Package: npRmpi (via r-universe)

October 13, 2026

Version 0.70-5

Date 2026-07-15

Depends R (>= 3.5.0)

Imports boot, cubature, methods, quadprog, quantreg, stats, parallel, tools, crs (>= 0.15-44)

SystemRequirements MPI

Suggests MASS, logspline, ks, testthat, np, withr, knitr, rmarkdown, rgl

LinkingTo crs (>= 0.15-44)

VignetteBuilder knitr

Config/testthat/edition 3

RoxygenNote 0.0.0

Title Parallel Nonparametric Kernel Smoothing Methods for Mixed Data Types Using 'MPI'

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

Description Nonparametric (and semiparametric) kernel methods that seamlessly handle a mix of continuous, unordered, and ordered factor data types. This package is a parallel implementation of the 'np' package based on the 'MPI' specification that incorporates the 'Rmpi' package (Hao Yu <hyu@stats.uwo.ca>) with minor modifications and we are extremely grateful to Hao Yu for his contributions to the 'R' community. We 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 (SHARCNET, <https://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

Encoding UTF-8

URL https://github.com/JeffreyRacine/R-Package-np

BugReports https://github.com/JeffreyRacine/R-Package-np/issues

NeedsCompilation yes

Author Jeffrey S. Racine [aut, cre], Tristen Hayfield [aut], Hao Yu [ctb, cph], The GSL Team [cph], Numerical Recipes Software [cph]

Config/pak/sysreqs make libopenmpi-dev

Repository https://jeffreyracine.r-universe.dev

Date/Publication 2026-07-15 00:20:02 UTC

RemoteUrl https://github.com/cran/npRmpi

RemoteRef HEAD

RemoteSha 6a4e5c99d577be8aa9e591a700f9cd1abf0b1587

Contents

b.star . . . 4
cps71 . . . 5
dimBS . . . 7
Engel95 . . . 8
gradients . . . 11
Italy . . . 13
lamhosts . . . 15
mpi.barrier . . . 16
mpi.bcast . . . 16
mpi.bcast.cmd . . . 17
mpi.bcast.Robj . . . 19
mpi.close.Rslaves . . . 20
mpi.comm.free . . . 21
mpi.comm.size . . . 22
mpi.exit . . . 23
mpi.get.processor.name . . . 24
mpi.get.version . . . 24
mpi.hostinfo . . . 25
np.kernels . . . 26
np.mpi.initialize . . . 29
np.options . . . 30
np.pairs . . . 33
np_plot_controls . . . 35
npcdens . . . 36

npcdensbw . . . 44
npcdenshat . . . 59
npcdist . . . 61
npcdistbw . . . 69
npcdisthat . . . 84
npcmstest . . . 86
npconmode . . . 90
npcopula . . . 99
npdeneqtest . . . 107
npdeptest . . . 110
npindex . . . 114
npindexbw . . . 120
npksum . . . 131
nplsqreg . . . 137
nplsqregbw . . . 141
npplreg . . . 144
npplregbw . . . 150
npqcmstest . . . 160
npqreg . . . 164
npquantile . . . 171
npreg . . . 175
npregbw . . . 182
npreghat . . . 196
npregiv . . . 198
npregivderiv . . . 208
npRmpi . . . 213
npRmpi.init . . . 218
npscoef . . . 222
npscoefbw . . . 228
npsdeptest . . . 241
npseed . . . 244
npsemihat . . . 246
npsigtest . . . 250
npsymtest . . . 255
nptgauss . . . 259
npudens . . . 260
npudensbw . . . 266
npudenshat . . . 277
npudist . . . 278
npudistbw . . . 284
npudisthat . . . 295
npuniden.boundary . . . 297
npuniden.reflect . . . 301
npuniden.sc . . . 305
npunitest . . . 311
oecdpanel . . . 314
plot.np . . . 316
se . . . 331

uocquantile 333
wage1 334

Index **336**

`b.star` *Compute Optimal Block Length for Stationary and Circular Bootstrap*

Description

`b.star` is a function which computes the optimal block length for the continuous variable `data` using the method described in Patton, Politis and White (2009).

Usage

```
b.star(data,
       Kn = NULL,
       mmax= NULL,
       Bmax = NULL,
       c = NULL,
       round = FALSE)
```

Arguments

Data Input: Time-series data used for automatic block-length selection.

`data`	data, an n x k matrix, each column being a data series.

Block-Length Selection Controls: Tuning constants from Politis and White (2004) and Patton, Politis, and White (2009).

`Kn`	See footnote c, page 59, Politis and White (2004). Defaults to `ceiling(log10(n))`.
`mmax`	See Politis and White (2004). Defaults to `ceiling(sqrt(n))+Kn`.
`Bmax`	See Politis and White (2004). Defaults to `ceiling(min(3*sqrt(n),n/3))`.
`c`	See Politis and White (2004). Defaults to `qnorm(0.975)`.

Output Rounding: Control for rounding the selected block lengths.

`round`	whether to round the result or not. Defaults to FALSE.

Details

`b.star` is a function which computes optimal block lengths for the stationary and circular bootstraps. This allows the use of `tsboot` from the **boot** package to be fully automatic by using the output from `b.star` as an input to the argument `l =` in `tsboot`. See below for an example.

Value

A kx2 matrix of optimal bootstrap block lengths computed from `data` for the stationary bootstrap and circular bootstrap (column 1 is for the stationary bootstrap, column 2 the circular).

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Patton, A. and D.N. Politis and H. White (2009), "CORRECTION TO "Automatic block-length selection for the dependent bootstrap" by D. Politis and H. White", Econometric Reviews 28(4), 372-375.

Politis, D.N. and J.P. Romano (1994), "Limit theorems for weakly dependent Hilbert space valued random variables with applications to the stationary bootstrap", Statistica Sinica 4, 461-476.

Politis, D.N. and H. White (2004), "Automatic block-length selection for the dependent bootstrap", Econometric Reviews 23(1), 53-70.

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
set.seed(12345)

# Function to generate an AR(1) series

ar.series <- function(phi,epsilon) {
  n <- length(epsilon)
  series <- numeric(n)
  series[1] <- epsilon[1]/(1-phi)
  for(i in 2:n) {
    series[i] <- phi*series[i-1] + epsilon[i]
  }
  return(series)
}

yt <- ar.series(0.1,rnorm(10000))
b.star(yt,round=TRUE)

yt <- ar.series(0.9,rnorm(10000))
b.star(yt,round=TRUE)

## End(Not run)
```

---

cps71 *Canadian High School Graduate Earnings*

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

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave). Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

data("cps71", package = "npRmpi")
mpi.bcast.Robj2slave(cps71)

if (interactive()) with(cps71, plot(age, logwage, xlab="Age", ylab="log(wage)"))
```

```
## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

dimBS *Local-Polynomial Basis Dimension Helper*

Description

dimBS returns the number of columns implied by an additive, generalized local-polynomial, or tensor-product basis specification. It is a compatibility wrapper around the internal dim_basis() helper used by **npRmpi**.

Usage

```
dimBS(basis = "additive",
      kernel = TRUE,
      degree = NULL,
      segments = NULL,
      include = NULL,
      categories = NULL)
```

Arguments

Basis Specification: Basis family, continuous-kernel counting mode, polynomial degree, and segment controls.

basis	basis family. One of "additive", "glp", or "tensor".
kernel	logical indicating whether only the continuous-kernel basis should be counted. When FALSE, optional categorical augmentation controlled by include and categories is included.

`degree` non-negative integer vector of local-polynomial degrees.

`segments` positive integer vector giving the number of segments for each continuous predictor. Defaults to one segment per degree entry.

Categorical Augmentation: Optional categorical-component controls used when `kernel = FALSE`.

`include` non-negative integer vector indicating which categorical components are included when `kernel = FALSE`.

`categories` non-negative integer vector giving category counts for included categorical components when `kernel = FALSE`.

Details

`dimBS()` is provided for compatibility with **crs**. In **npRmpi**, the underlying implementation lives in `dim_basis()`, which is used internally for LP basis-dimension checks and safe NOMAD restart initialization.

Value

A numeric scalar giving the implied basis dimension.

Examples

```
dimBS(basis = "tensor", degree = c(2, 2))
dimBS(basis = "glp", degree = c(3, 1, 0))
```

`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:
## Not run in checks: this IV example is computationally expensive and can
## exceed check time limits in MPI environments.
## Example - compute nonparametric instrumental regression using
## Landweber-Fridman iteration of Fredholm integral equations of the
## first kind.

## We consider an equation with an endogenous regressor (`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.

## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave). Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.
npRmpi.init(nslaves=1)
```

```
data(Engel95)

## Sort on logexp (the endogenous regressor) for plotting purposes

Engel95 <- Engel95[order(Engel95$logexp),]
mpi.bcast.Robj2slave(Engel95)

mpi.bcast.cmd(model.iv <- npregiv(y=Engel95$food,
                                  z=Engel95$logexp,
                                  w=Engel95$logwages,
                                  method="Landweber-Fridman"),
              caller.execute=TRUE)
phi <- model.iv$phi

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

mpi.bcast.cmd(ghat <- npreg(xdat=Engel95$logexp,
                            ydat=Engel95$food,
                            regtype="ll"),
              caller.execute=TRUE)

## For the plots, 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)

trim <- 0.0025

plot(Engel95$logexp,Engel95$food,
     ylab="Food Budget Share",
     xlab="log(Total Expenditure)",
     xlim=quantile(Engel95$logexp,c(trim,1-trim)),
     ylim=quantile(Engel95$food,c(trim,1-trim)),
     main="Nonparametric Instrumental Kernel Regression",
     type="p",
     cex=.5,
     col="lightgrey")

lines(Engel95$logexp,phi,col="blue",lwd=2,lty=2)

lines(Engel95$logexp,fitted(ghat),col="red",lwd=2,lty=4)

legend(quantile(Engel95$logexp,trim),quantile(Engel95$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))

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
```

```
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close

## End(Not run)
```

`gradients` *Extract Gradients*

Description

`gradients` is a generic function which extracts gradients from objects.

Usage

```
gradients(x, ...)

## S3 method for class 'condensity'
gradients(x, errors = FALSE, gradient.order = NULL, ...)

## S3 method for class 'condistribution'
gradients(x, errors = FALSE, gradient.order = NULL, ...)

## S3 method for class 'lsqregression'
gradients(x, errors = FALSE, gradient.order = NULL, ...)

## S3 method for class 'npregression'
gradients(x, errors = FALSE, gradient.order = NULL, ...)

## S3 method for class 'qregression'
gradients(x, errors = FALSE, ...)

## S3 method for class 'singleindex'
gradients(x, errors = FALSE, ...)
```

Arguments

Object And Output Controls: Object to interrogate and whether gradient standard errors are requested.

`x`	an object for which the extraction of gradients is meaningful.

`errors` a logical value specifying whether or not standard errors of gradients are desired. Defaults to `FALSE`.

Derivative Order Controls: Optional local-polynomial derivative order controls.

`gradient.order` for `npregression`, `condensity`, `condistribution`, and `lsqregression` objects fitted with `regtype="lp"`, optional derivative order request (scalar or one entry per continuous predictor). For `npregression`, `condensity`, and `condistribution`, unavailable components whose order exceeds the fitted polynomial degree are returned as `NA` when another requested continuous component remains available. For `lsqregression`, a supplied order must exactly identify the stored order; unavailable stored components remain `NA`, while any different requested order is rejected. If another derivative order is desired, compute the fit, prediction, or evaluation with `gradients=TRUE` and that `gradient.order`; `gradients()` does not recompute derivatives after fitting.

Additional Arguments: Further method-specific arguments.

`...` other arguments.

Details

This function provides a generic interface for extraction of gradients from objects. For `npregression`, `condensity`, `condistribution`, and `lsqregression` objects fitted with `regtype="lp"`, `gradient.order` identifies the stored continuous-predictor derivative order to extract; it is not a post-fit recomputation control. For `lsqregression`, the requested order must exactly match the stored order. For `qregression` objects, `errors=TRUE` returns asymptotic standard errors for the quantile gradients when the object was fitted with `gradients=TRUE`.

Value

Gradients extracted from the model object `x`.

Note

This method currently only supports objects from the `npRmpi` library.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

See the references for the method being interrogated via `gradients` in the appropriate help file. For example, for the particulars of the gradients for nonparametric regression see the references in `npreg`

See Also

`fitted`, `residuals`, `coef`, and `se`, for related methods; `npRmpi` for supported objects; `npRmpi.init` for MPI session startup.

Examples

```
## Not run in checks: excluded to keep MPI examples stable and check times short.
## Not run:

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

x <- runif(10)
y <- x + rnorm(10, sd = 0.1)
model <- npreg(y~x, gradients=TRUE)
gradients(model)

npRmpi.quit()
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

`Italy` *Italian GDP Panel*

Description

Italian GDP growth panel for 21 regions covering the period 1951-1998 (millions of Lire, 1990=base). There are 1008 observations in total.

Usage

```
data("Italy")
```

Format

A data frame with 2 columns, and 1008 rows.

year the first column, of type `ordered`

gdp the second column, of type `numeric`: millions of Lire, 1990=base

Source

Giovanni Baiocchi

References

Baiocchi, G. (2006), "Economic Applications of Nonparametric Methods," Ph.D. Thesis, University of York.

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave). Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

data("Italy")
mpi.bcast.Robj2slave(Italy)

with(Italy, {

plot(ordered(year), gdp, xlab="Year (ordered factor)",
     ylab="GDP (millions of Lire, 1990=base)")

})

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
```

```
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

`lamhosts` *Hosts Information*

Description

`lamhosts` finds the host name associated with its node number. It can be used with `npRmpi.init` (`mode="spawn"`), which internally uses `mpi.spawn.Rslaves`, to start R slaves on selected hosts. This is a MPI implementation specific function.

`mpi.is.master` checks if it is running on master or slaves.

Usage

```
lamhosts()
mpi.is.master()
```

Value

`lamhosts` returns CPUs nodes numbers with their host names.

`mpi.is.master` returns TRUE if it is on master and FALSE otherwise.

Author(s)

Hao Yu (minor modifications by Jeffrey S. Racine <racinej@mcmaster.ca>)

See Also

`npRmpi.init`, `mpi.hostinfo`, `slave.hostinfo`

mpi.barrier *MPI_Barrier API*

Description

`mpi.barrier` blocks the caller until all members have called it.

Usage

```
mpi.barrier(comm = 1)
```

Arguments

Communicator Input: MPI communicator on which to synchronize ranks.

`comm`	a communicator number

Value

1 if success. Otherwise 0.

Author(s)

Hao Yu

References

https://www.mpich.org/, https://www.mpich.org/static/docs/latest/www3/

mpi.bcast *MPI_Bcast API*

Description

`mpi.bcast` is a collective call among all members in a comm. It broadcasts a message from the specified rank to all members.

Usage

```
mpi.bcast(x,
          type,
          rank = 0,
          comm = 1,
          buffunit=100)
```

Arguments

Message Payload: Object and low-level MPI datatype sent or received by the broadcast.

`x`	data to be sent or received. Must be the same type among all members.
`type`	1 for integer, 2 for double, and 3 for character. Others are not supported.

Communication Controls: Sender rank, communicator, and buffer-unit controls.

`rank`	the sender.
`comm`	a communicator number.
`buffunit`	a buffer unit number.

Details

`mpi.bcast` is a blocking call among all members in a comm, i.e, all members have to wait until everyone calls it. All members have to prepare the same type of messages (buffers). Hence it is relatively difficult to use in R environment since the receivers may not know what types of data to receive, not to mention the length of data. Users should use various extensions of `mpi.bcast` in R. They are `mpi.bcast.Robj`, `mpi.bcast.cmd`, and `mpi.bcast.Robj2slave`.

When type=5, MPI continuous datatype (double) is defined with unit given by `buffunit`. It is used to transfer huge data where a double vector or matrix is divided into many chunks with unit `buffunit`. Total ceiling(length(obj)/buffunit) units are transferred. Due to MPI specification, both `buffunit` and total units transferred cannot be over 2^31-1. Notice that the last chunk may not have full length of data due to rounding. Special care is needed.

Value

`mpi.bcast` returns the message broadcasted by the sender (specified by the rank).

References

https://www.mpich.org/, https://www.mpich.org/static/docs/latest/www3/

See Also

`mpi.bcast.Robj`, `mpi.bcast.cmd`, `mpi.bcast.Robj2slave`.

`mpi.bcast.cmd` *Extension of MPI_Bcast API*

Description

`mpi.bcast.cmd` is an extension of `mpi.bcast`. It is mainly used to transmit a command from master to all R slaves spawned by using slavedaemon.R script.

Usage

```
mpi.bcast.cmd(cmd=NULL,
              ...,
              rank = 0,
              comm = 1,
              nonblock=FALSE,
              sleep=0.1,
              caller.execute = FALSE)
```

Arguments

Command Payload: Command sent from the master and optional arguments evaluated on workers.

`cmd`	a command to be sent from master.

Communication Controls: Sender rank, communicator, receiver polling, and caller-execution controls.

`rank`	the sender
`comm`	a communicator number
`nonblock`	logical. If TRUE, a nonblock procedure is used on all receivers so that they will consume none or little CPUs while waiting.
`sleep`	a sleep interval, used when nonblock=TRUE. The smaller sleep is, the more responsive slaves are, the more CPUs consume.
`caller.execute`	a logical value indicating whether the master node is additionally to execute the command

Additional Command Arguments: Arguments supplied to the transmitted function command.

`...`	used as arguments to cmd (function command) for passing their (master) values to R slaves, i.e., if 'myfun(x)' will be executed on R slaves with 'x' as master variable, use mpi.bcast.cmd(cmd=myfun, x=x).

Details

`mpi.bcast.cmd` is a collective call. This means all members in a communicator must execute it at the same time. Under `npRmpi.init(mode="spawn")` this is handled by spawned slave daemons. Under `npRmpi.init(mode="attach")` (batch `mpiexec` workflows), worker ranks enter the same idle-loop coordination internally, so no external bootstrap is needed. On the master, `cmd` and `...` are put together as a list which is then broadcast (after serialization) to all slaves (using a for-loop with `mpi.send`/`mpi.recv`). All slaves return an expression that is evaluated by the worker loop.

If nonblock=TRUE, then on receiving side, a nonblock procedure is used to check if there is a message. If not, it will sleep for the specied amount and repeat itself.

Please use `mpi.remote.exec` if you want the executed results returned from R slaves.

Value

`mpi.bcast.cmd` returns no value for the sender and an expression of the transmitted command for others.

Warning

Be cautious of using `mpi.bcast.cmd` alone by master in the middle of comptuation. Only all slaves in idle states (waiting instructions from master) can be used. Othewise it may result in miscommunication with other MPI calls.

Author(s)

Hao Yu (minor modifications by Jeffrey S. Racine <racinej@mcmaster.ca>)

`mpi.bcast.Robj` *Extensions of MPI_Bcast API*

Description

`mpi.bcast.Robj` and `mpi.bcast.Robj2slave` are used to move a general R object around among master and all slaves.

Usage

```
mpi.bcast.Robj(obj = NULL, rank = 0, comm = 1)
mpi.bcast.Robj2slave(obj, comm = 1, all = FALSE)
```

Arguments

Object Payload: R object sent from the sender rank or master process.

`obj`	an R object to be transmitted from the sender

Communication Controls: Sender rank, communicator, and all-object broadcast control.

`rank`	the sender.
`comm`	a communicator number.
`all`	a logical. If TRUE, all R objects on master are transmitted to slaves.

Details

`mpi.bcast.Robj` is an extension of `mpi.bcast` for moving a general R object around from a sender to everyone. `mpi.bcast.Robj2slave` does an R object transmission from master to all slaves unless all=TRUE in which case, all master's objects with the global enviroment are transmitted to all slavers.

Value

`mpi.bcast.Robj` returns no value for the sender and the transmitted one for others. `mpi.bcast.Robj2slave` returns no value for the master and the transmitted R object along its name on slaves.

Author(s)

Hao Yu

See Also

`mpi.bcast`,

`mpi.close.Rslaves` *Close and Inspect R Slaves*

Description

`mpi.close.Rslaves` shuts down (or soft-closes) R slave daemons managed by `npRmpi`. `tailslave.log` shows tail output from slave log files.

Usage

```
mpi.close.Rslaves(dellog = TRUE, comm = 1, force = FALSE)
tailslave.log(nlines = 3, comm = 1)
```

Arguments

Slave Shutdown Controls: Communicator, log-deletion, and hard-shutdown controls for slave daemons.

`dellog`	a logical specifying if R slave log files are deleted.
`comm`	a communicator number.
`force`	a logical. If `TRUE`, force a hard shutdown of slave daemons. When `options(npRmpi.reuse.slaves=TRUE` and `force=FALSE`, `mpi.close.Rslaves()` performs a soft-close (keeps daemons alive for reuse).

Slave Log Inspection: Tail length used when inspecting slave log files.

`nlines`	number of lines shown from the tail of each slave log file.

Details

In normal user workflows, call `npRmpi.quit()` rather than using `mpi.close.Rslaves()` directly.

`tailslave.log()` is useful for debugging worker startup or teardown issues.

Value

`mpi.close.Rslaves` returns a status code (in soft-close mode this may be an invisible no-op code). `tailslave.log` returns the tail output from slave log files.

Author(s)

Hao Yu

See Also

`npRmpi.init`, `npRmpi.quit`.

Examples

```
## Not run:
# Inspect slave logs from the current communicator.
tailslave.log()

# Close active slave daemons.
mpi.close.Rslaves()

## End(Not run)
```

---

mpi.comm.free *MPI_Comm_free API*

Description

`mpi.comm.free` deallocates a communicator so it points to MPI_COMM_NULL.

Usage

```
mpi.comm.free(comm=1)
```

Arguments

Communicator Input: MPI communicator to deallocate.

`comm`	a communicator number

Details

When members associated with a communicator finish jobs or exit, they have to call `mpi.comm.free` to release resource so `mpi.comm.size` will return 0.

Value

1 if success. Otherwise 0.

Author(s)

Hao Yu

References

https://www.mpich.org/, https://www.mpich.org/static/docs/latest/www3/

---

mpi.comm.size — *MPI_Comm_dup, MPI_Comm_rank, and MPI_Comm_size APIs*

Description

`mpi.comm.dup` duplicates (copies) a comm to a new comm. `mpi.comm.rank` returns its rank in a comm. `mpi.comm.size` returns the total number of members in a comm.

Usage

```
mpi.comm.dup(comm, newcomm)
mpi.comm.rank(comm = 1)
mpi.comm.size(comm = 1)
```

Arguments

Communicator Inputs: Existing communicator and optional target communicator for duplication.

`comm`	a communicator number
`newcomm`	a new communicator number

Value

- `mpi.comm.dup`: integer identifier of the duplicated communicator.
- `mpi.comm.rank`: integer rank within the communicator.
- `mpi.comm.size`: integer size of the communicator.

Author(s)

Hao Yu

References

https://www.mpich.org/, https://www.mpich.org/static/docs/latest/www3/

Examples

```
## Not run:
## Not run in checks when toggled to dontrun: communicator examples are
## documented for manual MPI sessions.
mpi.comm.rank(comm=0)
mpi.comm.size(comm=0)
mpi.comm.dup(comm=0, newcomm=5)

## End(Not run)
```

`mpi.exit` *Exit MPI Environment*

Description

`mpi.exit` terminates MPI execution environment and detaches the package `npRmpi`. After that, you can still work in R.

`mpi.quit` terminates MPI execution environment and quits R.

Usage

```
mpi.exit()
mpi.quit(save = "no")
```

Arguments

Exit Controls: R-session save behavior used by `mpi.quit()`.

`save` the same argument as `quit` but default to "no".

Details

Normally, MPI finalization is used to clean all MPI states. However, it will not detach the loaded `npRmpi` package. To be safer when leaving MPI, `mpi.exit` not only calls `mpi.finalize` but also detaches the `npRmpi` package. This will make reloading `npRmpi` impossible in the same R session.

If leaving MPI and R altogether, one simply uses `mpi.quit`.

Value

`mpi.exit` always returns 1

Author(s)

Hao Yu

mpi.get.processor.name *MPI_Get_processor_name API*

Description

`mpi.get.processor.name` returns the host name (a string) where it is executed.

Usage

```
mpi.get.processor.name(short = TRUE)
```

Arguments

Hostname Format: Control for abbreviated versus full processor names.

`short`	a logical.

Value

a base host name if `short = TRUE` and a full host name otherwise.

Author(s)

Hao Yu

References

https://www.mpich.org/, https://www.mpich.org/static/docs/latest/www3/

mpi.get.version *MPI_Get_version API*

Description

`mpi.get.version` returns the runtime MPI API version as reported by `MPI_Get_version`.

Usage

```
mpi.get.version()
```

Value

An integer vector of length two named `major` and `minor`.

Author(s)

Hao Yu

References

https://www.mpich.org/, https://www.mpich.org/static/docs/latest/www3/

---

mpi.hostinfo *Host Information Utilities*

Description

mpi.hostinfo prints host and rank information for the calling rank. slave.hostinfo prints host/rank summaries for active slave ranks.

Usage

```
mpi.hostinfo(comm = 1)
slave.hostinfo(comm = 1, short = TRUE)
```

Arguments

Host-Information Controls: Communicator and output-abbreviation controls for host/rank summaries.

comm	a communicator number.
short	a logical; if TRUE, abbreviate output when there are many slaves.

Details

slave.hostinfo() must be called on rank 0 of the target communicator.

Value

Both functions print informational output and return invisibly.

Author(s)

Hao Yu

See Also

mpi.get.processor.name, mpi.comm.size, mpi.comm.rank.

np.kernels *Kernel Functions Used In* **npRmpi**

Description

Summary of continuous, unordered-categorical, and ordered-categorical kernels used by **npRmpi** (including higher-order continuous kernels and compact-support variants used in C-level code paths).

Details

For MPI startup/performance guidance (including message-passing tradeoffs and the manual-broadcast template), see npRmpi.init details and inst/Rprofile. Documentation guide: see np.options for global options For interactive and cluster batch workflows, see npRmpi.init. and plot for plotting options.

Kernel option names used in **npRmpi**:

- Continuous kernels: ckertype (and ckerorder, ckerbound where applicable).
- Unordered kernels: ukertype.
- Ordered kernels: okertype.
- Conditional density/distribution bandwidth objects split kernel choices by response and regressor blocks: cykertype/cxkertype, uykertype/uxkertype, oykertype/oxkertype (with matching order/bound options for continuous kernels).

Let $u = (x_i - x)/h$ for continuous variables.

Continuous kernels (called via ckertype):

$$K_{G,2}(u) = \phi(u)$$

$$K_{G,4}(u) = \left(\frac{3}{2} - \frac{1}{2}u^2\right)\phi(u)$$

$$K_{G,6}(u) = \left(\frac{15}{8} - \frac{5}{4}u^2 + \frac{1}{8}u^4\right)\phi(u)$$

$$K_{G,8}(u) = \left(\frac{35}{16} - \frac{35}{16}u^2 + \frac{7}{16}u^4 - \frac{1}{48}u^6\right)\phi(u)$$

where $\phi(u)$ is the standard normal density.

ckertype="gaussian" with ckerorder=2,4,6,8.

The compact-support Epanechnikov-family kernels implemented in C use support $|u| < \sqrt{5}$:

$$K_{E,2}(u) = \frac{3}{4\sqrt{5}}\left(1 - \frac{u^2}{5}\right)\mathbf{1}(|u| < \sqrt{5})$$

$$K_{E,4}(u) = 0.008385254916(-15 + 7u^2)(-5 + u^2)\mathbf{1}(u^2 < 5)$$

$$K_{E,6}(u) = 0.33541019662496845446\left(2.734375 - 3.28125u^2 + 0.721875u^4\right)\left(1 - 0.2u^2\right)\mathbf{1}(u^2 < 5)$$

$$K_{E,8}(u) = 0.33541019662496845446\left(3.5888671875 - 7.8955078125u^2 + 4.1056640625u^4 - 0.5865234375u^6\right)\left(1 - 0.\right.$$

`ckertype="epanechnikov"` with `ckerorder=2,4,6,8`.

Uniform (rectangular) kernel:

$$K_U(u) = \frac{1}{2}\mathbf{1}(|u| < 1)$$

via `ckertype="uniform"` (order ignored).

Truncated-Gaussian (second-order) kernel via `ckertype="truncated gaussian"`:

$$K_{TG,2}(u) = [\alpha\phi(u) - c_0]\,\mathbf{1}(|u| < b)$$

with defaults $b = 3$ and internal constants calibrated in C.

Bounded continuous-kernel normalization (`ckerbound` and, for conditional objects, `cxkerbound`/`cykerbound`) reuses the selected continuous kernel and renormalizes it on the declared support. For a base kernel K and support $[a, b]$, the bounded kernel is

$$K_{[a,b]}(u; x, h) = \frac{K(u)}{\int_{(a-x)/h}^{(b-x)/h} K(t)dt}$$

with $u = (x_i - x)/h$. Option `ckerbound="range"` uses sample bounds for a, b; `ckerbound="fixed"` uses user-supplied bounds via `ckerlb`/`ckerub` (or the corresponding `cx*`/`cy*` arguments). Infinite bounds recover the unbounded kernel. This support-normalization strategy follows the same Racine-Li-Yan finite-support normalization principle and is useful when data exhibit non-negligible probability mass near boundaries.

Computationally, let $H_K(r) = \int_0^r K(t)dt$ and $S_K(r) = H_K(r)/r$, with the continuous value $S_K(0) = K(0)$. For finite bounds the implementation evaluates the scaled normalization mass directly as

$$hD(x, h) = (x - a)S_K((x - a)/h) + (b - x)S_K((b - x)/h).$$

Each supported kernel uses its analytic centered primitive. This avoids subtracting nearly equal whole-kernel CDF values and avoids constructing a normalization mass that vanishes numerically as h grows. Consequently, on finite $[a, b]$ the bounded density kernel converges to the uniform density $1/(b - a)$ as $h \to \infty$. One-sided infinite bounds use the corresponding exact half-mass term; two infinite bounds retain the ordinary unbounded kernel path.

For distribution estimation, finite bounds use the distinct observation-centered truncated-kernel CDF

$$B_i(y; h) = \frac{G((y - Y_i)/h) - G((a - Y_i)/h)}{G((b - Y_i)/h) - G((a - Y_i)/h)}, \quad y \in [a, b].$$

Thus $B_i(a; h) = 0$, $B_i(b; h) = 1$, and $B_i(y; h) \to (y - a)/(b - a)$ as $h \to \infty$. The implementation evaluates centered analytic interval masses rather than subtracting nearly equal whole-kernel CDF values. This distribution operator is not the integral of the evaluation-centered bounded density kernel above. For nonnegative second-order kernels each contribution is monotone and lies in $[0, 1]$; higher-order signed kernels retain their usual possible interior nonmonotonicity or overshoot.

Typical bounded-kernel calls:

```
## Unconditional density on [0,1]
bw <- npudensbw(dat=data.frame(x),
                ckertype="gaussian",
```

```
                ckerbound="fixed", ckerlb=0, ckerub=1)

## Regression with automatic sample-range bounds
bw <- npregbw(xdat=data.frame(x), ydat=y, ckerbound="range")

## Conditional density with separate x/y support controls
bw <- npcdensbw(xdat=data.frame(x), ydat=data.frame(y),
                cxkerbound="fixed", cxkerlb=0, cxkerub=1,
                cykerbound="range")
```

Unordered-categorical kernels (called via `ukertype`; for category count c):

$$L_{AA}(x_i, x; \lambda) = \mathbf{1}(x_i = x)(1-\lambda) + \mathbf{1}(x_i \neq x)\frac{\lambda}{c-1}$$

(Aitchison-Aitken) via `ukertype="aitchisonaitken"`.

$$L_{LR,u}(x_i, x; \lambda) = \mathbf{1}(x_i = x) + \mathbf{1}(x_i \neq x)\lambda$$

(Li-Racine unordered kernel) via `ukertype="liracine"`.

Ordered-categorical kernels (called via `okertype`):

$$L_{WvR}(x_i, x; \lambda) = \begin{cases} 1-\lambda, & x_i = x \\ \frac{1-\lambda}{2}\lambda^{|x_i - x|}, & x_i \neq x \end{cases}$$

(Wang-van Ryzin) via `okertype="wangvanryzin"`.

$$L_{LR,o}(x_i, x; \lambda) = \lambda^{|x_i - x|}$$

(Li-Racine ordered kernel) via `okertype="liracine"`.

$$L_{NLR,o}(x_i, x; \lambda) = \lambda^{|x_i - x|}\frac{1-\lambda}{1+\lambda}$$

(normalized Li-Racine ordered kernel; used internally)

$$L_{RLY}(x_i, x; \lambda) = \frac{\lambda^{|x_i - x|}}{\sum_{z \in \mathcal{S}(x)} \lambda^{|x_i - z|}}$$

(Racine-Li-Yan ordered kernel, normalized on support $\mathcal{S}(x)$). exposed as `okertype="racineliyan"`.

These univariate kernels are combined as generalized product kernels over mixed data types in the estimators and cross-validation criteria.

References

Aitchison, J. and Aitken, C. G. G. (1976), "Multivariate binary discrimination by the kernel method," *Biometrika*, **63**, 413–420.

Wang, M. C. and Van Ryzin, J. (1981), "A class of smooth estimators for discrete distributions," *Biometrika*, **68**, 301–309.

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

Racine, J. S. and Li, Q. (2004), "Nonparametric estimation of regression functions with both categorical and continuous data," *Journal of Econometrics*, **119**, 99–130.

Racine, J. S., Li, Q., and Yan, K. X. (2020), "Kernel Smoothed Probability Mass Functions for Ordered Datatypes," *Journal of Nonparametric Statistics*, **32**(3), 563–586. doi:10.1080/10485252.2020.1759595

Hall, P., Racine, J. S., and Li, Q. (2004), "Cross-validation and the estimation of conditional probability densities," *Journal of the American Statistical Association*, **99**, 1015–1026.

See Also

`np.options`, `plot` `npregbw`, `npudensbw`, `npudistbw`, `npcdensbw`, `npcdistbw`, `npksum`, `np.options`.

`np.mpi.initialize` *Initialize Ranks for Manual-Broadcast npRmpi Workflows*

Description

`np.mpi.initialize` initializes the caller and worker ranks for the profile/manual-broadcast **npRmpi** workflow.

Usage

```
np.mpi.initialize()
```

Details

`np.mpi.initialize()` is the helper used after ranks have already been started with the profile/manual-broadcast route. The usual pattern is: `mpi.bcast.cmd(np.mpi.initialize(), caller.execute=TRUE)`.

This helper is *not* the ordinary entry point for session or attach workflows. For those routes, use `npRmpi.init` and `npRmpi.quit` instead.

For MPI startup/performance guidance (including message-passing tradeoffs and the manual-broadcast template), see `npRmpi.init` details and `inst/Rprofile`. Documentation guide: see `np.kernels` for kernels, `np.options` for global options, and `plot` for plotting options.

Value

`np.mpi.initialize` returns no value for the sender and an expression of the transmitted command for others.

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

See Also

`np.kernels`, `np.options`, `plot`, `npRmpi.init`.

`np.options` *Global Package Options for* **npRmpi**

Description

Global options controlling selected computational and display behavior for the **npRmpi** package.

Details

For MPI startup/performance guidance (including message-passing tradeoffs and the manual-broadcast template), see `npRmpi.init` details and `inst/Rprofile`. Documentation guide: see `np.kernels` for kernels and `plot` for plotting options.

The following options are recognized by **npRmpi**.

Output and Plot Progress:

- `np.messages` (logical): controls console/progress output. Default is `TRUE`.
- `np.plot.progress` (logical): controls bounded plot/bootstrap progress heartbeats on the master rank. Default is `TRUE`.
- `np.plot.progress.start.grace.sec` (numeric): delay before the first plot/bootstrap progress line is shown. Default is `0.75`.
- `np.plot.progress.interval.sec` (numeric): minimum elapsed time between plot/bootstrap heartbeat lines once progress reporting has started. Default is `0.5`.
- `np.plot.progress.max.intermediate` (integer): maximum number of mid-run plot/bootstrap heartbeat lines emitted between the initial start notice and final completion line. Default is `3`.

Kernel Routing and Acceleration:

- `np.tree` (logical or `"auto"`): controls kd-tree acceleration for continuous variables. `TRUE` forces eligible tree routes on, `FALSE` disables them, and `"auto"` enables trees only when active continuous kernels are bounded support (`"epanechnikov"` or `"uniform"`). Default is `"auto"`.
- `np.categorical.compress` (logical): enables exact unique-profile compression for eligible all-categorical routes. This is distinct from `np.tree`: no continuous kd-tree is requested merely because categorical compression is enabled. The default is `TRUE`. Set `options(np.categorical.compress = FALSE, np.tree = FALSE)` to force the historical dense categorical route for debugging or timing comparisons.
- `np.macMseries.accelerate` (logical or `"auto"`): controls macOS arm64 Accelerate/vDSP/vForce fast paths for eligible Gaussian-kernel and weighted-reduction computations. `"auto"` enables these paths only when they were compiled for macOS arm64 with Accelerate support; `TRUE` requests them when available; and `FALSE` disables them exactly. Default is `"auto"`. On non-macOS, non-arm64, or non-Accelerate builds, this option falls back to the ordinary portable code path. The active option value is synchronized to MPI workers in autodispatch sessions. Accelerated and portable paths may use different math-library kernels or summation orders, so platform or option comparisons should use numerical tolerances; reserve exact equality for controlled runs using the same acceleration setting.

Search and Objective Caching:

- `np.objective.cache` (logical): enables package-side evaluation caching for eligible repeated bandwidth-search candidates. For all Powell routes using the shared C optimizer wrapper, exact repeated raw candidate vectors within one bandwidth solve reuse the previous objective value, including repeated candidates reached by later Powell restarts. For eligible continuous-only generalized/adaptive nearest-neighbor routes, this also enables the existing integer nearest-neighbor objective cache. Mixed continuous/categorical rows use only the exact full-candidate cache, so categorical smoothing parameters remain part of the cache key. Default is `TRUE`. Eligible continuous `npscoefbw` and `npindexbw` routes also use package-side caches around their family-native R `optim` evaluations. Set `options(np.objective.cache = FALSE)` to force repeated eligible candidates through the evaluator for diagnostic timing comparisons. This option is synchronized to MPI workers in autodispatch sessions. MADS/NOMAD solver caching is separate and is not controlled by this option, and the saturated extended-NN distance cache is governed by `np.extendednn`. Bandwidth summaries keep `Number of Function Evaluations` as the established cumulative optimizer evaluation count and report NOMAD, Powell, R-`optim`, nearest-neighbor, degree-search, guarded-evaluation, and other fast-CV diagnostics on separate lines when the corresponding counters are available. These diagnostics describe their own optimizer/cache layer and are not intended to decompose the function-evaluation count arithmetically.
- `np.nomad.degree.start.policy` (character): controls automatic polynomial-degree starting values for NOMAD local-polynomial searches when `degree.start` is not supplied. The default, `"low_first_full_random"`, uses `degree.min` for the first start and draws later starts over the full user-specified `degree.min`/`degree.max` range. Other accepted values are `"mid_first_full_random"`, `"anchor_then_random"`, `"spread_then_random"`, and `"random_full_only"`. Explicit `degree.start` values remain the first start. The active option value is synchronized to MPI workers in autodispatch sessions.

Bandwidth Shortcuts:

- `np.largeh` (logical): enables continuous large-h shortcut evaluations. Default is `TRUE`. Set `options(np.largeh = FALSE)` to keep tree/dense routing unchanged while forcing continuous large-bandwidth objective rows through the ordinary kernel-summation machinery for diagnostic timing comparisons.
- `np.extendednn` (logical): enables extended nearest-neighbor bandwidth search and manually supplied nearest-neighbor bandwidths above the ordinary $n-1$ neighbor-count limit for eligible generalized/adaptive nearest-neighbor routes. When `TRUE`, a nearest-neighbor search parameter above $n-1$ scales the saturated nearest-neighbor bandwidth. Default is `TRUE`. This option is synchronized to MPI workers in autodispatch sessions.
- `np.largeh.rel.tol` (numeric): relative tolerance used by the continuous large-h shortcut. When all standardized distances for a continuous predictor are sufficiently close to zero, the corresponding kernel factor is approximated by $K(0)$ to reduce repeated kernel evaluations. Default is `1e-3`. Valid range is `(0, 0.1)`.
- `np.largelambda` (logical): enables discrete near-upper bandwidth shortcut evaluations. Default is `TRUE`. Set `options(np.largelambda = FALSE)` to keep categorical routing unchanged while forcing near-upper discrete bandwidth rows through the ordinary category-comparison machinery for diagnostic timing comparisons.
- `np.disc.upper.rel.tol` (numeric): relative tolerance used by the discrete upper-bound shortcut for bandwidths near their feasible upper bounds. The near-upper check is applied relative to each kernel's own feasible upper bound (e.g., Aitchison-Aitken depends on category cardinality), with a tiny machine-precision floor for numerical robustness. When

same/different-category kernel values are numerically close, the corresponding discrete kernel factor is treated as constant to reduce repeated category comparisons. Default is `1e-2`. Valid range is `(0, 0.5)`.

Plot Layout:

- `plot.par.mfrow` (logical): used by `plot` to determine whether plotting layout is automatically managed via `par(mfrow=...)`. If `NULL` (default behavior), **npRmpi** uses its internal plotting defaults.

MPI Dispatch:

- `npRmpi.autodispatch` (logical): when `TRUE`, eligible `np*` calls are auto-dispatched across MPI ranks without explicit `mpi.bcast.cmd(...)` wrapping. Default is `FALSE`. For formula interfaces under autodispatch, provide explicit `data=` (or explicit `xdat`/`ydat`-style arguments) to avoid unresolved-symbol failures on slave ranks. Changing the active worker count can change summation/reduction order at roundoff scale; compare cross-worker-count bandwidths or objective values with numerical tolerances unless running a controlled single-process comparison.

Option values can be set globally via `options` and restored with `on.exit` in scripts/functions for reproducibility.

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

See Also

`npRmpi.init`. `np.kernels`, `plot npRmpi`, `plot`, `options`

Examples

```
## Not run:
npRmpi.init(nslaves=1)

old <- options(
  np.tree = "auto",
  np.categorical.compress = TRUE,
  np.macMseries.accelerate = "auto",
  np.objective.cache = TRUE,
  np.largeh = TRUE,
  np.extendednn = TRUE,
  np.largelambda = TRUE,
  np.messages = FALSE,
  np.largeh.rel.tol = 1e-3,
  np.disc.upper.rel.tol = 1e-2
)
on.exit(options(old), add = TRUE)
on.exit(npRmpi.quit(force=TRUE), add = TRUE)

## ... run bandwidth selection / estimation ...

## End(Not run)
```

np.pairs — *Cross-Validated Pairs Plot (Helper Functions)*

Description

Compute pairwise nonparametric regressions and densities for a set of variables, then plot a pairs-style display with fitted smoothers.

Usage

```
np.pairs(y_vars, y_dat, ...)
np.pairs.plot(pair_list)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the variables, data, and pair specifications to plot.

