper[2] = { 1.0,  1.0};
  double solution[2] = {R_NaN, R_NaN};
  double outputs[1] = {R_NaN};
  int status;

  PROTECT(eval_f);
  PROTECT(eval_env);
  PROTECT(x0_s = coerceVector(x0_s, REALSXP));
  if (XLENGTH(x0_s) != 2) {
    UNPROTECT(3);
    error("x0 must have length 2");
  }

  solve = (crs_nomad_solve_fn)
    R_GetCCallable("crs", "crs_nomad_solve");
  if (solve == NULL) {
    UNPROTECT(3);
    error("failed to resolve crs_nomad_solve");
  }

  options[0].name = "NB_THREADS_PARALLEL_EVAL";
  options[0].value = "1";

  memset(&callback, 0, sizeof(callback));
  callback.eval_f = (void *) eval_f;
  callback.environment = (void *) eval_env;

  memset(&problem, 0, sizeof(problem));
  problem.api_version = CRS_NOMAD_API_VERSION;
  problem.struct_size = sizeof(problem);
  problem.callback_mode = CRS_NOMAD_CALLBACK_R;
  problem.n = 2;
  problem.m = 1;
}
```

```

problem.x0 = REAL(x0_s);
problem.bb_input_type = input_type;
problem.bb_output_type = output_type;
problem.lower = lower;
problem.upper = upper;
problem.random_seed = 42;
problem.quiet = 1;
problem.option_count = 1;
problem.options = options;
problem.start_count = 1;

memset(&result, 0, sizeof(result));
result.api_version = CRS_NOMAD_API_VERSION;
result.struct_size = sizeof(result);
result.solution = solution;
result.solution_len = 2;
result.outputs = outputs;
result.outputs_len = 1;

status = solve(&problem, NULL, &callback, &result);

if (status != CRS_NOMAD_OK || result.status != CRS_NOMAD_OK) {
    UNPROTECT(3);
    error("NOMAD failed:");
}

UNPROTECT(3);
return ScalarReal(result.objective);
}

```

For R callbacks, `eval_f` must accept a numeric vector and return a numeric vector of length at least `problem.m`. Because this bridge calls back into R, do not request parallel black-box evaluation: `NB_THREADS_PARALLEL_EVAL > 1` is rejected.

Callback Contract

C callbacks return 0 on successful evaluation, fill exactly `problem->m` black-box outputs, and return nonzero on evaluation failure. They must not retain pointers supplied by **crs**.

R callback bridge users must ensure the R function and environment stored in `crs_nomad_r_callback` remain protected for the duration of `crs_nomad_solve`. R errors are trapped and reported as callback failures, with the first trapped R diagnostic retained in the fixed-capacity result message. R callbacks are main-thread callbacks; explicit `NB_THREADS_PARALLEL_EVAL > 1` is rejected for R callbacks, including when supplied through `'nomad.opt'`. Use the default or set it explicitly to 1.

Interrupt Semantics

Serial consumers normally leave interrupt ownership at `CRS_NOMAD_INTERRUPT_OWNERSHIP_DEFAULT`. The solve-scoped handler records only signal-safe state; NOMAD consumes it from ordinary solver

control flow, performs native cleanup, restores the previous process handler, and returns CRS_NOMAD_INTERRUPTED. If a callback failure occurs before the same solve is interrupted, the interrupt status takes precedence.

Lockstep consumers such as MPI-backed packages may use CRS_NOMAD_INTERRUPT_OWNERSHIP_EXTERNAL. They must own protected polling and defer any R unwind until a rank-common synchronization boundary. This option does not add synchronization, messages, or payloads inside **crs**.

Numeric And Option Semantics

`random_seed > 0` sets NOMAD's SEED. When **crs** generates multi-start points, the same seed also controls that start generation; `random_seed = 0` uses time-varying generated starts, matching the historical `snomadr()` posture.

Bounds may use `-Inf` and `Inf` for unbounded coordinates. NaN is invalid in starting points and bounds. The embedded NOMAD4 C interface represents categorical inputs through its integer input type, so callers remain responsible for category encoding and decoding.

NaN black-box outputs are callback failures for all output types. Infinite objective outputs are coerced to `+/-DBL_MAX`, matching `snomadr()` compatibility. Infinite PB or EB constraint outputs are passed through to NOMAD as infeasibility signals.

The native API currently requires NOMAD's evaluation cache (`EVAL_USE_CACHE = TRUE`, the NOMAD default), because result recovery is based on NOMAD's cache-backed incumbent state. Supplying `EVAL_USE_CACHE = FALSE` to `crs_nomad_solve` is rejected early with a clear error. The public R-level `snomadr()` interface retains its separate cache option behavior.

Ownership

The problem structure and all pointer fields are borrowed and read-only. The caller owns `result->solution` and `result->outputs`; **crs** does not return memory that callers must free.

Common Result Fields

The most useful result fields for package authors are:

- `status`: **crs** API status, where `CRS_NOMAD_OK` indicates that the native wrapper completed.
- `nomad_run_flag`: NOMAD's native run flag.
- `objective`: best objective reported by NOMAD.
- `solution`: caller-owned solution buffer filled by **crs**.
- `outputs`: caller-owned output buffer for the best solution.
- `callback_evaluations`: number of true package callback evaluations.
- `total_evaluations` and `cache_hits`: NOMAD point lookups and native cache hits. These are lookup diagnostics; cache hits are not expensive objective recomputations.
- `message`: diagnostic text when the wrapper reports a failure.

See Also

[snomadr](#)

Engel95

1995 British Family Expenditure Survey

Description

British cross-section data consisting of a random sample taken from the British Family Expenditure Survey for 1995. The households consist of married couples with an employed head-of-household between the ages of 25 and 55 years. There are 1655 household-level observations in total.

Usage

```
data("Engel95")
```

Format

A data frame with 10 columns, and 1655 rows.

food expenditure share on food, of type numeric

catering expenditure share on catering, of type numeric

alcohol expenditure share on alcohol, of type numeric

fuel expenditure share on fuel, of type numeric

motor expenditure share on motor, of type numeric

fares expenditure share on fares, of type numeric

leisure expenditure share on leisure, of type numeric

logexp logarithm of total expenditure, of type numeric

logwages logarithm of total earnings, of type numeric

nkids number of children, of type numeric

Source

Richard Blundell and Dennis Kristensen

References

Blundell, R. and X. Chen and D. Kristensen (2007), "Semi-Nonparametric IV Estimation of Shape-Invariant Engel Curves," *Econometrica*, 75, 1613-1669.

Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.

Examples

```
## Not run:
## Example - we compute nonparametric instrumental regression of an
## Engel curve for food expenditure shares using Landweber-Fridman
## iteration of Fredholm integral equations of the first kind.

## We consider an equation with an endogenous predictor ('z') and an
## instrument ('w'). Let  $y = \phi(z) + u$  where  $\phi(z)$  is the function of
## interest. Here  $E(u|z)$  is not zero hence the conditional mean  $E(y|z)$ 
## does not coincide with the function of interest, but if there exists
## an instrument  $w$  such that  $E(u|w) = 0$ , then we can recover the
## function of interest by solving an ill-posed inverse problem.

data(Engel95)

## Sort on logexp (the endogenous predictor) for plotting purposes
## (i.e. so we can plot a curve for the fitted values versus logexp)

Engel95 <- Engel95[order(Engel95$logexp),]

attach(Engel95)

## Setting cv.threshold = 0 forces NOMAD search instead of exhaustive search
## when no categorical predictors are present. This avoids unnecessary
## evaluation of all degree/segment combinations in the examples and, for
## crsiv() and crsivderiv(), ensures that the warm-start strategy is used.
model.iv <- crsiv(y=food,z=logexp,w=logwages,method="Landweber-Fridman",cv.threshold=0)
phihat <- model.iv$phi

## Compute the non-IV regression (i.e. regress y on z)

## Setting cv.threshold = 0 forces NOMAD search instead of exhaustive search
## when no categorical predictors are present. This avoids unnecessary
## evaluation of all degree/segment combinations in the examples and, for
## crsiv() and crsivderiv(), ensures that the warm-start strategy is used.
ghat <- crs(food~logexp,cv.threshold=0)

## For the plots, we restrict focal attention to the bulk of the data
## (i.e. for the plotting area trim out 1/4 of one percent from each
## tail of y and z). This is often helpful as estimates in the tails of
## the support are less reliable (i.e. more variable) so we are
## interested in examining the relationship 'where the action is'.

trim <- 0.0025

plot(logexp,food,
      ylab="Food Budget Share",
      xlab="log(Total Expenditure)",
      xlim=quantile(logexp,c(trim,1-trim)),
      ylim=quantile(food,c(trim,1-trim)),
      main="Nonparametric Instrumental Regression Splines",
      type="p",
```

```

      cex=.5,
      col="lightgrey")

lines(logexp, phihat, col="blue", lwd=2, lty=2)

lines(logexp, fitted(ghat), col="red", lwd=2, lty=4)

legend(quantile(logexp, trim), quantile(food, 1-trim),
      c(expression(paste("Nonparametric IV: ", hat(varphi)(logexp))),
        "Nonparametric Regression: E(food | logexp)"),
      lty=c(2, 4),
      col=c("blue", "red"),
      lwd=c(2, 2),
      bty="n")

## End(Not run)

```

frscv

Categorical Factor Regression Spline Cross-Validation

Description

frscv computes exhaustive cross-validation directed search for a regression spline estimate of a one (1) dimensional dependent variable on an r -dimensional vector of continuous predictors and nominal/ordinal ([factor/ordered](#)) predictors.

Usage

```

frscv(xz,
      y,
      basis = c("additive", "tensor", "glp", "auto"),
      complexity = c("degree-knots", "degree", "knots"),
      cv.func = c("cv.ls", "cv.gcv", "cv.aic"),
      degree = degree,
      degree.max = 10,
      degree.min = 0,
      display.nomad.progress = TRUE,
      display.warnings = TRUE,
      knots = c("quantiles", "uniform", "auto"),
      segments = segments,
      segments.max = 10,
      segments.min = 1,
      singular.ok = FALSE,
      tau = NULL,
      weights = NULL)

```

Arguments

Data, Model Inputs And Formula Interface: These arguments identify explicit data inputs for exhaustive spline search.

xz	continuous and/or nominal/ordinal (factor/ordered) predictors
y	continuous univariate vector

Basis And Spline Complexity: These arguments control basis type and spline complexity.

basis	a character string (default basis="additive") indicating whether the additive or tensor product B-spline basis matrix for a multivariate polynomial spline or generalized B-spline polynomial basis should be used. Note this can be automatically determined by cross-validation if cv=TRUE and basis="auto", and is an 'all or none' proposition (i.e. interaction terms for all predictors or for no predictors given the nature of 'tensor products'). Note also that if there is only one predictor this defaults to basis="additive" to avoid unnecessary computation as the spline bases are equivalent in this case
complexity	a character string (default complexity="degree-knots") indicating whether model 'complexity' is determined by the degree of the spline or by the number of segments ('knots'). This option allows the user to use cross-validation to select either the spline degree (number of knots held fixed) or the number of knots (spline degree held fixed) or both the spline degree and number of knots
degree	integer/vector specifying the degree of the B-spline basis for each dimension of the continuous x
degree.max	the maximum degree of the B-spline basis for each of the continuous predictors (default degree.max=10)
degree.min	the minimum degree of the B-spline basis for each of the continuous predictors (default degree.min=0)
knots	a character string (default knots="quantiles") specifying where knots are to be placed. 'quantiles' specifies knots placed at equally spaced quantiles (equal number of observations lie in each segment) and 'uniform' specifies knots placed at equally spaced intervals. If knots="auto", the knot type will be automatically determined by cross-validation
segments	integer/vector specifying the number of segments of the B-spline basis for each dimension of the continuous x (i.e. number of knots minus one)
segments.max	the maximum segments of the B-spline basis for each of the continuous predictors (default segments.max=10)
segments.min	the minimum segments of the B-spline basis for each of the continuous predictors (default segments.min=1)

Exhaustive Search Controls: These arguments control exhaustive cross-validation search.

cv.func	a character string (default cv.func="cv.ls") indicating which method to use to select smoothing parameters. cv.gcv specifies generalized cross-validation (Craven and Wahba (1979)), cv.aic specifies expected Kullback-Leibler cross-validation (Hurvich, Simonoff, and Tsai (1998)), and cv.ls specifies least-squares cross-validation
---------	---

`singular.ok` a logical value (default `singular.ok=FALSE`) that, when `FALSE`, discards singular bases during cross-validation (a check for ill-conditioned bases is performed).

Quantile And Weights: These arguments control quantile level and observation weights.

`tau` if non-null a number in (0,1) denoting the quantile for which a quantile regression spline is to be estimated rather than estimating the conditional mean (default `tau=NULL`)

`weights` an optional vector of weights to be used in the fitting process. Should be 'NULL' or a numeric vector. If non-NULL, weighted least squares is used with weights 'weights' (that is, minimizing $\sum(w \cdot e^2)$); otherwise ordinary least squares is used.

Warnings And Progress: These arguments control warnings and displayed optimizer progress.

`display.nomad.progress`

a logical value indicating whether to display NOMAD progress (default `display.nomad.progress=TRUE`)

`display.warnings`

a logical value indicating whether to display warnings (default `display.warnings=TRUE`)

Details

`frscv` computes exhaustive cross-validation for a regression spline estimate of a one (1) dimensional dependent variable on an r -dimensional vector of continuous and nominal/ordinal ([factor/ordered](#)) predictors. The optimal K/I combination (i.e. degree/segments/I) is returned along with other results (see below for return values).

For the continuous predictors the regression spline model employs either the additive or tensor product B-spline basis matrix for a multivariate polynomial spline via the B-spline routines in the GNU Scientific Library (<https://www.gnu.org/software/gsl/>) and the [tensor.prod.model.matrix](#) function.

For the nominal/ordinal ([factor/ordered](#)) predictors the regression spline model uses indicator basis functions.

Value

`frscv` returns a `crscv` object. Furthermore, the function [summary](#) supports objects of this type. The returned objects have the following components:

<code>K</code>	scalar/vector containing optimal degree(s) of spline or number of segments
<code>I</code>	scalar/vector containing an indicator of whether the predictor is included or not for each dimension of the nominal/ordinal (factor/ordered) predictors
<code>K.mat</code>	vector/matrix of values of K evaluated during search
<code>cv.func</code>	objective function value at optimum
<code>cv.func.vec</code>	vector of objective function values at each degree of spline or number of segments in <code>K.mat</code>

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

References

Craven, P. and G. Wahba (1979), “Smoothing Noisy Data With Spline Functions,” *Numerische Mathematik*, 13, 377-403.

Hurvich, C.M. and J.S. Simonoff and C.L. Tsai (1998), “Smoothing Parameter Selection in Non-parametric Regression Using an Improved Akaike Information Criterion,” *Journal of the Royal Statistical Society B*, 60, 271-293.

Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.

Ma, S. and J.S. Racine and L. Yang (2015), “Spline Regression in the Presence of Categorical Predictors,” *Journal of Applied Econometrics*, Volume 30, 705-717.

Ma, S. and J.S. Racine (2013), “Additive Regression Splines with Irrelevant Categorical and Continuous Regressors,” *Statistica Sinica*, Volume 23, 515-541.

See Also

[loess](#), [npregbw](#)

Examples

```
set.seed(42)
## Simulated data
n <- 1000

x <- runif(n)
z <- round(runif(n,min=-0.5,max=1.5))
z.unique <- uniquecombs(as.matrix(z))
ind <- attr(z.unique,"index")
ind.vals <- sort(unique(ind))
dgp <- numeric(length=n)
for(i in 1:nrow(z.unique)) {
  zz <- ind == ind.vals[i]
  dgp[zz] <- z[zz]+cos(2*pi*x[zz])
}

y <- dgp + rnorm(n,sd=.1)

xdata <- data.frame(x,z=factor(z))

## Compute the optimal K and I, determine optimal number of knots, set
## spline degree for x to 3

cv <- frscv(x=xdata,y=y,complexity="knots",degree=c(3))
summary(cv)
```

frscvNOMAD

*Categorical Factor Regression Spline Cross-Validation***Description**

frscvNOMAD computes NOMAD-based (Nonsmooth Optimization by Mesh Adaptive Direct Search, Abramson, Audet, Couture and Le Digabel (2011)) cross-validation directed search for a regression spline estimate of a one (1) dimensional dependent variable on an r -dimensional vector of continuous predictors and nominal/ordinal ([factor/ordered](#)) predictors.

Usage