`pair_list`	list returned by `np.pairs`.
`y_dat`	data frame containing the variables listed in `y_vars`.
`y_vars`	character vector of column names in `y_dat`. If `y_vars` is named, the names are used as plot labels.

Additional Arguments: Further graphical arguments are passed through to plotting methods.

`...`	additional arguments passed to `npudens` and `npreg`.

Details

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, `plot` for plotting options, and `npRmpi.init` for interactive/cluster MPI startup. See `npRmpi.init` details for performance tradeoffs (message passing/startup mode) and the `inst/Rprofile` manual-broadcast template.

On the diagonal, `npudens` is used to compute kernel density estimates. Off-diagonal panels use `npreg` with residuals to draw scatterplots and smoothers.

Value

`np.pairs` returns a list with components `y_vars`, `pair_names`, and `pair_kerns`. `np.pairs.plot` returns `NULL` (invisibly).

See Also

`np.kernels`, `np.options`, `plotnpudens`, `npreg`

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

data("USArrests")
y_vars <- c("Murder", "UrbanPop")
names(y_vars) <- c("Murder Arrests per 100K", "Pop. Percent Urban")

pair_list <- np.pairs(y_vars = y_vars, y_dat = USArrests,
                      ckertype = "epanechnikov",
                      bwscaling = TRUE)

np.pairs.plot(pair_list)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
```

```
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

`np_plot_controls` *Validated plot control helpers*

Description

Constructors for detailed `plot` controls used by the redesigned **npRmpi** plotting interface. Common choices such as `errors`, `band`, `alpha`, `bootstrap`, and `B` remain direct `plot` arguments; these helpers collect less common options.

Usage

```
np_boot_control(nonfixed = c("exact", "frozen"),
                wild = c("rademacher", "mammen"),
                blocklen = NULL)

np_grid_control(xtrim = NULL, xq = NULL, slices = NULL)

np_render_control(style = c("band", "bar"),
                  bar = c("|", "I"),
                  bar_num = NULL)
```

Arguments

`nonfixed`	Bootstrap refit behavior for non-fixed bandwidth routes.
`wild`	Wild-bootstrap multiplier distribution.
`blocklen`	Optional block length for block bootstrap methods.
`xtrim`	Optional two-element trim range for evaluation grids.
`xq`	Optional evaluation quantiles.
`slices`	Optional slice specification for routes that support it.
`style`	Interval display style.
`bar`	Bar style when `style = "bar"`.
`bar_num`	Optional number of bars.

Details

These helpers validate names and values early. Raw lists are deliberately not accepted as substitutes, so misspelled detailed controls fail before plot computation or MPI fanout begins.

New public plot arguments use `snake_case`. Dotted argument names are reserved for S3 methods and internal plot-engine plumbing. Common rendering controls such as `data_overlay`, `data_rug`, `layout`, and `output` are supplied directly to `plot(...)` rather than through `np_render_control()`.

Value

A small classed list consumed by `plot` methods.

Examples

```
## Not run:
plot(fit,
     errors = "bootstrap",
     bootstrap = "inid",
     B = 1999,
     band = "pointwise",
     boot_control = np_boot_control(nonfixed = "exact"))

## End(Not run)
```

`npcdens` *Kernel Conditional Density Estimation with Mixed Data Types*

Description

`npcdens` computes kernel conditional density estimates on $p+q$-variate evaluation data, given a set of training data (both explanatory and dependent) and a bandwidth specification (a `conbandwidth` object or a bandwidth vector, bandwidth type, and kernel type) using the method of Hall, Racine, and Li (2004). The data may be continuous, discrete (unordered and ordered factors), or some combination thereof.

Usage

```
npcdens(bws, ...)

## S3 method for class 'formula'
npcdens(bws, data = NULL, newdata = NULL, ...)


## S3 method for class 'conbandwidth'
npcdens(bws,
        txdat = stop("invoked without training data 'txdat'"),
        tydat = stop("invoked without training data 'tydat'"),
        exdat,
        eydat,
        gradients = FALSE,
        gradient.order = 1L,
        proper = FALSE,
        proper.method = c("project"),
        proper.control = list(),
        ...)
```

```
## Default S3 method:
npcdens(bws, txdat, tydat, nomad = FALSE, ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and training data.

`bws` a bandwidth specification. This can be set as a `conbandwidth` object returned from a previous invocation of `npcdensbw`, or as a $p+q$-vector of bandwidths, with each element i up to $i=q$ corresponding to the bandwidth for column i in `tydat`, and each element i from $i=q+1$ to $i=p+q$ corresponding to the bandwidth for column $i-q$ in `txdat`. If specified as a vector, then additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, training data, and so on.

`data` an optional data frame, list or environment (or object coercible to a data frame by `as.data.frame`) containing the variables in the model. If not found in data, the variables are taken from `environment(bws)`, typically the environment from which `npcdensbw` was called.

`txdat` a p-variate data frame of sample realizations of explanatory data (training data). Defaults to the corresponding training data recoverable from the bandwidth object in the current session.

`tydat` a q-variate data frame of sample realizations of dependent data (training data). Defaults to the corresponding training data recoverable from the bandwidth object in the current session.

Local-Polynomial Degree And Bandwidth Search: This argument controls the recommended automatic local-polynomial NOMAD route, which jointly selects continuous polynomial degree and bandwidths when these are computed inside `npcdens`.

`nomad` logical shortcut passed through to `npcdensbw` when bandwidths are computed inside `npcdens`. When `TRUE`, the bandwidth route fills any missing values among `regtype`, `search.engine`, `degree.select`, `bernstein.basis`, `degree.min`, `degree.max`, `degree.verify`, and `bwtype` with the recommended automatic local-polynomial degree-and-bandwidth NOMAD preset documented in `npcdensbw`. Additional NOMAD tuning arguments such as `nomad.nmulti` may also be supplied through `...`; `nmulti` remains the outer restart count while `nomad.nmulti` controls inner native **crs** NOMAD API multistarts within each outer restart. After fitting, inspect `fit$bws$nomad.shortcut` on the returned object `fit` to see the normalized shortcut metadata.

Evaluation Data And Returned Quantities: These arguments control where the fitted conditional density is evaluated and which estimates are returned.

`exdat` a p-variate data frame of explanatory data on which conditional densities will be evaluated. By default, evaluation takes place on the data provided by `txdat`.

`eydat` a q-variate data frame of dependent data on which conditional densities will be evaluated. By default, evaluation takes place on the data provided by `tydat`.

`gradients` a logical value specifying whether to return estimates of the gradients at the evaluation points. Defaults to `FALSE`.

`gradient.order` derivative order for continuous explanatory-variable gradients when `gradients = TRUE` and `regtype = "lp"`. A scalar is recycled across continuous explanatory variables, or one value may be supplied per continuous explanatory variable. The default `1L` returns first derivatives. Higher orders are available for continuous explanatory variables only and must not exceed the corresponding local-polynomial degree; unordered and ordered predictors retain their usual first-order discrete effects.

`newdata` An optional data frame in which to look for evaluation data. If omitted, the training data are used.

Fit Properization Controls: These arguments control optional post-estimation properization of the fitted conditional density.

`proper` a logical value specifying whether to post-process the estimated conditional density so that it integrates to one over the evaluation grid. Defaults to `FALSE`.

`proper.control` a named list of control parameters for properization. Supported entries are `tol`, `grid.check`, `store.raw`, and `fail.on.unsupported`.

`proper.method` the properization method. Currently only `"project"` is supported.

Additional Arguments: Further arguments are passed to `npcdensbw` when bandwidths are computed internally, or used to interpret a numeric `bws` vector.

`...` additional arguments supplied to `npcdensbw` when `npcdens` computes bandwidths internally, or arguments needed to interpret a numeric `bws` vector. This is where bandwidth selection controls such as `bwmethod`, `bwtype`, kernel/support controls such as `cxkertype`, `cykertype`, `cxkerbound`, and `cykerbound`, search controls such as `nmulti`, `scale.factor.search.lower`, and `nomad.nmulti`, and local-polynomial controls such as `regtype`, `degree`, `basis`, and `bernstein.basis` are supplied. See `npcdensbw` for the complete bandwidth-selection argument surface.

Details

Documentation guide: see `npcdensbw` for bandwidth selection and search controls, `np.kernels` for kernels, `np.options` for global options, `plot`, `plot.np` for plotting options, and `npRmpi.init` for interactive/cluster MPI startup. See `npRmpi.init` details for performance tradeoffs (message passing/startup mode) and the `inst/Rprofile` manual-broadcast template.

When `bws` is omitted, the formula and default methods call `npcdensbw` first and pass bandwidth-selection arguments from `...` to that call. When `bws` is already a `conbandwidth` object, `npcdens` estimates with the stored bandwidth metadata in that object.

Argument groups for bandwidth selection are documented on `npcdensbw`. The most common workflow is to initialize MPI execution if needed, choose data and bandwidth inputs, then bandwidth criterion and representation, then kernel/support controls, numerical search controls, bounded `cv.ls` quadrature controls if relevant, and finally local-polynomial/NOMAD controls for polynomial-adaptive fits.

For S3 plotting help, see plot.np. You can list available plot methods with methods("plot").

npcdens implements a variety of methods for estimating multivariate conditional distributions ($p+q$-variate) defined over a set of possibly continuous and/or discrete (unordered, ordered) data. The approach is based on Li and Racine (2004) who employ 'generalized product kernels' that admit a mix of continuous and discrete data types.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i, when estimating the density at the point x. Generalized nearest-neighbor bandwidths change with the point at which the density is estimated, x. Fixed bandwidths are constant over the support of x.

Training and evaluation input data may be a mix of continuous (default), unordered discrete (to be specified in the data frames using factor), and ordered discrete (to be specified in the data frames using ordered). Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see npRmpi for details).

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth order Gaussian and Epanechnikov kernels, and the uniform kernel. Unordered discrete data types use a variation on Aitchison and Aitken's (1976) kernel, while ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

For practitioners who want the recommended automatic local-polynomial degree-and-bandwidth NOMAD route without spelling out all LP tuning arguments, npcdens(..., nomad=TRUE) and npcdensbw(..., nomad=TRUE) expand missing settings to the same documented preset. Explicit incompatible settings fail fast rather than being silently rewritten.

With regtype = "lp", gradients = TRUE and gradient.order greater than one expose higher-order derivative estimates with respect to continuous explanatory variables. These derivatives use the same local-polynomial basis, degree, Bernstein option, bandwidths, and kernels as the fitted conditional density. Asymptotic standard errors for these higher-order derivative columns are not currently reported; the corresponding entries of congerr are NA. Use bootstrap intervals when inference on higher-order derivatives is required.

Value

npcdens returns a condensity object. The generic accessor functions fitted, se, and gradients, extract estimated values, asymptotic standard errors on estimates, and gradients, respectively, from the returned object. Furthermore, the functions predict, summary and plot support objects of both classes. The returned objects have the following components:

xbw	bandwidth(s), scale factor(s) or nearest neighbours for the explanatory data, txdat
ybw	bandwidth(s), scale factor(s) or nearest neighbours for the dependent data, tydat
xeval	the evaluation points of the explanatory data
yeval	the evaluation points of the dependent data
condens	estimates of the conditional density at the evaluation points
conderr	standard errors of the conditional density estimates
congrad	if invoked with gradients = TRUE, estimates of the gradients at the evaluation points. For local-polynomial fits, continuous-coordinate derivative orders are controlled by gradient.order.

congerr	if invoked with gradients = TRUE, standard errors of the gradients at the evaluation points. Higher-order continuous-coordinate derivative standard errors are currently returned as NA; bootstrap inference is preferred for those targets.
log_likelihood	log likelihood of the conditional density estimate

Book And Method Pointers

The conditional density target is $f(y \mid x)$. In the product-kernel formulation this can be viewed as estimating the joint mixed-data density of (Y, X) and normalizing by the marginal density of X, with local-polynomial routes using the selected continuous-coordinate polynomial degree in the conditional fit. The returned standard errors are asymptotic standard errors for the fitted conditional density values, and gradient standard errors are returned when gradients are requested and defined.

For book-length derivations, see Li and Racine (2007), Chapter 5 *Conditional Density Estimation*, especially Sections 5.1-5.4, and Chapter 4 *Kernel Estimation with Mixed Data*. The later workflow treatment is Racine (2019), Chapter 4 *Conditional Probability Density and Cumulative Distribution Functions*.

Usage Issues

If you are using data of mixed types, then it is advisable to use the data.frame function to construct your input data and not cbind, since cbind will typically not work as intended on mixed data types and will coerce the data to the same type.

When reusing a bandwidth object without explicit training or evaluation data, the required data are recovered from object-retained state or from the stored call environment when available. This is a convenience for ordinary same-session use, not a guarantee that a saved object is self-contained. For reproducible replay after saveRDS() and readRDS() in a fresh R session, supply the required data arguments explicitly.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

Hall, P. and J.S. Racine and Q. Li (2004), "Cross-validation and the estimation of conditional probability densities," Journal of the American Statistical Association, 99, 1015-1026.

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

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

Scott, D.W. (1992), *Multivariate Density Estimation. Theory, Practice and Visualization,* New York: Wiley.

Silverman, B.W. (1986), *Density Estimation,* London: Chapman and Hall.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

np.kernels, np.options, plot, plot.np npudens

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

data("Italy")

bw <- npcdensbw(formula=gdp~ordered(year), data=Italy)

fhat <- npcdens(bws=bw)

summary(fhat)

## Variations on local polynomial conditional density estimation with
## proper = TRUE.

Italy2 <- within(Italy, {
  year <- as.numeric(as.character(year))
})

## Plot only: make the plotted surface proper on the plot evaluation grid.

fhat <- npcdens(gdp ~ year, data = Italy2,
                regtype = "lp", degree = 3, nmulti = 1)
```

```
plot(fhat, proper = TRUE)

## Fit an object whose fitted values are themselves proper.

ctrl_fit <- list(
  mode = "slice",
  apply = "fitted",
  slice.grid.size = 101L,
  slice.extend.factor = 0.1
)

fhat_fit <- npcdens(
  gdp ~ year,
  data = Italy2,
  regtype = "lp",
  degree = 3,
  nmulti = 1,
  proper = TRUE,
  proper.control = ctrl_fit
)

fit_proper <- fitted(fhat_fit)
fit_raw <- fhat_fit$condens.raw

## Display the repaired and raw fitted values for cases where the raw
## fitted density is negative.

head(cbind(fit_proper, fit_raw)[which(fit_raw < 0), ])

## Predict on a common explicit y-grid for several years, and render
## those predictions proper.

g.grid <- seq(min(Italy2$gdp), max(Italy2$gdp), length.out = 200)

nd_grid <- expand.grid(
  gdp = g.grid,
  year = c(1955, 1975, 1995)
)

pred_grid <- predict(fhat, newdata = nd_grid, proper = TRUE)

## Predict on paired rows with different gdp grids by year, and still
## make the predictions proper via slice mode.

g1 <- seq(quantile(Italy2$gdp, 0.10),
          quantile(Italy2$gdp, 0.60), length.out = 60)
g2 <- seq(quantile(Italy2$gdp, 0.30),
          quantile(Italy2$gdp, 0.90), length.out = 35)

nd_slice <- rbind(
  data.frame(gdp = g1, year = rep(1960, length(g1))),
  data.frame(gdp = g2, year = rep(1985, length(g2)))
)
```

```
pred_slice <- predict(
  fhat,
  newdata = nd_slice,
  proper = TRUE,
  proper.control = list(mode = "slice")
)

## One object that carries properization for fitted values and for later
## predict() calls.

ctrl_both <- list(
  mode = "slice",
  apply = "both",
  slice.grid.size = 101L,
  slice.extend.factor = 0.1
)

fhat_both <- npcdens(
  gdp ~ year,
  data = Italy2,
  regtype = "lp",
  degree = 3,
  nmulti = 1,
  proper = TRUE,
  proper.control = ctrl_both
)

fit_both <- fitted(fhat_both)
pred_both <- predict(
  fhat_both,
  newdata = nd_slice,
  proper.control = ctrl_both
)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}
```

```
## End(Not run)
```

---

npcdensbw *Kernel Conditional Density Bandwidth Selection with Mixed Data Types*

Description

npcdensbw computes a conbandwidth object for estimating the conditional density of a $p + q$-variate kernel density estimator defined over mixed continuous and discrete (unordered, ordered) data using either the normal-reference rule-of-thumb, likelihood cross-validation, or least-squares cross validation using the method of Hall, Racine, and Li (2004).

Usage

```
npcdensbw(...)

## S3 method for class 'formula'
npcdensbw(formula,
          data,
          subset,
          na.action,
          call,
          ...)

## S3 method for class 'conbandwidth'
npcdensbw(xdat = stop("data 'xdat' missing"),
          ydat = stop("data 'ydat' missing"),
          bws,
          bandwidth.compute = TRUE,
          cfac.dir = 2.5*(3.0-sqrt(5)),
          scale.factor.init = 0.5,
          dfac.dir = 0.25*(3.0-sqrt(5)),
          dfac.init = 0.375,
          dfc.dir = 3,
          ftol = 1.490116e-07,
          scale.factor.init.upper = 2.0,
          hbd.dir = 1,
          hbd.init = 0.9,
          initc.dir = 1.0,
          initd.dir = 1.0,
          invalid.penalty = c("baseline","dbmax"),
          itmax = 10000,
          lbc.dir = 0.5,
          scale.factor.init.lower = 0.1,
          lbd.dir = 0.1,
```

```
            lbd.init = 0.1,
            memfac = 500,
            nmulti,
            penalty.multiplier = 10,
            powell.remin = TRUE,
            bwsolver = c("powell", "mads", "mads+powell"),
            scale.init.categorical.sample = FALSE,
            scale.factor.search.lower = NULL,
            cvls.quadrature.grid = NULL,
            cvls.quadrature.extend.factor = NULL,
            cvls.quadrature.points = NULL,
            cvls.quadrature.ratios = NULL,
            small = 1.490116e-05,
            tol = 1.490116e-04,
            transform.bounds = FALSE,
            ...,
            nomad.opts = list())

## Default S3 method:
npcdensbw(xdat = stop("data 'xdat' missing"),
            ydat = stop("data 'ydat' missing"),
            bws,
            bandwidth.compute = TRUE,
            bwmethod,
            bwscaling,
            bwtype,
            cfac.dir,
            scale.factor.init,
            cxkerbound,
            cxkerlb,
            cxkerorder,
            cxkertype,
            cxkerub,
            cykerbound,
            cykerlb,
            cykerorder,
            cykertype,
            cykerub,
            dfac.dir,
            dfac.init,
            dfc.dir,
            ftol,
            scale.factor.init.upper,
            hbd.dir,
            hbd.init,
            initc.dir,
            initd.dir,
            invalid.penalty,
```

```
itmax,
lbc.dir,
scale.factor.init.lower,
lbd.dir,
lbd.init,
memfac,
nmulti,
oxkertype,
oykertype,
penalty.multiplier,
nomad.remin = FALSE,
powell.remin,
bwsolver = c("powell", "mads", "mads+powell"),
scale.init.categorical.sample,
scale.factor.search.lower = NULL,
cvls.quadrature.grid = c("hybrid", "uniform", "sample"),
cvls.quadrature.extend.factor = 1,
cvls.quadrature.points = c(100L, 50L),
cvls.quadrature.ratios = c(0.20, 0.55, 0.25),
small,
tol,
transform.bounds,
uxkertype,
uykertype,
regtype = c("lc", "ll", "lp"),
basis = c("glp", "additive", "tensor"),
degree = NULL,
degree.select = c("manual", "coordinate", "exhaustive"),
search.engine = c("nomad+powell", "cell", "nomad"),
nomad = FALSE,
nomad.nmulti = 0L,
  degree.min = NULL,
degree.max = NULL,
degree.start = NULL,
degree.restarts = 0L,
degree.max.cycles = 20L,
degree.verify = FALSE,
bernstein.basis = FALSE,
...,
nomad.opts = list())
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the data, formula interface, and whether bandwidths are supplied or computed.

`bandwidth.compute`
: a logical value which specifies whether to do a numerical search for bandwidths or not. If set to `FALSE`, a `conbandwidth` object will be returned with bandwidths

set to those specified in bws. Defaults to TRUE.

bws — a bandwidth specification. This can be set as a conbandwidth object returned from a previous invocation, or as a $p+q$-vector of bandwidths, with each element i up to $i = q$ corresponding to the bandwidth for column i in ydat, and each element i from $i = q + 1$ to $i = p + q$ corresponding to the bandwidth for column $i - q$ in xdat. In either case, the bandwidth supplied will serve as a starting point in the numerical search for optimal bandwidths. If specified as a vector, then additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, selection methods, and so on. This can be left unset.

call — the original function call. This is passed internally by npRmpi when a bandwidth search has been implied by a call to another function. It is not recommended that the user set this.

data — an optional data frame, list or environment (or object coercible to a data frame by as.data.frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which the function is called.

formula — a symbolic description of variables on which bandwidth selection is to be performed. The details of constructing a formula are described below.

na.action — a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting of options, and is na.fail if that is unset. The (recommended) default is na.omit.

subset — an optional vector specifying a subset of observations to be used in the fitting process.

xdat — a p-variate data frame of explanatory data on which bandwidth selection will be performed. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.

ydat — a q-variate data frame of dependent data on which bandwidth selection will be performed. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.

Automatic Degree Search Controls: These arguments control automatic local-polynomial degree search when regtype="lp".

degree.max — optional scalar or integer vector giving upper bounds for automatic degree search over continuous xdat predictors when degree.select != "manual".

degree.max.cycles — positive integer giving the maximum number of coordinate-search sweeps over the degree vector. Ignored for "manual" and "exhaustive" degree selection.

degree.min — optional scalar or integer vector giving lower bounds for automatic degree search over continuous xdat predictors when degree.select != "manual".

degree.restarts — non-negative integer giving the number of additional deterministic coordinate-search restarts. Ignored for "manual" and "exhaustive" degree selection.

`degree.select` character string controlling local-polynomial degree handling when `regtype="lp"`. `"manual"` (default) treats `degree` as fixed. `"coordinate"` performs cached coordinate-wise search over admissible degree vectors for the continuous `xdat` predictors. `"exhaustive"` evaluates the full admissible degree grid when `search.engine="cell"`. For NOMAD-based search engines, any non-`"manual"` value requests direct joint search over degree and bandwidth coordinates.

`degree.start` optional starting degree vector for automatic coordinate search. If omitted, the search starts from the degree-zero local-constant baseline on the continuous `xdat` predictors.

`degree.verify` logical value indicating whether a coordinate-search solution should be exhaustively verified over the admissible degree grid after the heuristic phase completes. Available only for `search.engine="cell"`.

Bandwidth Criterion And Representation: These arguments choose the selection criterion and the way continuous bandwidths are represented.

`bwmethod` which method to use to select bandwidths. `cv.ml` specifies likelihood cross-validation, `cv.ls` specifies least-squares cross-validation, and `normal-reference` just computes the 'rule-of-thumb' bandwidth h_j using the standard formula $h_j = 1.06\sigma_j n^{-1/(2P+l)}$, where σ_j is an adaptive measure of spread of the jth continuous variable defined as min(standard deviation, mean absolute deviation/1.4826, interquartile range/1.349), n the number of observations, P the order of the kernel, and l the number of continuous variables. Note that when there exist factors and the normal-reference rule is used, there is zero smoothing of the factors. Defaults to `cv.ml`.

`bwscaling` a logical value that when set to `TRUE` the supplied bandwidths are interpreted as 'scale factors' (c_j), otherwise when the value is `FALSE` they are interpreted as 'raw bandwidths' (h_j for continuous data types, λ_j for discrete data types). For continuous data types, c_j and h_j are related by the formula $h_j = c_j\sigma_j n^{-1/(2P+l)}$, where σ_j is an adaptive measure of spread of continuous variable j defined as min(standard deviation, mean absolute deviation/1.4826, interquartile range/1.349), n the number of observations, P the order of the kernel, and l the number of continuous variables. For discrete data types, c_j and h_j are related by the formula $h_j = c_j n^{-2/(2P+l)}$, where here j denotes discrete variable j. Defaults to `FALSE`.

`bwtype` character string used for the continuous variable bandwidth type, specifying the type of bandwidth to compute and return in the `conbandwidth` object. Defaults to `fixed`. Option summary:
`fixed`: compute fixed bandwidths
`generalized_nn`: compute generalized nearest neighbors
`adaptive_nn`: compute adaptive nearest neighbors

Categorical Search Initialization: These controls set categorical search starts and categorical direction-set initialization.

`dfac.dir` stretch factor for direction set search for Powell's algorithm for categorical variables. See Details

`dfac.init` non-random initial values for scale factors for categorical variables for Powell's algorithm. See Details

`hbd.dir` upper bound for direction set search for Powell's algorithm for categorical variables. See Details

`hbd.init` upper bound for scale factors for categorical variables for Powell's algorithm. See Details

`initd.dir` initial non-random values for direction set search for Powell's algorithm for categorical variables. See Details

`lbd.dir` lower bound for direction set search for Powell's algorithm for categorical variables. See Details

`lbd.init` lower bound for scale factors for categorical variables for Powell's algorithm. See Details

`scale.init.categorical.sample` a logical value that when set to `TRUE` scales `lbd.dir`, `hbd.dir`, `dfac.dir`, and `initd.dir` by $n^{-2/(2P+l)}$, n the number of observations, P the order of the kernel, and l the number of `numeric` variables. See Details

Continuous Direction-Set Search Controls: These controls set Powell direction-set initialization for continuous variables.

`cfac.dir` stretch factor for direction set search for Powell's algorithm for `numeric` variables. See Details

`dfc.dir` chi-square degrees of freedom for direction set search for Powell's algorithm for `numeric` variables. See Details

`initc.dir` initial non-random values for direction set search for Powell's algorithm for `numeric` variables. See Details

`lbc.dir` lower bound for direction set search for Powell's algorithm for `numeric` variables. See Details

Continuous Kernel Support Controls: These controls choose and parameterize bounded support for continuous kernels.

`cxkerbound` character string controlling continuous-kernel support handling for `xdat`. Can be set as `none` (default kernel on full support), `range` (use sample min/max), or `fixed` (use `cxkerlb`/`cxkerub`). The bounded-kernel route reuses the selected continuous kernel and renormalizes it on the chosen support; see `np.kernels`.

`cxkerlb` numeric scalar/vector of lower bounds for continuous `xdat` variables used when `cxkerbound="fixed"`. Must satisfy lower-bound validity for each variable (e.g., `<= min(variable)`). Use `-Inf` for unbounded below. See `np.kernels` for bounded-kernel normalization details.

`cxkerub` numeric scalar/vector of upper bounds for continuous `xdat` variables used when `cxkerbound="fixed"`. Must satisfy upper-bound validity for each variable (e.g., `>= max(variable)`). Use `Inf` for unbounded above. See `np.kernels` for bounded-kernel normalization details.

`cykerbound` character string controlling continuous-kernel support handling for `ydat`. Can be set as `none` (default kernel on full support), `range` (use sample min/max), or `fixed` (use `cykerlb`/`cykerub`). The bounded-kernel route reuses the selected continuous kernel and renormalizes it on the chosen support; see `np.kernels`.

cykerlb
: numeric scalar/vector of lower bounds for continuous `ydat` variables used when `cykerbound="fixed"`. Must satisfy lower-bound validity for each variable (e.g., `<= min(variable)`). Use `-Inf` for unbounded below. See `np.kernels` for bounded-kernel normalization details.

cykerub
: numeric scalar/vector of upper bounds for continuous `ydat` variables used when `cykerbound="fixed"`. Must satisfy upper-bound validity for each variable (e.g., `>= max(variable)`). Use `Inf` for unbounded above. See `np.kernels` for bounded-kernel normalization details.

Continuous Scale-Factor Search Initialization: These controls define deterministic and random continuous scale-factor starts and the lower admissibility floor for fixed-bandwidth search.

scale.factor.init
: deterministic initial scale factor for continuous fixed-bandwidth search. Defaults to `0.5`. The value supplied by the user is not rewritten, but the effective first start passed to the optimizer is `max(scale.factor.init, scale.factor.search.lower)`. See Details.

scale.factor.init.lower
: lower endpoint for random continuous scale-factor starts. Defaults to `0.1`. The value supplied by the user is not rewritten, but the effective random-start lower endpoint is `max(scale.factor.init.lower, scale.factor.search.lower)`. See Details.

scale.factor.init.upper
: upper endpoint for random continuous scale-factor starts. Defaults to `2.0`. It must be greater than or equal to the effective lower endpoint, `max(scale.factor.init.lower, scale.factor.search.lower)`; otherwise bandwidth search errors rather than silently expanding the interval. See Details.

scale.factor.search.lower
: optional nonnegative scalar giving the hard lower admissibility bound for continuous fixed-bandwidth search candidates. Defaults to `NULL`. If `NULL`, an existing bandwidth object's stored value is inherited when available; otherwise the package default `0.1` is used. This floor applies to computed/search bandwidth candidates and to effective search starts only. It does not rewrite explicit bandwidths supplied for storage with `bandwidth.compute = FALSE`. Final fixed-bandwidth search candidates must also have a finite valid raw objective value.

Kernel Type Controls: These controls choose continuous, unordered, and ordered kernels for `xdat` and `ydat`.

cxkerorder
: numeric value specifying kernel order for `xdat` (one of `(2,4,6,8)`). Kernel order specified along with a `uniform` continuous kernel type will be ignored. Defaults to `2`.

cxkertype
: character string used to specify the continuous kernel type for `xdat`. Can be set as `gaussian`, `epanechnikov`, or `uniform`. Defaults to `gaussian`.

cykerorder
: numeric value specifying kernel order for `ydat` (one of `(2,4,6,8)`). Kernel order specified along with a `uniform` continuous kernel type will be ignored. Defaults to `2`.

cykertype
: character string used to specify the continuous kernel type for ydat. Can be set as gaussian, epanechnikov, or uniform. Defaults to gaussian.

oxkertype
: character string used to specify the ordered categorical kernel type for xdat. Can be set as wangvanryzin, liracine, or racineliyan. Defaults to liracine.

oykertype
: character string used to specify the ordered categorical kernel type for ydat. Can be set as wangvanryzin, liracine, or racineliyan. Defaults to liracine.

uxkertype
: character string used to specify the unordered categorical kernel type for xdat. Can be set as aitchisonaitken or liracine. Defaults to aitchisonaitken.

uykertype
: character string used to specify the unordered categorical kernel type for ydat. Can be set as aitchisonaitken or liracine. Defaults to aitchisonaitken.

Least-Squares Quadrature Controls: These controls tune quadrature for bounded continuous-response least-squares cross-validation.

cvls.quadrature.extend.factor
: a positive finite scalar controlling the finite numerical integration window used by bounded conditional-density cv.ls quadrature when one or both continuous response-side bounds are infinite. Finite bounds are used literally. Defaults to 1.

cvls.quadrature.grid
: character string specifying the one-dimensional bounded cv.ls I_1 quadrature grid. "uniform" uses only evenly spaced nodes over the finite quadrature window, "sample" uses deterministic ranked sample-y nodes, and "hybrid" uses a fixed-size merge controlled by cvls.quadrature.ratios. The default is "hybrid" for scalar continuous responses and "uniform" for two continuous responses.

cvls.quadrature.points
: a two-element integer vector giving the bounded cv.ls quadrature point counts for one and two continuous response variables, respectively. Defaults to c(100L, 50L). For two continuous response variables, the second entry is used per dimension.

cvls.quadrature.ratios
: a three-element non-negative numeric vector summing to one, giving the uniform, ranked sample-y, and composite Gauss-Legendre proportions used by the scalar bounded cv.ls "hybrid" quadrature grid. Defaults to c(0.20, 0.55, 0.25), which gives an 20/55/25 split when cvls.quadrature.points = c(100L, 50L) and the nearest deterministic exact-count split at other scalar grid sizes. The setting is ignored by explicit "uniform" and "sample" grid modes.

When response-side bounds are set explicitly to fixed infinite endpoints, bounded cv.ls uses a finite numerical quadrature surrogate over the data range extended by cvls.quadrature.extend.factor. In that edge case, callers who want tighter agreement with the ordinary unbounded convolution route should set cvls.quadrature.points explicitly.

Local-Polynomial Model Specification: These arguments control the local-polynomial estimator, basis, and fixed degree specification.

basis
: character string specifying the polynomial basis used when regtype="lp". Options are "glp", "additive", and "tensor".

`bernstein.basis`
: logical value controlling Bernstein basis evaluation for `regtype="lp"`. When automatic degree search is requested and `bernstein.basis` is not explicitly supplied, the search route defaults to `TRUE` for numerical stability. Explicit `bernstein.basis=FALSE` is honored, but raw-polynomial search can be poorly conditioned at higher degrees.

`degree`
: integer scalar or integer vector of polynomial degrees for continuous `xdat` variables when `regtype="lp"`. If scalar, the value is recycled to all continuous `xdat` variables. With no continuous `xdat` predictors, `regtype="lp"` is only admissible when `degree=0`, the local-constant equivalent.

`regtype`
: character string specifying the conditional local method used for the `xdat` regression weight operator. Options are `"lc"`, `"ll"`, and `"lp"`. For `npc*` methods, `"ll"` is implemented via the canonical local polynomial engine with `degree = 1` and `basis = "glp"`. `"ll"` and positive-degree `"lp"` require at least one continuous `xdat` predictor; for categorical-only `xdat`, use `"lc"` or `"lp"` with `degree=0`. If local-linear `cv.ls` search fails while using this canonical raw basis, retrying explicitly with `regtype="lp"`, `degree=1`, and `bernstein.basis=TRUE`, or centering/scaling the continuous regressors, can improve numerical conditioning without changing package defaults or invoking an automatic fallback.

NOMAD Search Controls: These arguments control the optional NOMAD direct-search route for local-polynomial degree and bandwidth search.

`nomad`
: logical or character shortcut for the recommended automatic local-polynomial NOMAD route. When `TRUE`, any missing values among `regtype`, `search.engine`, `degree.select`, `bernstein.basis`, `degree.min`, `degree.max`, `degree.verify`, and `bwtype` are filled with `regtype="lp"`, `search.engine="nomad+powell"`, `degree.select="coordinate"`, `bernstein.basis=TRUE`, `degree.min=0L`, `degree.max=10L`, `degree.verify=FALSE`, and `bwtype="fixed"`. Explicit incompatible settings error immediately; in particular, explicit `nomad=TRUE` currently requires `regtype="lp"`, `bwtype="fixed"`, automatic degree search, no explicit `degree`, at least one continuous `xdat` predictor, and `search.engine %in% c("nomad", "nomad+powell")`. This shortcut does not change the meaning of `nmulti` or `nomad.nmulti`: `nmulti` remains the outer restart count, while `nomad.nmulti` controls inner native **crs** NOMAD API multistarts within each outer restart. Returned bandwidth objects retain this normalized preset metadata in `bw$nomad.shortcut` for a returned object `bw`; when available, `nomad.time` and `powell.time` record the direct-search and Powell-polish timing components.

`nomad.nmulti`
: non-negative integer controlling the inner the native **crs** NOMAD C API multistart count used within each outer NOMAD restart when `regtype="lp"` and automatic degree search uses `search.engine="nomad"` or `"nomad+powell"`. Defaults to `0L`, which preserves the current one-start-per- restart behavior. This does not replace `nmulti`: `nmulti` controls outer restarts, while `nomad.nmulti` controls inner NOMAD multistarts within each outer restart.

`nomad.opts`
: optional named list of native NOMAD parameters forwarded to the **crs** NOMAD C API for NOMAD-backed bandwidth searches. `NULL` is treated as no extra NOMAD options. Non-list values and invalid or unsupported parameters

are rejected before solver entry rather than silently ignored. Routes that evaluate R callbacks require serial callback evaluation and reject parallel callback settings such as `NB_THREADS_PARALLEL_EVAL > 1`.

`nomad.remin` logical flag controlling the optional second NOMAD hot start. When `TRUE`, NOMAD is restarted once from the best full candidate found, including both bandwidth and degree coordinates. Defaults to `FALSE`; current simulation evidence favors the one-pass NOMAD default for routine use, while leaving this switch available for sensitivity checks.

`search.engine` character string controlling the automatic local-polynomial search backend when `regtype="lp"` and `degree.select != "manual"`. `"nomad+powell"` (default) performs direct joint search over the `xdat`-side continuous bandwidth coordinates and degree vector using the native **crs** NOMAD C API, then applies one Powell hot start from the NOMAD solution. `"nomad"` omits the Powell refinement. `"cell"` profiles the criterion over the admissible degree grid using repeated fixed-degree bandwidth solves. NOMAD-based search supports `bwtype="fixed"`, `bwtype="generalized_nn"`, and `bwtype="adaptive_nn"`; it currently requires `degree.verify=FALSE` and the suggested package **crs** to be installed.

Numerical Search And Tolerance Controls: These controls set optimizer tolerances, restart behavior, invalid-candidate penalties, memory blocking, and bounded search transformations.

`ftol` fractional tolerance on the value of the cross-validation function evaluated at located minima (of order the machine precision or perhaps slightly larger so as not to be diddled by roundoff). Defaults to `1.490116e-07` (1.0e+01*sqrt(.Machine$double.eps)).

`invalid.penalty` a character string specifying the penalty used when the optimizer encounters invalid bandwidths. `"baseline"` returns a finite penalty based on a baseline objective; `"dbmax"` returns `DBL\_MAX`. Defaults to `"baseline"`.

`itmax` integer number of iterations before failure in the numerical optimization routine. Defaults to `10000`.

`memfac` The algorithm to compute the least-squares objective function uses a block-based algorithm to eliminate or minimize redundant kernel evaluations. Due to memory, hardware and software constraints, a maximum block size must be imposed by the algorithm. This block size is roughly equal to memfac*10^5 elements. Empirical tests on modern hardware find that a memfac of 500 performs well. If you experience out of memory errors, or strange behaviour for large data sets (>100k elements) setting memfac to a lower value may fix the problem.

`nmulti` integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points

`penalty.multiplier` a numeric multiplier applied to the baseline penalty when `invalid.penalty="baseline"`. Defaults to `10`.

`powell.remin` logical flag controlling Powell restart-from-minimum behavior. For ordinary fixed-degree Powell-style search, `TRUE` restarts the local search from the located minimum. For `search.engine="nomad+powell"`, this controls only the final

Powell bandwidth-polish step. The default is TRUE for ordinary Powell routes and FALSE for the Powell polish after NOMAD unless explicitly supplied.

bwsolver
: bandwidth optimizer for fixed-degree searches. The default "powell" preserves the historical Powell route. "mads" uses the native **crs** NOMAD C API over the bandwidth coordinates, and "mads+powell" applies one Powell hot start from the MADS solution. Use search.engine rather than bwsolver for automatic degree search. Advanced native NOMAD parameters may be supplied through the explicit nomad.opts argument; invalid or unsupported parameters are rejected by NOMAD rather than silently ignored.

small
: a small number used to bracket a minimum (it is hopeless to ask for a bracketing interval of width less than sqrt(epsilon) times its central value, a fractional width of only about 10-04 (single precision) or 3x10-8 (double precision)). Defaults to small = 1.490116e-05 (1.0e+03*sqrt(.Machine$double.eps)).

tol
: tolerance on the position of located minima of the cross-validation function (tol should generally be no smaller than the square root of your machine's floating point precision). Defaults to 1.490116e-04 (1.0e+04*sqrt(.Machine$double.eps)).

transform.bounds
: a logical value that when set to TRUE applies an internal transformation that maps the unconstrained search to the feasible bandwidth domain. Defaults to FALSE.

Additional Arguments: These arguments collect remaining controls passed through S3 methods.

...
: additional arguments supplied to specify the bandwidth type, kernel types, selection methods, and so on, detailed below.

Details

The scale.factor.* controls are dimensionless search controls. The package converts scale factors to bandwidths using the estimator-specific scaling encoded in the bandwidth object, including kernel order and the number of continuous variables relevant for the estimator. Users should not pre-multiply these controls by sample-size or standard-deviation factors.

scale.factor.init controls the deterministic first search start. scale.factor.init.lower and scale.factor.init.upper define the random multistart interval. scale.factor.search.lower is the lower admissibility bound for continuous fixed-bandwidth search candidates. The effective first start is max(scale.factor.init, scale.factor.search.lower), and the effective random-start lower endpoint is max(scale.factor.init.lower, scale.factor.search.lower). scale.factor.init.upper must be at least that effective lower endpoint; the package errors rather than silently expanding the user's interval.

When scale.factor.search.lower is NULL, an existing bandwidth object's stored floor is inherited when available; otherwise the package default 0.1 is used. Explicit bandwidths supplied for storage with bandwidth.compute = FALSE are not rewritten by the search floor.

Categorical search-start controls such as dfac.init, lbd.init, and hbd.init have separate semantics and are not affected by scale.factor.search.lower.

For bwmethod="cv.ml", the ordinary contribution is based on $\log(\hat{f}_i)$. Compact-support kernels can yield zero leave-one-out conditional-density estimates, and local-polynomial or higher-order-kernel fits can yield negative estimates. In those cases the package optimizes a guarded CVML

criterion rather than the literal log likelihood. Let ϵ denote the positive machine floor used for the guard. Positive fitted values use the ordinary branch $-\log(\hat{f}_i)$. Near-zero fitted values use $-\log(\epsilon)$, and negative fitted values use the smooth penalty $\log(-\hat{f}_i) - 2\log(\epsilon)$. The `guarded` count reported in bandwidth summaries records objective evaluations in which this guard was used. The `bwmethod="cv.ls"` criterion remains well defined for negative kernel density estimates and is not modified by this CVML guard.

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, `plot`, `plot.np` for plotting options, and `npRmpi.init` for interactive/cluster MPI startup. See `npRmpi.init` details for performance tradeoffs (message passing/startup mode) and the `inst/Rprofile` manual-broadcast template.

The bandwidth-selection argument surface is easiest to read by decision group. Start by initializing MPI execution if needed, then choose the data and bandwidth inputs (`xdat`, `ydat`, `bws`, and `bandwidth.compute`), then choose the bandwidth criterion and representation (`bwmethod`, `bwscaling`, and `bwtype`). Next choose continuous kernel and support controls (`cxker*` and `cyker*`), categorical kernel controls (`uxkertype`, `uykertype`, `oxkertype`, and `oykertype`), and numerical search controls including `nmulti`, tolerances, penalties, and the `scale.factor.*` search-start and admissibility controls. Bounded continuous-response `cv.ls` fits may also use the `cvls.quadrature.*` controls. Local-polynomial and NOMAD controls (`regtype`, `basis`, `degree*`, `search.engine`, `nomad`, `nomad.nmulti`, and `bernstein.basis`) are relevant when using the explicit local-polynomial route.

For S3 plotting help, see `plot.np`. You can list available plot methods with `methods("plot")`.

`npcdensbw` implements a variety of methods for choosing bandwidths for multivariate distributions ($p + q$-variate) defined over a set of possibly continuous and/or discrete (unordered, ordered) data. The approach is based on Li and Racine (2004) who employ 'generalized product kernels' that admit a mix of continuous and discrete data types.

The cross-validation methods employ multivariate numerical search algorithms (direction set (Powell's) methods in multidimensions).

Bandwidths can (and will) differ for each variable which is, of course, desirable.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i, when estimating the density at the point x. Generalized nearest-neighbor bandwidths change with the point at which the density is estimated, x. Fixed bandwidths are constant over the support of x.

`npcdensbw` may be invoked *either* with a formula-like symbolic description of variables on which bandwidth selection is to be performed *or* through a simpler interface whereby data is passed directly to the function via the `xdat` and `ydat` parameters. Use of these two interfaces is **mutually exclusive**.

Data contained in the data frames `xdat` and `ydat` may be a mix of continuous (default), unordered discrete (to be specified in the data frames using `factor`), and ordered discrete (to be specified in the data frames using `ordered`). Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see `npRmpi` for details).

Data for which bandwidths are to be estimated may be specified symbolically. A typical description has the form `dependent data ~ explanatory data`, where `dependent data` and `explanatory data` are both series of variables specified by name, separated by the separation character '+'. For example, `y1 + y2 ~ x1 + x2` specifies that the bandwidths for the joint distribution of variables `y1` and `y2` conditioned on `x1` and `x2` are to be estimated. See below for further examples.

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth order Gaussian and Epanechnikov kernels, and the uniform kernel. Unordered discrete data types use a variation on Aitchison and Aitken's (1976) kernel, while ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

When `regtype="lp"` and `degree.select != "manual"`, `npcdensbw` can jointly determine the `xdat`-side local polynomial degree vector and the fixed bandwidth coordinates entering the conditional density criterion. With `search.engine="cell"`, the criterion is profiled over the admissible degree grid using cached coordinate-wise or exhaustive search. With `search.engine="nomad"` or `"nomad+powell"`, the criterion is optimized directly over the joint degree/bandwidth space using the native **crs** NOMAD C API; `"nomad+powell"` then performs one Powell hot start from the NOMAD solution and keeps the better of the direct NOMAD and polished answers. This polynomial-adaptive joint-search route is motivated by Hall and Racine (2015) together with Li, Li, and Racine (under revision). When `bernstein.basis` is not explicitly supplied, the automatic search route defaults to `bernstein.basis=TRUE` for numerical stability.

Setting `nomad=TRUE` is a convenience preset for this automatic LP route, not a generic optimizer alias. For conditional density bandwidth selection it expands any missing values to the equivalent long-form call