```
frscvNOMAD(xz,
  y,
  basis = c("additive", "tensor", "glp", "auto"),
  complexity = c("degree-knots", "degree", "knots"),
  cv.df.min = 1,
  cv.func = c("cv.ls", "cv.gcv", "cv.aic"),
  degree = degree,
  degree.max = 10,
  degree.min = 0,
  display.nomad.progress = TRUE,
  display.warnings = TRUE,
  include = include,
  initial.mesh.size.integer = "1",
  knots = c("quantiles", "uniform", "auto"),
  max.bb.eval = 10000,
  max.eval = NULL,
  min.mesh.size.integer = "1",
  min.frame.size.integer = "1",
  nmulti = 0,
  opts=list(),
  random.seed = 42,
  segments = segments,
  segments.max = 10,
  segments.min = 1,
  singular.ok = FALSE,
  tau = NULL,
  weights = NULL)
```

Arguments

Data, Model Inputs And Formula Interface: These arguments identify explicit data inputs for NOMAD spline search.

xz	continuous and/or nominal/ordinal (factor/ordered) predictors
y	continuous univariate vector

Basis And Spline Complexity: These arguments control basis type and spline complexity.

basis	a character string (default basis="additive") indicating whether the additive or tensor product B-spline basis matrix for a multivariate polynomial spline or generalized B-spline polynomial basis should be used. Note this can be automatically determined by cross-validation if cv=TRUE and basis="auto", and is an 'all or none' proposition (i.e. interaction terms for all predictors or for no predictors given the nature of 'tensor products'). Note also that if there is only one predictor this defaults to basis="additive" to avoid unnecessary computation as the spline bases are equivalent in this case
complexity	a character string (default complexity="degree-knots") indicating whether model 'complexity' is determined by the degree of the spline or by the number of segments ('knots'). This option allows the user to use cross-validation to select either the spline degree (number of knots held fixed) or the number of knots (spline degree held fixed) or both the spline degree and number of knots
degree	integer/vector specifying the degree of the B-spline basis for each dimension of the continuous x
degree.max	the maximum degree of the B-spline basis for each of the continuous predictors (default degree.max=10)
degree.min	the minimum degree of the B-spline basis for each of the continuous predictors (default degree.min=0)
knots	a character string (default knots="quantiles") specifying where knots are to be placed. 'quantiles' specifies knots placed at equally spaced quantiles (equal number of observations lie in each segment) and 'uniform' specifies knots placed at equally spaced intervals. If knots="auto", the knot type will be automatically determined by cross-validation
segments	integer/vector specifying the number of segments of the B-spline basis for each dimension of the continuous x (i.e. number of knots minus one)
segments.max	the maximum segments of the B-spline basis for each of the continuous predictors (default segments.max=10)
segments.min	the minimum segments of the B-spline basis for each of the continuous predictors (default segments.min=1)

Factor Inclusion Controls: These arguments control factor inclusion during search.

include	integer/vector for the categorical predictors. If it is not NULL, it will be the initial value for the fitting
---------	--

NOMAD Search Controls: These arguments control NOMAD search, cross-validation objective selection, and restart behavior.

cv.df.min	the minimum degrees of freedom to allow when conducting cross-validation (default cv.df.min=1)
cv.func	a character string (default cv.func="cv.ls") indicating which method to use to select smoothing parameters. cv.gcv specifies generalized cross-validation (Craven and Wahba (1979)), cv.aic specifies expected Kullback-Leibler cross-validation (Hurvich, Simonoff, and Tsai (1998)), and cv.ls specifies least-squares cross-validation

<code>initial.mesh.size.integer</code>	argument passed to the NOMAD solver (see snomadr for further details)
<code>max.bb.eval</code>	argument passed to the NOMAD solver (default 10000). This default was set on the basis of simulation evidence and real-world applications for the continuous-only frscvNOMAD search route; see snomadr for further details.
<code>max.eval</code>	optional NOMAD total point-lookup budget. This is distinct from <code>max.bb.eval</code> : <code>max.bb.eval</code> limits true blackbox objective computations, while <code>max.eval</code> limits total NOMAD point lookups, including cache hits. The default NULL uses <code>MAX_EVAL = 1000</code> for this continuous-only NOMAD route. If supplied, <code>max.eval</code> is passed to NOMAD as <code>MAX_EVAL</code> . Supplying both <code>max.eval</code> and <code>opts\$MAX_EVAL</code> with conflicting values is an error. Use an explicit <code>max.eval</code> or <code>opts\$MAX_EVAL</code> to request a larger point-lookup budget.
<code>min.frame.size.integer</code>	arguments passed to the NOMAD solver (see snomadr for further details)
<code>min.mesh.size.integer</code>	arguments passed to the NOMAD solver (see snomadr for further details)
<code>nmulti</code>	integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points (default <code>nmulti=0</code>)
<code>opts</code>	list of optional arguments to be passed to snomadr . If not user-specified, this function applies the NOMAD4 path defaults <code>QUAD_MODEL_SEARCH="no"</code> , <code>EVAL_QUEUE_SORT="DIR_LAST"</code> , <code>SIMPLE_LINE_SEARCH="yes"</code> , and <code>SPECULATIVE_SEARCH="no"</code> , and <code>DIRECTION_TYPE="ORTHO N+1 NEG"</code> for faster mixed-integer search in this specific frscvNOMAD path. User-supplied <code>opts</code> entries always take precedence.
<code>random.seed</code>	when it is not missing and not equal to 0, the initial points will be generated using this seed when <code>nmulti > 0</code>
<code>singular.ok</code>	a logical value (default <code>singular.ok=FALSE</code>) that, when FALSE, discards singular bases during cross-validation (a check for ill-conditioned bases is performed).

Quantile And Weights: These arguments control quantile level and observation weights.

<code>tau</code>	if non-null a number in (0,1) denoting the quantile for which a quantile regression spline is to be estimated rather than estimating the conditional mean (default <code>tau=NULL</code>)
<code>weights</code>	an optional vector of weights to be used in the fitting process. Should be 'NULL' or a numeric vector. If non-NULL, weighted least squares is used with weights 'weights' (that is, minimizing 'sum(w*e^2)'); otherwise ordinary least squares is used.

Warnings And Progress: These arguments control warnings and displayed optimizer progress.

<code>display.nomad.progress</code>	a logical value indicating whether to display the progress of the NOMAD solver (default <code>display.nomad.progress=TRUE</code>)
<code>display.warnings</code>	a logical value indicating whether to display warnings (default <code>display.warnings=TRUE</code>)

Details

frscvNOMAD computes NOMAD-based cross-validation for a regression spline estimate of a one (1) dimensional dependent variable on an r -dimensional vector of continuous and nominal/ordinal (**factor/ordered**) predictors. Numerical search for the optimal degree/segments/I is undertaken using **snomadr**.

The optimal K/I combination is returned along with other results (see below for return values).

For the continuous predictors the regression spline model employs either the additive or tensor product B-spline basis matrix for a multivariate polynomial spline via the B-spline routines in the GNU Scientific Library (<https://www.gnu.org/software/gsl/>) and the **tensor.prod.model.matrix** function.

For the nominal/ordinal (**factor/ordered**) predictors the regression spline model uses indicator basis functions.

Value

frscvNOMAD returns a crscv object. Furthermore, the function **summary** supports objects of this type. The returned objects have the following components:

K	scalar/vector containing optimal degree(s) of spline or number of segments
I	scalar/vector containing an indicator of whether the predictor is included or not for each dimension of the nominal/ordinal (factor/ordered) predictors
K.mat	vector/matrix of values of K evaluated during search
degree.max	the maximum degree of the B-spline basis for each of the continuous predictors (default degree.max=10)
segments.max	the maximum segments of the B-spline basis for each of the continuous predictors (default segments.max=10)
degree.min	the minimum degree of the B-spline basis for each of the continuous predictors (default degree.min=0)
segments.min	the minimum segments of the B-spline basis for each of the continuous predictors (default segments.min=1)
cv.func	objective function value at optimum
cv.func.vec	vector of objective function values at each degree of spline or number of segments in K.mat

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca> and Zhenghua Nie <niez@mcmaster.ca>

References

- Abramson, M.A. and C. Audet and G. Couture and J.E. Dennis Jr. and S. Le Digabel (2011), “The NOMAD project”. Software available at <https://www.gerad.ca/nomad>.
- Craven, P. and G. Wahba (1979), “Smoothing Noisy Data With Spline Functions,” *Numerische Mathematik*, 13, 377-403.

Hurvich, C.M. and J.S. Simonoff and C.L. Tsai (1998), “Smoothing Parameter Selection in Nonparametric Regression Using an Improved Akaike Information Criterion,” *Journal of the Royal Statistical Society B*, 60, 271-293.

Le Digabel, S. (2011), “Algorithm 909: NOMAD: Nonlinear Optimization With the MADS Algorithm”. *ACM Transactions on Mathematical Software*, 37(4):44:1-44:15.

Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.

Ma, S. and J.S. Racine and L. Yang (2015), “Spline Regression in the Presence of Categorical Predictors,” *Journal of Applied Econometrics*, Volume 30, 705-717.

Ma, S. and J.S. Racine (2013), “Additive Regression Splines with Irrelevant Categorical and Continuous Regressors,” *Statistica Sinica*, Volume 23, 515-541.

See Also

[loess](#), [npregbw](#)

Examples

```
set.seed(42)
## Simulated data
n <- 1000

x <- runif(n)
z <- round(runif(n,min=-0.5,max=1.5))
z.unique <- uniquecombs(as.matrix(z))
ind <- attr(z.unique,"index")
ind.vals <- sort(unique(ind))
dgp <- numeric(length=n)
for(i in 1:nrow(z.unique)) {
  zz <- ind == ind.vals[i]
  dgp[zz] <- z[zz]+cos(2*pi*x[zz])
}

y <- dgp + rnorm(n,sd=.1)

xdata <- data.frame(x,z=factor(z))

## Compute the optimal K and I, determine optimal number of knots, set
## spline degree for x to 3

cv <- frscvNOMAD(x=xdata,y=y,complexity="knots",degree=c(3),segments=c(5))
summary(cv)
```

Description

Produce model matrices for a generalized polynomial smooth from the model matrices for the marginal bases of the smooth.

Usage

```
glp.model.matrix(X)
```

Arguments

`X` a list of model matrices for the marginal bases of a smooth

Details

This function computes a generalized polynomial where the orders of each term entering the polynomial may vary.

Value

A model matrix for a generalized polynomial smooth.

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

References

Hall, P. and J.S. Racine (2015), “Infinite Order Cross-Validated Local Polynomial Regression,” *Journal of Econometrics*, 185, 510-525.

Examples

```
X <- list(matrix(1:4,2,2),matrix(5:10,2,3))
glp.model.matrix(X)
```

Description

`gsl.bs` generates the B-spline basis matrix for a polynomial spline and (optionally) the B-spline basis matrix derivative of a specified order with respect to each predictor

Usage

```
gsl.bs(..., display.warnings = TRUE)
## Default S3 method:
gsl.bs(x,
       degree = 3,
       nbreak = 2,
       deriv = 0,
       x.min = NULL,
       x.max = NULL,
       intercept = FALSE,
       knots = NULL,
       display.warnings = TRUE,
       ...)
```

Arguments

Data, Model Inputs And Formula Interface: This argument identifies the predictor variable for spline-basis construction.

`x` the predictor variable. Missing values are not allowed

Spline Basis Controls: These arguments control spline degree, derivative order, intercept inclusion, knots, and support bounds.

<code>degree</code>	degree of the piecewise polynomial - default is '3' (cubic spline)
<code>deriv</code>	the order of the derivative to be computed-default if 0
<code>intercept</code>	if 'TRUE', an intercept is included in the basis; default is 'FALSE'
<code>knots</code>	a vector (default knots="NULL") specifying knots for the spline basis (default enables uniform knots, otherwise those provided are used)
<code>nbreak</code>	number of breaks in each interval - default is '2'
<code>x.max</code>	the upper bound on which to construct the spline - defaults to <code>max(x)</code>
<code>x.min</code>	the lower bound on which to construct the spline - defaults to <code>min(x)</code>

Warnings And Progress: This argument controls warnings.

`display.warnings`
a logical value indicating whether to display warnings (default `display.warnings=TRUE`)

Additional Arguments: Further optional arguments are passed through to lower-level routines.

`...` optional arguments

Details

Typical usages are (see below for a list of options and also the examples at the end of this help file)

```
B <- gsl.bs(x, degree=10)
B.predict <- predict(gsl.bs(x, degree=10), newx=xeval)
```

Value

`gsl.bs` returns a `gsl.bs` object. A matrix of dimension `'c(length(x), degree+nbreak-1)'`. The generic function `predict` extracts (or generates) predictions from the returned object.

A primary use is in modelling formulas to directly specify a piecewise polynomial term in a model. See <https://www.gnu.org/software/gsl/> for further details.

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

References

Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.

Ma, S. and J.S. Racine and L. Yang (2015), "Spline Regression in the Presence of Categorical Predictors," *Journal of Applied Econometrics*, Volume 30, 705-717.

Ma, S. and J.S. Racine (2013), "Additive Regression Splines with Irrelevant Categorical and Continuous Regressors," *Statistica Sinica*, Volume 23, 515-541.

See Also

`bs`

Examples

```
## Plot the spline bases and their first order derivatives
x <- seq(0,1,length=100)
matplot(x,gsl.bs(x,degree=5),type="l")
matplot(x,gsl.bs(x,degree=5,deriv=1),type="l")

## Regression example
n <- 1000
x <- sort(runif(n))
y <- cos(2*pi*x) + rnorm(n,sd=.25)
B <- gsl.bs(x,degree=5,intercept=FALSE)
plot(x,y,cex=.5,col="grey")
lines(x,fitted(lm(y~B)))
```

Description

`krscv` computes exhaustive cross-validation directed search for a regression spline estimate of a one (1) dimensional dependent variable on an `r`-dimensional vector of continuous and nominal/ordinal (`factor/ordered`) predictors.

Usage

```
krscv(xz,
      y,
      basis = c("additive", "tensor", "glp", "auto"),
      complexity = c("degree-knots", "degree", "knots"),
      cv.func = c("cv.ls", "cv.gcv", "cv.aic"),
      degree = degree,
      degree.max = 10,
      degree.min = 0,
      display.nomad.progress = TRUE,
      display.warnings = TRUE,
      knots = c("quantiles", "uniform", "auto"),
      restarts = 0,
      segments = segments,
      segments.max = 10,
      segments.min = 1,
      singular.ok = FALSE,
      tau = NULL,
      weights = NULL)
```

Arguments

Data, Model Inputs And Formula Interface: These arguments identify explicit data inputs for exhaustive kernel/spline search.

xz	continuous and/or nominal/ordinal (factor/ordered) predictors
y	continuous univariate vector

Basis And Spline Complexity: These arguments control basis type and spline complexity.

basis	a character string (default basis="additive") indicating whether the additive or tensor product B-spline basis matrix for a multivariate polynomial spline or generalized B-spline polynomial basis should be used. Note this can be automatically determined by cross-validation if cv=TRUE and basis="auto", and is an 'all or none' proposition (i.e. interaction terms for all predictors or for no predictors given the nature of 'tensor products'). Note also that if there is only one predictor this defaults to basis="additive" to avoid unnecessary computation as the spline bases are equivalent in this case
complexity	a character string (default complexity="degree-knots") indicating whether model 'complexity' is determined by the degree of the spline or by the number of segments ('knots'). This option allows the user to use cross-validation to select either the spline degree (number of knots held fixed) or the number of knots (spline degree held fixed) or both the spline degree and number of knots
degree	integer/vector specifying the degree of the B-spline basis for each dimension of the continuous x
degree.max	the maximum degree of the B-spline basis for each of the continuous predictors (default degree.max=10)

<code>degree.min</code>	the minimum degree of the B-spline basis for each of the continuous predictors (default <code>degree.min=0</code>)
<code>knots</code>	a character string (default <code>knots="quantiles"</code>) specifying where knots are to be placed. 'quantiles' specifies knots placed at equally spaced quantiles (equal number of observations lie in each segment) and 'uniform' specifies knots placed at equally spaced intervals. If <code>knots="auto"</code> , the knot type will be automatically determined by cross-validation
<code>segments</code>	integer/vector specifying the number of segments of the B-spline basis for each dimension of the continuous x (i.e. number of knots minus one)
<code>segments.max</code>	the maximum segments of the B-spline basis for each of the continuous predictors (default <code>segments.max=10</code>)
<code>segments.min</code>	the minimum segments of the B-spline basis for each of the continuous predictors (default <code>segments.min=1</code>)

Exhaustive Search And Kernel Bandwidth Controls: These arguments control exhaustive cross-validation search and kernel-bandwidth restarts.

<code>cv.func</code>	a character string (default <code>cv.func="cv.ls"</code>) indicating which method to use to select smoothing parameters. <code>cv.gcv</code> specifies generalized cross-validation (Craven and Wahba (1979)), <code>cv.aic</code> specifies expected Kullback-Leibler cross-validation (Hurvich, Simonoff, and Tsai (1998)), and <code>cv.ls</code> specifies least-squares cross-validation
<code>restarts</code>	number of times to restart <code>optim</code> from different initial random values (default <code>restarts=0</code>) when searching for optimal bandwidths for the categorical predictors for each unique K combination (i.e. \ degree/segments)
<code>singular.ok</code>	a logical value (default <code>singular.ok=FALSE</code>) that, when <code>FALSE</code> , discards singular bases during cross-validation (a check for ill-conditioned bases is performed).

Quantile And Weights: These arguments control quantile level and observation weights.

<code>tau</code>	if non-null a number in (0,1) denoting the quantile for which a quantile regression spline is to be estimated rather than estimating the conditional mean (default <code>tau=NULL</code>)
<code>weights</code>	an optional vector of weights to be used in the fitting process. Should be 'NULL' or a numeric vector. If non-NULL, weighted least squares is used with weights 'weights' (that is, minimizing ' $\sum(w \cdot e^2)$ '); otherwise ordinary least squares is used.

Warnings And Progress: These arguments control warnings and displayed optimizer progress.

<code>display.nomad.progress</code>	a logical value indicating whether to display NOMAD progress (default <code>display.nomad.progress=TRUE</code>)
<code>display.warnings</code>	a logical value indicating whether to display warnings (default <code>display.warnings=TRUE</code>)

Details

krscv computes exhaustive cross-validation for a regression spline estimate of a one (1) dimensional dependent variable on an r -dimensional vector of continuous and nominal/ordinal ([factor/ordered](#)) predictors. The optimal K/λ combination is returned along with other results (see below for return values). The method uses kernel functions appropriate for categorical (ordinal/nominal) predictors which avoids the loss in efficiency associated with sample-splitting procedures that are typically used when faced with a mix of continuous and nominal/ordinal ([factor/ordered](#)) predictors.

For the continuous predictors the regression spline model employs either the additive or tensor product B-spline basis matrix for a multivariate polynomial spline via the B-spline routines in the GNU Scientific Library (<https://www.gnu.org/software/gsl/>) and the [tensor.prod.model.matrix](#) function.

For the discrete predictors the product kernel function is of the ‘Li-Racine’ type (see Li and Racine (2007) for details).

For each unique combination of degree and segment, numerical search for the bandwidth vector λ is undertaken using [optim](#) and the box-constrained L-BFGS-B method (see [optim](#) for details). The user may restart the [optim](#) algorithm as many times as desired via the `restarts` argument. The approach ascends from $K=0$ through `degree.max/segments.max` and for each value of K searches for the optimal bandwidths for this value of K . After the most complex model has been searched then the optimal K/λ combination is selected. If any element of the optimal K vector coincides with `degree.max/segments.max` a warning is produced and the user ought to restart their search with a larger value of `degree.max/segments.max`.

Value

krscv returns a `krscv` object. Furthermore, the function [summary](#) supports objects of this type. The returned objects have the following components:

<code>K</code>	scalar/vector containing optimal degree(s) of spline or number of segments
<code>K.mat</code>	vector/matrix of values of K evaluated during search
<code>restarts</code>	number of restarts during search, if any
<code>lambda</code>	optimal bandwidths for categorical predictors
<code>lambda.mat</code>	vector/matrix of optimal bandwidths for each degree of spline
<code>cv.func</code>	objective function value at optimum
<code>cv.func.vec</code>	vector of objective function values at each degree of spline or number of segments in <code>K.mat</code>

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

References

Craven, P. and G. Wahba (1979), “Smoothing Noisy Data With Spline Functions,” *Numerische Mathematik*, 13, 377-403.

Hurvich, C.M. and J.S. Simonoff and C.L. Tsai (1998), “Smoothing Parameter Selection in Nonparametric Regression Using an Improved Akaike Information Criterion,” *Journal of the Royal Statistical Society B*, 60, 271-293.

Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.

Ma, S. and J.S. Racine and L. Yang (2015), “Spline Regression in the Presence of Categorical Predictors,” *Journal of Applied Econometrics*, Volume 30, 705-717.

Ma, S. and J.S. Racine (2013), “Additive Regression Splines with Irrelevant Categorical and Continuous Regressors,” *Statistica Sinica*, Volume 23, 515-541.

See Also

[loess](#), [npregbw](#)

Examples

```
set.seed(42)
## Simulated data
n <- 1000

x <- runif(n)
z <- round(runif(n,min=-0.5,max=1.5))
z.unique <- uniquecombs(as.matrix(z))
ind <- attr(z.unique,"index")
ind.vals <- sort(unique(ind))
dgp <- numeric(length=n)
for(i in 1:nrow(z.unique)) {
  zz <- ind == ind.vals[i]
  dgp[zz] <- z[zz]+cos(2*pi*x[zz])
}
y <- dgp + rnorm(n,sd=.1)

xdata <- data.frame(x,z=factor(z))

## Compute the optimal K and lambda, determine optimal number of knots, set
## spline degree for x to 3

cv <- krscv(x=xdata,y=y,complexity="knots",degree=c(3))
summary(cv)
```

Description

krscvNOMAD computes NOMAD-based (Nonsmooth Optimization by Mesh Adaptive Direct Search, Abramson, Audet, Couture and Le Digabel (2011)) cross-validation directed search for a regression spline estimate of a one (1) dimensional dependent variable on an r -dimensional vector of continuous and nominal/ordinal ([factor/ordered](#)) predictors.

Usage