```
npcdensbw(...,
          regtype = "lp",
          search.engine = "nomad+powell",
          degree.select = "coordinate",
          bernstein.basis = TRUE,
          degree.min = 0L,
          degree.max = 10L,
          degree.verify = FALSE,
          bwtype = "fixed")
```

Compatible explicit tuning arguments are respected. Incompatible explicit settings fail fast so the shortcut never silently changes user-selected semantics. The character value `nomad="auto"` applies the same LP shortcut but leaves `search.engine` and `degree.select` eligible for data-driven resolution when both were not supplied explicitly: scalar continuous degree searches use the exhaustive degree-grid route, while higher-dimensional degree searches keep the NOMAD/Powell route. Explicit `search.engine` or `degree.select` choices are honored.

The optimizer invoked for search is Powell's conjugate direction method which requires the setting of (non-random) initial values and search directions for bandwidths, and, when restarting, random values for successive invocations. Bandwidths for `numeric` variables are scaled by robust measures of spread, the sample size, and the number of `numeric` variables where appropriate. Two sets of parameters for bandwidths for `numeric` can be modified, those for initial values for the parameters themselves, and those for the directions taken (Powell's algorithm does not involve explicit computation of the function's gradient). The default values are set by considering search performance for a variety of difficult test cases and simulated cases. We highly recommend restarting search a large number of times to avoid the presence of local minima (achieved by modifying `nmulti`). Further refinement for difficult cases can be achieved by modifying these sets of parameters. However, these parameters are intended more for the authors of the package to enable 'tuning' for various methods rather than for the user themselves.

Value

npcdensbw returns a conbandwidth object, with the following components:

xbw	bandwidth(s), scale factor(s) or nearest neighbours for the explanatory data, xdat
ybw	bandwidth(s), scale factor(s) or nearest neighbours for the dependent data, ydat
fval	objective function value at minimum

if bwtype is set to fixed, an object containing bandwidths (or scale factors if bwscaling = TRUE) is returned. If it is set to generalized_nn or adaptive_nn, then instead the kth nearest neighbors are returned for the continuous variables while the discrete kernel bandwidths are returned for the discrete variables.

The functions predict, summary and plot support objects of type conbandwidth.

Book And Method Pointers

npcdensbw selects response and conditioning bandwidths, and when requested local-polynomial degree/search metadata, for the conditional density target $f(y \mid x)$ estimated by npcdens.

For book-length derivations, see Li and Racine (2007), Chapter 5 *Conditional Density Estimation*, especially Sections 5.1-5.4, and Chapter 4 *Kernel Estimation with Mixed Data*. The later workflow treatment is Racine (2019), Chapter 4 *Conditional Probability Density and Cumulative Distribution Functions*.

Usage Issues

If you are using data of mixed types, then it is advisable to use the data.frame function to construct your input data and not cbind, since cbind will typically not work as intended on mixed data types and will coerce the data to the same type.

Caution: multivariate data-driven bandwidth selection methods are, by their nature, *computationally intensive*. Virtually all methods require dropping the ith observation from the data set, computing an object, repeating this for all observations in the sample, then averaging each of these leave-one-out estimates for a *given* value of the bandwidth vector, and only then repeating this a large number of times in order to conduct multivariate numerical minimization/maximization. Furthermore, due to the potential for local minima/maxima, *restarting this procedure a large number of times may often be necessary*. This can be frustrating for users possessing large datasets. For exploratory purposes, you may wish to override the default search tolerances, say, setting ftol=.01 and tol=.01 and conduct multistarting (the default is to restart min(2, ncol(xdat,ydat)) times) as is done for a number of examples. Once the procedure terminates, you can restart search with default tolerances using those bandwidths obtained from the less rigorous search (i.e., set bws=bw on subsequent calls to this routine where bw is the initial bandwidth object). A version of this package using the Rmpi wrapper is under development that allows one to deploy this software in a clustered computing environment to facilitate computation involving large datasets.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

Hall, P. and J.S. Racine and Q. Li (2004), "Cross-validation and the estimation of conditional probability densities," Journal of the American Statistical Association, 99, 1015-1026.

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

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

Li, A. and Q. Li and J.S. Racine (under revision), "Boundary Adjusted, Polynomial Adaptive, Nonparametric Kernel Conditional Density Estimation," Econometric Reviews.

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

Scott, D.W. (1992), *Multivariate Density Estimation. Theory, Practice and Visualization,* New York: Wiley.

Silverman, B.W. (1986), *Density Estimation,* London: Chapman and Hall.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

`np.kernels`, `np.options`, `plot`, `plot.np` `bw.nrd`, `bw.SJ`, `hist`, `npudens`, `npudist`

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )
```

```
if (!in.check) {
npRmpi.init(nslaves=1)

data("Italy")

bw <- npcdensbw(formula=gdp~ordered(year), data=Italy)

summary(bw)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

npcdenshat *Conditional Density Hat Operator*

Description

Constructs the conditional density hat operator associated with npcdens bandwidth objects. The returned operator maps a right-hand side y to Hy; with y = 1 this reproduces the fitted conditional density.

Usage

```
npcdenshat(bws,
           txdat = stop("training data 'txdat' missing"),
           tydat = stop("training data 'tydat' missing"),
           exdat,
           eydat,
           y = NULL,
           output = c("matrix", "apply"),
           deriv = NULL,
           s = NULL)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the fitted bandwidth object, training data, and evaluation data.

bws	A fitted conditional density bandwidth object of class "conbandwidth".
exdat	Optional evaluation conditioning data. If omitted, the operator is built on the training conditioning data.
eydat	Optional evaluation response data. If omitted, the operator is built on the training response data.
txdat	Training conditioning data used to construct the operator.
tydat	Training response data used to construct the operator.

Operator Output: These arguments control whether the operator is returned as a matrix or applied directly.

deriv	Convenience alias for s.
output	Either "matrix" for the hat matrix or "apply" for direct application to y.
s	Optional first-derivative multi-index over continuous conditioning variables. By default the conditional-density operator itself is returned. When s requests one first derivative, the returned operator maps y to the corresponding npcdens(..., gradients = TRUE) conditioning-variable gradient component.
y	Optional right-hand side vector or matrix with one row per training observation.

Details

For output = "matrix", the return value is a matrix with class c("npcdenshat", "matrix") and attributes storing the bandwidth object, training data, evaluation data, and call metadata.

For output = "apply", the function returns H y directly. Matrix right-hand sides are applied columnwise.

The optional s argument follows the derivative multi-index convention used by npreghat for continuous conditioning variables. The default s = NULL returns the ordinary conditional-density hat operator. Supplying one first derivative, for example s = 1L in a univariate continuous-x problem or s = c(x2 = 1L) in a multivariate problem, returns the hat operator for that component of the conditional-density gradient. This derivative route is intended to match npcdens(..., gradients = TRUE)$congrad.

This helper is intended for object-fed repeated evaluation once a bandwidth object has already been constructed. It does not perform bandwidth selection.

Value

Either a hat matrix of class "npcdenshat" or the applied result H y, depending on output.

Examples

```
## Not run:
npRmpi.init(nslaves = 1)
data(cps71)
tx <- data.frame(age = cps71$age)
ty <- data.frame(logwage = cps71$logwage)

bw <- npcdensbw(xdat = tx, ydat = ty, bwtype = "fixed",
                bandwidth.compute = FALSE, bws = c(1.0, 1.0))

H <- npcdenshat(bws = bw, txdat = tx, tydat = ty)
dens.hat <- npcdenshat(bws = bw, txdat = tx, tydat = ty,
                       y = rep(1, nrow(tx)),
                       output = "apply")
dens.core <- fitted(npcdens(bws = bw, txdat = tx, tydat = ty))

head(cbind(dens.core, dens.hat), n = 2L)

grad.hat <- npcdenshat(bws = bw, txdat = tx, tydat = ty,
                       y = rep(1, nrow(tx)),
                       output = "apply", s = 1L)
grad.core <- gradients(npcdens(bws = bw, txdat = tx, tydat = ty,
                               gradients = TRUE))[, 1L]
head(cbind(grad.core, grad.hat), n = 2L)

npRmpi.quit()

## End(Not run)
```

---

npcdist *Kernel Conditional Distribution Estimation with Mixed Data Types*

Description

npcdist computes kernel cumulative conditional distribution estimates on $p + q$-variate evaluation data, given a set of training data (both explanatory and dependent) and a bandwidth specification (a condbandwidth object or a bandwidth vector, bandwidth type, and kernel type) using the method of Li and Racine (2008) and Li, Lin, and Racine (2013). The data may be continuous, discrete (unordered and ordered factors), or some combination thereof.

Usage

```
npcdist(bws,
        ...)

## S3 method for class 'formula'
npcdist(bws,
        data = NULL,
        newdata = NULL,
```

```
        ...)


## S3 method for class 'condbandwidth'
npcdist(bws,
        txdat = stop("invoked without training data 'txdat'"),
        tydat = stop("invoked without training data 'tydat'"),
        exdat,
        eydat,
        gradients = FALSE,
        gradient.order = 1L,
        proper = FALSE,
        proper.method = c("isotonic"),
        proper.control = list(),
        ...)

## Default S3 method:
npcdist(bws,
        txdat,
        tydat,
        nomad = FALSE,
        ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and training data.

`bws` a bandwidth specification. This can be set as a `condbandwidth` object returned from a previous invocation of `npcdistbw`, or as a $p+q$-vector of bandwidths, with each element i up to $i=q$ corresponding to the bandwidth for column i in `tydat`, and each element i from $i=q+1$ to $i=p+q$ corresponding to the bandwidth for column $i-q$ in `txdat`. If specified as a vector, then additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, training data, and so on.

`data` an optional data frame, list or environment (or object coercible to a data frame by `as.data.frame`) containing the variables in the model. If not found in data, the variables are taken from `environment(bws)`, typically the environment from which `npcdistbw` was called.

`txdat` a p-variate data frame of sample realizations of explanatory data (training data). Defaults to the corresponding training data recoverable from the bandwidth object in the current session.

`tydat` a q-variate data frame of sample realizations of dependent data (training data). Defaults to the corresponding training data recoverable from the bandwidth object in the current session.

Local-Polynomial Degree And Bandwidth Search: This argument controls the recommended

automatic local-polynomial NOMAD route, which jointly selects continuous polynomial degree and bandwidths when these are computed inside `npcdist`.

`nomad` logical shortcut passed through to `npcdistbw` when bandwidths are computed inside `npcdist`. When `TRUE`, the bandwidth route fills any missing values among `regtype`, `search.engine`, `degree.select`, `bernstein.basis`, `degree.min`, `degree.max`, `degree.verify`, and `bwtype` with the recommended automatic local-polynomial degree-and-bandwidth NOMAD preset documented in `npcdistbw`. Additional NOMAD tuning arguments such as `nomad.nmulti` may also be supplied through `...`; `nmulti` remains the outer restart count while `nomad.nmulti` controls inner native **crs** NOMAD API multistarts within each outer restart. After fitting, inspect `fit$bws$nomad.shortcut` on the returned object `fit` to see the normalized shortcut metadata.

Evaluation Data And Returned Quantities: These arguments control where the fitted conditional distribution is evaluated and which estimates are returned.

`exdat` a p-variate data frame of explanatory data on which cumulative conditional distributions will be evaluated. By default, evaluation takes place on the data provided by `txdat`.

`eydat` a q-variate data frame of dependent data on which cumulative conditional distributions will be evaluated. By default, evaluation takes place on the data provided by `tydat`.

`gradients` a logical value specifying whether to return estimates of the gradients at the evaluation points. Defaults to `FALSE`.

`gradient.order` derivative order for continuous explanatory-variable gradients when `gradients = TRUE` and `regtype = "lp"`. A scalar is recycled across continuous explanatory variables, or one value may be supplied per continuous explanatory variable. The default `1L` returns first derivatives. Higher orders are available for continuous explanatory variables only and must not exceed the corresponding local-polynomial degree; unordered and ordered predictors retain their usual first-order discrete effects.

`newdata` An optional data frame in which to look for evaluation data. If omitted, the training data are used.

Fit Properization Controls: These arguments control optional post-estimation properization of the fitted conditional distribution.

`proper` a logical value specifying whether to post-process the estimated conditional distribution so that it is monotone and bounded on the evaluation grid. Defaults to `FALSE`.

`proper.control` a named list of control parameters for properization. Supported entries are `tol`, `grid.check`, `store.raw`, and `fail.on.unsupported`.

`proper.method` the properization method. Currently only `"isotonic"` is supported.

Additional Arguments: Further arguments are passed to `npcdistbw` when bandwidths are computed internally, or used to interpret a numeric `bws` vector.

... additional arguments supplied to npcdistbw when npcdist computes bandwidths internally, or arguments needed to interpret a numeric bws vector. This is where bandwidth selection controls such as bwmethod, bwtype, kernel/support controls such as cxkertype, cykertype, cxkerbound, and cykerbound, search controls such as nmulti, scale.factor.search.lower, and nomad.nmulti, and local-polynomial controls such as regtype, degree, basis, and bernstein.basis are supplied. See npcdistbw for the complete bandwidth-selection argument surface.

Details

Documentation guide: see npcdistbw for bandwidth selection and search controls, np.kernels for kernels, np.options for global options, plot, plot.np for plotting options, and npRmpi.init for interactive/cluster MPI startup. See npRmpi.init details for performance tradeoffs (message passing/startup mode) and the inst/Rprofile manual-broadcast template.

When bws is omitted, the formula and default methods call npcdistbw first and pass bandwidth-selection arguments from ... to that call. When bws is already a condbandwidth object, npcdist estimates with the stored bandwidth metadata in that object.

With finite continuous response bounds, npcdist uses the observation-centered truncated-kernel CDF documented in np.kernels. Each response contribution is exactly zero and one at the declared lower and upper endpoints and has the linear uniform-CDF limit as the response bandwidth diverges.

Argument groups for bandwidth selection are documented on npcdistbw. The most common workflow is to initialize MPI execution if needed, choose data and bandwidth inputs, then bandwidth criterion and representation, then kernel/support controls, numerical search controls, and finally local-polynomial/NOMAD controls for polynomial-adaptive fits.

For S3 plotting help, see plot.np. You can list available plot methods with methods("plot").

npcdist implements a variety of methods for estimating multivariate conditional cumulative distributions ($p+q$-variate) defined over a set of possibly continuous and/or discrete (unordered, ordered) data. The approach is based on Li and Racine (2004) who employ 'generalized product kernels' that admit a mix of continuous and discrete data types.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i, when estimating the cumulative conditional distribution at the point x. Generalized nearest-neighbor bandwidths change with the point at which the cumulative conditional distribution is estimated, x. Fixed bandwidths are constant over the support of x.

Training and evaluation input data may be a mix of continuous (default), unordered discrete (to be specified in the data frames using factor), and ordered discrete (to be specified in the data frames using ordered). Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see npRmpi for details).

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth order Gaussian and Epanechnikov kernels, and the uniform kernel. Unordered discrete data types use a variation on Aitchison and Aitken's (1976) kernel, while ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

For practitioners who want the recommended automatic local-polynomial degree-and-bandwidth NOMAD route without spelling out all LP tuning arguments, `npcdist(..., nomad=TRUE)` and `npcdistbw(..., nomad=TRUE)` expand missing settings to the same documented preset. Explicit incompatible settings fail fast rather than being silently rewritten.

With `regtype = "lp"`, `gradients = TRUE` and `gradient.order` greater than one expose higher-order derivative estimates with respect to continuous explanatory variables. These derivatives use the same local-polynomial basis, degree, Bernstein option, bandwidths, and kernels as the fitted conditional distribution. Asymptotic standard errors for these higher-order derivative columns are not currently reported; the corresponding entries of `congerr` are `NA`. Use bootstrap intervals when inference on higher-order derivatives is required.

Value

`npcdist` returns a `condistribution` object. The generic accessor functions `fitted`, `se`, and `gradients`, extract estimated values, asymptotic standard errors on estimates, and gradients, respectively, from the returned object. Furthermore, the functions `predict`, `summary` and `plot` support objects of both classes. The returned objects have the following components:

`xbw`	bandwidth(s), scale factor(s) or nearest neighbours for the explanatory data, `txdat`
`ybw`	bandwidth(s), scale factor(s) or nearest neighbours for the dependent data, `tydat`
`xeval`	the evaluation points of the explanatory data
`yeval`	the evaluation points of the dependent data
`condist`	estimates of the conditional cumulative distribution at the evaluation points
`conderr`	standard errors of the cumulative conditional distribution estimates
`congrad`	if invoked with `gradients = TRUE`, estimates of the gradients at the evaluation points. For local-polynomial fits, continuous-coordinate derivative orders are controlled by `gradient.order`.
`congerr`	if invoked with `gradients = TRUE`, standard errors of the gradients at the evaluation points. Higher-order continuous-coordinate derivative standard errors are currently returned as `NA`; bootstrap inference is preferred for those targets.
`log_likelihood`	log likelihood of the cumulative conditional distribution estimate

Book And Method Pointers

The conditional distribution target is $F(y \mid x) = \Pr(Y \leq y \mid X = x)$. The estimator uses the selected mixed-data conditional distribution bandwidths and kernels for the response and conditioning coordinates; local-polynomial routes use the selected continuous-coordinate polynomial degree. These fitted conditional CDFs are also the object inverted by `npqreg` when computing conditional quantiles.

For book-length derivations, see Li and Racine (2007), Chapter 6 *Conditional CDF and Quantile Estimation*, especially Sections 6.1, 6.2, and 6.5, together with Chapter 5 *Conditional Density Estimation*. The later workflow treatment is Racine (2019), Chapter 4 *Conditional Probability Density and Cumulative Distribution Functions*.

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

When reusing a bandwidth object without explicit training or evaluation data, the required data are recovered from object-retained state or from the stored call environment when available. This is a convenience for ordinary same-session use, not a guarantee that a saved object is self-contained. For reproducible replay after `saveRDS()` and `readRDS()` in a fresh R session, supply the required data arguments explicitly.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

Hall, P. and J.S. Racine and Q. Li (2004), "Cross-validation and the estimation of conditional probability densities," Journal of the American Statistical Association, 99, 1015-1026.

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

Li, Q. and J.S. Racine (2008), "Nonparametric estimation of conditional CDF and quantile functions with mixed categorical and continuous data," Journal of Business and Economic Statistics, 26, 423-434.

Li, Q. and J. Lin and J.S. Racine (2013), "Optimal bandwidth selection for nonparametric conditional distribution and quantile functions", Journal of Business and Economic Statistics, 31, 57-65.

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

Scott, D.W. (1992), *Multivariate Density Estimation. Theory, Practice and Visualization,* New York: Wiley.

Silverman, B.W. (1986), *Density Estimation,* London: Chapman and Hall.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

`np.kernels`, `np.options`, `plot`, `plot.np` `npudens`

Examples

```
## Not run:
## Not run in checks: this example performs bandwidth search on panel data and
## can be too slow/unstable for automated MPI checks.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
```

```
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.
npRmpi.init(nslaves=1)

data("Italy")

bw <- npcdistbw(formula=gdp~ordered(year),
                data=Italy)

Fhat <- npcdist(bws=bw)

summary(Fhat)

## Variations on local polynomial conditional distribution estimation
## with proper = TRUE.

Italy2 <- within(Italy, {
  year <- as.numeric(as.character(year))
})

## Plot only: make the plotted surface proper on the plot evaluation grid.

Fhat <- npcdist(gdp ~ year, data = Italy2,
                regtype = "lp", degree = 3, nmulti = 1)

plot(Fhat, proper = TRUE)

## Fit an object whose fitted values are themselves proper.

ctrl_fit <- list(
  mode = "slice",
  apply = "fitted",
  slice.grid.size = 101L,
  slice.extend.factor = 0.1
)

Fhat_fit <- npcdist(
  gdp ~ year,
  data = Italy2,
  regtype = "lp",
  degree = 3,
  nmulti = 1,
  proper = TRUE,
  proper.control = ctrl_fit
)
```

```
fit_proper <- fitted(Fhat_fit)
fit_raw <- Fhat_fit$condist.raw

## Predict on a common explicit y-grid for several years, and render
## those predictions proper.

g.grid <- seq(min(Italy2$gdp), max(Italy2$gdp), length.out = 200)

nd_grid <- expand.grid(
  gdp = g.grid,
  year = c(1955, 1975, 1995)
)

pred_grid <- predict(Fhat, newdata = nd_grid, proper = TRUE)

## Predict on paired rows with different gdp grids by year, and still
## make the predictions proper via slice mode.

g1 <- seq(quantile(Italy2$gdp, 0.10),
          quantile(Italy2$gdp, 0.60), length.out = 60)
g2 <- seq(quantile(Italy2$gdp, 0.30),
          quantile(Italy2$gdp, 0.90), length.out = 35)

nd_slice <- rbind(
  data.frame(gdp = g1, year = rep(1960, length(g1))),
  data.frame(gdp = g2, year = rep(1985, length(g2)))
)

pred_slice <- predict(
  Fhat,
  newdata = nd_slice,
  proper = TRUE,
  proper.control = list(mode = "slice")
)

## One object that carries properization for fitted values and for later
## predict() calls.

ctrl_both <- list(
  mode = "slice",
  apply = "both",
  slice.grid.size = 101L,
  slice.extend.factor = 0.1
)

Fhat_both <- npcdist(
  gdp ~ year,
  data = Italy2,
  regtype = "lp",
  degree = 3,
  nmulti = 1,
  proper = TRUE,
  proper.control = ctrl_both
```

```
)

fit_both <- fitted(Fhat_both)
pred_both <- predict(
  Fhat_both,
  newdata = nd_slice,
  proper.control = ctrl_both
)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close

## End(Not run)
```

---

npcdistbw *Kernel Conditional Distribution Bandwidth Selection with Mixed Data Types*

Description

npcdistbw computes a condbandwidth object for estimating a $p + q$-variate kernel conditional cumulative distribution estimator defined over mixed continuous and discrete (unordered xdat, ordered xdat and ydat) data using either the normal-reference rule-of-thumb or least-squares cross validation method of Li and Racine (2008) and Li, Lin and Racine (2013).

Usage

```
npcdistbw(...)

## S3 method for class 'formula'
npcdistbw(formula,
          data,
          subset,
          na.action,
          call,
          gdata = NULL,
```

```
          ...)


## S3 method for class 'condbandwidth'
npcdistbw(xdat = stop("data 'xdat' missing"),
          ydat = stop("data 'ydat' missing"),
          gydat = NULL,
          bws,
          bandwidth.compute = TRUE,
          cfac.dir = 2.5*(3.0-sqrt(5)),
          scale.factor.init = 0.5,
          dfac.dir = 0.25*(3.0-sqrt(5)),
          dfac.init = 0.375,
          dfc.dir = 3,
          do.full.integral = FALSE,
          ftol = 1.490116e-07,
          scale.factor.init.upper = 2.0,
          hbd.dir = 1,
          hbd.init = 0.9,
          initc.dir = 1.0,
          initd.dir = 1.0,
          invalid.penalty = c("baseline","dbmax"),
          itmax = 10000,
          lbc.dir = 0.5,
          scale.factor.init.lower = 0.1,
          lbd.dir = 0.1,
          lbd.init = 0.1,
          memfac = 500.0,
          ngrid = 100,
          nmulti,
          penalty.multiplier = 10,
          powell.remin = TRUE,
          bwsolver = c("powell", "mads", "mads+powell"),
          scale.init.categorical.sample = FALSE,
          scale.factor.search.lower = NULL,
          small = 1.490116e-05,
          tol = 1.490116e-04,
          transform.bounds = FALSE,
          ...,
          nomad.opts = list())

## Default S3 method:
npcdistbw(xdat = stop("data 'xdat' missing"),
          ydat = stop("data 'ydat' missing"),
          gydat,
          bws,
          bandwidth.compute = TRUE,
          bwmethod,
```

```
bwscaling,
bwtype,
cfac.dir,
scale.factor.init,
cxkerbound,
cxkerlb,
cxkerorder,
cxkertype,
cxkerub,
cykerbound,
cykerlb,
cykerorder,
cykertype,
cykerub,
dfac.dir,
dfac.init,
dfc.dir,
do.full.integral,
ftol,
scale.factor.init.upper,
hbd.dir,
hbd.init,
initc.dir,
initd.dir,
invalid.penalty,
itmax,
lbc.dir,
scale.factor.init.lower,
lbd.dir,
lbd.init,
memfac,
ngrid,
nmulti,
oxkertype,
oykertype,
penalty.multiplier,
nomad.remin = FALSE,
powell.remin,
bwsolver = c("powell", "mads", "mads+powell"),
scale.init.categorical.sample,
scale.factor.search.lower = NULL,
small,
tol,
transform.bounds,
uxkertype,
regtype = c("lc", "ll", "lp"),
basis = c("glp", "additive", "tensor"),
degree = NULL,
```

```
degree.select = c("manual", "coordinate", "exhaustive"),
search.engine = c("nomad+powell", "cell", "nomad"),
nomad = FALSE,
nomad.nmulti = 0L,
  degree.min = NULL,
degree.max = NULL,
degree.start = NULL,
degree.restarts = 0L,
degree.max.cycles = 20L,
degree.verify = FALSE,
bernstein.basis = FALSE,
...,
nomad.opts = list())
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the data, formula interface, optional distribution grid, and whether bandwidths are supplied or computed.

`bandwidth.compute`
: a logical value which specifies whether to do a numerical search for bandwidths or not. If set to `FALSE`, a `condbandwidth` object will be returned with bandwidths set to those specified in `bws`. Defaults to `TRUE`.

`bws`
: a bandwidth specification. This can be set as a `condbandwidth` object returned from a previous invocation, or as a $p+q$-vector of bandwidths, with each element i up to $i = q$ corresponding to the bandwidth for column i in `ydat`, and each element i from $i = q + 1$ to $i = p + q$ corresponding to the bandwidth for column $i - q$ in `xdat`. In either case, the bandwidth supplied will serve as a starting point in the numerical search for optimal bandwidths. If specified as a vector, then additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, selection methods, and so on. This can be left unset.

`call`
: the original function call. This is passed internally by `npRmpi` when a bandwidth search has been implied by a call to another function. It is not recommended that the user set this.

`data`
: an optional data frame, list or environment (or object coercible to a data frame by `as.data.frame`) containing the variables in the model. If not found in data, the variables are taken from `environment(formula)`, typically the environment from which the function is called.

`formula`
: a symbolic description of variables on which bandwidth selection is to be performed. The details of constructing a formula are described below.

`gdata`
: a grid of data on which the indicator function for least-squares cross-validation is to be computed (can be the sample or a grid of quantiles).

`gydat`
: a grid of data on which the indicator function for least-squares cross-validation is to be computed (can be the sample or a grid of quantiles for `ydat`).

`na.action`
: a function which indicates what should happen when the data contain `NA`s. The default is set by the `na.action` setting of options, and is `na.fail` if that is unset. The (recommended) default is `na.omit`.

`subset` an optional vector specifying a subset of observations to be used in the fitting process.

`xdat` a p-variate data frame of explanatory data on which bandwidth selection will be performed. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.

`ydat` a q-variate data frame of dependent data on which bandwidth selection will be performed. The data types may be continuous, discrete (ordered factors), or some combination thereof.

Automatic Degree Search Controls: These arguments control automatic local-polynomial degree search when `regtype="lp"`.

`degree.max` optional scalar or integer vector giving upper bounds for automatic degree search over continuous `xdat` predictors when `degree.select != "manual"`.

`degree.max.cycles` positive integer giving the maximum number of coordinate-search sweeps over the degree vector. Ignored for `"manual"` and `"exhaustive"` degree selection.

`degree.min` optional scalar or integer vector giving lower bounds for automatic degree search over continuous `xdat` predictors when `degree.select != "manual"`.

`degree.restarts` non-negative integer giving the number of additional deterministic coordinate-search restarts. Ignored for `"manual"` and `"exhaustive"` degree selection.

`degree.select` character string controlling local-polynomial degree handling when `regtype="lp"`. `"manual"` (default) treats `degree` as fixed. `"coordinate"` performs cached coordinate-wise search over admissible degree vectors for the continuous `xdat` predictors. `"exhaustive"` evaluates the full admissible degree grid when `search.engine="cell"`. For NOMAD-based search engines, any non-`"manual"` value requests direct joint search over degree and bandwidth coordinates.

`degree.start` optional starting degree vector for automatic coordinate search. If omitted, the search starts from the degree-zero local-constant baseline on the continuous `xdat` predictors.

`degree.verify` logical value indicating whether a coordinate-search solution should be exhaustively verified over the admissible degree grid after the heuristic phase completes. Available only for `search.engine="cell"`.

Bandwidth Criterion And Representation: These arguments choose the selection criterion and the way continuous bandwidths are represented.

`bwmethod` which method to use to select bandwidths. `cv.ls` specifies least-squares cross-validation (Li, Lin and Racine (2013)). `normal-reference` computes an ad hoc Silverman-style rule-of-thumb intended for fast exploratory data analysis, not production bandwidth selection. For raw continuous bandwidths it uses $1.06\sigma_j n^{-1/(2P+l)}$ for conditioning variables and $1.587\sigma_j n^{-1/(2P+l)}$ for response variables, where σ_j is an adaptive measure of spread of the jth continuous variable defined as min(standard deviation, mean absolute deviation/1.4826, interquartile range/1.349), n the number of observations, P the order of the kernel, and l the number of continuous variables. Note that when there exist factors

and the normal-reference rule is used, there is zero smoothing of the factors. Defaults to `cv.ls`.

`bwscaling` a logical value that when set to `TRUE` the supplied bandwidths are interpreted as 'scale factors' (c_j), otherwise when the value is `FALSE` they are interpreted as 'raw bandwidths' (h_j for continuous data types, λ_j for discrete data types). For continuous data types, c_j and h_j are related by the formula $h_j = c_j \sigma_j n^{-1/(2P+l)}$, where σ_j is an adaptive measure of spread of continuous variable j defined as min(standard deviation, mean absolute deviation/1.4826, interquartile range/1.349), n the number of observations, P the order of the kernel, and l the number of continuous variables. For discrete data types, c_j and h_j are related by the formula $h_j = c_j n^{-2/(2P+l)}$, where here j denotes discrete variable j. Defaults to `FALSE`.

`bwtype` character string used for the continuous variable bandwidth type, specifying the type of bandwidth to compute and return in the `condbandwidth` object. Defaults to `fixed`. Option summary:
`fixed`: compute fixed bandwidths
`generalized_nn`: compute generalized nearest neighbors
`adaptive_nn`: compute adaptive nearest neighbors

Categorical Search Initialization: These controls set categorical search starts and categorical direction-set initialization.

`dfac.dir` stretch factor for direction set search for Powell's algorithm for categorical variables. See Details

`dfac.init` non-random initial values for scale factors for categorical variables for Powell's algorithm. See Details

`hbd.dir` upper bound for direction set search for Powell's algorithm for categorical variables. See Details

`hbd.init` upper bound for scale factors for categorical variables for Powell's algorithm. See Details

`initd.dir` initial non-random values for direction set search for Powell's algorithm for categorical variables. See Details

`lbd.dir` lower bound for direction set search for Powell's algorithm for categorical variables. See Details

`lbd.init` lower bound for scale factors for categorical variables for Powell's algorithm. See Details

`scale.init.categorical.sample`
a logical value that when set to `TRUE` scales `lbd.dir`, `hbd.dir`, `dfac.dir`, and `initd.dir` by $n^{-2/(2P+l)}$, n the number of observations, P the order of the kernel, and l the number of `numeric` variables. See Details

Continuous Direction-Set Search Controls: These controls set Powell direction-set initialization for continuous variables.

`cfac.dir` stretch factor for direction set search for Powell's algorithm for `numeric` variables. See Details

`dfc.dir` chi-square degrees of freedom for direction set search for Powell's algorithm for `numeric` variables. See Details

`initc.dir` initial non-random values for direction set search for Powell's algorithm for `numeric` variables. See Details

`lbc.dir` lower bound for direction set search for Powell's algorithm for `numeric` variables. See Details

Continuous Kernel Support Controls: These controls choose and parameterize bounded support for continuous kernels.

`cxkerbound` character string controlling continuous-kernel support handling for `xdat`. Can be set as `none` (default kernel on full support), `range` (use sample min/max), or `fixed` (use `cxkerlb`/`cxkerub`). The bounded-kernel route reuses the selected continuous kernel and renormalizes it on the chosen support; see `np.kernels`.

`cxkerlb` numeric scalar/vector of lower bounds for continuous `xdat` variables used when `cxkerbound="fixed"`. Must satisfy lower-bound validity for each variable (e.g., `<= min(variable)`). Use `-Inf` for unbounded below. See `np.kernels` for bounded-kernel normalization details.

`cxkerub` numeric scalar/vector of upper bounds for continuous `xdat` variables used when `cxkerbound="fixed"`. Must satisfy upper-bound validity for each variable (e.g., `>= max(variable)`). Use `Inf` for unbounded above. See `np.kernels` for bounded-kernel normalization details.

`cykerbound` character string controlling continuous-kernel support handling for `ydat`. Can be set as `none` (default kernel on full support), `range` (use sample min/max), or `fixed` (use `cykerlb`/`cykerub`). For the response-side distribution operator, the selected continuous kernel is integrated after observation-centered truncation and normalization on the chosen support; see `np.kernels`.

`cykerlb` numeric scalar/vector of lower bounds for continuous `ydat` variables used when `cykerbound="fixed"`. Must satisfy lower-bound validity for each variable (e.g., `<= min(variable)`). Use `-Inf` for unbounded below. See `np.kernels` for bounded-kernel normalization details.

`cykerub` numeric scalar/vector of upper bounds for continuous `ydat` variables used when `cykerbound="fixed"`. Must satisfy upper-bound validity for each variable (e.g., `>= max(variable)`). Use `Inf` for unbounded above. See `np.kernels` for bounded-kernel normalization details.

Continuous Scale-Factor Search Initialization: These controls define deterministic and random continuous scale-factor starts and the lower admissibility floor for fixed-bandwidth search.

`scale.factor.init`
deterministic initial scale factor for continuous fixed-bandwidth search. Defaults to `0.5`. The value supplied by the user is not rewritten, but the effective first start passed to the optimizer is `max(scale.factor.init, scale.factor.search.lower)`. See Details.

`scale.factor.init.lower`
lower endpoint for random continuous scale-factor starts. Defaults to `0.1`. The value supplied by the user is not rewritten, but the effective random-start lower endpoint is `max(scale.factor.init.lower, scale.factor.search.lower)`. See Details.

`scale.factor.init.upper`
: upper endpoint for random continuous scale-factor starts. Defaults to `2.0`. It must be greater than or equal to the effective lower endpoint, `max(scale.factor.init.lower, scale.factor.search.lower)`; otherwise bandwidth search errors rather than silently expanding the interval. See Details.

`scale.factor.search.lower`
: optional nonnegative scalar giving the hard lower admissibility bound for continuous fixed-bandwidth search candidates. Defaults to `NULL`. If `NULL`, an existing bandwidth object's stored value is inherited when available; otherwise the package default `0.1` is used. This floor applies to computed/search bandwidth candidates and to effective search starts only. It does not rewrite explicit bandwidths supplied for storage with `bandwidth.compute = FALSE`. Final fixed-bandwidth search candidates must also have a finite valid raw objective value.

Distribution Integral And Grid Controls: These controls tune the conditional distribution-function integral and grid calculations.

`do.full.integral`
: a logical value which when set as `TRUE` evaluates the moment-based integral on the entire sample.

`memfac`
: The algorithm to compute the least-squares objective function uses a block-based algorithm to eliminate or minimize redundant kernel evaluations. Due to memory, hardware and software constraints, a maximum block size must be imposed by the algorithm. This block size is roughly equal to memfac*10^5 elements. Empirical tests on modern hardware find that a memfac of around 500 performs well. If you experience out of memory errors, or strange behaviour for large data sets (>100k elements) setting memfac to a lower value may fix the problem.

`ngrid`
: integer number of grid points to use when computing the moment-based integral. Defaults to `100`.

Kernel Type Controls: These controls choose continuous, unordered, and ordered kernels for `xdat` and `ydat`.

`cxkerorder`
: numeric value specifying kernel order for `xdat` (one of `(2,4,6,8)`). Kernel order specified along with a `uniform` continuous kernel type will be ignored. Defaults to `2`.

`cxkertype`
: character string used to specify the continuous kernel type for `xdat`. Can be set as `gaussian`, `epanechnikov`, or `uniform`. Defaults to `gaussian`.

`cykerorder`
: numeric value specifying kernel order for `ydat` (one of `(2,4,6,8)`). Kernel order specified along with a `uniform` continuous kernel type will be ignored. Defaults to `2`.

`cykertype`
: character string used to specify the continuous kernel type for `ydat`. Can be set as `gaussian`, `epanechnikov`, or `uniform`. Defaults to `gaussian`.

`oxkertype`
: character string used to specify the ordered categorical kernel type for `xdat`. Can be set as `wangvanryzin`, `liracine`, or `racineliyan`. Defaults to `liracine`.

`oykertype`
: character string used to specify the ordered categorical kernel type for `ydat`. Can be set as `wangvanryzin`, `liracine`, or `racineliyan`. Defaults to `liracine`.

uxkertype character string used to specify the unordered categorical kernel type for xdat. Can be set as aitchisonaitken or liracine. Defaults to aitchisonaitken.

Local-Polynomial Model Specification: These arguments control the local-polynomial estimator, basis, and fixed degree specification.

basis character string specifying the polynomial basis used when regtype="lp". Options are "glp", "additive", and "tensor".

bernstein.basis logical value controlling Bernstein basis evaluation for regtype="lp". When automatic degree search is requested and bernstein.basis is not explicitly supplied, the search route defaults to TRUE for numerical stability. Explicit bernstein.basis=FALSE is honored, but raw-polynomial search can be poorly conditioned at higher degrees.

degree integer scalar or integer vector of polynomial degrees for continuous xdat variables when regtype="lp". If scalar, the value is recycled to all continuous xdat variables. With no continuous xdat predictors, regtype="lp" is only admissible when degree=0, the local-constant equivalent.

regtype character string specifying the conditional local method used for the xdat regression weight operator. Options are "lc", "ll", and "lp". For npc* methods, "ll" is implemented via the canonical local polynomial engine with degree = 1 and basis = "glp". "ll" and positive-degree "lp" require at least one continuous xdat predictor; for categorical-only xdat, use "lc" or "lp" with degree=0. If local-linear cv.ls search fails while using this canonical raw basis, retrying explicitly with regtype="lp", degree=1, and bernstein.basis=TRUE, or centering/scaling the continuous regressors, can improve numerical conditioning without changing package defaults or invoking an automatic fallback.