```
krscvNOMAD(xz,
           y,
           basis = c("additive", "tensor", "glp", "auto"),
           complexity = c("degree-knots", "degree", "knots"),
           cv.df.min = 1,
           cv.func = c("cv.ls", "cv.gcv", "cv.aic"),
           degree = degree,
           degree.max = 10,
           degree.min = 0,
           display.nomad.progress = TRUE,
           display.warnings = TRUE,
           initial.mesh.size.integer = "1",
           initial.mesh.size.real = "r0.1",
           knots = c("quantiles", "uniform", "auto"),
           lambda = lambda,
           lambda.discrete = FALSE,
           lambda.discrete.num = 100,
           max.bb.eval = 1000,
           max.eval = NULL,
           min.mesh.size.integer = "1",
           min.mesh.size.real = paste("r", sqrt(.Machine$double.eps), sep=""),
           min.frame.size.integer = "1",
           min.frame.size.real = "1",
           nmulti = 0,
           opts=list(),
           random.seed = 42,
           segments = segments,
           segments.max = 10,
           segments.min = 1,
           singular.ok = FALSE,
           tau = NULL,
           weights = NULL)
```

Arguments

Data, Model Inputs And Formula Interface: These arguments identify explicit data inputs for NOMAD kernel/spline search.

xz	continuous and/or nominal/ordinal (factor/ordered) predictors
y	continuous univariate vector

Basis And Spline Complexity: These arguments control basis type and spline complexity.

basis	a character string (default basis="additive") indicating whether the additive or tensor product B-spline basis matrix for a multivariate polynomial spline or generalized B-spline polynomial basis should be used. Note this can be automatically determined by cross-validation if cv=TRUE and basis="auto", and is
-------	---

	an ‘all or none’ proposition (i.e. interaction terms for all predictors or for no predictors given the nature of ‘tensor products’). Note also that if there is only one predictor this defaults to <code>basis="additive"</code> to avoid unnecessary computation as the spline bases are equivalent in this case
<code>complexity</code>	a character string (default <code>complexity="degree-knots"</code>) indicating whether model ‘complexity’ is determined by the degree of the spline or by the number of segments (‘knots’). This option allows the user to use cross-validation to select either the spline degree (number of knots held fixed) or the number of knots (spline degree held fixed) or both the spline degree and number of knots
<code>degree</code>	integer/vector specifying the degree of the B-spline basis for each dimension of the continuous x
<code>degree.max</code>	the maximum degree of the B-spline basis for each of the continuous predictors (default <code>degree.max=10</code>)
<code>degree.min</code>	the minimum degree of the B-spline basis for each of the continuous predictors (default <code>degree.min=0</code>)
<code>knots</code>	a character string (default <code>knots="quantiles"</code>) specifying where knots are to be placed. ‘quantiles’ specifies knots placed at equally spaced quantiles (equal number of observations lie in each segment) and ‘uniform’ specifies knots placed at equally spaced intervals. If <code>knots="auto"</code> , the knot type will be automatically determined by cross-validation
<code>segments</code>	integer/vector specifying the number of segments of the B-spline basis for each dimension of the continuous x (i.e. number of knots minus one)
<code>segments.max</code>	the maximum segments of the B-spline basis for each of the continuous predictors (default <code>segments.max=10</code>)
<code>segments.min</code>	the minimum segments of the B-spline basis for each of the continuous predictors (default <code>segments.min=1</code>)

Kernel Bandwidth Controls: These arguments control categorical-kernel bandwidth search.

<code>lambda</code>	real/vector for the categorical predictors. If it is not NULL, it will be the starting value(s) for <code>lambda</code>
<code>lambda.discrete</code>	if <code>lambda.discrete=TRUE</code> , the bandwidth will be discretized into <code>lambda.discrete.num+1</code> points and <code>lambda</code> will be chosen from these points
<code>lambda.discrete.num</code>	a positive integer indicating the number of discrete values that <code>lambda</code> can assume - this parameter will only be used when <code>lambda.discrete=TRUE</code>

NOMAD Search Controls: These arguments control NOMAD search, cross-validation objective selection, and restart behavior.

<code>cv.df.min</code>	the minimum degrees of freedom to allow when conducting cross-validation (default <code>cv.df.min=1</code>)
<code>cv.func</code>	a character string (default <code>cv.func="cv.ls"</code>) indicating which method to use to select smoothing parameters. <code>cv.gcv</code> specifies generalized cross-validation (Craven and Wahba (1979)), <code>cv.aic</code> specifies expected Kullback-Leibler cross-validation (Hurvich, Simonoff, and Tsai (1998)), and <code>cv.ls</code> specifies least-squares cross-validation

<code>initial.mesh.size.integer</code>	argument passed to the NOMAD solver (see snomadr for further details)
<code>initial.mesh.size.real</code>	argument passed to the NOMAD solver (see snomadr for further details)
<code>max.bb.eval</code>	argument passed to the NOMAD solver (see snomadr for further details). Default is 1000, set on the basis of simulation evidence and real-world applications for the kernel/categorical krscvNOMAD search route.
<code>max.eval</code>	optional NOMAD total point-lookup budget. This is distinct from <code>max.bb.eval</code> : <code>max.bb.eval</code> limits true blackbox objective computations, while <code>max.eval</code> limits total NOMAD point lookups, including cache hits. The default NULL preserves historical behavior by using <code>max.bb.eval</code> as <code>MAX_EVAL</code> . If supplied, <code>max.eval</code> is passed to NOMAD as <code>MAX_EVAL</code> . Supplying both <code>max.eval</code> and <code>opts\$MAX_EVAL</code> with conflicting values is an error.
<code>min.frame.size.integer</code>	arguments passed to the NOMAD solver (see snomadr for further details)
<code>min.frame.size.real</code>	arguments passed to the NOMAD solver (see snomadr for further details)
<code>min.mesh.size.integer</code>	arguments passed to the NOMAD solver (see snomadr for further details)
<code>min.mesh.size.real</code>	argument passed to the NOMAD solver (see snomadr for further details)
<code>nmulti</code>	integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points (default <code>nmulti=0</code>)
<code>opts</code>	list of optional arguments to be passed to snomadr . If not user-specified, this function applies the NOMAD4 path defaults <code>QUAD_MODEL_SEARCH="no"</code> , <code>EVAL_QUEUE_SORT="DIR_LAST"</code> and <code>DIRECTION_TYPE="ORTHO 2N"</code> for an aggressive, speed-oriented mixed-integer search profile in this specific krscvNOMAD path. User-supplied <code>opts</code> entries always take precedence.
<code>random.seed</code>	when it is not missing and not equal to 0, the initial points will be generated using this seed when <code>nmulti > 0</code>
<code>singular.ok</code>	a logical value (default <code>singular.ok=FALSE</code>) that, when FALSE, discards singular bases during cross-validation (a check for ill-conditioned bases is performed).

Quantile And Weights: These arguments control quantile level and observation weights.

<code>tau</code>	if non-null a number in (0,1) denoting the quantile for which a quantile regression spline is to be estimated rather than estimating the conditional mean (default <code>tau=NULL</code>)
<code>weights</code>	an optional vector of weights to be used in the fitting process. Should be 'NULL' or a numeric vector. If non-NULL, weighted least squares is used with weights 'weights' (that is, minimizing 'sum(w*e^2)'); otherwise ordinary least squares is used.

Warnings And Progress: These arguments control warnings and displayed optimizer progress.

`display.nomad.progress`
 a logical value indicating whether to display the progress of the NOMAD solver
 (default `display.nomad.progress=TRUE`)

`display.warnings`
 a logical value indicating whether to display warnings (default `display.warnings=TRUE`)

Details

`krscvNOMAD` computes NOMAD-based cross-validation for a regression spline estimate of a one (1) dimensional dependent variable on an r -dimensional vector of continuous and nominal/ordinal (`factor/ordered`) predictors. Numerical search for the optimal degree/segments/lambda is undertaken using `snomadr`.

The optimal K/λ combination is returned along with other results (see below for return values). The method uses kernel functions appropriate for categorical (ordinal/nominal) predictors which avoids the loss in efficiency associated with sample-splitting procedures that are typically used when faced with a mix of continuous and nominal/ordinal (`factor/ordered`) predictors.

For the continuous predictors the regression spline model employs either the additive or tensor product B-spline basis matrix for a multivariate polynomial spline via the B-spline routines in the GNU Scientific Library (<https://www.gnu.org/software/gsl/>) and the `tensor.prod.model.matrix` function.

For the discrete predictors the product kernel function is of the ‘Li-Racine’ type (see Li and Racine (2007) for details).

Value

`krscvNOMAD` returns a `crscv` object. Furthermore, the function `summary` supports objects of this type. The returned objects have the following components:

<code>K</code>	scalar/vector containing optimal degree(s) of spline or number of segments
<code>K.mat</code>	vector/matrix of values of K evaluated during search
<code>degree.max</code>	the maximum degree of the B-spline basis for each of the continuous predictors (default <code>degree.max=10</code>)
<code>segments.max</code>	the maximum segments of the B-spline basis for each of the continuous predictors (default <code>segments.max=10</code>)
<code>degree.min</code>	the minimum degree of the B-spline basis for each of the continuous predictors (default <code>degree.min=0</code>)
<code>segments.min</code>	the minimum segments of the B-spline basis for each of the continuous predictors (default <code>segments.min=1</code>)
<code>restarts</code>	number of restarts during search, if any
<code>lambda</code>	optimal bandwidths for categorical predictors
<code>lambda.mat</code>	vector/matrix of optimal bandwidths for each degree of spline
<code>cv.func</code>	objective function value at optimum
<code>cv.func.vec</code>	vector of objective function values at each degree of spline or number of segments in <code>K.mat</code>

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca> and Zhenghua Nie <niez@mcmaster.ca>

References

- Abramson, M.A. and C. Audet and G. Couture and J.E. Dennis Jr. and S. Le Digabel (2011), “The NOMAD project”. Software available at <https://www.gerad.ca/nomad>.
- Craven, P. and G. Wahba (1979), “Smoothing Noisy Data With Spline Functions,” *Numerische Mathematik*, 13, 377-403.
- Hurvich, C.M. and J.S. Simonoff and C.L. Tsai (1998), “Smoothing Parameter Selection in Non-parametric Regression Using an Improved Akaike Information Criterion,” *Journal of the Royal Statistical Society B*, 60, 271-293.
- Le Digabel, S. (2011), “Algorithm 909: NOMAD: Nonlinear Optimization With The MADS Algorithm”. *ACM Transactions on Mathematical Software*, 37(4):44:1-44:15.
- Li, Q. and J.S. Racine (2007), *Nonparametric Econometrics: Theory and Practice*, Princeton University Press.
- Ma, S. and J.S. Racine and L. Yang (2015), “Spline Regression in the Presence of Categorical Predictors,” *Journal of Applied Econometrics*, Volume 30, 705-717.
- Ma, S. and J.S. Racine (2013), “Additive Regression Splines with Irrelevant Categorical and Continuous Regressors,” *Statistica Sinica*, Volume 23, 515-541.

See Also

[loess](#), [npregbw](#)

Examples

```
set.seed(42)
## Simulated data
n <- 1000

x <- runif(n)
z <- round(runif(n,min=-0.5,max=1.5))
z.unique <- uniquecombs(as.matrix(z))
ind <- attr(z.unique,"index")
ind.vals <- sort(unique(ind))
dgp <- numeric(length=n)
for(i in 1:nrow(z.unique)) {
  zz <- ind == ind.vals[i]
  dgp[zz] <- z[zz]+cos(2*pi*x[zz])
}
y <- dgp + rnorm(n,sd=.1)

xdata <- data.frame(x,z=factor(z))

## Compute the optimal K and lambda, determine optimal number of knots, set
## spline degree for x to 3
```

```
cv <- krscvNOMAD(x=xdata,y=y,complexity="knots",degree=c(3),segments=c(5))
summary(cv)
```

plot.crs

*Plot Categorical Regression Spline Fits***Description**

Plot fitted conditional mean, conditional quantile, gradient, and bivariate surface displays for [crs](#) objects using the modern **np**-style plot interface. This page is the canonical reference for CRS plot options and plot control helper objects.

Usage

```
## S3 method for class 'crs'
plot(x, ...)

crs_boot_control(wild = c("rademacher", "mammen"),
                 blocklen = NULL,
                 ...)

crs_grid_control(xtrim = NULL, xq = NULL, slices = NULL)
```

Arguments

Plot Object: This argument identifies the fitted CRS object to plot.

x an object of class "crs" returned by [crs](#). Mean-regression objects plot fitted conditional means by default. Objects fit with tau plot fitted conditional quantiles.

Control Helper Constructors: These arguments configure optional S3 control objects consumed by CRS plot methods. The `crs_*_control()` helpers use the same argument conventions as the corresponding modern **np** plot controls where the CRS plot route supports the same feature.

wild	wild-bootstrap multiplier family.
blocklen	optional positive block length for <code>bootstrap="fixed"</code> and <code>bootstrap="geom"</code> . If omitted for mean crs plots, CRS uses the same automatic block-length selector as np .
xtrim	optional scalar trimming/expansion control stored on a grid control object. This uses the same scalar CRS contract as the top-level <code>xtrim</code> argument documented below.
xq	optional conditioning quantile or vector of conditioning quantiles.
slices	reserved for future explicit slice selection.

Plot Options Passed Through ...: Named options passed via ... include:

Data and evaluation controls

`grid_control` optional detailed grid controls created by [crs_grid_control](#).

`neval` the number of evaluation points used for each plotted continuous margin. For bivariate surfaces, the plotted grid has `neval * neval` evaluation points.

`xq` the conditioning quantile used to hold non-plotted continuous variables fixed when drawing one-dimensional slices.

`xtrim` scalar trimming/expansion control for the evaluation support. Values in $[0, 0.5)$ trim both tails by the requested quantile amount; negative values expand the support. The default evaluates over the observed support of the plotted continuous variable.

Plot content controls

`common_scale` logical; whether multiple one-dimensional panels should share a common vertical scale when feasible.

`gradient_order`, `gradients` derivative plotting controls. `gradients=TRUE` requests gradients/effects rather than the fitted function, and `gradient_order` selects the derivative order. Current CRS gradient plotting requires a scalar `gradient_order`.

`layout` "auto" or TRUE requests automatic `par(mfrow=...)` management for multi-panel plots. "current" or FALSE preserves the user's current graphics layout.

`output` one of "plot", "data", or "both". "both" is the public spelling for the historical "plot-data" behavior. "data" returns the plotting payload without drawing.

Interval and bootstrap controls

`alpha` the nominal significance level used for interval/band construction.

`B` the bootstrap replication count. The default is 1999.

`band` one of "pmzsd", "pointwise", "bonferroni", "simultaneous", or "all". `band="all"` draws pointwise, simultaneous, and Bonferroni bands with a package-drawn legend unless suppressed.

`boot_control` optional detailed bootstrap controls created by [crs_boot_control](#). This controls wild-bootstrap multipliers and dependent-bootstrap block length. CRS plot refit bootstraps currently keep fitted tuning parameters fixed.

`bootstrap` the bootstrap resampling scheme. Mean CRS plots support "wild", "inid", "fixed", and "geom". Quantile CRS plots default to "inid" and support "inid", "fixed", and "geom"; "wild" is mean-only. Mean CRS gradient/effect plots also support "wild", "inid", "fixed", and "geom"; quantile CRS gradient plots support the refit schemes only.

`center` interval centering control. CRS currently supports `center="estimate"` only and fails fast for unsupported centering requests.

`errors` one of "none", "bootstrap", or "asymptotic".

Rendering and view controls

`data_overlay` logical; when TRUE, overlay observed response data on fitted mean/quantile displays where meaningful and expand plot limits to include the data. Overlay points default to small, transparent markers and can be customized with ordinary `points()` graphical arguments passed through ...

data_rug logical; when TRUE, draw rug-style support cues for the plotted coordinates. For ordinary one-dimensional base plots this behaves like a standard rug. For eligible bivariate surface plots, the base renderer draws floor-rug ticks and `renderer="rgl"` draws analogous 3D floor-rug ticks.

legend legend control for legends drawn by the package, including `band="all"` uncertainty legends. TRUE draws the default legend, FALSE, NULL, or NA suppresses it, a single character string supplies the legend position, and a list is merged into the default `legend` call for base plots or the default `rgl::legend3d()` call for `renderer="rgl"`. For routes with more than one package-drawn legend, named sublists such as `legend=list(bands=FALSE)` may be used when a route distinguishes those legend roles.

perspective logical; whether two-continuous-predictor CRS fits should be displayed as bivariate fitted-function surfaces.

phi, theta, view view controls for bivariate continuous surface plots. As in **np**, the `rgl` backend remaps the inherited base defaults `theta=0`, `phi=20` to a more informative oblique view.

renderer the surface-rendering backend for eligible bivariate continuous plots. "base" uses `persp`-style graphics. "rgl" uses the suggested **rgl** package and displays through the active **rgl** backend. On supported surface routes, `data_overlay=TRUE` adds observed data as an interactive point cloud, `data_rug=TRUE` adds 3D floor-rug ticks, and `errors!="none"` adds interval surfaces. With `band="all"`, CRS draws a matching interactive legend using `rgl::legend3d()`.

Graphical parameters

`border`, `cex`, `cex.axis`, `cex.lab`, `cex.main`, `cex.sub`, `col`, `lty`, `lwd`, `main`, `pch`, `sub`, `type`, `xlab`, `xlim`, `ylab`, `ylim`, `zlab` standard graphical controls forwarded to the underlying plotting primitives when relevant.

`rgl.persp3d.<arg>`, `rgl.view3d.<arg>`, `rgl.par3d.<arg>`, `rgl.grid3d.<arg>`, `rgl.widget.<arg>`, `rgl.surface3d.<arg>` backend-specific **rgl** controls. For example, `rgl.legend3d.bty="o"` restores a box around the `band="all"` **rgl** legend, and `rgl.surface3d.alpha` customizes interval-surface transparency.

Additional Arguments: Further plotting controls are passed through ... to the relevant plot route.

... for `plot.crs`, additional arguments supplied to control plotting behavior or passed through to underlying plotting helpers where supported. Supported named plotting controls are summarized above; standard graphical parameters are forwarded where relevant. For `crs_boot_control()`, unsupported helper options fail fast rather than being silently ignored.

Details

The default display is the fitted function, matching the modern **np** regression plot convention. One-dimensional fits are shown as partial fitted curves. For two continuous predictors, `perspective=TRUE` requests a fitted surface.

Plot interval/error types are:

"pmzsd"	: point estimate +/- $z_{(1-\alpha/2)}$ * standard error
"pointwise"	: per-point two-sided interval
"bonferroni"	: pointwise interval with alpha adjusted for m grid points

"simultaneous" : joint-band interval from the bootstrap route

where m is the number of evaluation points used in the plotted curve or surface. `band="all"` overlays pointwise, simultaneous, and Bonferroni bands.

For mean-regression plots, `errors="bootstrap"` defaults to the fast fixed-design wild bootstrap used by **np**, with `bootstrap="wild"` and `B=1999`. The refit bootstrap remains available with `bootstrap="inid"`. Mean plots also support the **np**-style block refit bootstraps `bootstrap="fixed"` and `bootstrap="geom"`; use `crs_boot_control(blocklen=...)` to set the block length explicitly, otherwise the **np** automatic block-length selector is used.

Quantile CRS plots, i.e. objects fit with `tau`, follow the **npqreg** convention and default to the refit `bootstrap="inid"` selector. `bootstrap="fixed"` and `bootstrap="geom"` are also available for quantile fitted-curve plots, while `bootstrap="wild"` remains mean-only.

For mean CRS gradient/effect plots requested with `gradients=TRUE`, `errors="bootstrap"` also defaults to the fixed-design wild bootstrap. Numeric continuous panels report derivative intervals computed with the corresponding `crshat()` derivative operator. Categorical panels report focal-minus-baseline effect intervals. Refit gradient/effect bootstrap intervals remain available with `bootstrap="inid"`, `bootstrap="fixed"`, and `bootstrap="geom"`. For quantile CRS gradient plots, `bootstrap="wild"` is not available because quantile estimation is not linear in the response.

Unsupported legacy CRS plot arguments such as `ci`, `deriv`, and `mean` now fail fast. New public plot controls use the **np**-style names documented here.

Value

Invisibly returns the plotted object for ordinary plotting calls. When `output="data"` or `output="both"` is requested, returns the plotting payload. The control helper constructors return small S3 lists consumed by `plot.crs()` and related CRS plot methods.

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

See Also

[crs](#), [crsiv](#), [crsivderiv](#), [clsd](#)

Examples

```
set.seed(42)
n <- 100
x <- runif(n)
y <- sin(2*pi*x) + rnorm(n, sd = 0.25)
model <- crs(y ~ x, cv = "none", degree = 3, segments = 1)

plot(model)
plot(model, data_rug = TRUE)

## Not run:
plot(model, errors = "bootstrap", band = "all")
```

```

plot(model, errors = "bootstrap", bootstrap = "inid", B = 99)
plot(model, errors = "bootstrap", bootstrap = "fixed", B = 99,
      boot_control = crs_boot_control(blocklen = 5))

z <- runif(n)
y2 <- sin(2*pi*x) + z + rnorm(n, sd = 0.25)
smodel <- crs(y2 ~ x + z, cv = "none", degree = c(3, 1),
              segments = c(1, 1))
plot(smodel, perspective = TRUE, data_rug = TRUE)
plot(smodel, perspective = TRUE, renderer = "rgl",
      errors = "bootstrap", band = "all", B = 99)

qmodel <- crs(y ~ x, cv = "none", degree = 3, segments = 1, tau = 0.5)
plot(qmodel, errors = "bootstrap", B = 99)

## End(Not run)

```

snomadr

R interface to NOMAD

Description

snomadr is an R interface to NOMAD (Nonsmooth Optimization by Mesh Adaptive Direct Search, Abramson, Audet, Couture and Le Digabel (2011)), an open source software C++ implementation of the Mesh Adaptive Direct Search (MADS, Le Digabel (2011)) algorithm designed for constrained optimization of blackbox functions.

NOMAD is designed to find (local) solutions of mathematical optimization problems of the form

$$\begin{aligned}
 &\min_{x \in \mathbb{R}^n} && f(x) \\
 &\text{s.t.} && g(x) \leq 0 \\
 & && x_L \leq x \leq x_U
 \end{aligned}$$

where $f(x): \mathbb{R}^n \rightarrow \mathbb{R}^k$ is the objective function, and $g(x): \mathbb{R}^n \rightarrow \mathbb{R}^m$ are the constraint functions. The vectors x_L and x_U are the bounds on the variables x . The functions $f(x)$ and $g(x)$ can be nonlinear and nonconvex. The variables can be integer, continuous real number, binary, and categorical.

Kindly see <https://www.gerad.ca/en/software/nomad> and the references below for details.

Usage