NOMAD Search Controls: These arguments control the optional NOMAD direct-search route for local-polynomial degree and bandwidth search.

nomad logical or character shortcut for the recommended automatic local-polynomial NOMAD route. When TRUE, any missing values among regtype, search.engine, degree.select, bernstein.basis, degree.min, degree.max, degree.verify, and bwtype are filled with regtype="lp", search.engine="nomad+powell", degree.select="coordinate", bernstein.basis=TRUE, degree.min=0L, degree.max=10L, degree.verify=FALSE, and bwtype="fixed". Explicit incompatible settings error immediately; in particular, explicit nomad=TRUE currently requires regtype="lp", bwtype="fixed", automatic degree search, no explicit degree, at least one continuous xdat predictor, and search.engine %in% c("nomad", "nomad+powell"). This shortcut does not change the meaning of nmulti or nomad.nmulti: nmulti remains the outer restart count, while nomad.nmulti controls inner native **crs** NOMAD API multistarts within each outer restart. Returned bandwidth objects retain this normalized preset metadata in bw$nomad.shortcut for a returned object bw; when available, nomad.time and powell.time record the direct-search and Powell-polish timing components.

nomad.nmulti non-negative integer controlling the inner the native **crs** NOMAD C API multistart count used within each outer NOMAD restart when regtype="lp" and

automatic degree search uses `search.engine="nomad"` or `"nomad+powell"`. Defaults to `0L`, which preserves the current one-start-per- restart behavior. This does not replace `nmulti`: `nmulti` controls outer restarts, while `nomad.nmulti` controls inner NOMAD multistarts within each outer restart.

`nomad.opts` optional named list of native NOMAD parameters forwarded to the **crs** NOMAD C API for NOMAD-backed bandwidth searches. `NULL` is treated as no extra NOMAD options. Non-list values and invalid or unsupported parameters are rejected before solver entry rather than silently ignored. Routes that evaluate R callbacks require serial callback evaluation and reject parallel callback settings such as `NB_THREADS_PARALLEL_EVAL > 1`.

`nomad.remin` logical flag controlling the optional second NOMAD hot start. When `TRUE`, NOMAD is restarted once from the best full candidate found, including both bandwidth and degree coordinates. Defaults to `FALSE`; current simulation evidence favors the one-pass NOMAD default for routine use, while leaving this switch available for sensitivity checks.

`search.engine` character string controlling the automatic local-polynomial search backend when `regtype="lp"` and `degree.select != "manual"`. `"nomad+powell"` (default) performs direct joint search over the `xdat`-side continuous bandwidth coordinates and degree vector using the native **crs** NOMAD C API, then applies one Powell hot start from the NOMAD solution. `"nomad"` omits the Powell refinement. `"cell"` profiles the criterion over the admissible degree grid using repeated fixed-degree bandwidth solves. NOMAD-based search supports `bwtype="fixed"`, `bwtype="generalized_nn"`, and `bwtype="adaptive_nn"`; it currently requires `degree.verify=FALSE` and the suggested package **crs** to be installed.

Numerical Search And Tolerance Controls: These controls set optimizer tolerances, restart behavior, invalid-candidate penalties, memory blocking, and bounded search transformations.

`ftol` fractional tolerance on the value of the cross-validation function evaluated at located minima (of order the machine precision or perhaps slightly larger so as not to be diddled by roundoff). Defaults to `1.490116e-07` (1.0e+01*sqrt(.Machine$double.eps)).

`invalid.penalty` a character string specifying the penalty used when the optimizer encounters invalid bandwidths. `"baseline"` returns a finite penalty based on a baseline objective; `"dbmax"` returns `DBL\_MAX`. Defaults to `"baseline"`.

`itmax` integer number of iterations before failure in the numerical optimization routine. Defaults to `10000`.

`nmulti` integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points

`penalty.multiplier` a numeric multiplier applied to the baseline penalty when `invalid.penalty="baseline"`. Defaults to `10`.

`powell.remin` logical flag controlling Powell restart-from-minimum behavior. For ordinary fixed-degree Powell-style search, `TRUE` restarts the local search from the located minimum. For `search.engine="nomad+powell"`, this controls only the final Powell bandwidth-polish step. The default is `TRUE` for ordinary Powell routes and `FALSE` for the Powell polish after NOMAD unless explicitly supplied.

`bwsolver` bandwidth optimizer for fixed-degree searches. The default `"powell"` preserves the historical Powell route. `"mads"` uses the native **crs** NOMAD C API over the bandwidth coordinates, and `"mads+powell"` applies one Powell hot start from the MADS solution. Use `search.engine` rather than `bwsolver` for automatic degree search. Advanced native NOMAD parameters may be supplied through the explicit `nomad.opts` argument; invalid or unsupported parameters are rejected by NOMAD rather than silently ignored.

`small` a small number used to bracket a minimum (it is hopeless to ask for a bracketing interval of width less than sqrt(epsilon) times its central value, a fractional width of only about 10-04 (single precision) or 3x10-8 (double precision)). Defaults to `small = 1.490116e-05 (1.0e+03*sqrt(.Machine$double.eps))`.

`tol` tolerance on the position of located minima of the cross-validation function (tol should generally be no smaller than the square root of your machine's floating point precision). Defaults to `1.490116e-04 (1.0e+04*sqrt(.Machine$double.eps))`.

`transform.bounds`
a logical value that when set to `TRUE` applies an internal transformation that maps the unconstrained search to the feasible bandwidth domain. Defaults to `FALSE`.

Additional Arguments: These arguments collect remaining controls passed through S3 methods.

`...` additional arguments supplied to specify the bandwidth type, kernel types, selection methods, and so on, detailed below.

Details

The `scale.factor.*` controls are dimensionless search controls. The package converts scale factors to bandwidths using the estimator-specific scaling encoded in the bandwidth object, including kernel order and the number of continuous variables relevant for the estimator. Users should not pre-multiply these controls by sample-size or standard-deviation factors.

`scale.factor.init` controls the deterministic first search start. `scale.factor.init.lower` and `scale.factor.init.upper` define the random multistart interval. `scale.factor.search.lower` is the lower admissibility bound for continuous fixed-bandwidth search candidates. The effective first start is `max(scale.factor.init, scale.factor.search.lower)`, and the effective random-start lower endpoint is `max(scale.factor.init.lower, scale.factor.search.lower)`. `scale.factor.init.upper` must be at least that effective lower endpoint; the package errors rather than silently expanding the user's interval.

When `scale.factor.search.lower` is `NULL`, an existing bandwidth object's stored floor is inherited when available; otherwise the package default `0.1` is used. Explicit bandwidths supplied for storage with `bandwidth.compute = FALSE` are not rewritten by the search floor.

Categorical search-start controls such as `dfac.init`, `lbd.init`, and `hbd.init` have separate semantics and are not affected by `scale.factor.search.lower`.

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, `plot`, `plot.np` for plotting options, and `npRmpi.init` for interactive/cluster MPI startup. See `npRmpi.init` details for performance tradeoffs (message passing/startup mode) and the `inst/Rprofile` manual-broadcast template.

The bandwidth-selection argument surface is easiest to read by decision group. Start by initializing MPI execution if needed, then choose the data and bandwidth inputs (xdat, ydat, gydat, bws, and bandwidth.compute), then choose the bandwidth criterion and representation (bwmethod, bwscaling, and bwtype). Next choose continuous kernel and support controls (cxker* and cyker*), categorical kernel controls (uxkertype, oxkertype, and oykertype), and numerical search controls including nmulti, tolerances, penalties, and the scale.factor.* search-start and admissibility controls. Local-polynomial and NOMAD controls (regtype, basis, degree*, search.engine, nomad, nomad.nmulti, and bernstein.basis) are relevant when using the explicit local-polynomial route.

For S3 plotting help, see plot.np. You can list available plot methods with methods("plot").

npcdistbw implements a variety of methods for choosing bandwidths for multivariate distributions ($p + q$-variate) defined over a set of possibly continuous and/or discrete (unordered xdat, ordered xdat and ydat) data. The approach is based on Li and Racine (2004) who employ 'generalized product kernels' that admit a mix of continuous and discrete data types.

The cross-validation methods employ multivariate numerical search algorithms. For fixed local-constant/local-linear fits, and for local-polynomial fits with degree.select="manual", bandwidth search uses multidimensional Powell direction-set optimization.

Bandwidths can (and will) differ for each variable which is, of course, desirable.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i, when estimating the cumulative distribution at the point x. Generalized nearest-neighbor bandwidths change with the point at which the cumulative distribution is estimated, x. Fixed bandwidths are constant over the support of x.

npcdistbw may be invoked *either* with a formula-like symbolic description of variables on which bandwidth selection is to be performed *or* through a simpler interface whereby data is passed directly to the function via the xdat and ydat parameters. Use of these two interfaces is **mutually exclusive**.

Data contained in the data frame xdat may be a mix of continuous (default), unordered discrete (to be specified in the data frames using factor), and ordered discrete (to be specified in the data frames using ordered). Data contained in the data frame ydat may be a mix of continuous (default) and ordered discrete (to be specified in the data frames using ordered). Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see npRmpi for details).

Data for which bandwidths are to be estimated may be specified symbolically. A typical description has the form dependent data ~ explanatory data, where dependent data and explanatory data are both series of variables specified by name, separated by the separation character '+'. For example, y1 + y2 ~ x1 + x2 specifies that the bandwidths for the joint distribution of variables y1 and y2 conditioned on x1 and x2 are to be estimated. See below for further examples.

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth order Gaussian and Epanechnikov kernels, and the uniform kernel. Unordered discrete data types use a variation on Aitchison and Aitken's (1976) kernel, while ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

When regtype="lp" and degree.select != "manual", npcdistbw can jointly determine the xdat-side local polynomial degree vector and the fixed bandwidth coordinates entering the conditional distribution criterion. With search.engine="cell", the criterion is profiled over the admissible degree grid using cached coordinate-wise or exhaustive search. With search.engine="nomad" or

"nomad+powell", the criterion is optimized directly over the joint degree/bandwidth space using the native **crs** NOMAD C API; "nomad+powell" then performs one Powell hot start from the NOMAD solution and keeps the better of the direct NOMAD and polished answers. This polynomial-adaptive joint-search route is motivated by Hall and Racine (2015) together with Li, Li, and Racine (under revision). When bernstein.basis is not explicitly supplied, the automatic search route defaults to bernstein.basis=TRUE for numerical stability.

Setting nomad=TRUE is a convenience preset for this automatic LP route, not a generic optimizer alias. For conditional distribution bandwidth selection it expands any missing values to the equivalent long-form call

```
npcdistbw(...,
          regtype = "lp",
          search.engine = "nomad+powell",
          degree.select = "coordinate",
          bernstein.basis = TRUE,
          degree.min = 0L,
          degree.max = 10L,
          degree.verify = FALSE,
          bwtype = "fixed")
```

Compatible explicit tuning arguments are respected. Incompatible explicit settings fail fast so the shortcut never silently changes user-selected semantics. The character value nomad="auto" applies the same LP shortcut but leaves search.engine and degree.select eligible for data-driven resolution when both were not supplied explicitly: scalar continuous degree searches use the exhaustive degree-grid route, while higher-dimensional degree searches keep the NOMAD/Powell route. Explicit search.engine or degree.select choices are honored.

The optimizer invoked for search is Powell's conjugate direction method which requires the setting of (non-random) initial values and search directions for bandwidths, and, when restarting, random values for successive invocations. Bandwidths for numeric variables are scaled by robust measures of spread, the sample size, and the number of numeric variables where appropriate. Two sets of parameters for bandwidths for numeric can be modified, those for initial values for the parameters themselves, and those for the directions taken (Powell's algorithm does not involve explicit computation of the function's gradient). The default values are set by considering search performance for a variety of difficult test cases and simulated cases. We highly recommend restarting search a large number of times to avoid the presence of local minima (achieved by modifying nmulti). Further refinement for difficult cases can be achieved by modifying these sets of parameters. However, these parameters are intended more for the authors of the package to enable 'tuning' for various methods rather than for the user themselves.

Value

npcdistbw returns a condbandwidth object, with the following components:

xbw	bandwidth(s), scale factor(s) or nearest neighbours for the explanatory data, xdat
ybw	bandwidth(s), scale factor(s) or nearest neighbours for the dependent data, ydat
fval	objective function value at minimum

if bwtype is set to fixed, an object containing bandwidths (or scale factors if bwscaling = TRUE) is returned. If it is set to generalized_nn or adaptive_nn, then instead the kth nearest neighbors are returned for the continuous variables while the discrete kernel bandwidths are returned for the discrete variables.

The functions predict, summary and plot support objects of type condbandwidth.

Book And Method Pointers

npcdistbw selects response and conditioning bandwidths, and when requested local-polynomial degree/search metadata, for the conditional distribution target $F(y \mid x) = \Pr(Y \leq y \mid X = x)$ estimated by npcdist and inverted by npqreg.

For book-length derivations, see Li and Racine (2007), Chapter 6 *Conditional CDF and Quantile Estimation*, especially Sections 6.1, 6.2, 6.3, and 6.5, and Chapter 5 *Conditional Density Estimation*. The later workflow treatment is Racine (2019), Chapter 4.

Usage Issues

If you are using data of mixed types, then it is advisable to use the data.frame function to construct your input data and not cbind, since cbind will typically not work as intended on mixed data types and will coerce the data to the same type.

Caution: multivariate data-driven bandwidth selection methods are, by their nature, *computationally intensive*. Virtually all methods require dropping the ith observation from the data set, computing an object, repeating this for all observations in the sample, then averaging each of these leave-one-out estimates for a *given* value of the bandwidth vector, and only then repeating this a large number of times in order to conduct multivariate numerical minimization/maximization. Furthermore, due to the potential for local minima/maxima, *restarting this procedure a large number of times may often be necessary*. This can be frustrating for users possessing large datasets. For exploratory purposes, you may wish to override the default search tolerances, say, setting ftol=.01 and tol=.01 and conduct multistarting (the default is to restart min(2, ncol(xdat,ydat)) times) as is done for a number of examples. Once the procedure terminates, you can restart search with default tolerances using those bandwidths obtained from the less rigorous search (i.e., set bws=bw on subsequent calls to this routine where bw is the initial bandwidth object). A version of this package using the Rmpi wrapper is under development that allows one to deploy this software in a clustered computing environment to facilitate computation involving large datasets.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

Hall, P. and J.S. Racine and Q. Li (2004), "Cross-validation and the estimation of conditional probability densities," Journal of the American Statistical Association, 99, 1015-1026.

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

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

Li, Q. and J.S. Racine (2008), "Nonparametric estimation of conditional CDF and quantile functions with mixed categorical and continuous data," Journal of Business and Economic Statistics, 26, 423-434.

Li, Q. and J. Lin and J.S. Racine (2013), "Optimal bandwidth selection for nonparametric conditional distribution and quantile functions", Journal of Business and Economic Statistics, 31, 57-65.

Li, A. and Q. Li and J.S. Racine (under revision), "Boundary Adjusted, Polynomial Adaptive, Nonparametric Kernel Conditional Density Estimation," Econometric Reviews.

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

Scott, D.W. (1992), *Multivariate Density Estimation. Theory, Practice and Visualization,* New York: Wiley.

Silverman, B.W. (1986), *Density Estimation,* London: Chapman and Hall.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

`np.kernels`, `np.options`, `plot`, `plot.np` `bw.nrd`, `bw.SJ`, `hist`, `npudens`, `npudist`

Examples

```
## Not run:
## Not run in checks: data-driven conditional CDF bandwidth selection is
## computationally intensive and may exceed check limits under MPI.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.
npRmpi.init(nslaves=1)

data("Italy")

bw <- npcdistbw(formula=gdp~ordered(year),
                data=Italy)

summary(bw)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.
```

```
## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close

## End(Not run)
```

---

npcdisthat *Conditional Distribution Hat Operator*

Description

Constructs the conditional distribution hat operator associated with npcdist bandwidth objects. The returned operator maps a right-hand side y to Hy; with y = 1 this reproduces the fitted conditional distribution function.

Usage

```
npcdisthat(bws,
           txdat = stop("training data 'txdat' missing"),
           tydat = stop("training data 'tydat' missing"),
           exdat,
           eydat,
           y = NULL,
           output = c("matrix", "apply"),
           deriv = NULL,
           s = NULL)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the fitted bandwidth object, training data, and evaluation data.

bws	A fitted conditional distribution bandwidth object of class "condbandwidth".
exdat	Optional evaluation conditioning data. If omitted, the operator is built on the training conditioning data.
eydat	Optional evaluation response data. If omitted, the operator is built on the training response data.
txdat	Training conditioning data used to construct the operator.

`tydat` Training response data used to construct the operator.

Operator Output: These arguments control whether the operator is returned as a matrix or applied directly.

`output` Either `"matrix"` for the hat matrix or `"apply"` for direct application to `y`.

`y` Optional right-hand side vector or matrix with one row per training observation.

Derivative Controls: These arguments request derivative hat operators with respect to continuous conditioning variables.

`deriv` Optional derivative selector passed through to the continuous-conditioning derivative resolver.

`s` Optional derivative multi-index for continuous conditioning variables. The default `s = NULL` returns the ordinary conditional-distribution hat operator.

Details

For `output = "matrix"`, the return value is a matrix with class `c("npcdisthat", "matrix")` and attributes storing the bandwidth object, training data, evaluation data, and call metadata.

For `output = "apply"`, the function returns `H y` directly. Matrix right-hand sides are applied column-wise.

The optional `s` argument follows the derivative multi-index convention used by `npreghat` for continuous conditioning variables. The default `s = NULL` returns the ordinary conditional-distribution hat operator. Supplying one first derivative, for example `s = 1L` in a univariate continuous-x problem or `s = c(x2 = 1L)` in a multivariate problem, returns the hat operator for that component of the conditional-distribution gradient. This derivative route is intended to match `npcdist(..., gradients = TRUE)$congrad`.

This helper is intended for object-fed repeated evaluation once a bandwidth object has already been constructed. It does not perform bandwidth selection.

Value

Either a hat matrix of class `"npcdisthat"` or the applied result `H y`, depending on `output`.

Examples

```
## Not run:
npRmpi.init(nslaves = 1)
data(cps71)
tx <- data.frame(age = cps71$age)
ty <- data.frame(logwage = cps71$logwage)

bw <- npcdistbw(xdat = tx, ydat = ty, bwtype = "fixed",
                bandwidth.compute = FALSE, bws = c(1.0, 1.0))

H <- npcdisthat(bws = bw, txdat = tx, tydat = ty)
dist.hat <- npcdisthat(bws = bw, txdat = tx, tydat = ty,
                       y = rep(1, nrow(tx)),
```

```
                        output = "apply")
dist.core <- fitted(npcdist(bws = bw, txdat = tx, tydat = ty))

head(cbind(dist.core, dist.hat), n = 2L)

npRmpi.quit()

## End(Not run)
```

---

npcmstest *Kernel Consistent Model Specification Test with Mixed Data Types*

Description

npcmstest implements a consistent test for correct specification of parametric regression models (linear or nonlinear) as described in Hsiao, Li, and Racine (2007).

Usage

```
npcmstest(formula,
          data = NULL,
          subset,
          xdat,
          ydat,
          model = stop(paste(sQuote("model"),," has not been provided")),
          distribution = c("bootstrap", "asymptotic"),
          boot.method = c("iid","wild","wild-rademacher"),
          boot.num = 399,
          pivot = TRUE,
          density.weighted = TRUE,
          random.seed = 42,
          ...)
```

Arguments

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

data	an optional data frame, list or environment (or object coercible to a data frame by as.data.frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which the function is called.
formula	a symbolic description of variables on which the test is to be performed. The details of constructing a formula are described below.
model	a model object obtained from a call to lm (or glm). Important: the call to either glm or lm must have the arguments x=TRUE and y=TRUE or npcmstest will not work. Also, the test is based on residual bootstrapping hence the outcome must be continuous (which rules out Logit, Probit, and Count models).

subset — an optional vector specifying a subset of observations to be used.

xdat — a p-variate data frame of explanatory data (training data) used to calculate the regression estimators.

ydat — a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of xdat.

Bootstrap And Test Controls: These arguments control the test statistic, bootstrap procedure, and reproducibility settings.

boot.method — a character string used to specify the bootstrap method. iid will generate independent identically distributed draws. wild will use a wild bootstrap. wild-rademacher will use a wild bootstrap with Rademacher variables. Defaults to iid.

boot.num — an integer value specifying the number of bootstrap replications to use. Defaults to 399.

density.weighted — a logical value specifying whether the statistic should be weighted by the density of xdat. Defaults to TRUE.

distribution — a character string used to specify the method of estimating the distribution of the statistic to be calculated. bootstrap will conduct bootstrapping. asymptotic will use the normal distribution. Defaults to bootstrap.

pivot — a logical value specifying whether the statistic should be normalised such that it approaches $N(0,1)$ in distribution. Defaults to TRUE.

random.seed — an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.

Additional Arguments: Further arguments are passed to the bandwidth-selection routines used by the test.

... — additional arguments supplied to control bandwidth selection on the residuals. One can specify the bandwidth type, kernel types, and so on. To do this, you may specify any of bwscaling, bwtype, ckertype, ckerorder, ukertype, okertype, as described in npregbw. This is necessary if you specify bws as a p-vector and not a bandwidth object, and you do not desire the default behaviours.

Details

For MPI startup/performance guidance (including message-passing tradeoffs and the manual-broadcast template), see npRmpi.init details and inst/Rprofile. Documentation guide: see np.kernels for kernels, np.options for global options, and plot for plotting options.

Value

npcmstest returns an object of type cmstest with the following components, components will contain information related to Jn or In depending on the value of pivot:

Jn — the statistic Jn

In — the statistic In

Omega.hat	as described in Hsiao, C. and Q. Li and J.S. Racine.
q.*	the various quantiles of the statistic Jn (or In if pivot=FALSE) are in components q.90, q.95, q.99 (one-sided 1%, 5%, 10% critical values)
P	the P-value of the statistic
Jn.bootstrap	if pivot=TRUE contains the bootstrap replications of Jn
In.bootstrap	if pivot=FALSE contains the bootstrap replications of In

summary supports object of type cmstest.

Book And Method Pointers

npcmstest tests a parametric conditional-mean specification against a nonparametric alternative, using residual-based bootstrap or asymptotic calibration as selected by distribution. The statistic is designed for continuous outcomes and fitted lm or glm objects whose model frame, model matrix, and response are available.

For book-length background, see Li and Racine (2007), Chapter 12 *Model Specification Tests*, especially Section 12.1 and Section 12.1.2. For the conditional-mean estimation context used by the test, see Racine (2019), Chapter 6 *Conditional Mean Function Estimation*.

Usage Issues

npcmstest supports regression objects generated by lm and uses features specific to objects of type lm hence if you attempt to pass objects of a different type the function cannot be expected to work.

If you are using data of mixed types, then it is advisable to use the data.frame function to construct your input data and not cbind, since cbind will typically not work as intended on mixed data types and will coerce the data to the same type.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

Hsiao, C. and Q. Li and J.S. Racine (2007), "A consistent model specification test with mixed categorical and continuous data," Journal of Econometrics, 140, 802-826.

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

Maasoumi, E. and J.S. Racine and T. Stengos (2007), "Growth and convergence: a profile of distribution dynamics and mobility," Journal of Econometrics, 136, 483-508.

Murphy, K. M. and F. Welch (1990), "Empirical age-earnings profiles," Journal of Labor Economics, 8, 202-229.

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

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

npRmpi.init. np.kernels, np.options, plot, npregbw.

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

data(cps71)

model <- lm(logwage~age+I(age^2), data=cps71, x=TRUE, y=TRUE)

npcmstest(model = model, xdat = cps71$age, ydat = cps71$logwage,
          boot.num=9, nmulti = 1)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.
```

```
npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

npconmode *Kernel Modal Regression with Mixed Data Types*

Description

npconmode performs kernel modal regression on mixed data, and finds the conditional mode given a set of training data, consisting of explanatory data and dependent data, and possibly evaluation data. Automatically computes various in sample and out of sample measures of accuracy.

Usage

```
npconmode(bws, ...)

## S3 method for class 'formula'
npconmode(bws,
          data = NULL,
          newdata = NULL,
          ...)

## Default S3 method:
npconmode(bws,
          txdat,
          tydat,
          nomad = FALSE,
          proper = NULL,
          proper.control = list(),
          probabilities = FALSE,
          gradients = FALSE,
          level = NULL,
          ...)

## S3 method for class 'conbandwidth'
npconmode(bws,
          txdat = stop("invoked without training data 'txdat'"),
          tydat = stop("invoked without training data 'tydat'"),
          exdat,
          eydat,
          proper = NULL,
          proper.control = list(),
```

```
          probabilities = FALSE,
          gradients = FALSE,
          level = NULL,
          ...)

## S3 method for class 'conmode'
predict(object,
          newdata = NULL,
          type = c("class", "prob"),
          se.fit = FALSE,
          ...)

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

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and training data.

bws	a bandwidth specification. This can be set as a conbandwidth object returned from an invocation of npcdensbw
data	an optional data frame, list or environment (or object coercible to a data frame by as.data.frame) containing the variables in the model. If not found in data, the variables are taken from environment(bws), typically the environment from which npcdensbw was called.
txdat	a p-variate data frame of explanatory data (conditioning data) used to calculate the regression estimators. Defaults to the corresponding training data recoverable from the bandwidth object in the current session.
tydat	a one (1) dimensional vector of unordered or ordered factors, containing the dependent data. Defaults to the corresponding training data recoverable from the bandwidth object in the current session.
object	an object of class "conmode" returned by npconmode.
x	an object of class "conmode" returned by npconmode.

Evaluation Data: These arguments control where the conditional mode is evaluated.

exdat	a p-variate data frame of points on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by txdat.
eydat	a one (1) dimensional numeric or integer vector of the true values (outcomes) of the dependent variable. By default, evaluation takes place on the data provided by tydat.
newdata	An optional data frame in which to look for evaluation data. If omitted, the training data are used.

`type` prediction type for `predict.conmode`. Use `"class"` to return modal class predictions and `"prob"` to return the full class-probability matrix. Probability prediction requires either a stored probability matrix or an evaluation path with `probabilities=TRUE`.

`se.fit` a logical value for `predict.conmode`. If `TRUE` and `type="prob"`, return a list with components `fit` and `se.fit`, containing the class-probability matrix and matching asymptotic standard-error matrix. Standard errors are for the smooth class-probability target, not for the discrete modal class label.

Probability And Gradient Controls: These arguments control stored class probabilities, proper probability normalization, and level-specific class-probability effects.

`probabilities` a logical value. If `TRUE`, store the full matrix of fitted probabilities over the discrete response support in the returned object. The default `FALSE` keeps returned objects compact. If `gradients=TRUE`, probabilities are stored automatically because they are the object-fed target for `plot.conmode`.

`gradients` a logical value. If `TRUE`, compute and store class-probability gradients/effects with respect to the conditioning variables. These are derivatives for continuous conditioning variables and the corresponding smooth finite-change effects for discrete conditioning variables as produced by the underlying `npcdens` gradient route. The target is the smooth class probability $\Pr(Y = \ell \mid X = x)$, not the derivative of the discrete modal class label. Gradients are stored for one selected response level at a time.

`level` response level for class-probability gradients/effects and `plot.conmode` class-probability displays. If omitted, the default is the base/reference response level, `levels(y)[1]`, for both factor and ordered-factor responses.

`proper` a logical value or `NULL`. If `TRUE`, fitted probabilities over the discrete response support are projected onto the probability simplex before modal selection. If `FALSE`, legacy raw per-level values are used. If `NULL`, the default is estimator-aware: `FALSE` for `regtype="lc"` and `TRUE` otherwise.

`proper.control` a list of controls for discrete probability properization. Currently `tol` may be supplied to set the numerical tolerance for non-negativity and unit-mass checks.

Local-Polynomial Degree And Bandwidth Search: This argument controls the recommended automatic local-polynomial NOMAD route, which jointly selects continuous polynomial degree and bandwidths when conditional-density bandwidths are computed inside `npconmode`.

`nomad` logical shortcut passed through to `npcdensbw` when bandwidths are computed inside `npconmode`. When `TRUE`, the conditional-density bandwidth route fills any missing values among `regtype`, `search.engine`, `degree.select`, `bernstein.basis`, `degree.min`, `degree.max`, `degree.verify`, and `bwtype` with the recommended automatic local-polynomial degree-and-bandwidth NOMAD preset documented in `npcdensbw`. Additional NOMAD tuning arguments such as `nomad.nmulti` may also be supplied through `...`; `nmulti` remains the outer restart count while `nomad.nmulti` controls inner native **crs** NOMAD API multistarts within each outer restart. Automatic NOMAD degree search requires at least one continuous explanatory variable. After fitting, inspect `fit$bws$nomad.shortcut` on the returned object `fit` to see the normalized shortcut metadata.

Additional Arguments: Further arguments are passed to the bandwidth-selection counterpart, prediction/evaluation route, or plot route as appropriate.

. . . additional arguments supplied to npcdensbw when bandwidths are computed internally, or arguments needed to interpret a numeric bws vector. This is where bandwidth-selection controls such as bwmethod, bwtype, and bwscaling, kernel/support controls such as cxkertype, cykertype, cxkerorder, cykerorder, cxkerbound, and cykerbound, categorical kernel controls such as uxkertype, uykertype, oxkertype, and oykertype, search controls such as nmulti and scale.factor.search.lower, and local-polynomial/NOMAD controls such as regtype, degree, bernstein.basis, degree.select, and nomad.nmulti are supplied. In predict.conmode, additional arguments are passed to npconmode for evaluation with the stored bandwidth object; native exdat and eydat take precedence over newdata. In plot.conmode, additional arguments control the object-fed probability/effect display; common controls include gradients, level, output, data_rug, layout, legend, view, neval, grid_control, perspective, plot.vars, renderer, and graphics arguments. The default view="sample" draws the stored fitted probabilities/effects at the fitted evaluation points. The view="fixed" route constructs one-dimensional object-fed slices over each conditioning variable, holding the remaining variables at their uocquantile median/mode values; np_grid_control(xtrim=..., xq=...) may be used to adjust the slice range and holdout quantiles. With perspective=TRUE, the same fixed-grid route draws a base-graphics probability surface for one selected response level over two continuous conditioning variables; use plot.vars to name the displayed pair when more than two continuous conditioning variables are present; renderer="rgl" requests the corresponding interactive surface when **rgl** is available. The public output="both" spelling is accepted as an alias for the historical output="plot-data" behavior. For class-probability plots, errors="asymptotic" adds probability-level asymptotic standard errors and intervals for the selected response level. These intervals target the smooth probability $\Pr(Y = \ell \mid X = x)$, not the discrete modal class label. Rows whose probabilities were materially repaired by simplex projection report NA standard errors rather than treating raw conditional-density standard errors as projected-probability standard errors. Bootstrap intervals are available for one-dimensional view="sample" and view="fixed" probability plots and use the stored bandwidth object without recomputing bandwidths. Surface asymptotic intervals are available as output="data" payload columns; surface band rendering and surface bootstrap intervals are intentionally deferred. See npcdensbw and plot.np for the complete bandwidth-selection and plotting argument surfaces.

Details

For MPI startup/performance guidance (including message-passing tradeoffs and the manual-broadcast template), see npRmpi.init details and inst/Rprofile. Documentation guide: see np.kernels for kernels, np.options for global options, and plot for plotting options.

If bws is not an explicit conbandwidth object, npconmode computes conditional density bandwidths by forwarding the call to npcdensbw. The resulting conbandwidth object is stored unchanged in the returned object's bws component. Regression type, local-polynomial degree, and NOMAD/search

metadata are also mirrored on the returned conmode object for convenient inspection and summary reporting; the bandwidth object remains the canonical source.

For non-local-constant conditional-mode fits, npconmode constructs the full set of fitted probabilities over the discrete response support and projects them onto the probability simplex before selecting and reporting the modal outcome. Thus the probabilities used for modal selection are non-negative and sum to one over the discrete support. For binary outcomes this is the complement contract $\Pr(Y = \ell_2 \mid X = x) = 1 - \Pr(Y = \ell_1 \mid X = x)$. Local-constant fits are already proper by construction, so omitted proper resolves to FALSE for regtype="lc" and to TRUE otherwise.

The predict method follows the usual S3 newdata convention. With no evaluation arguments, predict(fit) extracts the stored modal class and predict(fit, type="prob") extracts stored class probabilities when the object was fitted with probabilities=TRUE. With newdata, predict evaluates the conditional mode at the supplied rows using the stored bandwidth object. Use type="prob" to return the full matrix of fitted class probabilities at the evaluation rows. Use se.fit=TRUE with type="prob" to return a list containing the class-probability matrix and the matching asymptotic standard-error matrix. Native evaluator arguments exdat and eydat remain available for advanced workflows and take precedence if supplied.

Value

npconmode returns a conmode object with the following components:

- bws — the conbandwidth object used to compute the conditional mode
- conmode — a vector of type factor (or ordered factor) containing the conditional mode at each evaluation point
- condens — a vector of numeric type containing the modal density estimates at each evaluation point
- conderr — a vector of numeric type containing asymptotic standard errors for the modal density estimates at each evaluation point. If a row is materially properized by probability projection, this value is set to NA for that row rather than reporting a standard error for the unrepaired raw probability.
- xeval — a data frame of evaluation points
- yeval — a vector of type factor (or ordered factor) containing the actual outcomes, or NA if not provided
- confusion.matrix — the confusion matrix or NA if outcomes are not available
- CCR.overall — the overall correct classification ratio, or NA if outcomes are not available
- CCR.byoutcome — a numeric vector containing the correct classification ratio by outcome, or NA if outcomes are not available
- fit.mcfadden — the McFadden-Puig-Kerschner performance measure or NA if outcomes are not available
- probabilities — if requested, a matrix of fitted probabilities over the discrete response support
- probability.levels — if probabilities are stored, the response support corresponding to the probability matrix columns

`probability.errors`
: if `probabilities` are stored, a matrix of asymptotic standard errors for the raw selected-level conditional probability estimates, with rows set to `NA` where simplex projection materially repaired the fitted probability vector

`probability.repaired.rows`
: if `probabilities` are stored, a logical vector identifying rows whose fitted probability vector was materially repaired by proper probability projection

`probability.gradients`
: if `gradients=TRUE`, a matrix whose entries are class-probability gradients/effects for one selected response level, indexed by evaluation row and conditioning variable

`probability.gradient.level`, `probability.gradient.names`, `probability.gradient.info`
: metadata describing the response support, conditioning variables, and interpretation of `probability.gradients`

`proper.requested`, `proper.applied`, `proper.info`
: metadata describing whether discrete probability properization was requested, whether any row was materially projected, and diagnostics for non-negativity and unit-mass checks

`regtype`, `degree`, `nomad`, `search.engine`
: metadata mirrored from `bws` describing the conditional density regression type, local-polynomial degree, whether NOMAD automatic degree search was used, and the search engine when available

`degree.search`, `nomad.shortcut`, `nomad.time`, `powell.time`
: detailed search metadata mirrored from `bws` when automatic degree search or NOMAD/Powell refinement was used

The function `predict` may be used to extract conditional mode class predictions, while `fitted` extracts the conditional density estimates at the conditional mode from the resulting object. The function `gradients` extracts class-probability gradients/effects when `gradients=TRUE`. Also, `summary` and `plot` support `conmode` objects. For `plot.conmode`, first fit with `probabilities=TRUE` so the plot can remain object-fed. The default plot displays the fitted probability for the base/reference response level `levels(y)[1]`; use `plot(fit, level=...)` to display another response level and `plot(fit, gradients=TRUE)` to display stored class-probability effects. The default `view="sample"` draws stored object-fed probability/effect payloads at the fitted evaluation points. Use `view="fixed"` and `neval` to draw object-fed one-dimensional slices over each conditioning variable, with other variables held at their median/mode values. Use `perspective=TRUE` to draw a base-graphics probability surface for one selected response level over two continuous conditioning variables, and `renderer="rgl"` for the corresponding interactive surface. Use `errors="asymptotic"` for probability-level standard errors and intervals in one-dimensional plots, or with `output="data"` for surface interval payloads. Bootstrap intervals and surface band rendering are not yet implemented for `plot.conmode`.

Book And Method Pointers

The conditional-mode target is $\arg\max_y \Pr(Y = y \mid X = x)$ for a discrete response. In practice `npconmode` estimates the conditional probability of each response support point using `npcdensbw` / `npcdens`, optionally projects non-local-constant fitted probabilities onto the probability simplex, and then reports the modal support point.

Setting `gradients=TRUE` stores class-probability effects for one response level. If `level` is omitted, the base/reference response level `levels(Y)[1]` is used. For a selected support point ℓ, the stored effects target

$$\nabla_x \Pr(Y = \ell \mid X = x),$$

with continuous variables interpreted as derivatives and discrete variables interpreted as the corresponding smooth finite-change effects. These effects are useful for asking how the fitted probability of a class changes with each covariate; they are not gradients of the `arg max` classification rule itself.

For book-length background, see Racine (2019), Chapter 4 *Conditional Probability Density and Cumulative Distribution Functions*, especially the binary and multinomial choice material, and Li and Racine (2007), Chapter 5 *Conditional Density Estimation* together with Chapter 4 *Kernel Estimation with Mixed Data*.

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

When reusing a bandwidth object without explicit training or evaluation data, the required data are recovered from object-retained state or from the stored call environment when available. This is a convenience for ordinary same-session use, not a guarantee that a saved object is self-contained. For reproducible replay after `saveRDS()` and `readRDS()` in a fresh R session, supply the required data arguments explicitly.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

Hall, P. and J.S. Racine and Q. Li (2004), "Cross-validation and the estimation of conditional probability densities," Journal of the American Statistical Association, 99, 1015-1026.

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

McFadden, D. and C. Puig and D. Kerschner (1977), "Determinants of the long-run demand for electricity," Proceedings of the American Statistical Association (Business and Economics Section), 109-117.

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

Scott, D.W. (1992), *Multivariate Density Estimation. Theory, Practice and Visualization,* New York: Wiley.

Silverman, B.W. (1986), *Density Estimation,* London: Chapman and Hall.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

npRmpi.init. np.kernels, np.options, plot, npcdensbw.

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

library(MASS)
data(birthwt)

birthwt$low <- factor(birthwt$low)
birthwt$smoke <- factor(birthwt$smoke)
birthwt$race <- factor(birthwt$race)
birthwt$ht <- factor(birthwt$ht)
birthwt$ui <- factor(birthwt$ui)
birthwt$ftv <- ordered(birthwt$ftv)

bw <- npcdensbw(low~
                smoke+
                race+
                ht+
                ui+
                ftv+
                age+
                lwt,
                data=birthwt)
```

```
summary(bw)

model <- npconmode(bws=bw)

summary(model)

## Predict modal classes and fitted class probabilities at selected rows.
new.birthwt <- birthwt[1:5, c("smoke", "race", "ht", "ui", "ftv", "age", "lwt")]
predict(model, newdata=new.birthwt)
predict(npconmode(bws=bw, probabilities=TRUE),
        newdata=new.birthwt, type="prob")

## Class-probability effects can be requested for interactive inspection.
## The default plot displays the fitted probability for the base/reference
## response level; level selects another response level when desired, and
## gradients=TRUE plots the corresponding class-probability effects.

set.seed(42)
n <- 100
x <- seq(-1, 1, length.out=n)
y <- factor(rbinom(n, 1, plogis(1.5*x)), levels=0:1)

model.effects <- npconmode(y~x,
                           regtype="ll",
                           bwmethod="cv.ls",
                           nmulti=1,
                           probabilities=TRUE,
                           gradients=TRUE)

gradients(model.effects)
plot(model.effects)
plot(model.effects, view="fixed", neval=25)
plot(model.effects, gradients=TRUE)
plot(model.effects, gradients=TRUE, view="fixed", neval=25)

## A two-continuous-predictor fit can be displayed as a probability surface.
## The default level is the base/reference response level.

z <- runif(n, -1, 1)
y2 <- factor(rbinom(n, 1, plogis(1.5*x - z)), levels=0:1)
model.surface <- npconmode(y2~x+z,
                           regtype="ll",
                           bwmethod="cv.ls",
                           nmulti=1,
                           probabilities=TRUE)
plot(model.surface, perspective=TRUE, neval=15)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
```

```
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

npcopula *Kernel Copula Estimation with Mixed Data Types*

Description

npcopula estimates a mixed-data kernel copula distribution or copula density. It can be called with an existing unconditional distribution/density bandwidth object, or with a one-sided formula in which case the appropriate bandwidth object is selected internally.