```

snomadr(eval.f,
        n,
        bbin = NULL,
        bbout = NULL,
        x0 = NULL,

```

```

lb = NULL,
ub = NULL,
nmulti = 0,
random.seed = 0,
opts = list(),
display.nomad.progress = TRUE,
information = list(),
snomadr.environment = new.env(),
... )

```

Arguments

Objective, Dimension, And Evaluation Environment: These arguments identify the objective, dimension, black-box types, and evaluation environment.

<code>bbin</code>	types of variables. Variable types are 0 (CONTINUOUS), 1 (INTEGER), 2 (CATEGORICAL), 3 (BINARY)
<code>bbout</code>	types of output of <code>eval.f</code> . See the NOMAD User Guide https://nomad-4-user-guide.readthedocs.io/en/latest/#
<code>eval.f</code>	function that returns the value of the objective function
<code>n</code>	the number of variables
<code>snomadr.environment</code>	environment that is used to evaluate the functions. Use this to pass additional data or parameters to a function

Multi-Start Controls: These arguments control multistart execution and reproducibility settings.

<code>nmulti</code>	when it is missing, or it is equal to 0 and <code>x0</code> is provided, <code>snomadRSolve</code> will be called to solve the problem. Otherwise, <code>smultinomadRSolve</code> will be called
<code>random.seed</code>	when it is not missing and not equal to 0, the initial points will be generated using this seed when <code>nmulti > 0</code>

NOMAD Options And Information Queries: These arguments control NOMAD options and optional information queries.

<code>information</code>	is a list. <code>snomadr</code> will call <code>snomadRInfo</code> to return the information about NOMAD according to the values of "info", "version" and "help". "info"="i": display the usage and copyright of NOMAD "version"="v": display the version of NOMAD you are using "help"="h": display all options You also can display a specific option, for example, "help"="h x0", this will tell you how to set <code>x0</code>
<code>opts</code>	list of options for NOMAD, see the NOMAD user guide https://nomad-4-user-guide.readthedocs.io/en/latest/#. Options can also be set by <code>nomad.opt</code> which should be in the folder where R (<code>snomadr</code>) is executed. This interface uses NOMAD 4 option names directly (for example <code>MIN_FRAME_SIZE</code>).

A complete option catalog generated from the embedded NOMAD 4.5.0 definitions is bundled at `system.file("nomad", "NOMAD_4_5_0_OPTIONS_REFERENCE.md", package="crs")`.

To improve comparability with historical NOMAD 3.9.1 behavior, `snomadr` applies a small NOMAD4 compatibility profile when these options are not user-supplied (user opts always take precedence): `QUAD_MODEL_SEARCH`, `SGTELIB_MODEL_SEARCH`, `NM_SEARCH`, `SPECULATIVE_SEARCH`, `EVAL_OPPORTUNISTIC`, `EVAL_QUEUE_SORT`, `DIRECTION_TYPE`, `QUAD_MODEL_SEARCH_BOX_FACTOR`, and `QUAD_MODEL_BOX_FACTOR`.

A few high-impact options are:

```
"MAX_BB_EVAL"=10000
```

```
"INITIAL_MESH_SIZE"=1
```

```
"MIN_MESH_SIZE"="r1.0e-10"
```

```
"MIN_FRAME_SIZE"=1
```

Note that the "r..." denotes relative measurement (relative to lb and ub)

Note that decreasing the maximum number of black box evaluations ("MAX_BB_EVAL") will terminate search sooner and may result in a less accurate solution. For complicated problems you may want to increase this value. When experimenting, "DISPLAY_DEGREE"=1 or "DISPLAY_DEGREE"=2 can provide additional native NOMAD output. Higher native display degrees are currently rejected by the embedded `snomadr()` interface because NOMAD4's verbose block output queue can abort repeated R-session solves.

Starting Values, Bounds, And Variable Types: These arguments identify starting values, bounds, and variable types.

lb	vector with lower bounds of the controls (use -1.0e19 for controls without lower bound)
ub	vector with upper bounds of the controls (use 1.0e19 for controls without upper bound)
x0	vector with starting values for the optimization. If it is provided and <code>nmulti</code> is bigger than 1, <code>x0</code> will be the first initial point for multiple initial points

Additional Arguments: These arguments control displayed progress and pass-through values.

...	arguments that will be passed to the user-defined objective and constraints functions. See the examples below
display.nomad.progress	when FALSE, no output from <code>snomadr</code> is displayed on the screen. If the NOMAD option "DISPLAY_DEGREE"=0, is set, there will also be no output from NOMAD. Supported native "DISPLAY_DEGREE" values are 0, 1, and 2 in this embedded interface.

Details

`snomadr` is used in the `crs` package to numerically minimize an objective function with respect to the spline degree, number of knots, and optionally the kernel bandwidths when using `crs` with the option `cv="nomad"` (default). This is a constrained mixed integer combinatoric problem and

is known to be computationally ‘hard’. See [frscvNOMAD](#) and [krscvNOMAD](#) for the functions called when `cv="nomad"` while using [crs](#).

However, the user should note that for simple problems involving one predictor exhaustive search may be faster and potentially more accurate, so please bear in mind that `cv="exhaustive"` can be useful when using [crs](#).

Naturally, exhaustive search is also useful for verifying solutions returned by `snomadr`. See [frscv](#) and [krscv](#) for the functions called when `cv="exhaustive"` while using [crs](#).

Value

The return value contains a list with the inputs, and additional elements

<code>call</code>	the call that was made to solve
<code>status</code>	integer value with the status of the optimization
<code>message</code>	more informative message with the status of the optimization
<code>bbe</code>	number of blackbox evaluations reported by the interface
<code>cache.hits</code>	number of NOMAD native cache hits during the run
<code>cache.size</code>	number of points in NOMAD’s native cache at the end of the run
<code>callback.evaluations</code>	number of evaluations sent to the R callback
<code>total.evaluations</code>	total evaluations represented by callback evaluations plus cache hits
<code>iterations</code>	number of iterations that were executed, if multiple initial points are set, this number will be the sum for each initial point.
<code>objective</code>	value if the objective function in the solution
<code>solution</code>	optimal value of the controls

Author(s)

Zhenghua Nie <niez@mcmaster.ca>

References

Abramson, M.A. and C. Audet and G. Couture and J.E. Dennis Jr. and S. Le Digabel (2011), “The NOMAD project”. Software available at <https://www.gerad.ca/en/software/nomad/>

Le Digabel, S. (2011), “Algorithm 909: NOMAD: Nonlinear Optimization With The MADS Algorithm”. *ACM Transactions on Mathematical Software*, 37(4):44:1-44:15.

See Also

[optim](#), [nlm](#), [nlminb](#)

Examples

```
## Not run:
## List all options
snomadr(information=list("help"="-h"))

## Print given option, for example, MESH_SIZE
snomadr(information=list("help"="-h MESH_SIZE"))

## Print the version of NOMAD
snomadr(information=list("version"="-v"))

## Print usage and copyright
snomadr(information=list("info"="-i"))

## This is the example found in
## NOMAD/examples/basic/library/single_obj/basic_lib.cpp

eval.f <- function ( x ) {

  f <- c(Inf, Inf, Inf);
  n <- length (x);

  if ( n == 5 && ( is.double(x) || is.integer(x) ) ) {
    f[1] <- x[5];
    f[2] <- sum ( (x-1)^2 ) - 25;
    f[3] <- 25 - sum ( (x+1)^2 );
  }

  return ( as.double(f) );
}

## Initial values
x0 <- rep(0.0, 5 )

bbin <-c(1, 1, 1, 1, 1)
## Bounds
lb <- rep(-6.0,5 )
ub <- c(5.0, 6.0, 7.0, 1000000, 1000000)

bbout <-c(0, 2, 1)
## Options
opts <-list("MAX_BB_EVAL"=500,
           "MIN_MESH_SIZE"=0.001,
           "INITIAL_MESH_SIZE"=0.1,
           "MIN_FRAME_SIZE"=1)

snomadr(eval.f=eval.f,n=5, x0=x0, bbin=bbin, bbout=bbout, lb=lb, ub=ub, opts=opts)

## How to transfer other parameters into eval.f
##
## First example: supply additional arguments in user-defined functions
```

```

##

## objective function and gradient in terms of parameters
eval.f.ex1 <- function(x, params) {
  return( params[1]*x^2 + params[2]*x + params[3] )
}

## Define parameters that we want to use
params <- c(1,2,3)

## Define initial value of the optimization problem
x0 <- 0

## solve using snomadr
snomadr( n      =1,
         x0      = x0,
         eval.f   = eval.f.ex1,
         params   = params )

##
## Second example: define an environment that contains extra parameters
##

## Objective function and gradient in terms of parameters
## without supplying params as an argument
eval.f.ex2 <- function(x) {
  return( params[1]*x^2 + params[2]*x + params[3] )
}

## Define initial value of the optimization problem
x0 <- 0

## Define a new environment that contains params
auxdata <- new.env()
auxdata$params <- c(1,2,3)

## pass The environment that should be used to evaluate functions to snomadr
snomadr(n      =1,
         x0      = x0,
         eval.f   = eval.f.ex2,
         snomadr.environment = auxdata )

## Solve using algebra
cat( paste( "Minimizing f(x) = ax^2 + bx + c\n" ) )
cat( paste( "Optimal value of control is -b/(2a) = ", -params[2]/(2*params[1]), "\n" ) )
cat( paste( "With value of the objective function f(-b/(2a)) = ",
           eval.f.ex1( -params[2]/(2*params[1]), params ), "\n" ) )

## The following example is NOMAD/examples/advanced/multi_start/multi.cpp
## This will call smultinomadRSolve to resolve the problem.
eval.f.ex1 <- function(x, params) {
  M<-as.numeric(params$M)

```

```

NBC<-as.numeric(params$NBC)

f<-rep(0, M+1)
x<-as.numeric(x)

x1 <- rep(0.0, NBC)
y1 <- rep(0.0, NBC)

x1[1]<-x[1]
x1[2]<-x[2]
y1[3]<-x[3]
x1[4]<-x[4]
y1[4]<-x[5]

epi <- 6
for(i in 5:NBC){
  x1[i]<-x[epi]
  epi <- epi+1
  y1[i]<-x[epi]
  epi<-epi+1
}
constraint <- 0.0
ic <- 1
f[ic]<-constraint
ic <- ic+1

constraint <- as.numeric(1.0)
distmax <- as.numeric(0.0)
avg_dist <- as.numeric(0.0)
dist1<-as.numeric(0.0)

for(i in 1:(NBC-1)){
  for (j in (i+1):NBC){
    dist1 <- as.numeric((x1[i]-x1[j])*(x1[i]-x1[j])+(y1[i]-y1[j])*(y1[i]-y1[j]))

    if((dist1 > distmax)) {distmax <- as.numeric(dist1)}
    if((dist1[1]) < 1) {constraint <- constraint*sqrt(dist1)}
    else if((dist1) > 14) {avg_dist <- avg_dist+sqrt(dist1)}
  }
}

if(constraint < 0.9999) constraint <- 1001.0-constraint
else constraint = sqrt(distmax)+avg_dist/(10.0*NBC)

f[2] <- 0.0
f[M+1] <- constraint

return(as.numeric(f) )
}

## Define parameters that we want to use
params<-list()

```

```

NBC <- 5
M <- 2
n<-2*NBC-3

params$NBC<-NBC
params$M<-M
x0<-rep(0.1, n)
lb<-rep(0, n)
ub<-rep(4.5, n)

eval.f.ex1(x0, params)

bbout<-c(2, 2, 0)
nmulti=5
bbin<-rep(0, n)
## Define initial value of the optimization problem

## Solve using snomadRSolve
snomadr(n          = as.integer(n),
        x0         = x0,
        eval.f     = eval.f.ex1,
        bbin       = bbin,
        bbout      = bbout,
        lb         = lb,
        ub         = ub,
        params     = params )

## Solve using smultinomadRSolve, if x0 is provided, x0 will
## be the first initial point, otherwise, the program will
## check best_x.txt, if it exists, it will be read in as
## the first initial point. Other initial points will be
## generated by uniform distribution.
## nmulti represents the number of mads will run.
##
snomadr(n          = as.integer(n),
        eval.f     = eval.f.ex1,
        bbin       = bbin,
        bbout      = bbout,
        lb         = lb,
        ub         = ub,
        nmulti     = as.integer(nmulti),
        display.nomad.progress = TRUE,
        params     = params )

## End(Not run)

```

Description

Produce model matrices or penalty matrices for a tensor product smooth from the model matrices or penalty matrices for the marginal bases of the smooth.

Usage

```
tensor.prod.model.matrix(X)
```

Arguments

`X` a list of model matrices for the marginal bases of a smooth

Details

If $X[[1]]$, $X[[2]]$... $X[[m]]$ are the model matrices of the marginal bases of a tensor product smooth then the i th row of the model matrix for the whole tensor product smooth is given by $X[[1]][i,] \% \times \% X[[2]][i,] \% \times \% \dots X[[m]][i,]$, where $\% \times \%$ is the Kronecker product. Of course the routine operates column-wise, not row-wise!

Value

Either a single model matrix for a tensor product smooth, or a list of penalty terms for a tensor product smooth.

Author(s)

Simon N. Wood <simon.wood@r-project.org>

References

Wood, S.N. (2006) "Low Rank Scale Invariant Tensor Product Smoother for Generalized Additive Mixed Models". *Biometrics* 62(4):1025-1036

See Also

[te](#), [smooth.construct.tensor.smooth.spec](#)

Examples

```
X <- list(matrix(1:4,2,2),matrix(5:10,2,3))
tensor.prod.model.matrix(X)
```

uniquecombs

*Find the unique rows in a matrix***Description**

This routine returns a matrix containing all the unique rows of the matrix supplied as its argument. That is, all the duplicate rows are stripped out. Note that the ordering of the rows on exit is not the same as on entry. It also returns an index attribute for relating the result back to the original matrix.

Usage

```
uniquecombs(x)
```

Arguments

`x` is an R matrix (numeric)

Details

Models with more parameters than unique combinations of covariates are not identifiable. This routine provides a means of evaluating the number of unique combinations of covariates in a model. The routine calls compiled C code.

Value

A matrix consisting of the unique rows of `x` (in arbitrary order).

The matrix has an "index" attribute. `index[i]` gives the row of the returned matrix that contains row `i` of the original matrix.

Author(s)

Simon N. Wood <simon.wood@r-project.org>

See Also

[unique](#) can do the same thing, including for non-numeric matrices, but more slowly and without returning the index.

Examples

```
X<-matrix(c(1,2,3,1,2,3,4,5,6,1,3,2,4,5,6,1,1,1),6,3,byrow=TRUE)
print(X)
Xu <- uniquecombs(X);Xu
ind <- attr(Xu,"index")
## find the value for row 3 of the original from Xu
Xu[ind[3],];X[3,]
```

wage1

*Cross-Sectional Data on Wages***Description**

Cross-section wage data consisting of a random sample taken from the U.S. Current Population Survey for the year 1976. There are 526 observations in total.

Usage

```
data("wage1")
```

Format

A data frame with 24 columns, and 526 rows.

wage column 1, of type numeric, average hourly earnings

educ column 2, of type numeric, years of education

exper column 3, of type numeric, years potential experience

tenure column 4, of type numeric, years with current employer

nonwhite column 5, of type factor, =“Nonwhite” if nonwhite, “White” otherwise

female column 6, of type factor, =“Female” if female, “Male” otherwise

married column 7, of type factor, =“Married” if Married, “Nonmarried” otherwise

numdep column 8, of type numeric, number of dependents

smsa column 9, of type numeric, =1 if live in SMSA

northcen column 10, of type numeric, =1 if live in north central U.S

south column 11, of type numeric, =1 if live in southern region

west column 12, of type numeric, =1 if live in western region

construc column 13, of type numeric, =1 if work in construc. indus.

ndurman column 14, of type numeric, =1 if in nondur. manuf. indus.

trcommpu column 15, of type numeric, =1 if in trans, commun, pub ut

trade column 16, of type numeric, =1 if in wholesale or retail

services column 17, of type numeric, =1 if in services indus.

profserv column 18, of type numeric, =1 if in prof. serv. indus.

profocc column 19, of type numeric, =1 if in profess. occupation

clerocc column 20, of type numeric, =1 if in clerical occupation

servocc column 21, of type numeric, =1 if in service occupation

lwage column 22, of type numeric, log(wage)

expersq column 23, of type numeric, exper^2

tenursq column 24, of type numeric, tenure^2

Source

Jeffrey M. Wooldridge

References

Wooldridge, J.M. (2000), *Introductory Econometrics: A Modern Approach*, South-Western College Publishing.

Examples

```
## Not run:
data(wage1)

## Cross-validated model selection for spline degree and bandwidths Note
## - we override the default degree.max and segments.max here given the
## large number of predictors and small sample size (the tensor spline basis
## will become singular hampering search)

model <- crs(lwage~married+
             female+
             nonwhite+
             educ+
             exper+
             tenure,
             degree.max=5,
             segments.max=5,
             data=wage1)

summary(model)

## Partial mean plots (control for non axis predictors)
plot(model)
## Partial first derivative plots (control for non axis predictors)
plot(model,gradients = TRUE)
## Partial second derivative plots (control for non axis predictors)
plot(model,gradients = TRUE, gradient_order = 2)

## End(Not run)
```

Index

- * **datasets**
 - cps71, 9
 - Engel95, 44
 - wage1, 81
 - * **instrument**
 - crsiv, 22
 - crsivderiv, 29
 - * **interface**
 - crs_nomad_api, 37
 - snomadr, 71
 - * **models**
 - glp.model.matrix, 54
 - tensor.prod.model.matrix, 78
 - uniquecombs, 80
 - * **nonparametric**
 - clsd, 3
 - crs, 10
 - crssigtest, 35
 - frscv, 46
 - frscvNOMAD, 50
 - gsl.bs, 55
 - krscv, 57
 - krscvNOMAD, 61
 - * **optimize**
 - snomadr, 71
 - * **package**
 - crs-package, 2
 - * **regression**
 - glp.model.matrix, 54
 - tensor.prod.model.matrix, 78
 - uniquecombs, 80
 - * **smooth**
 - glp.model.matrix, 54
 - tensor.prod.model.matrix, 78
- bs, 57
- class, 18
- clsd, 3, 70
- coef, 7
- cps71, 9
- crs, 10, 22, 25–27, 31–34, 67, 70, 73, 74
- crs-package, 2
- crs_boot_control, 68
- crs_boot_control (plot.crs), 67
- crs_grid_control, 68
- crs_grid_control (plot.crs), 67
- crs_nomad_api, 37
- crshat, 20
- crsiv, 22, 34, 70
- crsivderiv, 27, 29, 70
- crssigtest, 35
- Engel95, 44
- extendrange, 5
- factor, 10, 12, 13, 16, 18, 46–48, 50, 53, 57, 58, 60–62, 65
- fitted, 7, 17, 26, 32
- frscv, 46, 74
- frscvNOMAD, 14, 50, 74
- glp.model.matrix, 54
- gsl.bs, 55
- hatvalues, 18
- krscv, 57, 74
- krscvNOMAD, 14, 61, 74
- legend, 69
- lm, 15, 18
- loess, 19, 49, 54, 61, 66
- logspline, 8
- model.frame, 18
- nlm, 74
- nlminb, 74
- optim, 6, 16, 59, 60, 74

ordered, [10](#), [12](#), [13](#), [16](#), [18](#), [46–48](#), [50](#), [53](#), [57](#),
[58](#), [60–62](#), [65](#)

persp, [69](#)

plot, [7](#), [17](#), [26](#), [32](#), [33](#)

plot.crs, [7](#), [8](#), [17](#), [26](#), [27](#), [33](#), [34](#), [67](#)

predict, [17](#), [22](#), [26](#), [32](#), [57](#)

predict.lm, [17](#)

print, [26](#), [32](#)

print.crshat (crshat), [20](#)

residuals, [17](#), [26](#), [32](#)

smooth.construct.tensor.smooth.spec,
[79](#)

smooth.spline, [19](#)

snomadr, [6](#), [14](#), [15](#), [25](#), [32](#), [43](#), [52](#), [53](#), [64](#), [65](#),
[71](#)

stepAIC, [14](#), [17](#)

summary, [7](#), [17](#), [18](#), [26](#), [32](#), [36](#), [48](#), [53](#), [60](#), [65](#)

te, [79](#)

tensor.prod.model.matrix, [3](#), [7](#), [13](#), [16](#), [48](#),
[53](#), [60](#), [65](#), [78](#)

unique, [80](#)

uniquecombs, [80](#)

vignette, [2](#)

wage1, [81](#)