Usage

```
npcopula(bws, ...)

## S3 method for class 'formula'
npcopula(bws,
         data = NULL,
         u = NULL,
         target = c("distribution", "density"),
         evaluation = c("grid", "sample"),
         neval = 30,
         n.quasi.inv = 1000,
         er.quasi.inv = 1,
         ...)

## Default S3 method:
npcopula(bws,
         data,
         u = NULL,
         target = NULL,
         evaluation = c("sample", "grid"),
         neval = 30,
         n.quasi.inv = 1000,
         er.quasi.inv = 1,
```

```
          ...)

## S3 method for class 'npcopula'
predict(object,
        newdata = NULL,
        u = NULL,
        se.fit = FALSE,
        output = c("vector", "object", "data"),
        ...)

## S3 method for class 'npcopula'
plot(x,
     perspective = TRUE,
     view = c("rotate", "fixed", "surface", "contour", "image",
              "empirical", "all"),
     renderer = c("base", "rgl"),
     errors = c("none", "bootstrap", "asymptotic"),
     band = c("pointwise", "pmzsd", "bonferroni",
              "simultaneous", "all"),
     alpha = 0.05,
     bootstrap = c("inid", "fixed", "geom"),
     B = 1999,
     center = c("estimate", "bias-corrected"),
     boot_control = np_boot_control(),
     output = c("plot", "data", "plot-data", "both"),
     legend = TRUE,
     theta = 0.0,
     phi = 20.0,
     xlab = "u1",
     ylab = "u2",
     zlab = NULL,
     main = NULL,
     col = NULL,
     border = .np_plot_color("surface_border"),
     zlim = NULL,
     ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, source data, and formula route.

`bws` bandwidth specification or one-sided formula. A bandwidth specification is either an unconditional distribution bandwidth object returned by `npudistbw`, for a copula distribution, or an unconditional density bandwidth object returned by `npudensbw`, for a copula density. If `bws` is a formula such as `~ x + y`, `npcopula` first calls `npudistbw` when `target="distribution"` and `npudensbw` when `target="density"`; additional bandwidth-selection arguments in `...` are forwarded to that bandwidth selector.

`data` data frame containing the variables used to construct `bws`; when `bws` is a formula, `data` is passed to the bandwidth selector. Copulas are defined here for numeric and ordered variables; unordered factors are rejected.

Copula Target And Evaluation Grid: These arguments control whether a copula distribution or density is estimated and where it is evaluated.

`er.quasi.inv` fraction passed to `extendrange` when constructing the marginal quasi-inverse grid used for supplied probability values `u`. See Details.

`evaluation` evaluation route used when `bws` is a formula. The default `"grid"` constructs a plot-ready two-dimensional probability grid when `u` is omitted. Use `"sample"` to evaluate at the sample realizations.

`n.quasi.inv` number of grid points used to compute each marginal quasi-inverse when `u` is supplied or automatically generated.

`newdata` optional prediction data for `predict.npcopula`. This is a compatibility alias for probability coordinates in columns named `u1`, `u2`, ... It is not interpreted as raw data on the original variable scale. If both `u` and `newdata` are supplied, `u` takes precedence.

`neval` number of probability values per margin in the automatically generated two-dimensional grid used when `bws` is a formula, `evaluation="grid"`, and `u` is omitted.

`target` target used when `bws` is a formula. The default `"distribution"` estimates a copula distribution using `npudistbw`; `"density"` estimates a copula density using `npudensbw`. When `bws` is already a bandwidth object, the target is inferred from that object and a conflicting explicit `target` is rejected.

`u` optional matrix or data frame of marginal probability values in $[0, 1]$. Each column corresponds to one variable in the copula. If supplied, `npcopula` evaluates on the Cartesian product `expand.grid(u)`. For two-dimensional displays, the clearest spelling is often `data.frame(u1 = u1.seq, u2 = u2.seq)`; names matching the original variables are also accepted here because `u` explicitly denotes probability coordinates. If omitted with a formula route and `evaluation="grid"`, a two-dimensional grid is generated automatically.

`object` an object of class `"npcopula"` returned by `npcopula`.

`se.fit` logical value. If `TRUE`, `predict.npcopula` returns a list with fitted values and the stored standard-error slot.

Plot Display Controls: These arguments control how a two-dimensional grid copula object is displayed.

`x` an object of class `"npcopula"` returned by `npcopula`.

`border` border color for surface facets and interval wireframes when `view="surface"`.

`col` optional surface or image colors. If omitted, `plot.npcopula` uses the same viridis `hcl.colors` palette used by the other modern surface plot methods.

`legend` logical or legend-control value used for `band="all"` interval overlays. Use `FALSE`, `NULL`, or `NA` to suppress the interval legend.

main, xlab, ylab, zlab
: plot titles and axis labels.

output
: return mode. For plot.npcopula, "plot" draws the plot, "data" returns the plotted data with interval columns when requested, and "plot-data" draws and returns the plotted data. "both" is accepted as an alias for "plot-data". For predict.npcopula, "vector" returns fitted copula values, "object" returns the evaluated "npcopula" object, and "data" returns as.data.frame() on that object.

perspective
: logical value. If TRUE, draw a surface display; otherwise use view to choose "contour" or "image". The "empirical" and "all" views ignore this argument.

phi, theta
: viewing angles passed to persp or to the shared **rgl** surface renderer. The defaults match the package-wide surface-plot defaults used by the other perspective plot methods. As with those methods, the exact default pair theta = 0 and phi = 20 is remapped internally for renderer="rgl" to account for the different viewing-angle convention used by **rgl**; explicitly supplied non-default angles are passed through.

renderer
: plotting renderer for surface displays. "base" uses persp. "rgl" uses the shared interactive **rgl** surface renderer when the suggested package **rgl** is installed.

view
: display type for grid output. The default "rotate" draws a rotating base persp surface using the same frame step and delay as the other package perspective plots. Use "fixed" for a single fixed surface, "surface" as a backward-compatible fixed-surface alias, or "contour" and "image" with perspective=FALSE. Use "empirical" to plot the empirical copula coordinates. Use "all" for a base-graphics four-panel display containing the copula contour, copula surface, empirical copula coordinates, and a copula-density surface. The "all" view is not currently supported with renderer="rgl".

zlim
: optional z-axis limits for surface displays.

Plot Interval Controls: These arguments add asymptotic or bootstrap intervals to two-dimensional surface plots.

alpha
: nominal size used for asymptotic or bootstrap intervals when errors!="none".

B
: number of bootstrap replications when errors="bootstrap".

band
: interval type for plotted surfaces. Supported values are "pointwise", "pmzsd", "bonferroni", "simultaneous", and "all". The "all" option overlays pointwise, simultaneous, and Bonferroni wireframes where available.

bootstrap
: bootstrap resampling method used when errors="bootstrap"; supported values are "inid", "fixed", and "geom". Wild bootstrap is not defined for copula surfaces.

boot_control
: optional np_boot_control object. For copula surfaces the block length is used by the "fixed" and "geom" block bootstrap routes.

center
: centering convention for bootstrap intervals. Copula plots support "estimate" and "bias-corrected". Copula plots do not currently have a validated oversmoothed pilot correction; unsupported bootstrap bias-correction combinations fail clearly.

`errors` interval route for `plot.npcopula`: `"none"`, `"asymptotic"`, or `"bootstrap"`. Intervals are available for two-dimensional grid evaluation output and are drawn as transparent wireframes over the copula surface.

Additional Arguments: Further arguments are passed to the bandwidth-selection counterpart, prediction/evaluation route, or graphics renderer as appropriate.

`...` additional arguments supplied to `npudistbw` or `npudensbw` when `npcopula` computes bandwidths internally, or arguments needed to interpret a numeric `bws` vector. This is where bandwidth-selection controls such as `bwmethod`, `bwtype`, and `bwscaling`, kernel/support controls such as `ckertype`, `ckerorder`, and `ckerbound`, categorical kernel controls such as `ukertype` and `okertype`, and search controls such as `nmulti` and `scale.factor.search.lower` are supplied. In `predict.npcopula`, additional arguments are passed to `npcopula` for evaluation with the stored bandwidth object and training data. In `plot.npcopula`, additional arguments are passed to the selected graphics routine, such as `persp`, `contour`, `image`, or the shared **rgl** renderer.

Details

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, and `plot` for plotting options.

`npcopula` computes the nonparametric copula distribution or copula density using marginal quasi-inversion. For the distribution target, Sklar's theorem gives

$$C(u_1, \ldots, u_d) = H(F_1^{-1}(u_1), \ldots, F_d^{-1}(u_d)),$$

where H is the joint distribution and F_j^{-1} is the quasi-inverse of marginal distribution F_j. For the density target, the estimated copula density is

$$c(u) = \frac{f(x_u)}{\prod_{j=1}^{d} f_j(x_{u,j})}, \quad x_{u,j} = F_j^{-1}(u_j),$$

with numerator and marginal denominators estimated using the selected mixed-data kernel bandwidths.

If `u` is provided, `expand.grid` is called on `u`. As the dimension increases this can become unwieldy because a grid with m points in each of d margins has m^d rows. Therefore the formula route automatically generates a probability grid only for two-dimensional copulas. For higher-dimensional copulas, supply `u` explicitly or use `evaluation="sample"`. The `predict.npcopula` method follows these same probability-coordinate semantics: use `u` for probability values named however convenient, or use `newdata` only with `u1`, `u2`, ... columns.

The 'quasi-inverse' is computed via Definition 2.3.6 from Nelsen (2006). An equi-quantile grid on the data range of length `n.quasi.inv/2` is combined with an equi-spaced grid on the data range extended by `er.quasi.inv`; the sorted union forms the grid used for marginal inversion. If requested probability values lie outside the attainable estimated marginal distribution range, they are reset to the nearest attainable endpoint. Inspect the returned `u` columns when endpoint behavior matters. Higher-order continuous CDF kernels can yield nonmonotone estimated marginal CDFs on the finite inversion grid; the quasi-inverse follows the first grid point whose estimated CDF reaches the requested probability.

The `plot.npcopula` method supports base `persp`, `contour`, and `image` displays for two-dimensional grid output. Surface plots use the package-wide viridis default palette, detailed perspective ticks, and the same default viewing angles and base-graphics rotation cadence as the other surface plot methods. `renderer="rgl"` requests the shared interactive **rgl** surface renderer, using the same default-angle remapping used by the other package surface plots. For mixed ordered margins and endpoint-clamped continuous margins, grid displays are drawn against the requested probability grid, while the returned `u` columns retain the attainable marginal probability values produced by quasi-inversion. The `"empirical"` view plots empirical copula coordinates, and `"all"` gives a base-graphics four-panel diagnostic display.

For grid surfaces, `plot.npcopula` can add asymptotic or bootstrap intervals. Distribution-copula asymptotic intervals use the joint distribution standard error evaluated at the marginal quasi-inverse grid. Density-copula asymptotic intervals use the plug-in delta-method denominator correction corresponding to $c(u) = f(x_u)/\prod_j f_j(x_{u,j})$. Bootstrap intervals resample rows and recompute the plotted copula surface on the same probability grid. These interval conventions do not fully propagate uncertainty from the marginal quasi-inverse transform; `band="all"` overlays transparent pointwise, simultaneous, and Bonferroni wireframes.

Value

`npcopula` returns an object of class `"npcopula"`. The main components are:

`copula`	estimated copula distribution value or copula density value.
`u1`, `u2`, ...	marginal probability coordinates associated with the sample realizations or evaluation grid.
`x`, `y`, ...	marginal quasi-inverse coordinates corresponding to the requested probability grid when grid evaluation is used.
`bws`	selected unconditional distribution or density bandwidth object.
`eval`	data frame containing the copula values, probability coordinates, and quasi-inverse coordinates. `as.data.frame(object)` returns this component for data-frame workflows.
`copulaerr`	asymptotic or bootstrap standard-error slot. The fitted object stores `NA` unless an interval-producing plotting route constructs evaluation-specific intervals.

The source data, target, evaluation route, grid dimensions, and timing metadata are retained as list components. The functions `fitted`, `predict`, `se`, `summary`, `as.data.frame`, and `plot` support `"npcopula"` objects.

Book And Method Pointers

The copula distribution target is $C(u) = H(F_1^{-1}(u_1), \ldots, F_d^{-1}(u_d))$; the copula density target is $c(u) = f(x_u)/\prod_j f_j(x_{u,j})$. The mixed-data kernel implementation follows Racine (2015), with quasi-inversion in the sense of Nelsen (2006). For the underlying mixed-data density and distribution estimators, see Li and Racine (2007), Chapter 1 *Density Estimation*, Chapter 3 *Kernel Estimation with Mixed Data*, and Racine (2019), Chapter 2 *Continuous Density and Cumulative Distribution Functions*.

Usage Issues

Use a `data.frame` rather than `cbind` for mixed data so that ordered variables remain ordered. Unordered factors are not valid for copula estimation in this implementation.

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

References

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

Nelsen, R.B. (2006), *An Introduction to Copulas*, Second Edition, Springer.

Racine, J.S. (2015), "Mixed Data Kernel Copulas", *Empirical Economics*, 48, 37–59.

Racine, J.S. (2019), *An Introduction to the Advanced Theory and Practice of Nonparametric Econometrics: A Replicable Approach Using R*.

See Also

`npudistbw`, `npudist`, `npudensbw`, `npudens`, `np.kernels`, `np.options`, `plot`

Examples

```
## Not run:
library("MASS")

npRmpi.init(nslaves = 1)

## Example 1: bivariate mixed data, continuous x and ordered y.

set.seed(42)
n <- 1000
n.eval <- 30
rho <- 0.99
mu <- c(0, 0)
Sigma <- matrix(c(1, rho, rho, 1), 2, 2)
xy <- mvrnorm(n = n, mu = mu, Sigma = Sigma)
mydat <- data.frame(
  x = xy[, 1],
  y = ordered(as.integer(cut(xy[, 2],
    quantile(xy[, 2], seq(0, 1, by = .1)),
    include.lowest = TRUE)) - 1)
)

grid.seq <- seq(0, 1, length.out = n.eval)
grid.dat <- data.frame(u1 = grid.seq, u2 = grid.seq)

## Estimate the copula distribution from an npudistbw() object.

bw.cdf <- npudistbw(~ x + y, data = mydat, nmulti = 1)
```

```
copula <- npcopula(bws = bw.cdf, data = mydat, u = grid.dat)
summary(copula)

## Native plotting replaces the older manual contour(), persp(), and
## empirical scatterplot calls.

plot(copula, perspective = FALSE, view = "contour")
plot(copula, perspective = FALSE, view = "image")
plot(copula, view = "fixed", zlim = c(0, 1))
if (requireNamespace("rgl", quietly = TRUE))
  plot(copula, view = "fixed", renderer = "rgl", zlim = c(0, 1))
plot(copula)

## Plot empirical copula coordinates from the retained sample data.

plot(copula, view = "empirical")

## Or request the four-panel base-graphics diagnostic display.

plot(copula, view = "all")

## Estimate and plot the copula density from an npudensbw() object.

bw.pdf <- npudensbw(~ x + y, data = mydat, nmulti = 1)
copula.dens <- npcopula(bws = bw.pdf, data = mydat, u = grid.dat)
summary(copula.dens)
plot(copula.dens, view = "fixed")
if (requireNamespace("rgl", quietly = TRUE))
  plot(copula.dens, view = "fixed", renderer = "rgl")
plot(copula.dens)

## Intervals are available for two-dimensional grid surfaces.

plot(copula, errors = "asymptotic", band = "pointwise")
plot(copula, errors = "bootstrap", bootstrap = "inid", B = 399,
     band = "pointwise")

## Prediction evaluates the retained bandwidth object on a supplied
## probability grid.

predict(copula, u = data.frame(x = c(0.25, 0.75),
                               y = c(0.25, 0.75)))
predict(copula, newdata = data.frame(u1 = c(0.25, 0.75),
                                     u2 = c(0.25, 0.75)))

## Example 2: bivariate continuous data.

set.seed(42)
n <- 1000
n.eval <- 30
rho <- 0.99
mu <- c(0, 0)
Sigma <- matrix(c(1, rho, rho, 1), 2, 2)
```

```
xy <- mvrnorm(n = n, mu = mu, Sigma = Sigma)
mydat <- data.frame(x = xy[, 1], y = xy[, 2])

grid.seq <- seq(0, 1, length.out = n.eval)
grid.dat <- data.frame(u1 = grid.seq, u2 = grid.seq)

bw.cdf <- npudistbw(~ x + y, data = mydat, nmulti = 1)
copula <- npcopula(bws = bw.cdf, data = mydat, u = grid.dat)
summary(copula)

plot(copula, perspective = FALSE, view = "contour")
plot(copula, perspective = FALSE, view = "image")
plot(copula, view = "fixed", zlim = c(0, 1))
if (requireNamespace("rgl", quietly = TRUE))
  plot(copula, view = "fixed", renderer = "rgl", zlim = c(0, 1))
plot(copula)
plot(copula, view = "empirical")
plot(copula, view = "all")

bw.pdf <- npudensbw(~ x + y, data = mydat, nmulti = 1)
copula.dens <- npcopula(bws = bw.pdf, data = mydat, u = grid.dat)
summary(copula.dens)
plot(copula.dens, view = "fixed", zlim = c(0, 40))
if (requireNamespace("rgl", quietly = TRUE))
  plot(copula.dens, view = "fixed", renderer = "rgl",
       zlim = c(0, 40))
plot(copula.dens, zlim = c(0, 40))

## The formula interface is a shorter route when bandwidths do not need
## to be reused explicitly.

copula.short <- npcopula(~ x + y, data = mydat, neval = n.eval,
                         nmulti = 1)
plot(copula.short, view = "all")

npRmpi.quit()

## End(Not run)
```

---

npdeneqtest *Kernel Consistent Density Equality Test with Mixed Data Types*

Description

npdeneqtest implements a consistent integrated squared difference test for equality of densities as described in Li, Maasoumi, and Racine (2009).

Usage

```
npdeneqtest(x = NULL,
```

```
y = NULL,
bw.x = NULL,
bw.y = NULL,
boot.num = 399,
random.seed = 42,
...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the two samples and any supplied bandwidths.

`bw.x`, `bw.y`	optional bandwidth objects for `x`,`y`
`x`, `y`	data frames for the two samples for which one wishes to test equality of densities. The variables in each data frame must be the same (i.e. have identical names).

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

`boot.num`	an integer value specifying the number of bootstrap replications to use. Defaults to `399`.
`random.seed`	an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.

Additional Arguments: Further arguments are passed to the bandwidth-selection routines used by the test.

`...`	additional arguments supplied to specify the bandwidth type, kernel types, and so on. This is used if you do not pass in bandwidth objects and you do not desire the default behaviours. To do this, you may specify any of `bwscaling`, `bwtype`, `ckertype`, `ckerorder`, `ukertype`, `okertype`.

Details

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, `plot` for plotting options, and `npRmpi.init` for interactive/cluster MPI startup. See `npRmpi.init` details for performance tradeoffs (message passing/startup mode) and the `inst/Rprofile` manual-broadcast template.

`npdeneqtest` computes the integrated squared density difference between the estimated densities/probabilities of two samples having identical variables/datatypes. See Li, Maasoumi, and Racine (2009) for details.

Value

`npdeneqtest` returns an object of type `deneqtest` with the following components

`Tn`	the (standardized) statistic `Tn`
`In`	the (unstandardized) statistic `In`
`Tn.bootstrap`	contains the bootstrap replications of `Tn`

In.bootstrap	contains the bootstrap replications of In
Tn.P	the P-value of the Tn statistic
In.P	the P-value of the In statistic
boot.num	number of bootstrap replications

summary supports object of type deneqtest.

Book And Method Pointers

npdeneqtest compares two distributions by estimating the integrated squared difference between their density or probability functions. The two samples must have matching variables and data types because the statistic compares like with like.

For book-length background, see Li and Racine (2007), Chapter 12 *Model Specification Tests*, especially Section 12.2, and Chapter 13 *Nonsmoothing Tests*, especially Section 13.2. For density and mixed-data density context, see Racine (2019), Chapters 2 and 3.

Usage Issues

If you are using data of mixed types, then it is advisable to use the data.frame function to construct your input data and not cbind, since cbind will typically not work as intended on mixed data types and will coerce the data to the same type.

It is crucial that both data frames have the same variable names.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Li, Q. and E. Maasoumi and J.S. Racine (2009), "A Nonparametric Test for Equality of Distributions with Mixed Categorical and Continuous Data," Journal of Econometrics, 148, pp 186-200.

See Also

np.kernels, np.options, plot npdeptest,npsdeptest,npsymtest,npunitest

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
```

```
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)
set.seed(42)

n <- 100

sample.A <- data.frame(x=rnorm(n))
sample.B <- data.frame(x=rnorm(n))

output <- npdeneqtest(sample.A,sample.B,boot.num=29)

output


## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

npdeptest *Kernel Consistent Pairwise Nonlinear Dependence Test for Univariate Processes*

Description

npdeptest implements the consistent metric entropy test of pairwise independence as described in Maasoumi and Racine (2002).

Usage

```
npdeptest(data.x = NULL,
          data.y = NULL,
          method = c("integration","summation"),
          bootstrap = TRUE,
          boot.num = 399,
          random.seed = 42)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the paired samples being tested.

data.x, data.y	two univariate vectors containing two variables that are of type numeric.
method	a character string used to specify whether to compute the integral version or the summation version of the statistic. Can be set as integration or summation (see below for details). Defaults to integration.

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

boot.num	an integer value specifying the number of bootstrap replications to use. Defaults to 399.
bootstrap	a logical value which specifies whether to conduct the bootstrap test or not. If set to FALSE, only the statistic will be computed. Defaults to TRUE.
random.seed	an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.

Details

Documentation guide: see np.kernels for kernels, np.options for global options, plot for plotting options, and npRmpi.init for interactive/cluster MPI startup. See npRmpi.init details for performance tradeoffs (message passing/startup mode) and the inst/Rprofile manual-broadcast template.

npsdeptest computes the nonparametric metric entropy (normalized Hellinger of Granger, Maasoumi and Racine (2004)) for testing pairwise nonlinear dependence between the densities of two data series. See Maasoumi and Racine (2002) for details. Default bandwidths are of the Kullback-Leibler variety obtained via likelihood cross-validation. The null distribution is obtained via bootstrap resampling under the null of pairwise independence.

npdeptest computes the distance between the joint distribution and the product of marginals (i.e. the joint distribution under the null), $D[f(y,\hat{y}), f(y) \times f(\hat{y})]$. Examples include, (a) a measure/test of "fit", for in-sample values of a variable y and its fitted values, $\hat{y}$, and (b) a measure of "predictability" for a variable y and its predicted values $\hat{y}$ (from a user implemented model).

The summation version of this statistic will be numerically unstable when `data.x` and `data.y` lack common support or are sparse (the summation version involves division of densities while the integration version involves differences). Warning messages are produced should this occur ('integration recommended') and should be heeded.

Value

`npdeptest` returns an object of type `deptest` with the following components

`Srho`	the statistic `Srho`
`Srho.bootstrap.vec`	contains the bootstrap replications of `Srho`
`P`	the P-value of the Srho statistic
`bootstrap`	a logical value indicating whether bootstrapping was performed
`boot.num`	number of bootstrap replications
`bw.data.x`	the numeric bandwidth for `data.x` marginal density
`bw.data.y`	the numeric bandwidth for `data.y` marginal density
`bw.joint`	the numeric matrix of bandwidths for `data` and lagged `data` joint density at lag `num.lag`

`summary` supports object of type `deptest`.

Book And Method Pointers

`npdeptest` compares the joint density or probability law of two variables with the product of their marginals. Departures from the product form indicate dependence, lack of fit, or predictability depending on how the two inputs are constructed.

For book-length background, see Li and Racine (2007), Chapter 13 *Nonsmoothing Tests*, especially Sections 13.4 and 13.5, and Chapter 12 *Model Specification Tests*, especially Sections 12.4.1 and 12.4.3. For entropy and density context, see Racine (2019), Chapter 2.

Usage Issues

The `integration` version of the statistic uses multidimensional numerical methods from the **cubature** package. See **adaptIntegrate** for details. The `integration` version of the statistic will be substantially slower than the `summation` version, however, it will likely be both more accurate and powerful.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Granger, C.W. and E. Maasoumi and J.S. Racine (2004), "A dependence metric for possibly nonlinear processes", Journal of Time Series Analysis, 25, 649-669.

Maasoumi, E. and J.S. Racine (2002), "Entropy and Predictability of Stock Market Returns," Journal of Econometrics, 107, 2, pp 291-312.

See Also

np.kernels, np.options, plot npdeneqtest,npsdeptest,npsymtest,npunitest

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)
set.seed(42)

n <- 100

x <- rnorm(n)
y <- 1 + x + rnorm(n)
model <- lm(y~x)
y.fit <- fitted(model)

output <- npdeptest(y,
                    y.fit,
                    boot.num=29,
                    method="summation")

summary(output)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
```

```
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

npindex *Semiparametric Single Index Model*

Description

npindex computes a semiparametric single index model for a dependent variable and p-variate explanatory data using the model $Y = G(X\beta) + \epsilon$, given a set of evaluation points, training points (consisting of explanatory data and dependent data), and a npindexbw bandwidth specification. Note that for this semiparametric estimator, the bandwidth object contains parameters for the single index model and the (scalar) bandwidth for the index function.

Usage

```
npindex(bws, ...)

## S3 method for class 'formula'
npindex(bws,
        data = NULL,
        newdata = NULL,
        y.eval = FALSE,
        ...)


## Default S3 method:
npindex(bws,
        txdat,
        tydat,
        nomad = FALSE,
        ...)

## S3 method for class 'sibandwidth'
npindex(bws,
```

```
        txdat = stop("training data 'txdat' missing"),
        tydat = stop("training data 'tydat' missing"),
        exdat,
        eydat,
        boot.num = 399,
        errors = FALSE,
        gradients = FALSE,
        residuals = FALSE,
        ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and training data.

`bws` a bandwidth specification. This can be set as a `sibandwidth` object returned from an invocation of `npindexbw`, or as a vector of parameters (beta) with each element i corresponding to the coefficient for column i in `txdat` where the first element is normalized to 1, and a scalar bandwidth (h).

`data` an optional data frame, list or environment (or object coercible to a data frame by `as.data.frame`) containing the variables in the model. If not found in data, the variables are taken from `environment(bws)`, typically the environment from which `npindexbw` was called.

`txdat` a p-variate data frame of explanatory data (training data) used to calculate the regression estimators. Defaults to the corresponding training data recoverable from the bandwidth object in the current session.

`tydat` a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of `txdat`. Defaults to the corresponding training data recoverable from the bandwidth object in the current session.

Local-Polynomial Degree And Bandwidth Search: This argument controls the recommended automatic local-polynomial NOMAD route, which jointly selects continuous polynomial degree and bandwidths when these are computed inside `npindex`.

`nomad` logical shortcut passed through to `npindexbw` when bandwidths are computed inside `npindex`. When `TRUE`, the single-index bandwidth route fills any missing values among `regtype`, `search.engine`, `degree.select`, `bernstein.basis`, `degree.min`, `degree.max`, `degree.verify`, and `bwtype` with the recommended automatic local-polynomial degree-and-bandwidth NOMAD preset documented in `npindexbw`. Additional NOMAD tuning arguments such as `nomad.nmulti` may also be supplied through `...`; `nmulti` remains the outer restart count while `nomad.nmulti` controls inner native **crs** NOMAD API multistarts within each outer restart. After fitting, inspect `fit$bws$nomad.shortcut` on the returned object `fit` to see the normalized shortcut metadata.

Evaluation Data And Returned Quantities: These arguments control where the single-index fit is evaluated and which evaluation quantities are returned.

exdat — a p-variate data frame of points on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by `txdat`.

eydat — a one (1) dimensional numeric or integer vector of the true values of the dependent variable. Optional, and used only to calculate the true errors.

newdata — An optional data frame in which to look for evaluation data. If omitted, the training data are used.

y.eval — If `newdata` contains dependent data and `y.eval = TRUE`, `npRmpi` will compute goodness of fit statistics on these data and return them. Defaults to `FALSE`.

Fitted Quantities And Inference: These arguments control residuals, gradients, and bootstrap standard errors.

boot.num — an integer specifying the number of bootstrap replications to use when performing standard error calculations. Defaults to `399`.

errors — a logical value indicating that you want (bootstrapped) standard errors for the conditional mean, gradients (when `gradients=TRUE` is set), and average gradients (when `gradients=TRUE` is set), computed and returned in the resulting `singleindex` object. Defaults to `FALSE`.

gradients — a logical value indicating that you want gradients and the asymptotic covariance matrix for beta computed and returned in the resulting `singleindex` object. Defaults to `FALSE`.

residuals — a logical value indicating that you want residuals computed and returned in the resulting `singleindex` object. Defaults to `FALSE`.

Additional Arguments: Further arguments are passed to the bandwidth-selection counterpart when bandwidths are not supplied.

... — additional arguments supplied to specify the parameters to the `sibandwidth` S3 method, which is called during estimation.

Details

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, `plot`, `plot.np` for plotting options, and `npRmpi.init` for interactive/cluster MPI startup. See `npRmpi.init` details for performance tradeoffs (message passing/startup mode) and the `inst/Rprofile` manual-broadcast template.

For S3 plotting help, see `plot.np`. You can list available plot methods with `methods("plot")`.

A matrix of gradients along with average derivatives are computed and returned if `gradients=TRUE` is used.

Interpreting β Coefficients: Single-index coefficients are identified only up to scale, so `npindexbw` normalizes the first coefficient to one. The remaining coefficients describe relative movement in the index $X'\beta$, not standalone linear-model slopes. Holding other variables fixed, a one-unit increase in X_j changes the index by the same amount as a β_j-unit increase in X_1. Where the link derivative exists and is nonzero, this is also the local marginal-effect ratio relative to X_1. Choose the first continuous regressor deliberately when coefficient interpretation is important.

For $E[Y \mid X] = g(X'\beta)$, the component plotted for predictor X_j is

$$\frac{\partial E[Y \mid X]}{\partial X_j} = \beta_j g'(X'\beta)$$

Accordingly, `plot(..., gradients=TRUE)` displays every component against the common scalar index $X'\beta$; the y-axis identifies the predictor with respect to which the derivative is taken. These panels are not partial-dependence plots against the individual X_j values.

Optimizer Choice When Bandwidths Are Computed: When `npindex` computes its bandwidth object internally, optimizer controls in `...` are passed to `npindexbw`. The default `optim.method="Nelder-Mead"` is a robust derivative-free starting point for low-dimensional searches, such as one bandwidth plus only a few index coefficients. For larger p, `optim.method="BFGS"` can be useful as a performance and sensitivity comparison, but users should compare objective values, convergence behavior, and fitted objects because the single-index criterion can be nonconvex and route-sensitive.

For practitioners who want the recommended automatic local-polynomial degree-and-bandwidth NOMAD route without spelling out all LP tuning arguments, `npindex(..., nomad=TRUE)` and `npindexbw(..., nomad=TRUE)` expand missing settings to the same documented preset. Explicit incompatible settings fail fast rather than being silently rewritten.

Value

`npindex` returns a `npsingleindex` object. The generic functions `fitted`, `residuals`, `coef`, `vcov`, `se`, `predict`, and `gradients`, extract (or generate) estimated values, residuals, coefficients, variance-covariance matrix, bootstrapped standard errors on estimates, predictions, and gradients, respectively, from the returned object. Furthermore, the functions `summary` and `plot` support objects of this type. The returned object has the following components:

`eval`	evaluation points
`mean`	estimates of the regression function (conditional mean) at the evaluation points
`beta`	the model coefficients
`betavcov`	the asymptotic covariance matrix for the model coefficients
`merr`	standard errors of the regression function estimates
`grad`	estimates of the gradients at each evaluation point
`gerr`	standard errors of the gradient estimates
`mean.grad`	mean (average) gradient over the evaluation points
`mean.gerr`	bootstrapped standard error of the mean gradient estimates
`R2`	if `method="ichimura"`, coefficient of determination (Doksum and Samarov (1995))
`MSE`	if `method="ichimura"`, mean squared error
`MAE`	if `method="ichimura"`, mean absolute error
`MAPE`	if `method="ichimura"`, mean absolute percentage error
`CORR`	if `method="ichimura"`, absolute value of Pearson's correlation coefficient
`SIGN`	if `method="ichimura"`, fraction of observations where fitted and observed values agree in sign

`confusion.matrix`
: if `method="kleinspady"`, the confusion matrix or `NA` if outcomes are not available

`CCR.overall`
: if `method="kleinspady"`, the overall correct classification ratio, or `NA` if outcomes are not available

`CCR.byoutcome`
: if `method="kleinspady"`, a numeric vector containing the correct classification ratio by outcome, or `NA` if outcomes are not available

`fit.mcfadden`
: if `method="kleinspady"`, the McFadden-Puig-Kerschner performance measure or `NA` if outcomes are not available

Book And Method Pointers

The single-index model reduces a multivariate predictor to an index, typically written $E[Y \mid X] = g(X'\beta)$ for continuous outcomes, with a normalization on β for identification. For binary outcomes, the Klein-Spady route estimates the corresponding binary-choice probability through the index. The fitted index regression, gradients, and average derivatives are computed from the selected index direction and kernel regression fit.

For book-length derivations, see Li and Racine (2007), Chapter 8 *Semiparametric Single Index Models*, especially Sections 8.1, 8.2.1, 8.4, 8.5, and 8.10. The later workflow treatment is Racine (2019), Chapter 8 *Semiparametric Conditional Mean Function Estimation*, especially the single-index material.

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

When reusing a bandwidth object without explicit training or evaluation data, the required data are recovered from object-retained state or from the stored call environment when available. This is a convenience for ordinary same-session use, not a guarantee that a saved object is self-contained. For reproducible replay after `saveRDS()` and `readRDS()` in a fresh R session, supply the required data arguments explicitly.

`vcov` requires that `gradients=TRUE` be set.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

Doksum, K. and A. Samarov (1995), "Nonparametric estimation of global functionals and a measure of the explanatory power of covariates regression," The Annals of Statistics, 23 1443-1473.

Ichimura, H., (1993), "Semiparametric least squares (SLS) and weighted SLS estimation of single-index models," Journal of Econometrics, 58, 71-120.

Klein, R. W. and R. H. Spady (1993), "An efficient semiparametric estimator for binary response models," Econometrica, 61, 387-421.

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

McFadden, D. and C. Puig and D. Kerschner (1977), "Determinants of the long-run demand for electricity," Proceedings of the American Statistical Association (Business and Economics Section), 109-117.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

npRmpi.init for MPI startup and workflow guidance. np.kernels, np.options, plot, plot.np, npindexbw

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)
set.seed(42)

n <- 500

x1 <- runif(n, min=-1, max=1)
x2 <- runif(n, min=-1, max=1)
y <- x1 - x2 + rnorm(n)
```

```
## Ichimura, continuous y

bw <- npindexbw(formula=y~x1+x2)

summary(bw)

model <- npindex(bws=bw,
                 gradients=TRUE)

summary(model)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

npindexbw *Semiparametric Single Index Model Parameter and Bandwidth Selection*

Description

npindexbw computes a npindexbw bandwidth specification using the model $Y = G(X\beta) + \epsilon$. For continuous Y, the approach is that of Hardle, Hall and Ichimura (1993) which jointly minimizes a least-squares cross-validation function with respect to the parameters and bandwidth. For binary Y, a likelihood-based cross-validation approach is employed which jointly maximizes a likelihood cross-validation function with respect to the parameters and bandwidth. The bandwidth object contains parameters for the single index model and the (scalar) bandwidth for the index function.

Usage

```
npindexbw(...)

## S3 method for class 'formula'
```

```
npindexbw(formula,
          data,
          subset,
          na.action,
          call,
          ...)

## Default S3 method:
npindexbw(xdat = stop("training data xdat missing"),
          ydat = stop("training data ydat missing"),
          bws,
          bandwidth.compute = TRUE,
          basis = c("glp", "additive", "tensor"),
          bernstein.basis = FALSE,
          degree = NULL,
          degree.select = c("manual", "coordinate", "exhaustive"),
          search.engine = c("nomad+powell", "cell", "nomad"),
          nomad = FALSE,
          nomad.nmulti = 0L,
            degree.min = NULL,
          degree.max = NULL,
          degree.start = NULL,
          degree.restarts = 0L,
          degree.max.cycles = 20L,
          degree.verify = FALSE,
          nmulti,
          nomad.remin = FALSE,
          powell.remin = TRUE,
          only.optimize.beta,
          optim.abstol,
          optim.maxattempts,
          optim.maxit,
          optim.method,
          optim.reltol,
          random.seed,
          regtype = c("lc", "ll", "lp"),
          scale.factor.init.lower = 0.1,
          scale.factor.init.upper = 2.0,
          scale.factor.init = 0.5,
          scale.factor.search.lower = NULL,
          ...,
          nomad.opts = list())

## S3 method for class 'sibandwidth'
npindexbw(xdat = stop("training data xdat missing"),
          ydat = stop("training data ydat missing"),
          bws,
          bandwidth.compute = TRUE,
```

```
nmulti,
only.optimize.beta = FALSE,
optim.abstol = .Machine$double.eps,
optim.maxattempts = 10,
optim.maxit = 500,
optim.method = c("Nelder-Mead", "BFGS", "CG"),
optim.reltol = sqrt(.Machine$double.eps),
random.seed = 42,
scale.factor.init.lower = 0.1,
scale.factor.init.upper = 2.0,
scale.factor.init = 0.5,
scale.factor.search.lower = NULL,
...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the data, formula interface, method label, and whether bandwidths are supplied or computed.

`bandwidth.compute`
: a logical value which specifies whether to do a numerical search for bandwidths or not. If set to `FALSE`, a `bandwidth` object will be returned with bandwidths set to those specified in `bws`. Defaults to `TRUE`.

`bws`
: a bandwidth specification. This can be set as a `singleindexbandwidth` object returned from an invocation of `npindexbw`, or as a vector of parameters (beta) with each element i corresponding to the coefficient for column i in `xdat` where the first element is normalized to 1, and a scalar bandwidth (h). If specified as a vector, then additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, and so on.

`call`
: the original function call. This is passed internally by `npRmpi` when a bandwidth search has been implied by a call to another function. It is not recommended that the user set this.

`data`
: an optional data frame, list or environment (or object coercible to a data frame by `as.data.frame`) containing the variables in the model. If not found in data, the variables are taken from `environment(formula)`, typically the environment from which the function is called.

`formula`
: a symbolic description of variables on which bandwidth selection is to be performed. The details of constructing a formula are described below.

`method`
: the single index model method, one of either "ichimura" (Ichimura (1993)) or "kleinspady" (Klein and Spady (1993)). Defaults to `ichimura`.

`na.action`
: a function which indicates what should happen when the data contain `NA`s. The default is set by the `na.action` setting of options, and is `na.fail` if that is unset. The (recommended) default is `na.omit`.

`subset`
: an optional vector specifying a subset of observations to be used in the fitting process.

`xdat` a p-variate data frame of explanatory data (training data) used to calculate the regression estimators.

`ydat` a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of `xdat`.

Automatic Degree Search Controls: These arguments control automatic local-polynomial degree search.

`degree.max` optional scalar or integer vector giving upper bounds for automatic degree search when `degree.select != "manual"`.

`degree.max.cycles` positive integer giving the maximum number of coordinate-search sweeps over the degree vector. Ignored for `"manual"` and `"exhaustive"` degree selection.

`degree.min` optional scalar or integer vector giving lower bounds for automatic degree search when `degree.select != "manual"`.

`degree.restarts` non-negative integer giving the number of additional deterministic coordinate-search restarts. Ignored for `"manual"` and `"exhaustive"` degree selection.

`degree.select` character string controlling local-polynomial degree handling when `regtype="lp"`. `"manual"` (default) treats `degree` as fixed. `"coordinate"` performs cached coordinate-wise search over admissible degree values for the index smoother. `"exhaustive"` evaluates the full admissible degree grid when `search.engine="cell"`. For NOMAD-based search engines, any non-`"manual"` value requests direct joint search over the degree and bandwidth coordinates.

`degree.start` optional starting degree vector for automatic coordinate search. If omitted, the search starts from the degree-zero local-constant baseline for the index smoother.

`degree.verify` logical value indicating whether an automatic degree-search solution should be exhaustively certified over the admissible degree grid after the heuristic phase completes. With `search.engine="cell"`, this verifies a coordinate-search solution using repeated fixed-degree solves. With NOMAD-based search engines, this first runs the joint NOMAD search over bandwidth and degree coordinates, then verifies the result over the requested degree grid using repeated fixed-degree solves. The default is `FALSE` because certification can be substantially slower than the heuristic search. Set `degree.verify=TRUE` when degree-selection certification or sensitivity checking is more important than runtime.

Continuous Scale-Factor Search Initialization: These controls define deterministic and random continuous scale-factor starts and the lower admissibility floor for fixed-bandwidth search.

`scale.factor.init` deterministic initial scale factor for continuous fixed-bandwidth search. Defaults to `0.5`. The value supplied by the user is not rewritten, but the effective first start passed to the optimizer is `max(scale.factor.init, scale.factor.search.lower)`. See Details.

`scale.factor.init.lower` lower endpoint for random continuous scale-factor starts. Defaults to `0.1`. The value supplied by the user is not rewritten, but the effective random-start lower endpoint is `max(scale.factor.init.lower, scale.factor.search.lower)`. See Details.

`scale.factor.init.upper`
: upper endpoint for random continuous scale-factor starts. Defaults to `2.0`. It must be greater than or equal to the effective lower endpoint, `max(scale.factor.init.lower, scale.factor.search.lower)`; otherwise bandwidth search errors rather than silently expanding the interval. See Details.

`scale.factor.search.lower`
: optional nonnegative scalar giving the hard lower admissibility bound for continuous fixed-bandwidth search candidates. Defaults to `NULL`. If `NULL`, an existing bandwidth object's stored value is inherited when available; otherwise the package default `0.1` is used. This floor applies to computed/search bandwidth candidates and to effective search starts only. It does not rewrite explicit bandwidths supplied for storage with `bandwidth.compute = FALSE`. Final fixed-bandwidth search candidates must also have a finite valid raw objective value.

Local-Polynomial Model Specification: These arguments control the index smoother, local-polynomial basis, and fixed degree specification.

`basis`
: local polynomial basis selector used when `regtype="lp"`: one of `"glp"`, `"additive"`, or `"tensor"`. Ignored for `"lc"` and `"ll"`.

`bernstein.basis`
: logical flag used when `regtype="lp"`; if `TRUE`, use a Bernstein/B-spline basis for local polynomial terms. When automatic degree search is requested and `bernstein.basis` is not explicitly supplied, the search route defaults to `TRUE` for numerical stability. Explicit `bernstein.basis=FALSE` is honored, but raw-polynomial search can be poorly conditioned at higher degrees.

`degree`
: integer degree vector for continuous predictors when `regtype="lp"`. When `degree.select="manual"`, this must be supplied explicitly. For single-index regression this is typically length one.

`regtype`
: a character string specifying local smoothing type for the nonparametric index regression fit used downstream in `npindex`. Supported values are `"lc"`, `"ll"`, and `"lp"`.

NOMAD Search Controls: These arguments control the optional NOMAD direct-search route for local-polynomial degree and bandwidth search.

`nomad`
: logical or character shortcut for the recommended automatic local-polynomial NOMAD route. When `TRUE`, any missing values among `regtype`, `search.engine`, `degree.select`, `bernstein.basis`, `degree.min`, `degree.max`, `degree.verify`, and `bwtype` are filled with `regtype="lp"`, `search.engine="nomad+powell"`, `degree.select="coordinate"`, `bernstein.basis=TRUE`, `degree.min=0L`, `degree.max=10L`, `degree.verify=FALSE`, and `bwtype="fixed"`. Explicit incompatible settings error immediately; in particular, explicit `nomad=TRUE` currently requires `regtype="lp"`, `bwtype="fixed"`, automatic degree search, no explicit `degree`, and `search.engine %in% c("nomad", "nomad+powell")`. This shortcut does not change the meaning of `nmulti` or `nomad.nmulti`: `nmulti` remains the outer restart count, while `nomad.nmulti` controls inner native **crs** NOMAD API multistarts within each outer restart. Returned bandwidth objects retain this normalized preset metadata in `bw$nomad.shortcut` for a returned object `bw`; when available, `nomad.time`

records the direct-search component. The legacy field `powell.time` records the subsequent family-native R `optim` refinement selected by `optim.method`.

`nomad.nmulti` non-negative integer controlling the inner the native **crs** NOMAD C API multistart count used within each outer NOMAD restart when `regtype="lp"` and automatic degree search uses `search.engine="nomad"` or `"nomad+powell"`. Defaults to `0L`, which preserves the current one-start-per- restart behavior. This does not replace `nmulti`: `nmulti` controls outer restarts, while `nomad.nmulti` controls inner NOMAD multistarts within each outer restart.

`nomad.opts` optional named list of native NOMAD parameters forwarded to the **crs** NOMAD C API for NOMAD-backed bandwidth searches. `NULL` is treated as no extra NOMAD options. Non-list values and invalid or unsupported parameters are rejected before solver entry rather than silently ignored. Routes that evaluate R callbacks require serial callback evaluation and reject parallel callback settings such as `NB_THREADS_PARALLEL_EVAL > 1`.

`nomad.remin` logical flag controlling the optional second NOMAD hot start. When `TRUE`, NOMAD is restarted once from the best full candidate found, including both bandwidth and degree coordinates. Defaults to `FALSE`; current simulation evidence favors the one-pass NOMAD default for routine use, while leaving this switch available for sensitivity checks.

`search.engine` character string controlling the automatic local-polynomial search backend when `regtype="lp"` and `degree.select != "manual"`. The compatibility value `"nomad+powell"` (default) performs direct joint search over the index-smoother bandwidth and degree using the native **crs** NOMAD C API, then applies one family-native R `optim` hot start from the NOMAD solution using `optim.method`. The value is retained for API compatibility; `"nomad"` omits this R-optimizer refinement. `"cell"` profiles the criterion over the admissible degree grid using repeated fixed-degree solves. Direct exhaustive degree-grid search is available with `degree.select="exhaustive"` and `search.engine="cell"`. NOMAD-based search is intended to avoid exhaustive degree enumeration; users who want post-search certification over the requested degree grid can set `degree.verify=TRUE`. NOMAD-based search requires the suggested package **crs** to be installed.

Numerical Search Controls: These arguments control outer restart behavior for bandwidth and index-parameter search.

`nmulti` integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points. Defaults to `min(2,ncol(xdat))`.

`powell.remin` compatibility argument shared with bandwidth families that use the Powell direction-set solver. Single-index fixed-degree and post-NOMAD refinement use R `optim` with `optim.method`; consequently, this argument does not alter their refinement behavior.

Optimization Controls: These arguments control outer optimization behavior for the semiparametric search.

`only.optimize.beta`
signals the routine to only minimize the objective function with respect to beta

`optim.abstol` the absolute convergence tolerance used by `optim`. Only useful for non-negative functions, as a tolerance for reaching zero. Defaults to `.Machine$double.eps`.

`optim.maxattempts` maximum number of attempts taken trying to achieve successful convergence in `optim`. Defaults to `10`.

`optim.maxit` maximum number of iterations used by `optim`. Defaults to `500`.

`optim.method` method used by `optim` for minimization of the objective function. See `?optim` for references. Defaults to `"Nelder-Mead"`.

the default method is an implementation of that of Nelder and Mead (1965), that uses only function values and is robust but relatively slow. It will work reasonably well for non-differentiable functions.

method `"BFGS"` is a quasi-Newton method (also known as a variable metric algorithm), specifically that published simultaneously in 1970 by Broyden, Fletcher, Goldfarb and Shanno. This uses function values and gradients to build up a picture of the surface to be optimized.

method `"CG"` is a conjugate gradients method based on that by Fletcher and Reeves (1964) (but with the option of Polak-Ribiere or Beale-Sorenson updates). Conjugate gradient methods will generally be more fragile than the BFGS method, but as they do not store a matrix they may be successful in much larger optimization problems.

The default `"Nelder-Mead"` method is a reasonable, robust derivative-free default when the search dimension is small, such as one scalar bandwidth plus only a few index coefficients. As the number of predictors grows, the β search becomes higher dimensional and `optim.method="BFGS"` can be a useful performance and sensitivity comparison. Because the single-index objective can be nonconvex and route-sensitive, compare the selected objective value, convergence behavior, and resulting bandwidth object against the default search before relying on the result.

`optim.reltol` relative convergence tolerance used by `optim`. The algorithm stops if it is unable to reduce the value by a factor of 'reltol * (abs(val) + reltol)' at a step. Defaults to `sqrt(.Machine$double.eps)`, typically about `1e-8`.

`random.seed` an integer used to seed R's random number generator. This ensures replicability of the numerical search. Defaults to 42.

Additional Arguments: These arguments collect remaining controls passed through S3 methods.

`...` additional arguments supplied to specify the parameters to the `sibandwidth` S3 method, which is called during the numerical search. In particular, `bwtype` may be supplied here to request `"fixed"`, `"generalized_nn"`, or `"adaptive_nn"` bandwidth types.

Details

The `scale.factor.*` controls are dimensionless search controls. The package converts scale factors to bandwidths using the estimator-specific scaling encoded in the bandwidth object, including

kernel order and the number of continuous variables relevant for the estimator. Users should not pre-multiply these controls by sample-size or standard-deviation factors.

`scale.factor.init` controls the deterministic first search start when that control is exposed. `scale.factor.init.lower` and `scale.factor.init.upper` define the random multistart interval when exposed. `scale.factor.search.lower` is the lower admissibility bound for continuous fixed-bandwidth search candidates. The effective first start is `max(scale.factor.init, scale.factor.search.lower)` when both controls are present, and the effective random-start lower endpoint is `max(scale.factor.init.lower, scale.factor.search.lower)`. `scale.factor.init.upper` must be at least that effective lower endpoint; the package errors rather than silently expanding the user's interval.

When `scale.factor.search.lower` is `NULL`, an existing bandwidth object's stored floor is inherited when available; otherwise the package default `0.1` is used. Explicit bandwidths supplied for storage with `bandwidth.compute = FALSE` are not rewritten by the search floor.

Categorical search-start controls such as `dfac.init`, `lbd.init`, and `hbd.init` have separate semantics and are not affected by `scale.factor.search.lower`.

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, `plot`, `plot.np` for plotting options, and `npRmpi.init` for interactive/cluster MPI startup. See `npRmpi.init` details for performance tradeoffs (message passing/startup mode) and the `inst/Rprofile` manual-broadcast template.

For S3 plotting help, see `plot.np`. You can list available plot methods with `methods("plot")`.

We implement Ichimura's (1993) method via joint estimation of the bandwidth and coefficient vector using leave-one-out nonlinear least squares. We implement Klein and Spady's (1993) method maximizing the leave-one-out log likelihood function jointly with respect to the bandwidth and coefficient vector. Note that Klein and Spady's (1993) method is for *binary outcomes only*, while Ichimura's (1993) method can be applied for any outcome data type (i.e., continuous or discrete).

We impose the identification condition that the first element of the coefficient vector beta is equal to one, while identification also requires that the explanatory variables contain *at least one* continuous variable.

Interpreting β Coefficients: The scale of $X'\beta$ and the scale of the unknown link function are not separately identified, so `npindexbw` normalizes the first coefficient to one. The remaining coefficients should therefore be read relative to the first explanatory variable, not as standalone linear-model slopes. Holding other variables fixed, a one-unit increase in X_j changes the index by the same amount as a β_j-unit increase in X_1. Where the link derivative exists and is nonzero, marginal-effect ratios satisfy $\partial g(X'\beta)/\partial X_j \ / \ \partial g(X'\beta)/\partial X_1 = \beta_j$. Signs and magnitudes are therefore relative to the chosen first variable and its units; choose the first continuous regressor deliberately when interpretation is important.

Choosing `optim.method`: The default `"Nelder-Mead"` search is often a good starting point for low-dimensional single-index problems because it uses only objective values and is comparatively tolerant of rough objective surfaces. When p is larger, especially when many coefficients enter β, `"BFGS"` may reduce runtime for some problems, but it is more dependent on local smoothness and on the quality of numerical gradient information. For important fits, try both methods with adequate multistarts and compare the final objective values, convergence messages, and fitted objects.

`npindexbw` may be invoked *either* with a formula-like symbolic description of variables on which bandwidth selection is to be performed *or* through a simpler interface whereby data is passed di-

rectly to the function via the xdat and ydat parameters. Use of these two interfaces is **mutually exclusive**.

Note that, unlike most other bandwidth methods in the npRmpi package, this implementation uses the R optim nonlinear minimization routines and npksum. We have implemented multistarting and strongly encourage its use in practice. For exploratory purposes, you may wish to override the default search tolerances, say, setting optim.reltol=.1 and conduct multistarting (the default is to restart min(2, ncol(xdat)) times) as is done for a number of examples. See the optimizer guidance above when choosing between the default "Nelder-Mead" method and optim.method="BFGS" for higher-dimensional index searches.

Data for which bandwidths are to be estimated may be specified symbolically. A typical description has the form dependent data ~ explanatory data, where dependent data is a univariate response, and explanatory data is a series of variables specified by name, separated by the separation character '+'. For example y1 ~ x1 + x2 specifies that the bandwidth object for the regression of response y1 and semiparametric regressors x1 and x2 are to be estimated. See below for further examples.

When regtype="lp" and degree.select != "manual", npindexbw can jointly determine the local-polynomial degree for the index smoother together with its bandwidth coordinate. With search.engine="cell", the criterion is profiled over the admissible degree grid using cached coordinate-wise or exhaustive search. With search.engine="nomad" or "nomad+powell", the criterion is optimized directly over the joint degree/bandwidth space using the native **crs** NOMAD C API. For this family, the compatibility value "nomad+powell" then performs one family-native R optim hot start using optim.method and retains the better of the direct NOMAD and refined solutions. For the index-smoother local-polynomial component, this polynomial-adaptive joint-search route follows Hall and Racine (2015).

Setting nomad=TRUE is a convenience preset for this automatic LP route, not a generic optimizer alias. For single-index bandwidth selection it expands any missing values to the equivalent long-form call

```
npindexbw(...,
          regtype = "lp",
          search.engine = "nomad+powell",
          degree.select = "coordinate",
          bernstein.basis = TRUE,
          degree.min = 0L,
          degree.max = 10L,
          degree.verify = FALSE,
          bwtype = "fixed")
```

Compatible explicit tuning arguments are respected. Incompatible explicit settings fail fast so the shortcut never silently changes user-selected semantics. The character value nomad="auto" applies the same LP shortcut but leaves search.engine and degree.select eligible for data-driven resolution when both were not supplied explicitly: scalar continuous degree searches use the exhaustive degree-grid route, while higher-dimensional degree searches keep the compatibility "nomad+powell" route. Explicit search.engine or degree.select choices are honored.

Value

npindexbw returns a sibandwidth object, with the following components:

bw	bandwidth(s), scale factor(s) or nearest neighbours for the data, xdat
beta	coefficients of the model
fval	objective function value at minimum

If bwtype is set to fixed, an object containing a scalar bandwidth for the function $G(X\beta)$ and an estimate of the parameter vector β is returned.

If bwtype is set to generalized_nn or adaptive_nn, then instead the scalar kth nearest neighbor is returned.

The functions coef, predict, summary, and plot support objects of this class.

Book And Method Pointers

npindexbw selects the index direction, bandwidth, and when requested local-polynomial degree/search metadata for the single-index model. The continuous-outcome target is typically written $E[Y \mid X] = g(X'\beta)$, with β normalized for identification; binary outcomes use the Klein-Spady route.

For book-length derivations, see Li and Racine (2007), Chapter 8 *Semiparametric Single Index Models*, especially Sections 8.1, 8.2.1, 8.4, 8.5, and 8.10, and Racine (2019), Chapter 8.

Usage Issues

If you are using data of mixed types, then it is advisable to use the data.frame function to construct your input data and not cbind, since cbind will typically not work as intended on mixed data types and will coerce the data to the same type.

Caution: multivariate data-driven bandwidth selection methods are, by their nature, *computationally intensive*. Virtually all methods require dropping the ith observation from the data set, computing an object, repeating this for all observations in the sample, then averaging each of these leave-one-out estimates for a *given* value of the bandwidth vector, and only then repeating this a large number of times in order to conduct multivariate numerical minimization/maximization. Furthermore, due to the potential for local minima/maxima, *restarting this procedure a large number of times may often be necessary*. This can be frustrating for users possessing large datasets. For exploratory purposes, you may wish to override the default search tolerances, say, setting optim.reltol=.1 and conduct multistarting (the default is to restart min(2, ncol(xdat)) times). Once the procedure terminates, you can restart search with default tolerances using those bandwidths obtained from the less rigorous search (i.e., set bws=bw on subsequent calls to this routine where bw is the initial bandwidth object). A version of this package using the Rmpi wrapper is under development that allows one to deploy this software in a clustered computing environment to facilitate computation involving large datasets.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

Hardle, W. and P. Hall and H. Ichimura (1993), "Optimal Smoothing in Single-Index Models," The Annals of Statistics, 21, 157-178.

Ichimura, H., (1993), "Semiparametric least squares (SLS) and weighted SLS estimation of single-index models," Journal of Econometrics, 58, 71-120.

Klein, R. W. and R. H. Spady (1993), "An efficient semiparametric estimator for binary response models," Econometrica, 61, 387-421.

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

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

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

npRmpi.init for MPI startup and workflow guidance.

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)
set.seed(42)
```

```
n <- 500

x1 <- runif(n, min=-1, max=1)
x2 <- runif(n, min=-1, max=1)
y <- x1 - x2 + rnorm(n)

## Ichimura, continuous y

bw <- npindexbw(formula=y~x1+x2)

summary(bw)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

npksum *Kernel Sums with Mixed Data Types*

Description

npksum computes kernel sums on evaluation data, given a set of training data, data to be weighted (optional), and a bandwidth specification (any bandwidth object).

Usage

```
npksum(...)

## S3 method for class 'formula'
npksum(formula,
       data,
       newdata,
```

```
        subset,
        na.action,
        ...)

## Default S3 method:
npksum(bws,
        txdat = stop("training data 'txdat' missing"),
        tydat = NULL,
        exdat = NULL,
        weights = NULL,
        bandwidth.divide = FALSE,
        compute.ocg = FALSE,
        compute.score = FALSE,
        kernel.pow = 1.0,
        leave.one.out = FALSE,
        operator = names(ALL_OPERATORS),
        permutation.operator = names(PERMUTATION_OPERATORS),
        return.kernel.weights = FALSE,
        ...)

## S3 method for class 'numeric'
npksum(bws,
        txdat = stop("training data 'txdat' missing"),
        tydat,
        exdat,
        weights,
        bandwidth.divide,
        compute.ocg,
        compute.score,
        kernel.pow,
        leave.one.out,
        operator,
        permutation.operator,
        return.kernel.weights,
        ...)
```

Arguments

Bandwidth And Data Inputs: Core inputs defining the training data, optional weighted response, and evaluation points for the generalized product kernel sum.

bws	a bandwidth specification. This can be set as any suitable bandwidth object returned from a bandwidth-generating function, or a numeric vector.
txdat	a p-variate data frame of sample realizations (training data) used to compute the sum.
tydat	a numeric vector of data to be weighted. The ith kernel weight is applied to the ith element. Defaults to 1.
exdat	a p-variate data frame of sum evaluation points (if omitted, defaults to the training data itself).

`weights` a n by q matrix of weights which can optionally be applied to `tydat` in the sum. See details.

Kernel-Sum Operators And Output: Controls for bandwidth normalization, kernel powers, leave-one-out evaluation, operator variants, and returned kernel weights.

`bandwidth.divide`
: a logical specifying whether or not to divide continuous kernel weights by their bandwidths. Use this with nearest-neighbor methods. Defaults to `FALSE`.

`compute.ocg` a logical specifying whether or not to return a separate result for each unordered and ordered dimension, where the product kernel term for that dimension is evaluated at an appropriate reference category. This is used primarily in `npRmpi` to compute ordered and unordered categorical gradients. See details.

`compute.score` a logical specifying whether or not to return the score (the 'grad h' terms) for each dimension in addition to the kernel sum. Cannot be `TRUE` if a permutation operator other than `"none"` is selected.

`kernel.pow` an integer specifying the power to which the kernels will be raised in the sum. Defaults to `1`.

`leave.one.out` a logical value to specify whether or not to compute the leave one out sums. Will not work if `exdat` is specified. Defaults to `FALSE`.

`operator` a string specifying whether the `normal`, `convolution`, `derivative`, or `integral` kernels are to be used. Operators scale results by factors of h or $1/h$ where appropriate. Defaults to `normal` and applies to all elements in a multivariate object. See details.

`permutation.operator`
: a string which can have a value of `none`, `normal`, `derivative`, or `integral`. If set to something other than `none` (the default), then a separate result will be returned for each term in the product kernel, with the operator applied to that term. Permutation operators scale results by factors of h or $1/h$ where appropriate. This is useful for computing gradients, for example.

`return.kernel.weights`
: a logical specifying whether or not to return the matrix of generalized product kernel weights. Defaults to `FALSE`. See details.

Formula Interface: Formula-method arguments for symbolic kernel-sum specifications.

`formula` a symbolic description of variables on which the sum is to be performed. The details of constructing a formula are described below.

`data` an optional data frame, list or environment (or object coercible to a data frame by `as.data.frame`) containing the variables in the model. If not found in data, the variables are taken from `environment(formula)`, typically the environment from which the function is called.

`newdata` An optional data frame in which to look for evaluation data. If omitted, `data` is used.

`subset` an optional vector specifying a subset of observations to be used.

`na.action` a function which indicates what should happen when the data contain `NA`s. The default is set by the `na.action` setting of options, and is `na.fail` if that is unset. The (recommended) default is `na.omit`.

Additional Arguments: Further arguments passed to the default kernel-sum method.

`...` additional arguments supplied to specify the parameters to the `default` S3 method, which is called during estimation.

Details

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, `plot` for plotting options, and `npRmpi.init` for interactive/cluster MPI startup. See `npRmpi.init` details for performance tradeoffs (message passing/startup mode) and the `inst/Rprofile` manual-broadcast template.

`npksum` exists so that you can create your own kernel objects with or without a variable to be weighted (default $Y = 1$). With the options available, you could create new nonparametric tests or even new kernel estimators. The convolution kernel option would allow you to create, say, the least squares cross-validation function for kernel density estimation.

`npksum` uses highly-optimized C code that strives to minimize its 'memory footprint', while there is low overhead involved when using repeated calls to this function (see, by way of illustration, the example below that conducts leave-one-out cross-validation for a local constant regression estimator via calls to the `R` function `nlm`, and compares this to the `npregbw` function).

`npksum` implements a variety of methods for computing multivariate kernel sums (p-variate) defined over a set of possibly continuous and/or discrete (unordered, ordered) data. The approach is based on Li and Racine (2003) who employ 'generalized product kernels' that admit a mix of continuous and discrete data types.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i, when estimating the kernel sum at the point x. Generalized nearest-neighbor bandwidths change with the point at which the sum is computed, x. Fixed bandwidths are constant over the support of x.

`npksum` computes $\sum_{j=1}^{n} W_j' Y_j K(X_j)$, where W_j represents a row vector extracted from W. That is, it computes the kernel weighted sum of the outer product of the rows of W and Y. In the examples, the uses of such sums are illustrated.

`npksum` may be invoked *either* with a formula-like symbolic description of variables on which the sum is to be performed *or* through a simpler interface whereby data is passed directly to the function via the `txdat` and `tydat` parameters. Use of these two interfaces is **mutually exclusive**.

Data contained in the data frame `txdat` (and also `exdat`) may be a mix of continuous (default), unordered discrete (to be specified in the data frame `txdat` using the `factor` command), and ordered discrete (to be specified in the data frame `txdat` using the `ordered` command). Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see `npRmpi` for details).

Data for which bandwidths are to be estimated may be specified symbolically. A typical description has the form `dependent data ~ explanatory data`, where `dependent data` and `explanatory data` are both series of variables specified by name, separated by the separation character '+'. For

example, y1 ~ x1 + x2 specifies that y1 is to be kernel-weighted by x1 and x2 throughout the sum. See below for further examples.

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth order Gaussian and Epanechnikov kernels, and the uniform kernel. Unordered discrete data types use a variation on Aitchison and Aitken's (1976) kernel, while ordered data types use a variation of the Wang and van Ryzin (1981) kernel (see npRmpi for details).

The option operator= can be used to 'mix and match' operator strings to create a 'hybrid' kernel provided they match the dimension of the data. For example, for a two-dimensional data frame of numeric datatypes, operator=c("normal","derivative") will use the normal (i.e. PDF) kernel for variable one and the derivative of the PDF kernel for variable two. Please note that applying operators will scale the results by factors of h or $1/h$ where appropriate.

The option permutation.operator= can be used to 'mix and match' operator strings to create a 'hybrid' kernel, in addition to the kernel sum with no operators applied, one for each continuous dimension in the data. For example, for a two-dimensional data frame of numeric datatypes, permutation.operator=c("derivative") will return the usual kernel sum as if operator = c("normal","normal") in the ksum member, and in the p.ksum member, it will return kernel sums for operator = c("derivative","normal"), and operator = c("normal","derivative"). This makes the computation of gradients much easier.

The option compute.score= can be used to compute the gradients with respect to h in addition to the normal kernel sum. Like permutations, the additional results are returned in the p.ksum. This option does not work in conjunction with permutation.operator.

The option compute.ocg= works much like permutation.operator, but for discrete variables. The kernel is evaluated at a reference category in each dimension: for ordered data, the next lowest category is selected, except in the case of the lowest category, where the second lowest category is selected; for unordered data, the first category is selected. These additional data are returned in the p.ksum member. This option can be set simultaneously with permutation.operator.

The option return.kernel.weights=TRUE returns a matrix of dimension 'number of training observations' by 'number of evaluation observations' and contains only the generalized product kernel weights ignoring all other objects and options that may be provided to npksum (e.g. bandwidth.divide=TRUE will be ignored, etc.). Summing the columns of the weight matrix and dividing by 'number of training observations' times the product of the bandwidths (i.e. colMeans(foo$kw)/prod(h)) would produce the kernel estimator of a (multivariate) density (operator="normal") or multivariate cumulative distribution (operator="integral").

Value

npksum returns a npkernelsum object with the following components:

eval	the evaluation points
ksum	the sum at the evaluation points
kw	the kernel weights (when return.kernel.weights=TRUE is specified)

Usage Issues

If you are using data of mixed types, then it is advisable to use the data.frame function to construct your input data and not cbind, since cbind will typically not work as intended on mixed data types and will coerce the data to the same type.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), " Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

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

Li, Q. and J.S. Racine (2003), "Nonparametric estimation of distributions with categorical and continuous data," Journal of Multivariate Analysis, 86, 266-292.

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

Scott, D.W. (1992), *Multivariate Density Estimation. Theory, Practice and Visualization,* New York: Wiley.

Silverman, B.W. (1986), *Density Estimation,* London: Chapman and Hall.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

npRmpi.init for MPI startup and workflow guidance.

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
```

```
npRmpi.init(nslaves=1)

n <- 100000
x <- rnorm(n)
x.eval <- seq(-4, 4, length=50)

bw <- npudensbw(dat=x, bwmethod="normal-reference")

den.ksum <- npksum(txdat=x, exdat=x.eval, bws=bw$bw,
                   bandwidth.divide=TRUE)$ksum/n

den.ksum

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

nplsqreg *Location-Scale Kernel Quantile Regression with Mixed Data Types*

Description

nplsqreg computes a location-scale kernel estimate of the conditional quantile function for a one dimensional dependent variable and mixed continuous, unordered factor, and ordered factor explanatory data. Unlike npqreg, which obtains conditional quantiles by inverting an estimated conditional distribution, nplsqreg estimates the requested conditional quantile surface directly using a locally weighted quantile-kernel construction.

Usage

```
nplsqreg(bws, ...)

## S3 method for class 'formula'
```

```
nplsqreg(bws, data = NULL, newdata = NULL, tau = 0.5,
       gradients = FALSE, residuals = FALSE, subset, na.action,
       gradient.order = 1L, ...)

## S3 method for class 'lsqregressionbandwidth'
nplsqreg(bws, txdat = NULL, tydat = NULL,
       tau = bws$tau, ...)

## Default S3 method:
nplsqreg(bws,
       txdat = stop("training data 'txdat' missing"),
       tydat = stop("training data 'tydat' missing"),
       tau = 0.5,
       exdat,
       gradients = FALSE,
       residuals = FALSE,
       gradient.order = 1L,
       ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and training data.

`bws` a formula, an `lsqregressionbandwidth` object returned by `nplsqregbw`, an `rbandwidth` object, a numeric bandwidth vector, or omitted for automatic bandwidth selection. Exact nplsqreg reuse is through the fitted object's `$bws` component; `$reg.bws` is internal regression state.

`data` an optional data frame, list or environment containing the variables in the model. If not found in `data`, the variables are taken from `environment(bws)`.

`subset` an optional vector specifying a subset of observations to be used by the formula method.

`na.action` a function specifying the action to take when missing values are found by the formula method.

`txdat` a p-variate data frame of explanatory data used as training data. Defaults to the training data stored in `bws`.

`tydat` a one dimensional numeric vector of dependent data. Defaults to the training response stored in `bws`.

Evaluation Data And Returned Quantities: These arguments control where the quantile regression is evaluated and which fitted quantities are returned.

`newdata` an optional data frame in which to look for evaluation covariates for formula fits. If omitted, the training data are used.

`exdat` a p-variate data frame of evaluation points. By default, evaluation takes place on `txdat`. The native `exdat` argument takes precedence over `newdata` when both are supplied.

`gradients` a logical value indicating whether gradients and categorical effects of the conditional quantile with respect to the conditioning variables should be computed and returned. Defaults to `FALSE`.

`gradient.order` for a final `regtype="lp"` fit with `gradients=TRUE`, a positive integer scalar or one positive integer per continuous predictor. Each entry requests that coordinate's derivative order; it does not request mixed partial derivatives. The default is the first derivative. Ordered and unordered predictors retain their existing first-difference effect semantics.

`residuals` a logical value indicating whether residuals should be returned for training-data fits. Defaults to `FALSE`.

Quantile Index And Additional Controls: These arguments control the requested quantile probability and the bandwidth selection, prediction, or plotting route.

`tau` a numeric scalar or vector specifying the quantile probability or probabilities τ. Values must lie strictly in $(0, 1)$.

`...` additional arguments supplied to `nplsqregbw` when bandwidths are computed internally, to `npreg` for the final transformed-response fit, or to plotting and prediction methods as appropriate. Common bandwidth-selection controls include `regtype`, `bwtype`, `nmulti`, `degree`, `nomad`, `search.engine`, `tau.search`, `delta`, `scale`, `regtype.pilot`, `nomad.pilot`, and `pilot.args`.

Details

The estimator follows the locally weighted quantile-kernel approach of Racine and Li (2017). Given a conditional scale pilot $\hat{\sigma}(X_i)$, define

$$Y_i^\delta = Y_i + \hat{\sigma}(X_i)\Phi^{-1}(\delta),$$

where $0 < \delta < 1$ and Φ^{-1} is the standard normal quantile function. For a requested quantile probability τ, `nplsqregbw` selects the bandwidths and δ by leave-one-out check-loss cross-validation. With the selected bandwidths and δ, `nplsqreg` then fits a kernel regression of Y_i^δ on X_i using the ordinary mixed-data machinery in `npreg`. The fitted mean of the transformed response is the estimated conditional quantile.

The scale pilot is interpreted as a conditional standard deviation. The default pilot estimates the conditional mean, smooths squared residuals, floors the fitted variance before taking square roots, and then uses the resulting positive scale vector in the quantile-kernel transformation. The local-linear residual-variance pilot follows the idea of Fan and Yao (1998); `regtype.pilot` can be used to select the pilot regression type independently of the final quantile fit.

If `tau` has length greater than one, `tau.search="full"` performs a separate bandwidth/δ search for each quantile while sharing the same pilot scale. The explicit `tau.search="refined"` route fits the central quantile first and warm-starts the remaining quantiles, recording the search order and warm-start provenance in the returned object. The conservative default is `tau.search="full"`.

Gradients and categorical effects are those returned by the final `npreg` fit on the transformed response. Thus ordered-factor effects are finite-difference contrasts and unordered-factor effects follow the corresponding mixed-data regression semantics. For a local-polynomial fit, `gradient.order` is applied coordinate by coordinate to continuous predictors. Availability follows `npreg`: when at least one requested continuous derivative is available, components whose order exceeds their fitted

polynomial degree are returned as NA with a warning; a request with no available continuous derivative component is rejected. No mixed partials, Hessians, derivative tensors, or multiple derivative orders are stored. With vector tau, the existing evaluation-by-coordinate-by-tau array layout is retained. The gradients method extracts exactly the stored order and rejects any different requested order; it does not recompute derivatives after fitting.

Standard errors are the asymptotic standard errors from that final transformed-response regression, conditional on the selected bandwidths, pilot scale, and δ. For vector tau, per-quantile fits are independent, so estimated quantile curves are not monotonized and may cross in finite samples.

Value

nplsqreg returns an object of class lsqregression. The generic functions fitted, quantile, se, predict, residuals, and gradients extract estimated conditional quantiles, asymptotic standard errors from the transformed-response regression, predictions, residuals when requested, and gradients or categorical effects. The functions summary, print, and plot support objects of this type. Local-polynomial fit objects retain the compact continuous-coordinate derivative order in $gradient.order.

Usage Issues

If you are using data of mixed types, then it is advisable to use the data.frame function to construct your input data and not cbind, since cbind will typically not work as intended on mixed data types and will coerce the data to the same type.

When reusing a bandwidth object without explicit training or evaluation data, the required data are recovered from object-retained state or from the stored call environment when available. This is a convenience for ordinary same-session use, not a guarantee that a saved object is self-contained. For reproducible replay after saveRDS() and readRDS() in a fresh R session, supply the required data arguments explicitly.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Fan, J. and Q. Yao (1998), "Efficient Estimation of Conditional Variance Functions in Stochastic Regression," Biometrika, 85, 645-660. doi:10.1093/biomet/85.3.645

Racine, J.S. and K. Li (2017), "Nonparametric conditional quantile estimation: A locally weighted quantile kernel approach," Journal of Econometrics, 201, 72-94. doi:10.1016/j.jeconom.2017.06.020

Racine, J.S. and I. Van Keilegom (2020), "A smooth nonparametric, multivariate, mixed-data location-scale test," Journal of Business & Economic Statistics, 38, 784-795. doi:10.1080/07350015.2019.1574227

See Also

nplsqregbw, npreg, npqreg, np.kernels, np.options, plot.np

Examples

```
## Not run:
data("Italy")

model.q <- nplsqreg(gdp ~ ordered(year), data = Italy,
                    tau = c(0.25, 0.50, 0.75))
plot(model.q)

model.med <- nplsqreg(gdp ~ ordered(year), data = Italy, tau = 0.50)
model.q2 <- nplsqreg(bws = model.med$bws, tau = 0.50)
plot(model.med, gradient = TRUE)

## End(Not run)
```

---

nplsqregbw *Bandwidth Selection for Location-Scale Kernel Quantile Regression*

Description

nplsqregbw selects bandwidths and the scalar location-scale shift parameter delta for nplsqreg by minimizing leave-one-out check loss.

Usage

```
nplsqregbw(bws, ...)

## S3 method for class 'formula'
nplsqregbw(bws, data = NULL, tau = 0.5,
       subset, na.action, ...)

## S3 method for class 'lsqregressionbandwidth'
nplsqregbw(bws, tau = bws$tau, ...)

## Default S3 method:
nplsqregbw(bws,
       xdat = stop("invoked without data 'xdat'"),
       ydat = stop("invoked without data 'ydat'"),
       tau = 0.5,
       tau.search = c("full", "refined"),
       delta = NULL,
       scale = NULL,
       regtype = c("lc", "ll", "lp"),
       regtype.pilot = c("auto", "ll", "lc", "lp"),
       nomad = FALSE,
       nomad.pilot = FALSE,
       pilot.args = list(),
       bandwidth.compute = TRUE,
```

```
        delta.bounds = c(1e-4, 1 - 1e-4),
        optim.control = list(maxit = 50L),
        ...,
        nomad.opts = list())
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and training data.

`bws` a formula, an `lsqregressionbandwidth` object, an `rbandwidth` object, a numeric bandwidth vector, or omitted for automatic selection. Ordinary `npregbw` `rbandwidth` objects may be used as starting values. Exact nplsqreg reuse is through an `lsqregressionbandwidth` object, typically `fit$bws`; `fit$reg.bws` is internal regression state.

`data` an optional data frame, list or environment containing the variables in the model. If not found in `data`, the variables are taken from `environment(bws)`.

`subset` an optional vector specifying a subset of observations to be used by the formula method.

`na.action` a function specifying the action to take when missing values are found by the formula method.

`xdat` a p-variate data frame of explanatory data used as training data.

`ydat` a one dimensional numeric vector of dependent data.

Quantile Index And Vector-Tau Search: These arguments identify the quantile probabilities and how vector-`tau` fits are orchestrated.

`tau` a numeric scalar or vector specifying the quantile probability or probabilities τ. Values must lie strictly in $(0, 1)$.

`tau.search` either `"full"` or `"refined"`. The default `"full"` route performs a full bandwidth/`delta` search for each `tau`, sharing only the pilot scale. The explicit `"refined"` route fits the central `tau` first and uses the resulting bandwidths, `delta`, and when applicable selected degree vector as warm starts for the remaining quantiles.

Scale Pilot: These arguments control or supply the conditional standard deviation used in the location-scale transformation.

`scale` an optional strictly positive scale vector interpreted as the conditional standard deviation at the training observations. If omitted, a pilot scale is computed automatically.

`regtype.pilot` regression type used for the pilot mean and residual-variance smooths. The default `"auto"` uses local linear smoothing when continuous predictors are present and local constant smoothing for categorical-only designs.

`nomad.pilot` logical value indicating whether the pilot regressions should use the automatic NOMAD local-polynomial route when compatible with `regtype.pilot`. Defaults to `FALSE`.

`pilot.args` optional named list of additional arguments supplied to the pilot `npreg` calls.

Bandwidth, Delta And Degree Search: These arguments control the main check-loss bandwidth and `delta` optimization.

`regtype` regression type for the main transformed-response fit and check-loss bandwidth search. This is independent of `regtype.pilot` and follows `npregbw` semantics.

`delta` optional starting value for the shift parameter. If omitted, the first start uses `0.5`; subsequent multistarts draw starting values from $(0, 1)$.

`delta.bounds` two numeric values giving the lower and upper bounds for `delta`. The interval must lie strictly inside $(0, 1)$.

`nomad` logical value indicating whether to use the local-polynomial NOMAD route for joint bandwidth, `delta`, and continuous degree-vector search. Defaults to `FALSE`.

`nomad.opts` optional named list of native NOMAD parameters forwarded to the **crs** NOMAD C API for NOMAD-backed bandwidth searches. `NULL` is treated as no extra NOMAD options. Non-list values and invalid or unsupported parameters are rejected before solver entry.

`bandwidth.compute` logical value indicating whether automatic bandwidth selection should be performed. Defaults to `TRUE`.

`optim.control` a list of controls passed to the Powell optimizer used for fixed degree searches and NOMAD hot-start refinement.

`...` additional bandwidth, kernel, local-polynomial, and search controls forwarded to the package bandwidth machinery. Common examples include `regtype`, `bwtype`, `bwmethod`, `nmulti`, `degree`, `basis`, `bernstein.basis`, `search.engine`, `degree.min`, `degree.max`, `degree.start`, `degree.restarts`, `powell.remin`, `nomad.remin`, and `nomad.nmulti`. The explanatory-variable kernel aliases used by `npqreg` and `npcdistbw`, such as `cxkertype`, `cxkerorder`, `uxkertype`, and `oxkertype`, are accepted and mapped to the corresponding regression-kernel controls. Response-side conditional-distribution kernel controls such as `cykertype` are not meaningful for the location-scale transformed-response estimator and fail clearly rather than being ignored.

Details

For a requested quantile probability τ, `nplsqregbw` chooses bandwidths and `delta` for the transformed response

$$Y_i^\delta = Y_i + \hat{\sigma}(X_i)\Phi^{-1}(\delta)$$

by minimizing the leave-one-out check loss. The selected bandwidths are then used by `nplsqreg` for the final mixed-data `npreg` fit of the transformed response.

The default pilot scale is a residual scale: first `npreg` estimates the conditional mean, then `npreg` smooths squared residuals using the same bandwidth object, and the square root of the floored fitted variance is used as the scale. The local-linear pilot option follows Fan and Yao (1998). A user-supplied `scale` must be strictly positive and is interpreted as a conditional standard deviation, not a variance.

When nomad=TRUE, the search can include the continuous local-polynomial degree vector in addition to bandwidths and delta. The search.engine="nomad+powell" route uses NOMAD for degree search and Powell for hot-start refinement at the selected degree.

Value

nplsqregbw returns an object of class lsqregressionbandwidth. For vector tau, the object stores the per-tau bandwidth objects, selected delta values, objectives, fit order, warm-start provenance, and shared pilot scale.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Fan, J. and Q. Yao (1998), "Efficient Estimation of Conditional Variance Functions in Stochastic Regression," Biometrika, 85, 645-660. doi:10.1093/biomet/85.3.645

Racine, J.S. and K. Li (2017), "Nonparametric conditional quantile estimation: A locally weighted quantile kernel approach," Journal of Econometrics, 201, 72-94. doi:10.1016/j.jeconom.2017.06.020

Racine, J.S. and I. Van Keilegom (2020), "A smooth nonparametric, multivariate, mixed-data location-scale test," Journal of Business & Economic Statistics, 38, 784-795. doi:10.1080/07350015.2019.1574227

See Also

nplsqreg, npregbw, npreg, np.kernels, np.options

---

npplreg *Partially Linear Kernel Regression with Mixed Data Types*

Description

npplreg computes a partially linear kernel regression estimate of a one (1) dimensional dependent variable on $p + q$-variate explanatory data, using the model $Y = X\beta + \Theta(Z) + \epsilon$ given a set of estimation points, training points (consisting of explanatory data and dependent data), and a bandwidth specification, which can be a rbandwidth object, or a bandwidth vector, bandwidth type and kernel type.

Usage

```
npplreg(bws, ...)

## S3 method for class 'formula'
npplreg(bws,
        data = NULL,
        newdata = NULL,
        y.eval = FALSE,
```

```
        ...)

## Default S3 method:
npplreg(bws,
        txdat,
        tydat,
        tzdat,
        nomad = FALSE,
        ...)

## S3 method for class 'plbandwidth'
npplreg(bws,
        txdat = stop("training data txdat missing"),
        tydat = stop("training data tydat missing"),
        tzdat = stop("training data tzdat missing"),
        exdat,
        eydat,
        ezdat,
        residuals = FALSE,
        ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and partially linear training data.

`bws` a bandwidth specification. This can be set as a `plbandwidth` object returned from an invocation of `npplregbw`, or as a matrix of bandwidths, where each row is a set of bandwidths for Z, with a column for each variable Z_i. In the first row are the bandwidths for the regression of Y on Z, the following rows contain the bandwidths for the regressions of the columns of X on Z. If specified as a matrix additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, training data, and so on.

`data` an optional data frame, list or environment (or object coercible to a data frame by `as.data.frame`) containing the variables in the model. If not found in data, the variables are taken from `environment(bws)`, typically the environment from which `npplregbw` was called.

`txdat` a p-variate data frame of explanatory data (training data), corresponding to X in the model equation, whose linear relationship with the dependent data Y is posited. Defaults to the training data used to compute the bandwidth object.

`tydat` a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of `txdat`. Defaults to the corresponding training data recoverable from the bandwidth object in the current session.

`tzdat` a q-variate data frame of explanatory data (training data), corresponding to Z in the model equation, whose relationship to the dependent variable is unspecified (nonparametric). Defaults to the training data used to compute the bandwidth object.

Local-Polynomial Degree And Bandwidth Search: This argument controls the recommended automatic local-polynomial NOMAD route, which jointly selects continuous polynomial degree and bandwidths when these are computed inside `npplreg`.

`nomad` logical shortcut passed through to `npplregbw` when bandwidths are computed inside `npplreg`. When `TRUE`, the partially linear bandwidth route fills any missing values among `regtype`, `search.engine`, `degree.select`, `bernstein.basis`, `degree.min`, `degree.max`, `degree.verify`, and `bwtype` with the recommended automatic local-polynomial degree-and-bandwidth NOMAD preset documented in `npplregbw`. The preset selects local-polynomial degrees and bandwidths separately for the nuisance regressions $E[Y \mid Z]$ and $E[X_j \mid Z]$ before the final partially linear solve. Additional NOMAD tuning arguments such as `nomad.nmulti` may also be supplied through `...`; `nmulti` remains the outer restart count while `nomad.nmulti` controls inner native **crs** NOMAD API multistarts within each outer restart. After fitting, inspect `fit$bws$nomad.shortcut` on the returned object `fit` to see the normalized shortcut metadata.

Evaluation Data And Returned Quantities: These arguments control where the partially linear regression is evaluated and which fitted quantities are returned.

`exdat` a p-variate data frame of points on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by `txdat`.

`eydat` a one (1) dimensional numeric or integer vector of the true values of the dependent variable. Optional, and used only to calculate the true errors. By default, evaluation takes place on the data provided by `tydat`.

`ezdat` a q-variate data frame of points on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by `tzdat`.

`newdata` An optional data frame in which to look for evaluation data. If omitted, the training data are used.

`residuals` a logical value indicating that you want residuals computed and returned in the resulting `plregression` object. Defaults to `FALSE`.

`y.eval` If `newdata` contains dependent data and `y.eval = TRUE`, `npRmpi` will compute goodness of fit statistics on these data and return them. Defaults to `FALSE`.

Additional Arguments: Further arguments are passed to `npplregbw` and its component `npregbw` searches when bandwidths are computed internally.

`...` additional arguments supplied to `npplregbw` when `npplreg` computes bandwidths internally, or arguments needed to interpret a numeric or matrix `bws` specification. This is where bandwidth selection controls such as `bwmethod`, `bwtype`, and `bwscaling`, kernel/support controls such as `ckertype`, `ckerorder`, and `ckerbound`, categorical kernel controls such as `ukertype` and `okertype`, search controls such as `nmulti` and `scale.factor.search.lower`, and local-polynomial/NOMAD controls such as `regtype`, `degree`, `bernstein.basis`, `degree.select`, and `nomad.nmulti` are supplied. See `npplregbw` and `npregbw` for the complete bandwidth-selection argument surface, including the child-indexed matrix/list `degree` form for fixed local-polynomial partially linear fits.

Details

Documentation guide: see npplregbw for partially linear bandwidth selection, npregbw for the component nonparametric regression search controls, np.kernels for kernels, np.options for global options, plot, plot.np for plotting options, and npRmpi.init for interactive/cluster MPI startup. See npRmpi.init details for performance tradeoffs (message passing/startup mode) and the inst/Rprofile manual-broadcast template.

When bws is omitted, the formula and default methods call npplregbw first and pass bandwidth-selection arguments from ... to that call. When bws is already a plbandwidth object, npplreg estimates with the stored bandwidth metadata in that object.

Argument groups for bandwidth selection are documented on npplregbw and, for the component nonparametric regressions, npregbw. The most common workflow is to choose the linear X variables and nonparametric Z variables first, then bandwidth/search controls for the Z-side nonparametric regressions, and finally local-polynomial/NOMAD controls when using polynomial-adaptive fits.

For S3 plotting help, see plot.np. You can list available plot methods with methods("plot").

npplreg uses a combination of OLS and nonparametric regression to estimate the parameter β in the model $Y = X\beta + \Theta(Z) + \epsilon$.

npplreg implements a variety of methods for nonparametric regression on multivariate (q-variate) explanatory data defined over a set of possibly continuous and/or discrete (unordered, ordered) data. The approach is based on Li and Racine (2003) who employ 'generalized product kernels' that admit a mix of continuous and discrete data types.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i, when estimating the density at the point x. Generalized nearest-neighbor bandwidths change with the point at which the density is estimated, x. Fixed bandwidths are constant over the support of x.

Data contained in the data frame tzdat may be a mix of continuous (default), unordered discrete (to be specified in the data frame tzdat using factor), and ordered discrete (to be specified in the data frame tzdat using ordered). Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see npRmpi for details).

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth order Gaussian and Epanechnikov kernels, and the uniform kernel. Unordered discrete data types use a variation on Aitchison and Aitken's (1976) kernel, while ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

For practitioners who want the recommended automatic local-polynomial degree-and-bandwidth NOMAD route without spelling out all LP tuning arguments, npplreg(..., nomad=TRUE) and npplregbw(..., nomad=TRUE) expand missing settings to the same documented preset. Explicit incompatible settings fail fast rather than being silently rewritten.

Value

npplreg returns a plregression object. The generic accessor functions coef, fitted, residuals, predict, and vcov, extract (or estimate) coefficients, estimated values, residuals, predictions, and variance-covariance matrices, respectively, from the returned object. Furthermore, the functions summary and plot support objects of this type. The returned object has the following components:

For `plregression` objects, `predict(object, se.fit = TRUE)` returns a list with components `fit` and `se.fit`. The `se.fit` component contains asymptotic prediction standard errors computed from the partially linear fitted-value linear operator and the squared residuals from the training fit.

`evalx`	evaluation points
`evalz`	evaluation points
`mean`	estimation of the regression, or conditional mean, at the evaluation points
`xcoef`	coefficient(s) corresponding to the components β_i in the model
`xcoeferr`	standard errors of the coefficients
`xcoefvcov`	covariance matrix of the coefficients
`bws`	the canonical bandwidth object, stored as a `plbandwidth` object
`bw`	backward-compatible alias for `bws`
`resid`	if `residuals = TRUE`, in-sample or out-of-sample residuals where appropriate (or possible)
`R2`	coefficient of determination (Doksum and Samarov (1995))
`MSE`	mean squared error
`MAE`	mean absolute error
`MAPE`	mean absolute percentage error
`CORR`	absolute value of Pearson's correlation coefficient
`SIGN`	fraction of observations where fitted and observed values agree in sign

Book And Method Pointers

The partially linear model is $Y = X\beta + \Theta(Z) + \epsilon$. The Robinson-style construction residualizes Y and each column of X on Z, then solves the final least-squares problem using those residualized variables. Thus the nonparametric part of the problem is a sequence of conditional-mean regressions $E[Y \mid Z]$ and $E[X_j \mid Z]$, followed by the parametric solve for β. In local-polynomial/NOMAD fits, the nuisance regressions may select child-specific continuous polynomial degrees.

For book-length derivations, see Li and Racine (2007), Chapter 7 *Semiparametric Partially Linear Models*, especially Sections 7.1, 7.2, and 7.4, and Racine (2019), Chapter 8 *Semiparametric Conditional Mean Function Estimation*, especially Robinson's partially linear model.

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

When reusing a bandwidth object without explicit training or evaluation data, the required data are recovered from object-retained state or from the stored call environment when available. This is a convenience for ordinary same-session use, not a guarantee that a saved object is self-contained. For reproducible replay after `saveRDS()` and `readRDS()` in a fresh R session, supply the required data arguments explicitly.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

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.

Gao, Q. and L. Liu and J.S. Racine (2015), "A partially linear kernel estimator for categorical data," Econometric Reviews, 34 (6-10), 958-977.

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

Li, Q. and J.S. Racine (2004), "Cross-validated local linear nonparametric regression," Statistica Sinica, 14, 485-512.

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

Racine, J.S. and Q. Li (2004), "Nonparametric estimation of regression functions with both categorical and continuous data," Journal of Econometrics, 119, 99-130.

Robinson, P.M. (1988), "Root-n-consistent semiparametric regression," Econometrica, 56, 931-954.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

np.kernels, np.options, plot, plot.np npregbw, npreg

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
```

```
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

set.seed(42)

n <- 250
x1 <- rnorm(n)
x2 <- rbinom(n, 1, .5)

z1 <- rbinom(n, 1, .5)
z2 <- rnorm(n)

y <- 1 + x1 + x2 + z1 + sin(z2) + rnorm(n)

x2 <- factor(x2)
z1 <- factor(z1)

bw <- npplregbw(formula=y~x1+x2|z1+z2)

pl <- npplreg(bws=bw)

summary(pl)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

npplregbw *Partially Linear Kernel Regression Bandwidth Selection with Mixed Data Types*

Description

npplregbw computes a bandwidth object for a partially linear kernel regression estimate of a one (1) dimensional dependent variable on $p + q$-variate explanatory data, using the model $Y = X\beta + \Theta(Z) + \epsilon$ given a set of estimation points, training points (consisting of explanatory data and dependent data), and a bandwidth specification, which can be a rbandwidth object, or a bandwidth vector, bandwidth type and kernel type.

Usage

```
npplregbw(...)

## S3 method for class 'formula'
npplregbw(formula,
          data,
          subset,
          na.action,
          call,
          ...)

## Default S3 method:
npplregbw(xdat = stop("invoked without data `xdat'"),
          ydat = stop("invoked without data `ydat'"),
          zdat = stop("invoked without data `zdat'"),
          bandwidth.compute = TRUE,
          bws,
          degree = NULL,
          degree.select = c("manual", "coordinate", "exhaustive"),
          search.engine = c("nomad+powell", "cell", "nomad"),
          nomad = FALSE,
          nomad.nmulti = 0L,
            degree.min = NULL,
          degree.max = NULL,
          degree.start = NULL,
          degree.restarts = 0L,
          degree.max.cycles = 20L,
          degree.verify = FALSE,
          scale.factor.search.lower = NULL,
          ftol,
          itmax,
          nmulti,
          nomad.remin,
          powell.remin,
          small,
          tol,
          ...)

## S3 method for class 'plbandwidth'
npplregbw(xdat = stop("invoked without data `xdat'"),
```

```
          ydat = stop("invoked without data `ydat'"),
          zdat = stop("invoked without data `zdat'"),
          bws,
          nmulti,
          ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the linear, nonparametric, formula, and bandwidth inputs.

`bandwidth.compute`
: a logical value which specifies whether to do a numerical search for bandwidths or not. If set to `FALSE`, a `plbandwidth` object will be returned with bandwidths set to those specified in `bws`. Defaults to `TRUE`.

`bws`
: a bandwidth specification. This can be set as a `plbandwidth` object returned from an invocation of `npplregbw`, or as a matrix of bandwidths, where each row is a set of bandwidths for Z, with a column for each variable Z_i. In the first row are the bandwidths for the regression of Y on Z. The following rows contain the bandwidths for the regressions of the columns of X on Z. If specified as a matrix, additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, and so on.

  If left unspecified, `npplregbw` will search for optimal bandwidths using `npregbw` in the course of calculations. If specified, `npplregbw` will use the given bandwidths as the starting point for the numerical search for optimal bandwidths, unless you specify bandwidth.compute = FALSE.

`call`
: the original function call. This is passed internally by `npRmpi` when a bandwidth search has been implied by a call to another function. It is not recommended that the user set this.

`data`
: an optional data frame, list or environment (or object coercible to a data frame by `as.data.frame`) containing the variables in the model. If not found in data, the variables are taken from `environment(formula)`, typically the environment from which the function is called.

`formula`
: a symbolic description of variables on which bandwidth selection is to be performed. The details of constructing a formula are described below.

`na.action`
: a function which indicates what should happen when the data contain `NA`s. The default is set by the `na.action` setting of options, and is `na.fail` if that is unset. The (recommended) default is `na.omit`.

`subset`
: an optional vector specifying a subset of observations to be used in the fitting process.

`xdat`
: a p-variate data frame of explanatory data (training data), corresponding to X in the model equation, whose linear relationship with the dependent data Y is posited.

`ydat`
: a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of `xdat`.

zdat — a q-variate data frame of explanatory data (training data), corresponding to Z in the model equation, whose relationship to the dependent variable is unspecified (nonparametric)

Automatic Degree Search Controls: These arguments control automatic local-polynomial degree search.

degree.max — optional scalar or integer vector giving upper bounds for automatic degree search over continuous zdat predictors when degree.select != "manual".

degree.max.cycles — positive integer giving the maximum number of coordinate-search sweeps over the degree vector. Ignored for "manual" and "exhaustive" degree selection.

degree.min — optional scalar or integer vector giving lower bounds for automatic degree search over continuous zdat predictors when degree.select != "manual".

degree.restarts — non-negative integer giving the number of additional deterministic coordinate-search restarts. Ignored for "manual" and "exhaustive" degree selection.

degree.select — character string controlling local-polynomial degree handling for the nuisance regressions of Y on Z and each column of X on Z. "manual" (default) treats degree as fixed. "coordinate" performs cached coordinate-wise search over admissible degree vectors. "exhaustive" evaluates the full admissible degree grid when search.engine="cell". For NOMAD-based search engines, any non-"manual" value requests a separate degree and bandwidth search for each nuisance regression.

degree.start — optional starting degree vector for automatic coordinate search. If omitted, the search starts from the degree-zero local-constant baseline on the continuous zdat predictors.

degree.verify — logical value indicating whether a coordinate-search solution should be exhaustively verified over the admissible degree grid after the heuristic phase completes. Available only for search.engine="cell".

Continuous Scale-Factor Search Controls: These controls define lower admissibility bounds for continuous fixed-bandwidth search.

scale.factor.search.lower — optional nonnegative scalar giving the hard lower admissibility bound for continuous fixed-bandwidth search candidates. Defaults to NULL. If NULL, an existing bandwidth object's stored value is inherited when available; otherwise the package default 0.1 is used. This floor applies to computed/search bandwidth candidates and to effective search starts only. It does not rewrite explicit bandwidths supplied for storage with bandwidth.compute = FALSE. Final fixed-bandwidth search candidates must also have a finite valid raw objective value.

Local-Polynomial Model Specification: These arguments control fixed local-polynomial specification for the nonparametric component.

degree — for local-polynomial partially linear fits, polynomial degree specification for each continuous nonparametric regressor in zdat. When supplied with degree.select="manual",

bandwidth optimization treats this input as fixed. A numeric vector is used as a common degree vector for every nuisance regression. To specify child-specific fixed degrees, supply either a matrix with $p + 1$ rows and one column per continuous `zdat` predictor, or a list with $p + 1$ entries. The child order matches the bandwidth matrix/object order: first the regression of Y on Z (row or entry `yzbw`), followed by the regressions of the columns of X on Z. Named matrix rows or list entries are accepted and reordered to this canonical order.

NOMAD Search Controls: These arguments control the optional NOMAD direct-search route for local-polynomial degree and bandwidth search.

`nomad` logical or character shortcut for the recommended automatic local-polynomial NOMAD route for the partially linear nuisance regressions. When `TRUE`, any missing values among `regtype`, `search.engine`, `degree.select`, `bernstein.basis`, `degree.min`, `degree.max`, `degree.verify`, and `bwtype` are filled with `regtype="lp"`, `search.engine="nomad+powell"`, `degree.select="coordinate"`, `bernstein.basis=TRUE`, `degree.min=0L`, `degree.max=10L`, `degree.verify=FALSE`, and `bwtype="fixed"`. Explicit incompatible settings error immediately; in particular, explicit `nomad=TRUE` currently requires `regtype="lp"`, `bwtype="fixed"`, automatic degree search, no explicit `degree`, and `search.engine %in% c("nomad", "nomad+powell")`. This shortcut does not change the meaning of `nmulti` or `nomad.nmulti`: `nmulti` remains the outer restart count, while `nomad.nmulti` controls inner native **crs** NOMAD API multistarts within each outer restart. Returned bandwidth objects retain this normalized preset metadata in `bw$nomad.shortcut` for a returned object `bw`; when available, `nomad.time` and `powell.time` record the direct-search and Powell-polish timing components. Each nuisance regression selects its own admissible degree and bandwidths.

`nomad.nmulti` non-negative integer controlling the inner the native **crs** NOMAD C API multistart count used within each outer NOMAD restart when `regtype="lp"` and automatic degree search uses `search.engine="nomad"` or `"nomad+powell"`. Defaults to `0L`, which preserves the current one-start-per- restart behavior. This does not replace `nmulti`: `nmulti` controls outer restarts, while `nomad.nmulti` controls inner NOMAD multistarts within each outer restart. Advanced native NOMAD parameters may be supplied as `nomad.opts` through `...`; invalid or unsupported parameters are rejected by NOMAD rather than silently ignored. Routes that evaluate R callbacks require serial callback evaluation and reject parallel callback settings such as `NB_THREADS_PARALLEL_EVAL > 1`.

`nomad.remin` logical flag controlling the optional second NOMAD hot start. When `TRUE`, NOMAD is restarted once from the best full candidate found, including both bandwidth and degree coordinates. Defaults to `FALSE`; current simulation evidence favors the one-pass NOMAD default for routine use, while leaving this switch available for sensitivity checks.

`search.engine` character string controlling the automatic local-polynomial search backend for each nuisance regression when `degree.select != "manual"`. `"nomad+powell"` (default) performs direct search over fixed bandwidths and the degree vector using the native **crs** NOMAD C API, then applies one Powell hot start from the NOMAD solution, separately for each nuisance regression. `"nomad"` omits the Powell refinement. `"cell"` profiles the criterion over the admissible degree grid

using repeated fixed-degree bandwidth solves. NOMAD-based search supports fixed, generalized-nearest-neighbor, and adaptive-nearest-neighbor child templates; it currently requires `degree.verify=FALSE` and the suggested package **crs** to be installed.

Numerical Search And Tolerance Controls: These controls set optimizer tolerances and restart behavior.

`ftol` tolerance on the value of the cross-validation function evaluated at located minima. Defaults to `1.19e-07 (FLT_EPSILON)`

`itmax` integer number of iterations before failure in the numerical optimization routine. Defaults to `10000`

`nmulti` integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points. Defaults to `min(2,ncol(zdat))`.

`powell.remin` logical flag controlling Powell restart-from-minimum behavior. For ordinary fixed-degree Powell-style search, `TRUE` restarts the local search from the located minimum. For `search.engine="nomad+powell"`, this controls only the final Powell bandwidth-polish step. The default is `TRUE` for ordinary Powell routes and `FALSE` for the Powell polish after NOMAD unless explicitly supplied.

`small` a small number, at about the precision of the data type used. Defaults to `2.22e-16 (DBL_EPSILON)`

`tol` tolerance on the position of located minima of the cross-validation function. Defaults to `1.49e-08 (sqrt(DBL_EPSILON))`

Additional Arguments: These arguments collect remaining controls passed through S3 methods.

`...` additional arguments supplied to specify the regression type, bandwidth type, kernel types, selection methods, and so on. To do this, you may specify any of `regtype`, `bwmethod`, `bwscaling`, `bwtype` (one of `fixed`, `generalized_nn`, `adaptive_nn`), `ckertype`, `ckerorder`, `ukertype`, `okertype`, `bernstein.basis`, and `basis`, as described in `npregbw`.

Details

The `scale.factor.*` controls are dimensionless search controls. The package converts scale factors to bandwidths using the estimator-specific scaling encoded in the bandwidth object, including kernel order and the number of continuous variables relevant for the estimator. Users should not pre-multiply these controls by sample-size or standard-deviation factors.

`scale.factor.init` controls the deterministic first search start when that control is exposed. `scale.factor.init.lower` and `scale.factor.init.upper` define the random multistart interval when exposed. `scale.factor.search.lower` is the lower admissibility bound for continuous fixed-bandwidth search candidates. The effective first start is `max(scale.factor.init, scale.factor.search.lower)` when both controls are present, and the effective random-start lower endpoint is `max(scale.factor.init.lower, scale.factor.search.lower)`. `scale.factor.init.upper` must be at least that effective lower endpoint; the package errors rather than silently expanding the user's interval.

When scale.factor.search.lower is NULL, an existing bandwidth object's stored floor is inherited when available; otherwise the package default 0.1 is used. Explicit bandwidths supplied for storage with bandwidth.compute = FALSE are not rewritten by the search floor.

Categorical search-start controls such as dfac.init, lbd.init, and hbd.init have separate semantics and are not affected by scale.factor.search.lower.

Documentation guide: see npregbw for component nonparametric regression bandwidth controls, np.kernels for kernels, np.options for global options, plot, plot.np for plotting options, and npRmpi.init for interactive/cluster MPI startup. See npRmpi.init details for performance tradeoffs (message passing/startup mode) and the inst/Rprofile manual-broadcast template.

The partially linear bandwidth-selection argument surface is easiest to read by decision group: linear xdat inputs, nonparametric zdat inputs, and existing bandwidth inputs; local-polynomial/NOMAD controls for the nonparametric component; numerical search and feasibility controls; formula-interface controls; and additional bandwidth, kernel, and support controls that are passed to the component npregbw searches.

For S3 plotting help, see plot.np. You can list available plot methods with methods("plot").

npplregbw implements a variety of methods for nonparametric regression on multivariate (q-variate) explanatory data defined over a set of possibly continuous and/or discrete (unordered, ordered) data. The approach is based on Li and Racine (2003), who employ 'generalized product kernels' that admit a mix of continuous and discrete data types.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i, when estimating the density at the point x. Generalized nearest-neighbor bandwidths change with the point at which the density is estimated, x. Fixed bandwidths are constant over the support of x.

npplregbw may be invoked *either* with a formula-like symbolic description of variables on which bandwidth selection is to be performed *or* through a simpler interface whereby data is passed directly to the function via the xdat, ydat, and zdat parameters. Use of these two interfaces is **mutually exclusive**.

Data contained in the data frame zdat may be a mix of continuous (default), unordered discrete (to be specified in the data frame zdat using factor), and ordered discrete (to be specified in the data frame zdat using ordered). Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see npRmpi for details).

Data for which bandwidths are to be estimated may be specified symbolically. A typical description has the form dependent data ~ parametric explanatory data | nonparametric explanatory data, where dependent data is a univariate response, and parametric explanatory data and nonparametric explanatory data are both series of variables specified by name, separated by the separation character '+'. For example, y1 ~ x1 + x2 | z1 specifies that the bandwidth object for the partially linear model with response y1, linear parametric regressors x1 and x2, and nonparametric regressor z1 is to be estimated. See below for further examples.

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth order Gaussian and Epanechnikov kernels, and the uniform kernel. Unordered discrete data types use a variation on Aitchison and Aitken's (1976) kernel, while ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

When the nonparametric component is estimated with regtype="lp" and degree.select != "manual", npplregbw determines the zdat-side degree vector and associated bandwidth coordinates separately for the nuisance regressions $E[Y \mid Z]$ and $E[X_j \mid Z]$. With search.engine="cell", the

criterion is profiled over the degree grid using cached coordinate-wise or exhaustive search together with repeated fixed-degree bandwidth solves. With `search.engine="nomad"` or `"nomad+powell"`, each nuisance criterion is optimized directly over that child's degree/bandwidth space using the native **crs** NOMAD C API; `"nomad+powell"` then performs one Powell hot start for that child and keeps the better of the direct NOMAD and polished solutions. For the nonparametric regression components, this polynomial-adaptive joint-search route follows Hall and Racine (2015).

Setting `nomad=TRUE` is a convenience preset for this automatic LP route, not a generic optimizer alias. For partially linear regression it expands any missing values to the equivalent long-form call

```
npplregbw(...,
          regtype = "lp",
          search.engine = "nomad+powell",
          degree.select = "coordinate",
          bernstein.basis = TRUE,
          degree.min = 0L,
          degree.max = 10L,
          degree.verify = FALSE,
          bwtype = "fixed")
```

Compatible explicit tuning arguments are respected. Incompatible explicit settings fail fast so the shortcut never silently changes user-selected semantics. The character value `nomad="auto"` applies the same LP shortcut but leaves `search.engine` and `degree.select` eligible for data-driven resolution when both were not supplied explicitly: scalar continuous degree searches use the exhaustive degree-grid route, while higher-dimensional degree searches keep the NOMAD/Powell route. Explicit `search.engine` or `degree.select` choices are honored.

Value

if `bwtype` is set to `fixed`, an object containing bandwidths (or scale factors if `bwscaling = TRUE`) is returned. If it is set to `generalized_nn` or `adaptive_nn`, then instead the kth nearest neighbors are returned for the continuous variables while the discrete kernel bandwidths are returned for the discrete variables. Bandwidths are stored in a list under the component name `bw`. Each element is an `rbandwidth` object. The first element of the list corresponds to the regression of Y on Z. Each subsequent element is the bandwidth object corresponding to the regression of the ith column of X on Z. See examples for more information.

Book And Method Pointers

`npplregbw` selects the bandwidths and, for local-polynomial routes, child-specific degree/search metadata used by `npplreg`. The target model is $Y = X\beta + \Theta(Z) + \epsilon$; the bandwidth object controls the nuisance regressions $E[Y \mid Z]$ and $E[X_j \mid Z]$ used in the final partially linear solve.

For book-length derivations, see Li and Racine (2007), Chapter 7 *Semiparametric Partially Linear Models*, especially Sections 7.1, 7.2, and 7.4, and Racine (2019), Chapter 8 *Semiparametric Conditional Mean Function Estimation*.

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

Caution: multivariate data-driven bandwidth selection methods are, by their nature, *computationally intensive*. Virtually all methods require dropping the ith observation from the data set, computing an object, repeating this for all observations in the sample, then averaging each of these leave-one-out estimates for a *given* value of the bandwidth vector, and only then repeating this a large number of times in order to conduct multivariate numerical minimization/maximization. Furthermore, due to the potential for local minima/maxima, *restarting this procedure a large number of times may often be necessary*. This can be frustrating for users possessing large datasets. For exploratory purposes, you may wish to override the default search tolerances, say, setting ftol=.01 and tol=.01 and conduct multistarting (the default is to restart min(2, ncol(zdat)) times) as is done for a number of examples. Once the procedure terminates, you can restart search with default tolerances using those bandwidths obtained from the less rigorous search (i.e., set `bws=bw` on subsequent calls to this routine where `bw` is the initial bandwidth object). A version of this package using the `Rmpi` wrapper is under development that allows one to deploy this software in a clustered computing environment to facilitate computation involving large datasets.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

Gao, Q. and L. Liu and J.S. Racine (2015), "A partially linear kernel estimator for categorical data," Econometric Reviews, 34 (6-10), 958-977.

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

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

Li, Q. and J.S. Racine (2004), "Cross-validated local linear nonparametric regression," Statistica Sinica, 14, 485-512.

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

Racine, J.S. and Q. Li (2004), "Nonparametric estimation of regression functions with both categorical and continuous data," Journal of Econometrics, 119, 99-130.

Robinson, P.M. (1988), "Root-n-consistent semiparametric regression," Econometrica, 56, 931-954.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

`np.kernels`, `np.options`, `plot`, `plot.np` `npregbw`, `npreg`

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

set.seed(42)

n <- 250
x1 <- rnorm(n)
x2 <- rbinom(n, 1, .5)

z1 <- rbinom(n, 1, .5)
z2 <- rnorm(n)

y <- 1 + x1 + x2 + z1 + sin(z2) + rnorm(n)

x2 <- factor(x2)
z1 <- factor(z1)

bw <- npplregbw(formula=y~x1+x2|z1+z2)

summary(bw)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
```

```
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

npqcmstest *Kernel Consistent Quantile Regression Model Specification Test with Mixed Data Types*

Description

npqcmstest implements a consistent test for correct specification of parametric quantile regression models (linear or nonlinear) as described in Racine (2006) which extends the work of Zheng (1998).

Usage

```
npqcmstest(formula,
           data = NULL,
           subset,
           xdat,
           ydat,
           model = stop(paste(sQuote("model"),," has not been provided")),
           tau = 0.5,
           distribution = c("bootstrap", "asymptotic"),
           bwydat = c("y","varepsilon"),
           boot.method = c("iid","wild","wild-rademacher"),
           boot.num = 399,
           pivot = TRUE,
           density.weighted = TRUE,
           random.seed = 42,
           ...)
```

Arguments

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

`data` an optional data frame, list or environment (or object coercible to a data frame by `as.data.frame`) containing the variables in the model. If not found in data, the variables are taken from `environment(formula)`, typically the environment from which the function is called.

`formula` a symbolic description of variables on which the test is to be performed. The details of constructing a formula are described below.

`model` a model object obtained from a call to `rq`. Important: the call to `rq` must have the argument `model=TRUE` or `npqcmstest` will not work.

`subset` an optional vector specifying a subset of observations to be used.

`xdat` a p-variate data frame of explanatory data (training data) used to calculate the quantile regression estimators.

`ydat` a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of `xdat`.

Bootstrap And Test Controls: These arguments control the quantile level, test statistic, bootstrap procedure, and reproducibility settings.

`boot.method` a character string used to specify the bootstrap method. `iid` will generate independent identically distributed draws. `wild` will use a wild bootstrap. `wild-rademacher` will use a wild bootstrap with Rademacher variables. Defaults to `iid`.

`boot.num` an integer value specifying the number of bootstrap replications to use. Defaults to `399`.

`bwydat` a character string used to specify the left hand side variable used in bandwidth selection. `"varepsilon"` uses $1-\tau, -\tau$ for `ydat` while `"y"` will use y. Defaults to `"y"`.

`density.weighted` a logical value specifying whether the statistic should be weighted by the density of `xdat`. Defaults to `TRUE`.

`distribution` a character string used to specify the method of estimating the distribution of the statistic to be calculated. `bootstrap` will conduct bootstrapping. `asymptotic` will use the normal distribution. Defaults to `bootstrap`.

`pivot` a logical value specifying whether the statistic should be normalised such that it approaches $N(0,1)$ in distribution. Defaults to `TRUE`.

`random.seed` an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.

`tau` a numeric value specifying the τth quantile is desired

Additional Arguments: Further arguments are passed to the bandwidth-selection routines used by the test.

`...` additional arguments supplied to control bandwidth selection on the residuals. One can specify the bandwidth type, kernel types, and so on. To do this, you may specify any of `bwscaling`, `bwtype`, `ckertype`, `ckerorder`, `ukertype`, `okertype`, as described in `npregbw`. This is necessary if you specify `bws` as a p-vector and not a `bandwidth` object, and you do not desire the default behaviours.

Details

For MPI startup/performance guidance (including message-passing tradeoffs and the manual-broadcast template), see npRmpi.init details and inst/Rprofile. Documentation guide: see np.kernels for kernels, np.options for global options, and plot for plotting options.

Value

npqcmstest returns an object of type cmstest with the following components. Components will contain information related to Jn or In depending on the value of pivot:

Jn	the statistic Jn
In	the statistic In
Omega.hat	as described in Racine, J.S. (2006).
q.*	the various quantiles of the statistic Jn (or In if pivot=FALSE) are in components q.90, q.95, q.99 (one-sided 1%, 5%, 10% critical values)
P	the P-value of the statistic
Jn.bootstrap	if pivot=TRUE contains the bootstrap replications of Jn
In.bootstrap	if pivot=FALSE contains the bootstrap replications of In

summary supports object of type cmstest.

Book And Method Pointers

npqcmstest tests a parametric conditional-quantile specification against a nonparametric conditional-quantile alternative at the supplied τ. The test therefore sits at the intersection of conditional CDF/quantile estimation and model specification testing.

For book-length background, see Li and Racine (2007), Chapter 12 *Model Specification Tests* and Chapter 6 *Conditional CDF and Quantile Estimation*. For the later workflow treatment of conditional CDFs and quantiles, see Racine (2019), Chapter 4 *Conditional Probability Density and Cumulative Distribution Functions*.

Usage Issues

If you are using data of mixed types, then it is advisable to use the data.frame function to construct your input data and not cbind, since cbind will typically not work as intended on mixed data types and will coerce the data to the same type.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

Koenker, R.W. and G.W. Bassett (1978), "Regression quantiles," Econometrica, 46, 33-50.

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

Murphy, K. M. and F. Welch (1990), "Empirical age-earnings profiles," Journal of Labor Economics, 8, 202-229.

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

Racine, J.S. (2006), "Consistent specification testing of heteroskedastic parametric regression quantile models with mixed data," manuscript.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

Zheng, J. (1998), "A consistent nonparametric test of parametric regression models under conditional quantile restrictions," Econometric Theory, 14, 123-138.

See Also

npRmpi.init. np.kernels, np.options, plot, npregbw.

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

library("quantreg")

data("cps71")

model <- rq(logwage~age+I(age^2), data=cps71, tau=0.5, model=TRUE)
```

```
X <- data.frame(age=cps71$age)

# Note - this may take a few minutes depending on the speed of your
# computer...

output <- npqcmstest(model=model, xdat=X,
                     ydat=cps71$logwage, tau=0.5,
                     boot.num=29)

summary(output)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

npqreg *Kernel Quantile Regression with Mixed Data Types*

Description

npqreg computes a kernel quantile regression estimate of a one (1) dimensional dependent variable on p-variate explanatory data, given a set of evaluation points, training points (consisting of explanatory data and dependent data), and a bandwidth specification using the methods of Li and Racine (2008) and Li, Lin and Racine (2013). A bandwidth specification can be a condbandwidth object, or a bandwidth vector, bandwidth type and kernel type.

Usage

```
npqreg(bws, ...)

## S3 method for class 'formula'
npqreg(bws,
```

```
        data = NULL,
        newdata = NULL,
        ...)

## S3 method for class 'condbandwidth'
npqreg(bws,
        txdat = stop("training data 'txdat' missing"),
        tydat = stop("training data 'tydat' missing"),
        exdat,
        tau = 0.5,
        gradients = FALSE,
        tol = 1.490116e-04,
        small = 1.490116e-05,
        itmax = 10000,
        ...)

## Default S3 method:
npqreg(bws,
        txdat,
        tydat,
        nomad = FALSE,
        ...)

## S3 method for class 'qregression'
predict(object, se.fit = FALSE, ...)

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

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and training data.

`bws` a bandwidth specification. This can be set as a `condbandwidth` object returned from an invocation of `npcdistbw`, or as a vector of bandwidths, with each element i corresponding to the bandwidth for column i in `txdat`. If specified as a vector, then additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, and so on.

`data` an optional data frame, list or environment (or object coercible to a data frame by `as.data.frame`) containing the variables in the model. If not found in data, the variables are taken from `environment(bws)`, typically the environment from which `npcdistbw` was called.

`txdat` a p-variate data frame of explanatory data (training data) used to calculate the regression estimators. Defaults to the corresponding training data recoverable from the bandwidth object in the current session.

tydat — a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of `txdat`. Defaults to the corresponding training data recoverable from the bandwidth object in the current session.

object — an object of class `"qregression"` returned by `npqreg`.

x — an object of class `"qregression"` returned by `npqreg`.

Local-Polynomial Degree And Bandwidth Search: This argument controls the recommended automatic local-polynomial NOMAD route, which jointly selects continuous polynomial degree and bandwidths when conditional-distribution bandwidths are computed inside `npqreg`.

nomad — logical shortcut passed through to `npcdistbw` when bandwidths are computed inside `npqreg`. When `TRUE`, the conditional-distribution bandwidth route fills any missing values among `regtype`, `search.engine`, `degree.select`, `bernstein.basis`, `degree.min`, `degree.max`, `degree.verify`, and `bwtype` with the recommended automatic local-polynomial degree-and-bandwidth NOMAD preset documented in `npcdistbw`. Additional NOMAD tuning arguments such as `nomad.nmulti` may also be supplied through `...`; `nmulti` remains the outer restart count while `nomad.nmulti` controls inner native **crs** NOMAD API multistarts within each outer restart. After fitting, inspect `fit$bws$nomad.shortcut` on the returned object `fit` to see the normalized shortcut metadata.

Evaluation Data And Returned Quantities: These arguments control where the quantile regression is evaluated and which fitted quantities are returned.

exdat — a p-variate data frame of points on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by `txdat`.

gradients — a logical value indicating that you want gradients of the conditional quantile with respect to the conditioning variables computed and returned in the resulting `qregression` object. Defaults to `FALSE`.

newdata — An optional data frame in which to look for evaluation data. If omitted, the training data are used.

se.fit — logical value. If `TRUE`, `predict.qregression` returns a list with components `fit` and `se.fit`; otherwise it returns fitted conditional quantiles.

tau — a numeric scalar or vector specifying the quantile probability or probabilities τ. Defaults to `0.5`.

Quantile Solver Controls: These arguments control the one-dimensional numerical quantile extraction step.

itmax — integer maximum number of iterations allowed in the one-dimensional quantile refinement. Defaults to `10000`.

small — minimum interval width used by the one-dimensional quantile refinement. Defaults to `1.490116e-05` (approximately `1000*sqrt(.Machine$double.eps)`).

tol — tolerance on the one-dimensional quantile location refinement. Defaults to `1.490116e-04` (approximately `10000*sqrt(.Machine$double.eps)`).

Additional Arguments: Further arguments are passed to the bandwidth-selection counterpart, prediction/evaluation route, or plot route as appropriate.

. . . additional arguments supplied to npcdistbw when npqreg computes bandwidths internally, or arguments needed to interpret a numeric bws vector. This is where bandwidth selection controls such as bwmethod, bwtype, and bwscaling, kernel/support controls such as cxkertype, cykertype, cxkerorder, cykerorder, cxkerbound, and cykerbound, categorical kernel controls such as uxkertype, uykertype, oxkertype, and oykertype, search controls such as nmulti and scale.factor.search.lower, and local-polynomial/NOMAD controls such as regtype, degree, bernstein.basis, degree.select, and nomad.nmulti are supplied. In predict.qregression, additional arguments are passed to npqreg for evaluation with the stored bandwidth object; common examples are newdata, native exdat, and tau. In plot.qregression, additional arguments are passed through the package plot route; common controls include tau, gradients, output, legend, and graphics arguments. See npcdistbw and plot.np for the complete bandwidth-selection and plotting argument surfaces.

Details

Documentation guide: see np.kernels for kernels, np.options for global options, plot, plot.np for plotting options, and npRmpi.init for interactive/cluster MPI startup. See npRmpi.init details for performance tradeoffs (message passing/startup mode) and the inst/Rprofile manual-broadcast template.

Given a conditional distribution bandwidth object, npqreg estimates the conditional distribution function $F(y|x)$ and extracts the requested conditional quantile. For $0 < \tau < 1$, the conditional quantile at probability τ is

$$q_\tau(x) = \inf\{y : F(y|x) \geq \tau\}.$$

Equivalently, $q_\tau(x)$ is a quasi-inverse of the conditional distribution in the sense of Nelsen (2006): an inverse agrees with F on the range of F, while outside that range the generalized inverse is defined by the lower endpoint at which F reaches or exceeds the requested probability. Numerically, npqreg inverts the selected conditional distribution estimator represented by bws. This includes the selected bandwidth type, kernels, local-polynomial regression type, selected polynomial degree, basis, and Bernstein-basis setting inherited from npcdistbw. If the bandwidth object was selected with nomad=TRUE, the returned conditional-distribution bandwidth object is an LP object: its regtype/regtype.engine metadata identify the selected local-polynomial route and its degree/degree.engine metadata record the selected continuous-coordinate polynomial degree. npqreg, predict, and plot reuse this stored LP metadata; plotting additional tau values does not recompute or downgrade the selected degree. The one-dimensional inversion is carried out over the observed support of the dependent variable using the same selected conditional CDF estimator that is later used for quantile standard errors and gradients. The arguments tol, small, and itmax control this one-dimensional refinement. Conditional quantile inversion requires a monotone dependent-variable conditional distribution estimate; therefore npqreg supports cykerorder = 2 for this route and rejects higher-order dependent-variable CDF kernels, which can be nonmonotone.

Let $f(y|x) = \partial F(y|x)/\partial y$ denote the conditional density. The asymptotic standard error of the

conditional quantile is computed by the first-order delta method,

$$se\{\hat{q}_\tau(x)\} = \frac{se\{\hat{F}(\hat{q}_\tau(x)|x)\}}{\hat{f}(\hat{q}_\tau(x)|x)},$$

using the selected conditional distribution standard-error machinery and the selected conditional density evaluated at the fitted quantile. This corresponds to the quantile variance expression in Li, Lin and Racine (2013).

If `gradients=TRUE`, `npqreg` also computes gradients of the conditional quantile with respect to the conditioning variables for which gradients are defined. Differentiating $F(q_\tau(x)|x) = \tau$ gives

$$\nabla_x q_\tau(x) = -\frac{F_x(q_\tau(x)|x)}{f(q_\tau(x)|x)},$$

where $F_x(y|x)$ is the derivative of the same selected conditional distribution estimator with respect to x. For `regtype="lc"`, this uses the local-constant conditional-gradient machinery; for `regtype="ll"` it uses the canonical local-polynomial degree-one route; and for `regtype="lp"` it uses the selected or supplied degree vector. The corresponding first-order gradient standard errors are computed componentwise as

$$se\{\nabla_x \hat{q}_\tau(x)\} = \frac{se\{\hat{F}_x(\hat{q}_\tau(x)|x)\}}{\hat{f}(\hat{q}_\tau(x)|x)}.$$

When `npqreg` is called without an explicit `bws` object, it first computes conditional distribution bandwidths using `npcdistbw` and stores them in the returned object's `bws` component. If a scalar `tau` was used initially and additional quantiles are later desired as fitted objects, reuse those selected bandwidths directly, for example `npqreg(bws = fit$bws, tau = c(0.25, 0.5, 0.75))`. If the goal is only to inspect additional quantiles graphically, use `plot(fit, tau = c(0.25, 0.5, 0.75))`; this reuses the stored bandwidths and recomputes only the one-dimensional quantile extraction step for the requested `tau` values. Vector-`tau` plots are overlaid and include a legend; use `legend=FALSE`, `legend=NULL`, or a `legend=list(...)` control to suppress or customize it.

The `predict` method follows the usual S3 `newdata` convention. For formula fits, supply a data frame of evaluation covariates via `predict(fit, newdata=...)`. For non-formula fits, `newdata` is translated to the native evaluator argument `exdat` when `exdat` is not supplied. The native `exdat` argument remains available for advanced workflows and takes precedence if both `newdata` and `exdat` are supplied. If `tau` is omitted in `predict`, the fitted object's stored `tau` value is used.

Value

`npqreg` returns a `qregression` object. The generic functions `fitted` (or `quantile`), `se`, `predict`, and `gradients` extract (or generate) estimated values, asymptotic standard errors on estimates, predictions, and gradients, respectively, from the returned object. `predict` uses the object's stored `tau` value by default; supply `tau=` to override it. Furthermore, the functions `summary` and `plot` support objects of this type. The returned object has the following components:

`eval`	evaluation points
`quantile`	estimation of the quantile regression function (conditional quantile) at the evaluation points. If `tau` has length greater than one this is an evaluation-by-`tau` matrix.

quanterr	asymptotic standard errors of the quantile regression estimates, obtained from the conditional distribution standard error and the estimated conditional density at the fitted quantile. If tau has length greater than one this is an evaluation-by-tau matrix.
quantgrad	gradients of the conditional quantile with respect to the conditioning variables at each evaluation point, when gradients=TRUE. If tau has length greater than one this is an evaluation-by-gradient-by-tau array.
quantgerr	asymptotic standard errors for gradients, when gradients=TRUE. If tau has length greater than one this is an evaluation-by-gradient-by-tau array.
tau	the quantile probability or probabilities computed

Book And Method Pointers

The conditional quantile target is the generalized inverse $q_\tau(x) = \inf\{y : F(y \mid x) \geq \tau\}$ of the conditional distribution. The standard errors and gradients described above are first-order delta-method quantities evaluated using the same selected conditional CDF, conditional density, bandwidths, kernels, and local-polynomial degree inherited from the supplied npcdistbw object.

For book-length derivations, see Li and Racine (2007), Chapter 6 *Conditional CDF and Quantile Estimation*, especially Sections 6.3-6.5, and Racine (2019), Chapter 4 *Conditional Probability Density and Cumulative Distribution Functions*. The quasi-inverse terminology follows Nelsen (2006).

Usage Issues

If you are using data of mixed types, then it is advisable to use the data.frame function to construct your input data and not cbind, since cbind will typically not work as intended on mixed data types and will coerce the data to the same type.

When reusing a bandwidth object without explicit training or evaluation data, the required data are recovered from object-retained state or from the stored call environment when available. This is a convenience for ordinary same-session use, not a guarantee that a saved object is self-contained. For reproducible replay after saveRDS() and readRDS() in a fresh R session, supply the required data arguments explicitly.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

Hall, P. and J.S. Racine and Q. Li (2004), "Cross-validation and the estimation of conditional probability densities," Journal of the American Statistical Association, 99, 1015-1026.

Koenker, R. W. and G.W. Bassett (1978), "Regression quantiles," Econometrica, 46, 33-50.

Koenker, R. (2005), *Quantile Regression,* Econometric Society Monograph Series, Cambridge University Press.

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

Li, Q. and J.S. Racine (2008), "Nonparametric estimation of conditional CDF and quantile functions with mixed categorical and continuous data," Journal of Business and Economic Statistics, 26, 423-434.

Li, Q. and J. Lin and J.S. Racine (2013), "Optimal Bandwidth Selection for Nonparametric Conditional Distribution and Quantile Functions", Journal of Business and Economic Statistics, 31, 57-65.

Nelsen, R.B. (2006), *An Introduction to Copulas*, Second Edition, Springer.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

np.kernels, np.options, plot, plot.np, **quantreg**

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

data("Italy")

## A quantile regression example

bw <- npcdistbw(gdp~ordered(year),data=Italy)

summary(bw)
```

```
model <- npqreg(bws=bw, tau=c(0.25, 0.50, 0.75))

summary(model)

## If a scalar tau was used first, additional quantiles can reuse the
## selected bandwidths without recomputing cross-validation. Use npqreg()
## when the additional fitted values are needed as an object, or plot()
## when graphical inspection is all that is desired.

model.med <- npqreg(bws=bw, tau=0.50)
model.more <- npqreg(bws=model.med$bws, tau=c(0.25, 0.50, 0.75))
plot(model.med, tau=c(0.25, 0.50, 0.75))

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

npquantile *Kernel Univariate Quantile Estimation*

Description

npquantile computes smooth quantiles from a univariate unconditional kernel cumulative distribution estimate given data and, optionally, a bandwidth specification i.e. a dbandwidth object using the bandwidth selection method of Li, Li and Racine (2017).

Usage

```
npquantile(x = NULL,
           tau = c(0.01,0.05,0.25,0.50,0.75,0.95,0.99),
           num.eval = 10000,
           bws = NULL,
           f = 1,
           ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the distribution object, data, and bandwidth controls used for quantile extraction.

`bws` an optional `dbandwidth` specification (if already computed avoid unnecessary computation inside `npquantile`). This must be set as a `dbandwidth` object returned from an invocation of `npudistbw`. If not provided `npudistbw` is invoked with optional arguments passed via `....`

`x` a univariate vector of type `numeric` containing sample realizations (training data) used to estimate the cumulative distribution (must be the same training data used to compute the bandwidth object `bws` passed in).

Evaluation And Quantile Controls: These arguments control the target quantile level, evaluation size, and distribution interpolation.

`f` an optional argument fed to `extendrange`. Defaults to `1`. See `?extendrange` for details.

`num.eval` an optional integer specifying the length of the grid on which the quasi-inverse is computed. Defaults to `10000`.

`tau` an optional vector containing the probabilities for quantile(s) to be estimated (must contain numbers in $[0, 1]$). Defaults to `c(0.01,0.05,0.25,0.50,0.75,0.95,0.99)`.

Additional Arguments: Further arguments are passed to bandwidth or distribution routines as needed.

`...` additional arguments supplied to specify the bandwidth type, kernel types, bandwidth selection methods, and so on. See `?npudistbw` for details.

Details

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, `plot` for plotting options, and `npRmpi.init` for interactive/cluster MPI startup. See `npRmpi.init` details for performance tradeoffs (message passing/startup mode) and the `inst/Rprofile` manual-broadcast template.

Typical usage is

```
## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.
npRmpi.init(nslaves=1)
set.seed(42)
x <- rchisq(100,df=10)
npquantile(x)
npRmpi.quit()
```

The quantile function q_τ is defined to be the left-continuous inverse of the distribution function $F(x)$, i.e. $q_\tau = \inf\{x : F(x) \geq \tau\}$.

A traditional estimator of q_τ is the τth sample quantile. However, these estimates suffer from lack of efficiency arising from variability of individual order statistics; see Sheather and Marron (1990) and Hyndman and Fan (1996) for methods that interpolate/smooth the order statistics, each of which discussed in the latter can be invoked through `quantile` via `type=j`, `j=1,...,9`.

The function `npquantile` implements a method for estimating smooth quantiles based on the quasi-inverse of a `npudist` object where $F(x)$ is replaced with its kernel estimator and bandwidth selection is that appropriate for such objects; see Definition 2.3.6, page 21, Nelsen 2006 for a definition of the quasi-inverse of $F(x)$.

For construction of the quasi-inverse we create a grid of evaluation points based on the function `extendrange` along with the sample quantiles themselves computed from invocation of `quantile`. The coarseness of the grid defined by `extendrange` (which has been passed the option `f=1`) is controlled by `num.eval`.

Note that for any value of τ less/greater than the smallest/largest value of $F(x)$ computed for the evaluation data (i.e. that outlined in the paragraph above), the quantile returned for such values is that associated with the smallest/largest value of $F(x)$, respectively.

Value

`npquantile` returns a vector of quantiles corresponding to `tau`.

Usage Issues

Cross-validated bandwidth selection is used by default (`npudistbw`). For large datasets this can be computationally demanding. In such cases one might instead consider a rule-of-thumb bandwidth (`bwmethod="normal-reference"`) or bounded kernels such as `ckertype="epanechnikov"`. The default `options(np.tree="auto")` uses kd-trees for eligible bounded-kernel continuous routes; `options(np.tree=TRUE)` remains available to force kd-tree use explicitly.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Cheng, M.-Y. and Sun, S. (2006), "Bandwidth selection for kernel quantile estimation," Journal of the Chinese Statistical Association, **44**, 271-295.

Hyndman, R.J. and Fan, Y. (1996), "Sample quantiles in statistical packages," American Statistician, **50**, 361-365.

Li, Q. and J.S. Racine (2017), "Smooth Unconditional Quantile Estimation," Manuscript.

Li, C. and H. Li and J.S. Racine (2017), "Cross-Validated Mixed Datatype Bandwidth Selection for Nonparametric Cumulative Distribution/Survivor Functions," Econometric Reviews, **36**, 970-987.

Nelsen, R.B. (2006), *An Introduction to Copulas,* Second Edition, Springer-Verlag.

Sheather, S. and J.S. Marron (1990), "Kernel quantile estimators," Journal of the American Statistical Association, Vol. 85, No. 410, 410-416.

Yang, S.-S. (1985), "A Smooth Nonparametric Estimator of a Quantile Function," Journal of the American Statistical Association, **80**, 1004-1011.

See Also

np.kernels, np.options, plot

quantile for various types of sample quantiles; ecdf for empirical distributions of which quantile is an inverse; boxplot.stats and fivenum for computing other versions of quartiles; qlogspline for logspline density quantiles; qkde for alternative kernel quantiles, etc.

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

set.seed(42)

## Simulate data from a chi-square distribution
df <- 50
x <- rchisq(100,df=df)

## Vector of quantiles desired
tau <- c(0.01,0.05,0.25,0.50,0.75,0.95,0.99)

## Compute kernel smoothed sample quantiles
q <- npquantile(x,tau)

q
```

```
## Compute sample quantiles using the default method in R (Type 7)
quantile(x,tau)

## True quantiles based on known distribution
qchisq(tau,df=df)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

npreg *Kernel Regression with Mixed Data Types*

Description

npreg computes a kernel regression estimate of a one (1) dimensional dependent variable on p-variate explanatory data, given a set of evaluation points, training points (consisting of explanatory data and dependent data), and a bandwidth specification using the method of Racine and Li (2004) and Li and Racine (2004). A bandwidth specification can be a rbandwidth object, or a bandwidth vector, bandwidth type and kernel type.

Usage

```
npreg(bws, ...)

## S3 method for class 'formula'
npreg(bws,
      data = NULL,
      newdata = NULL,
      y.eval = FALSE,
      ...)
```

```
## Default S3 method:
npreg(bws,
      txdat,
      tydat,
      nomad = FALSE,
      ...)

## S3 method for class 'rbandwidth'
npreg(bws,
      txdat = stop("training data 'txdat' missing"),
      tydat = stop("training data 'tydat' missing"),
      exdat,
      eydat,
      gradient.order = 1L,
      gradients = FALSE,
      residuals = FALSE,
      ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and training data.

`bws` a bandwidth specification. This can be set as a `rbandwidth` object returned from an invocation of `npregbw`, or as a vector of bandwidths, with each element i corresponding to the bandwidth for column i in `txdat`. If specified as a vector, then additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, and so on.

`data` an optional data frame, list or environment (or object coercible to a data frame by `as.data.frame`) containing the variables in the model. If not found in data, the variables are taken from `environment(bws)`, typically the environment from which `npregbw` was called.

`txdat` a p-variate data frame of explanatory data (training data) used to calculate the regression estimators. Defaults to the corresponding training data recoverable from the bandwidth object in the current session.

`tydat` a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of `txdat`. Defaults to the corresponding training data recoverable from the bandwidth object in the current session.

Local-Polynomial Degree And Bandwidth Search: This argument controls the recommended automatic local-polynomial NOMAD route, which jointly selects continuous polynomial degree and bandwidths when these are computed inside `npreg`.

`nomad` logical shortcut passed through to `npregbw` when bandwidths are computed inside `npreg`. When `TRUE`, the regression bandwidth route fills any missing values among `regtype`, `search.engine`, `degree.select`, `bernstein.basis`,

degree.min, degree.max, degree.verify, and bwtype with the recommended automatic local-polynomial degree-and-bandwidth NOMAD preset documented in npregbw. Additional NOMAD tuning arguments such as nomad.nmulti may also be supplied through ...; nmulti remains the outer restart count while nomad.nmulti controls inner native **crs** NOMAD API multistarts within each outer restart. After fitting, inspect fitbwsnomad.shortcut on the returned object fit to see the normalized shortcut metadata.

Evaluation Data And Returned Quantities: These arguments control where the regression is evaluated and which fitted quantities are returned.

exdat — a p-variate data frame of points on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by txdat.

eydat — a one (1) dimensional numeric or integer vector of the true values of the dependent variable. Optional, and used only to calculate the true errors.

gradient.order — for regtype="lp" with gradients=TRUE, a positive integer (or integer vector with one entry per continuous predictor) specifying derivative order(s). Defaults to 1L. Orders exceeding degree for a variable are returned as NA. Higher-order derivatives are available for continuous predictors when the requested order does not exceed the corresponding local-polynomial degree.

gradients — a logical value indicating that you want gradients computed and returned in the resulting npregression object. Defaults to FALSE. For regtype="lp", derivative components that are not defined (requested order exceeds the variable-specific polynomial degree) are returned as NA. For ordered and unordered predictors the gradient columns are first-order effects/contrasts where supported; gradient.order controls derivative order for continuous predictors.

newdata — An optional data frame in which to look for evaluation data. If omitted, the training data are used.

residuals — a logical value indicating that you want residuals computed and returned in the resulting npregression object. Defaults to FALSE.

y.eval — If newdata contains dependent data and y.eval = TRUE, npRmpi will compute goodness of fit statistics on these data and return them. Defaults to FALSE.

Additional Arguments: Further arguments are passed to npregbw when bandwidths are computed internally, or used to interpret a numeric bws vector.

... — additional arguments supplied to npregbw when npreg computes bandwidths internally, or arguments needed to interpret a numeric bws vector. This is where bandwidth selection controls such as bwmethod, bwtype, bwscaling, kernel/support controls such as ckertype, ckerorder, and ckerbound, categorical kernel controls such as ukertype and okertype, search controls such as nmulti and scale.factor.search.lower, and local-polynomial/NOMAD controls such as regtype, degree, basis, bernstein.basis, degree.select, and nomad.nmulti are supplied. See npregbw for the complete bandwidth-selection argument surface.

Details

Documentation guide: see npregbw for bandwidth selection and search controls, np.kernels for kernels, np.options for global options, plot, plot.np for plotting options, and npRmpi.init for interactive/cluster MPI startup. See npRmpi.init details for performance tradeoffs (message passing/startup mode) and the inst/Rprofile manual-broadcast template.

When bws is omitted, the formula and default methods call npregbw first and pass bandwidth-selection arguments from ... to that call. When bws is already an rbandwidth object, npreg estimates with the stored bandwidth metadata in that object.

Argument groups for bandwidth selection are documented on npregbw. The most common workflow is to choose data and bandwidth inputs first, then bandwidth criterion and representation, then kernel/support controls, and finally local-polynomial/NOMAD controls when using polynomial-adaptive fits.

For S3 plotting help, see plot.np. You can list available plot methods with methods("plot").

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

```
Usage 1: first compute the bandwidth object via npregbw and then
compute the conditional mean:

bw <- npregbw(y~x)
ghat <- npreg(bw)

Usage 2: alternatively, compute the bandwidth object indirectly:

ghat <- npreg(y~x)

Usage 3: modify the default kernel and order:

ghat <- npreg(y~x, ckertype="epanechnikov", ckerorder=4)

Usage 4: use the data frame interface rather than the formula
interface:

ghat <- npreg(tydat=y, txdat=x, ckertype="epanechnikov", ckerorder=4)
```

npreg implements a variety of methods for regression on multivariate (p-variate) data, the types of which are possibly continuous and/or discrete (unordered, ordered). The approach is based on Li and Racine (2003) who employ 'generalized product kernels' that admit a mix of continuous and discrete data types.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i, when estimating the density at the point x. Generalized nearest-neighbor bandwidths change with the point at which the density is estimated, x. Fixed bandwidths are constant over the support of x.

Data contained in the data frame txdat may be a mix of continuous (default), unordered discrete (to be specified in the data frame txdat using factor), and ordered discrete (to be specified in the

data frame `txdat` using `ordered`). Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see `npRmpi` for details).

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth order Gaussian and Epanechnikov kernels, and the uniform kernel. Unordered discrete data types use a variation on Aitchison and Aitken's (1976) kernel, while ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

When bandwidths are obtained with `regtype="lp"`, C-level `npreg` supports heterogeneous continuous polynomial degrees via `degree`. The `basis` selector currently supports `basis="glp"`, `"additive"`, and `"tensor"`. For continuous predictors with degree vector d, additive basis size is $1 + \sum_j d_j$, tensor basis size is $\prod_j (d_j + 1)$, and GLP uses admissible multi-indices α with $\alpha_j \leq d_j$ and $0 < \sum_j \alpha_j \leq \max_j d_j$ plus an intercept. The optional flag `bernstein.basis` controls basis construction: `FALSE` (default) uses raw local-polynomial powers, while `TRUE` uses a Bernstein/B-spline basis. The homogeneous degree-0 and degree-1 cases remain equivalent to `lc` and `ll`, respectively. For continuous predictors, `gradients=TRUE` returns the derivative order(s) requested by `gradient.order`; the associated `gerr` entries are standard errors for those same derivative orders. Both raw and Bernstein/B-spline bases report fitted values and derivatives on the original response/predictor scale. In mixed-data GLP settings, unordered and ordered predictors contribute first-order effects/contrasts rather than higher-order derivatives. When `npregbw(..., regtype="lp")` is used with `degree.select="manual"`, the degree vector remains fixed user input. When `degree.select != "manual"`, `npregbw` can jointly select polynomial degree and bandwidth using either the cached cell-search backend or the direct `search.engine="nomad"/"nomad+powell"` route described in `npregbw`; the latter follows Hall and Racine (2015). For practitioners who want that recommended route without spelling out all LP tuning arguments, `npreg(..., nomad=TRUE)` and `npregbw(..., nomad=TRUE)` expand missing settings to the same documented automatic-LP NOMAD preset. Explicit incompatible settings fail fast rather than being silently rewritten. The direct NOMAD backend is provided by the suggested package `crs`, so install `crs` before using `search.engine="nomad"`, `"nomad+powell"`, or `nomad=TRUE`. For `bernstein.basis=TRUE`, evaluation points for continuous predictors must lie within training support; use `bernstein.basis=FALSE` for extrapolation. For `regtype="ll"` and `regtype="lp"`, the training continuous design is checked for rank deficiency and extreme condition number before estimation proceeds.

The use of compactly supported kernels or the occurrence of small bandwidths can lead to numerical problems for the local linear estimator when computing the locally weighted least squares solution. To overcome this problem we rely on a form or 'ridging' proposed by Cheng, Hall, and Titterington (1997), modified so that we solve the problem pointwise rather than globally (i.e. only when it is needed).

Value

`npreg` returns a `npregression` object. The generic functions `fitted`, `residuals`, `se`, `predict`, and `gradients`, extract (or generate) estimated values, residuals, asymptotic standard errors on estimates, predictions, and gradients, respectively, from the returned object. Furthermore, the functions `summary` and `plot` support objects of this type. The returned object has the following components:

`eval`	evaluation points
`mean`	estimates of the regression function (conditional mean) at the evaluation points
`merr`	standard errors of the regression function estimates
`grad`	estimates of the gradients at each evaluation point

gerr	standard errors of the gradient estimates
resid	if residuals = TRUE, in-sample or out-of-sample residuals where appropriate (or possible)
R2	coefficient of determination (Doksum and Samarov (1995))
MSE	mean squared error
MAE	mean absolute error
MAPE	mean absolute percentage error
CORR	absolute value of Pearson's correlation coefficient
SIGN	fraction of observations where fitted and observed values agree in sign

Book And Method Pointers

The regression target is the conditional mean $m(x) = E[Y \mid X = x]$. Local-constant estimation can be written schematically as $\hat{m}(x) = \sum_i W_i(x) Y_i$, where the weights are normalized generalized product-kernel weights over the continuous, unordered, and ordered components of X. Local-polynomial regression replaces this scalar weighted average by a weighted least squares fit in powers of the continuous coordinates around x; the fitted intercept is the estimate of $m(x)$ and the fitted slope terms give the reported gradients where available.

For book-length derivations, see Li and Racine (2007), Chapter 2 *Regression*, especially Sections 2.1, 2.2, 2.4, and 2.5, and Chapter 4 *Kernel Estimation with Mixed Data*, especially Sections 4.2 and 4.4. The later workflow treatment is Racine (2019), Chapter 6 *Conditional Mean Function Estimation*, including the local polynomial and mixed-data marginal-effect discussion.

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

When reusing a bandwidth object without explicit training or evaluation data, the required data are recovered from object-retained state or from the stored call environment when available. This is a convenience for ordinary same-session use, not a guarantee that a saved object is self-contained. For reproducible replay after `saveRDS()` and `readRDS()` in a fresh R session, supply the required data arguments explicitly.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

Cheng, M.-Y. and P. Hall and D.M. Titterington (1997), "On the shrinkage of local linear curve estimators," Statistics and Computing, 7, 11-17.

Fan, J. and I. Gijbels (1996), *Local Polynomial Modelling and Its Applications,* Chapman and Hall.

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.

Hall, P. and Q. Li and J.S. Racine (2007), "Nonparametric estimation of regression functions in the presence of irrelevant regressors," The Review of Economics and Statistics, 89, 784-789.

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

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

Li, Q. and J.S. Racine (2004), "Cross-validated local linear nonparametric regression," Statistica Sinica, 14, 485-512.

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

Racine, J.S. and Q. Li (2004), "Nonparametric estimation of regression functions with both categorical and continuous data," Journal of Econometrics, 119, 99-130.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

np.kernels, np.options, plot, plot.np loess

Examples

```
## Not run:
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").
##
## R CMD check can set up a process environment where spawn-mode MPI
## teardown may terminate the check parent process. Keep this example
## fully runnable for users while skipping MPI spawn only in check.
force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
  npRmpi.init(nslaves=1)

  set.seed(42)

  n <- 250
```

```
  x <- runif(n)
  z1 <- rbinom(n,1,.5)
  z2 <- rbinom(n,1,.5)
  y <- cos(2*pi*x) + z1 + rnorm(n,sd=.25)
  z1 <- factor(z1)
  z2 <- factor(z2)

  bw <- npregbw(y~x+z1+z2,
                regtype="lc",
                bwmethod="cv.ls",
                nmulti=1)

  summary(bw)

  model <- npreg(bws=bw,
                 gradients=FALSE)

  summary(model)

  npRmpi.quit()
  ## npRmpi.quit(force=TRUE)
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

npregbw *Kernel Regression Bandwidth Selection with Mixed Data Types*

Description

npregbw computes a bandwidth object for a p-variate kernel regression estimator defined over mixed continuous and discrete (unordered, ordered) data using expected Kullback-Leibler cross-validation, or least-squares cross validation using the method of Racine and Li (2004) and Li and Racine (2004).

Usage

```
npregbw(...)

## S3 method for class 'formula'
npregbw(formula,
        data,
        subset,
        na.action,
        call,
        ...)
```

```
## Default S3 method:
npregbw(xdat = stop("invoked without data 'xdat'"),
        ydat = stop("invoked without data 'ydat'"),
        bws,
        bandwidth.compute = TRUE,
        basis,
        bernstein.basis,
        bwmethod,
        bwscaling,
        bwtype,
        cfac.dir,
        scale.factor.init,
        ckerbound,
        ckerlb,
        ckerorder,
        ckertype,
        ckerub,
        degree,
        degree.select = c("manual", "coordinate", "exhaustive"),
        search.engine = c("nomad+powell", "cell", "nomad"),
        nomad = FALSE,
        nomad.nmulti = 0L,
        degree.min = NULL,
        degree.max = NULL,
        degree.start = NULL,
        degree.restarts = 0L,
        degree.max.cycles = 20L,
        degree.verify = FALSE,
        dfac.dir,
        dfac.init,
        dfc.dir,
        ftol,
        scale.factor.init.upper,
        hbd.dir,
        hbd.init,
        initc.dir,
        initd.dir,
        invalid.penalty = c("baseline","dbmax"),
        itmax,
        lbc.dir,
        scale.factor.init.lower,
        lbd.dir,
        lbd.init,
        nmulti,
        okertype,
        penalty.multiplier = 10,
        regtype,
```

```
        nomad.remin = FALSE,
        powell.remin,
        bwsolver = c("powell", "mads", "mads+powell"),
        scale.init.categorical.sample,
        scale.factor.search.lower = NULL,
        small,
        tol,
        transform.bounds = FALSE,
        ukertype,
        ...,
        nomad.opts = list())

## S3 method for class 'rbandwidth'
npregbw(xdat = stop("invoked without data 'xdat'"),
        ydat = stop("invoked without data 'ydat'"),
        bws,
        bandwidth.compute = TRUE,
        cfac.dir = 2.5*(3.0-sqrt(5)),
        scale.factor.init = 0.5,
        dfac.dir = 0.25*(3.0-sqrt(5)),
        dfac.init = 0.375,
        dfc.dir = 3,
        ftol = 1.490116e-07,
        scale.factor.init.upper = 2.0,
        hbd.dir = 1,
        hbd.init = 0.9,
        initc.dir = 1.0,
        initd.dir = 1.0,
        invalid.penalty = c("baseline","dbmax"),
        itmax = 10000,
        lbc.dir = 0.5,
        scale.factor.init.lower = 0.1,
        lbd.dir = 0.1,
        lbd.init = 0.1,
        nmulti,
        penalty.multiplier = 10,
        powell.remin = TRUE,
        bwsolver = c("powell", "mads", "mads+powell"),
        scale.init.categorical.sample = FALSE,
        scale.factor.search.lower = NULL,
        small = 1.490116e-05,
        tol = 1.490116e-04,
        transform.bounds = FALSE,
        ...,
        nomad.opts = list())
```

Arguments

Data And Bandwidth Inputs: These arguments identify the data and whether bandwidths are supplied or computed.

`xdat` a p-variate data frame of regressors on which bandwidth selection will be performed. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.

`ydat` a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of `xdat`.

`bws` a bandwidth specification. This can be set as a `rbandwidth` object returned from a previous invocation, or as a vector of bandwidths, with each element i corresponding to the bandwidth for column i in `xdat`. In either case, the bandwidth supplied will serve as a starting point in the numerical search for optimal bandwidths. If specified as a vector, then additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, selection methods, and so on. This can be left unset.

`bandwidth.compute` a logical value which specifies whether to do a numerical search for bandwidths or not. If set to `FALSE`, a `rbandwidth` object will be returned with bandwidths set to those specified in `bws`. Defaults to `TRUE`.

Local-Polynomial And NOMAD Controls: These arguments control regression type, local-polynomial specification, and optional automatic degree search.

`regtype` a character string specifying which type of kernel regression estimator to use. `lc` specifies a local-constant estimator (Nadaraya-Watson) and `ll` specifies a local-linear estimator. `lp` specifies a local polynomial estimator with polynomial degree(s) given by `degree` for continuous predictors, or selected automatically when `degree.select != "manual"`. `ll` and positive-degree `lp` require at least one continuous predictor; for categorical-only predictors use `lc` or `lp` with `degree=0`. Defaults to `lc`.

`basis` basis selector relevant only when `regtype="lp"`. Supported values are `"glp"`, `"additive"`, and `"tensor"`. Let d_j denote the degree for continuous predictor j and q the number of continuous predictors. With one segment per predictor (no internal knots), basis dimensions are: $1 + \sum_{j=1}^{q} d_j$ for additive, $\prod_{j=1}^{q}(d_j + 1)$ for tensor, and $1 + |\{\alpha : \alpha_j \leq d_j,\ 0 < \sum_j \alpha_j \leq \max_j d_j\}|$ for generalized local-polynomial (GLP) basis construction.

`bernstein.basis` logical flag relevant only when `regtype="lp"`. If `FALSE` (default), the GLP basis uses raw local-polynomial powers (stable for extrapolation). If `TRUE`, a Bernstein (B-spline) basis is used for continuous predictors. For `bernstein.basis=TRUE`, prediction/evaluation points must lie within the training support of each continuous predictor. For automatic degree search, if `bernstein.basis` is not explicitly supplied, the search route defaults to `TRUE` for numerical stability. Explicit `bernstein.basis=FALSE` is honored, but raw-polynomial search can be poorly conditioned at higher degrees. For `regtype="ll"` and `regtype="lp"`, a pre-optimization design-conditioning check is performed on the training continuous

design: rank deficiency triggers an error, and large condition number ($\kappa(B)$) triggers warning/error thresholds to avoid unstable optimization dominated by ridging.

`degree` a user-supplied vector of fixed polynomial degrees for the continuous predictors (exactly one degree per continuous predictor), relevant only when `regtype="lp"`. When `degree.select="manual"`, this must be supplied explicitly. Entries must be non-negative integers in `[0,12]`. With no continuous predictors, `regtype="lp"` is only admissible when `degree=0`, the local-constant equivalent. Bandwidth optimization treats this vector as fixed input and optimizes only bandwidths.

`degree.select` character string controlling local-polynomial degree handling when `regtype="lp"`. `"manual"` (default) treats `degree` as fixed. `"coordinate"` performs cached coordinate-wise search over admissible degree vectors. `"exhaustive"` evaluates the full admissible degree grid when `search.engine="cell"`. For NOMAD-based search engines, any non-`"manual"` value requests direct joint search over degree and bandwidth coordinates.

`search.engine` character string controlling the automatic local-polynomial search backend when `regtype="lp"` and `degree.select != "manual"`. `"nomad+powell"` (default) performs direct joint mixed discrete/continuous search over fixed bandwidths and the degree vector using the native **crs** NOMAD C API, followed by one Powell hot start from the NOMAD solution. `"nomad"` performs the direct joint NOMAD search without the Powell refinement. `"cell"` uses the legacy profiled degree-grid search built from repeated fixed-degree bandwidth solves. NOMAD-based search supports `bwtype="fixed"`, `bwtype="generalized_nn"`, and `bwtype="adaptive_nn"`; it currently requires `degree.verify=FALSE` and the suggested package **crs** to be installed.

`nomad` logical or character shortcut for the recommended automatic local-polynomial NOMAD route. When `TRUE`, any missing values among `regtype`, `search.engine`, `degree.select`, `bernstein.basis`, `degree.min`, `degree.max`, `degree.verify`, and `bwtype` are filled with `regtype="lp"`, `search.engine="nomad+powell"`, `degree.select="coordinate"`, `bernstein.basis=TRUE`, `degree.min=0L`, `degree.max=10L`, `degree.verify=FALSE`, and `bwtype="fixed"`. Explicit incompatible settings error immediately; in particular, explicit `nomad=TRUE` currently requires `regtype="lp"`, `bwtype="fixed"`, automatic degree search, no explicit `degree`, at least one continuous predictor, and `search.engine %in% c("nomad", "nomad+powell")`. This shortcut does not change the meaning of `nmulti` or `nomad.nmulti`: `nmulti` remains the outer restart count, while `nomad.nmulti` controls inner native **crs** NOMAD API multistarts within each outer restart. Returned bandwidth objects retain this normalized preset metadata in `bw$nomad.shortcut` for a returned object `bw`; when available, `nomad.time` and `powell.time` record the direct-search and Powell-polish timing components.

`nomad.nmulti` non-negative integer controlling the inner the native **crs** NOMAD C API multistart count used within each outer NOMAD restart when `regtype="lp"` and automatic degree search uses `search.engine="nomad"` or `"nomad+powell"`. Defaults to `0L`, which preserves the current one-start-per- restart behavior. This does not replace `nmulti`: `nmulti` controls outer restarts, while `nomad.nmulti` controls inner NOMAD multistarts within each outer restart.

`nomad.remin` logical flag controlling the optional second NOMAD hot start. When `TRUE`,

NOMAD is restarted once from the best full candidate found, including both bandwidth and degree coordinates. Defaults to `FALSE`; current simulation evidence favors the one-pass NOMAD default for routine use, while leaving this switch available for sensitivity checks.

`degree.min` optional scalar or integer vector giving lower bounds for automatic degree search when `degree.select != "manual"`. If scalar, the value is recycled over continuous predictors.

`degree.max` optional scalar or integer vector giving upper bounds for automatic degree search when `degree.select != "manual"`. If scalar, the value is recycled over continuous predictors.

`degree.start` optional starting degree vector for automatic degree search when `degree.select="coordinate"`. If omitted, cell-based search starts from the degree-zero local-constant baseline on the continuous predictors, while NOMAD-based search starts from a clipped degree-one vector on the searchable continuous predictors. For NOMAD multi-starts, later restart starts are generated reproducibly from a conservative proposal box and screened using `dim_basis()` so that the initial basis dimension remains well below the training-sample limit. This avoids wasting starts on flat penalty or heavily ridged designs while leaving the full user requested degree search region unchanged.

`degree.restarts` non-negative integer giving the number of additional deterministic restarts used by coordinate search. Ignored for `degree.select="manual"` and `"exhaustive"`.

`degree.max.cycles` positive integer giving the maximum number of coordinate-search sweeps over the continuous-predictor degree vector. Ignored for `degree.select="manual"` and `"exhaustive"`.

`degree.verify` logical value indicating whether a coordinate-search solution should be exhaustively verified over the admissible degree grid after the heuristic phase completes. Available only for `search.engine="cell"`.

Bandwidth Criterion And Representation: These arguments choose the selection criterion and the way continuous bandwidths are represented.

`bwmethod` which method to use to select bandwidths. `cv.aic` specifies expected Kullback-Leibler cross-validation (Hurvich, Simonoff, and Tsai (1998)), and `cv.ls` specifies least-squares cross-validation. Defaults to `cv.ls`.

`bwscaling` a logical value that when set to `TRUE` the supplied bandwidths are interpreted as 'scale factors' (c_j), otherwise when the value is `FALSE` they are interpreted as 'raw bandwidths' (h_j for continuous data types, λ_j for discrete data types). For continuous data types, c_j and h_j are related by the formula $h_j = c_j \sigma_j n^{-1/(2P+l)}$, where σ_j is an adaptive measure of spread of continuous variable j defined as min(standard deviation, mean absolute deviation/1.4826, interquartile range/1.349), n the number of observations, P the order of the kernel, and l the number of continuous variables. For discrete data types, c_j and h_j are related by the formula $h_j = c_j n^{-2/(2P+l)}$, where here j denotes discrete variable j. Defaults to `FALSE`.

`bwtype` character string used for the continuous variable bandwidth type, specifying the type of bandwidth to compute and return in the `bandwidth` object. Defaults to

fixed. Option summary:
fixed: compute fixed bandwidths
generalized_nn: compute generalized nearest neighbors
adaptive_nn: compute adaptive nearest neighbors

Search Initialization, Kernels, And Support: These controls set numerical search starts, kernel choices, and support bounds.

`cfac.dir` stretch factor for direction set search for Powell's algorithm for `numeric` variables. See Details

`scale.factor.init` non-random initial scale factor for `numeric` variables for Powell's algorithm. If `scale.factor.search.lower` is larger, the effective first search start is raised to that floor. See Details

`ckerbound` character string controlling continuous-kernel support handling. Can be set as `none` (default kernel on full support), `range` (use sample min/max), or `fixed` (use `ckerlb`/`ckerub`). The bounded-kernel route reuses the selected continuous kernel and renormalizes it on the chosen support; see `np.kernels`.

`ckerlb` numeric scalar/vector of lower bounds for continuous variables used when `ckerbound="fixed"`. Must satisfy lower-bound validity for each continuous variable (e.g., `<= min(variable)`). Use `-Inf` for unbounded below. See `np.kernels` for bounded-kernel normalization details.

`ckerorder` numeric value specifying kernel order (one of `(2,4,6,8)`). Kernel order specified along with a `uniform` continuous kernel type will be ignored. Defaults to `2`.

`ckertype` character string used to specify the continuous kernel type. Can be set as `gaussian`, `epanechnikov`, or `uniform`. Defaults to `gaussian`.

`ckerub` numeric scalar/vector of upper bounds for continuous variables used when `ckerbound="fixed"`. Must satisfy upper-bound validity for each continuous variable (e.g., `>= max(variable)`). Use `Inf` for unbounded above. See `np.kernels` for bounded-kernel normalization details.

`dfac.dir` stretch factor for direction set search for Powell's algorithm for categorical variables. See Details

`dfac.init` non-random initial values for scale factors for categorical variables for Powell's algorithm. See Details

`dfc.dir` chi-square degrees of freedom for direction set search for Powell's algorithm for `numeric` variables. See Details

`ftol` fractional tolerance on the value of the cross-validation function evaluated at located minima (of order the machine precision or perhaps slightly larger so as not to be diddled by roundoff). Defaults to `1.490116e-07` (1.0e+01*sqrt(.Machine$double.eps)).

`scale.factor.init.upper` upper endpoint for random scale-factor starts for `numeric` variables for Powell's algorithm. It must be no smaller than `max(scale.factor.init.lower, scale.factor.search.lower)`. See Details

`hbd.dir` upper bound for direction set search for Powell's algorithm for categorical variables. See Details

`hbd.init` upper bound for scale factors for categorical variables for Powell's algorithm. See Details

`initc.dir` initial non-random values for direction set search for Powell's algorithm for `numeric` variables. See Details

`initd.dir` initial non-random values for direction set search for Powell's algorithm for categorical variables. See Details

`invalid.penalty`
a character string specifying the penalty used when the optimizer encounters invalid bandwidths. `"baseline"` returns a finite penalty based on a baseline objective; `"dbmax"` returns `DBL\_MAX`. Defaults to `"baseline"`.

`itmax` integer number of iterations before failure in the numerical optimization routine. Defaults to `10000`.

`lbc.dir` lower bound for direction set search for Powell's algorithm for `numeric` variables. See Details

`scale.factor.init.lower`
lower endpoint for random scale-factor starts for `numeric` variables for Powell's algorithm. The effective lower endpoint is `max(scale.factor.init.lower, scale.factor.search.lower)`. See Details

`lbd.dir` lower bound for direction set search for Powell's algorithm for categorical variables. See Details

`lbd.init` lower bound for scale factors for categorical variables for Powell's algorithm. See Details

`nmulti` integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points. Defaults to `min(2,ncol(xdat))`.

`nomad.opts` optional named list of native NOMAD parameters forwarded to the **crs** NOMAD C API for MADS/NOMAD-backed bandwidth searches. `NULL` is treated as no extra NOMAD options. Non-list values and unsupported cache-off callback settings are rejected before solver entry.

`okertype` character string used to specify the ordered categorical kernel type. Can be set as `wangvanryzin`, `liracine`, or `racineliyan`. Defaults to `liracine`.

`penalty.multiplier`
a numeric multiplier applied to the baseline penalty when `invalid.penalty="baseline"`. Defaults to `10`.

`powell.remin` logical flag controlling Powell restart-from-minimum behavior. For ordinary fixed-degree Powell-style search, `TRUE` restarts the local search from the located minimum. For `search.engine="nomad+powell"`, this controls only the final Powell bandwidth-polish step. The default is `TRUE` for ordinary Powell routes and `FALSE` for the Powell polish after NOMAD unless explicitly supplied.

`bwsolver` bandwidth optimizer for fixed-degree searches. The default `"powell"` preserves the historical Powell route. `"mads"` uses the native **crs** NOMAD C API over the bandwidth coordinates, and `"mads+powell"` applies one Powell hot start from the MADS solution. Use `search.engine` rather than `bwsolver` for automatic degree search. Advanced native NOMAD parameters may be supplied through the explicit `nomad.opts` argument; invalid or unsupported parameters are rejected by NOMAD rather than silently ignored.

`scale.init.categorical.sample`
: a logical value that when set to `TRUE` scales `lbd.dir`, `hbd.dir`, `dfac.dir`, and `initd.dir` by $n^{-2/(2P+l)}$, n the number of observations, P the order of the kernel, and l the number of `numeric` variables. See Details

`scale.factor.search.lower`
: an optional nonnegative scalar controlling the hard lower bound used for continuous fixed-bandwidth search candidates. When omitted, the default coefficient `0.1` is used. Larger values impose a stricter search floor. This argument affects search admissibility and search-start sanitization only; explicit bandwidths supplied for storage when `bandwidth.compute = FALSE` are not rewritten. Final fixed-bandwidth search candidates must also have a finite valid raw objective value.

`small`
: a small number used to bracket a minimum (it is hopeless to ask for a bracketing interval of width less than sqrt(epsilon) times its central value, a fractional width of only about 10-04 (single precision) or 3x10-8 (double precision)). Defaults to `small = 1.490116e-05 (1.0e+03*sqrt(.Machine$double.eps))`.

`tol`
: tolerance on the position of located minima of the cross-validation function (tol should generally be no smaller than the square root of your machine's floating point precision). Defaults to `1.490116e-04 (1.0e+04*sqrt(.Machine$double.eps))`.

`transform.bounds`
: a logical value that when set to `TRUE` applies an internal transformation that maps the unconstrained search to the feasible bandwidth domain. Defaults to `FALSE`.

`ukertype`
: character string used to specify the unordered categorical kernel type. Can be set as `aitchisonaitken` or `liracine`. Defaults to `aitchisonaitken`.

Formula Interface: These arguments are used by the formula method and are normally supplied by the top-level call.

`formula`
: a symbolic description of variables on which bandwidth selection is to be performed. The details of constructing a formula are described below.

`data`
: an optional data frame, list or environment (or object coercible to a data frame by `as.data.frame`) containing the variables in the model. If not found in data, the variables are taken from `environment(formula)`, typically the environment from which the function is called.

`subset`
: an optional vector specifying a subset of observations to be used in the fitting process.

`na.action`
: a function which indicates what should happen when the data contain `NA`s. The default is set by the `na.action` setting of options, and is `na.fail` if that is unset. The (recommended) default is `na.omit`.

`call`
: the original function call. This is passed internally by `npRmpi` when a bandwidth search has been implied by a call to another function. It is not recommended that the user set this.

Additional Arguments: These arguments collect remaining controls passed through S3 methods.

`...`
: additional arguments supplied to specify the bandwidth type, kernel types, selection methods, and so on, detailed below.

Details

The scale.factor.* controls are dimensionless search controls. The package converts scale factors to bandwidths using the estimator-specific scaling already encoded in the bandwidth object, including the kernel order and the number of continuous variables relevant for that estimator. Users should not pre-multiply these controls by sample-size or standard-deviation factors. scale.factor.init controls the deterministic first search start, scale.factor.init.lower and scale.factor.init.upper control the random multistart interval, and scale.factor.search.lower is the lower admissibility bound for continuous fixed-bandwidth search candidates. Categorical search-start controls such as dfac.init, lbd.init, and hbd.init have separate semantics and are not affected by scale.factor.search.lower.

Documentation guide: see np.kernels for kernels, np.options for global options, plot, plot.np for plotting options, and npRmpi.init for interactive/cluster MPI startup. See npRmpi.init details for performance tradeoffs (message passing/startup mode) and the inst/Rprofile manual-broadcast template.

The bandwidth-selection argument surface is easiest to read by decision group: data and existing bandwidth inputs; local-polynomial/NOMAD controls when polynomial-adaptive regression is requested; bandwidth criterion and representation; continuous kernel and support controls beginning with cker*; categorical kernel controls ukertype and okertype; and numerical search initialization, tolerances, and feasibility controls. Users who call npreg without a bandwidth object can pass these same bandwidth-selection controls through that function's ....

For S3 plotting help, see plot.np. You can list available plot methods with methods("plot").

npregbw implements a variety of methods for choosing bandwidths for multivariate (p-variate) regression data defined over a set of possibly continuous and/or discrete (unordered, ordered) data. The approach is based on Li and Racine (2003) who employ 'generalized product kernels' that admit a mix of continuous and discrete data types.

The cross-validation methods employ multivariate numerical search algorithms. For fixed-degree local-constant/local-linear regression, and for local-polynomial regression with degree.select="manual", the bandwidth search uses multidimensional Powell direction-set optimization.

Bandwidths can (and will) differ for each variable which is, of course, desirable.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i, when estimating the density at the point x. Generalized nearest-neighbor bandwidths change with the point at which the density is estimated, x. Fixed bandwidths are constant over the support of x.

npregbw may be invoked *either* with a formula-like symbolic description of variables on which bandwidth selection is to be performed *or* through a simpler interface whereby data is passed directly to the function via the xdat and ydat parameters. Use of these two interfaces is **mutually exclusive**.

Data contained in the data frame xdat may be a mix of continuous (default), unordered discrete (to be specified in the data frame xdat using factor), and ordered discrete (to be specified in the data frame xdat using ordered). Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see npRmpi for details).

Data for which bandwidths are to be estimated may be specified symbolically. A typical description has the form dependent data ~ explanatory data, where dependent data is a univariate

response, and `explanatory data` is a series of variables specified by name, separated by the separation character '+'. For example, `y1 ~ x1 + x2` specifies that the bandwidths for the regression of response `y1` and nonparametric regressors `x1` and `x2` are to be estimated. See below for further examples.

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth order Gaussian and Epanechnikov kernels, and the uniform kernel. Unordered discrete data types use a variation on Aitchison and Aitken's (1976) kernel, while ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

When `regtype="lp"` and `degree.select != "manual"`, `npregbw` can jointly determine the continuous-predictor degree vector and bandwidth coordinates. With `search.engine="cell"`, the objective is profiled over the degree grid using cached coordinate-wise or exhaustive search together with the existing fixed-degree bandwidth optimizer. With `search.engine="nomad"` or `"nomad+powell"`, the package instead evaluates the cross-validation criterion directly over the joint space of fixed bandwidths and polynomial degrees using the native **crs** NOMAD C API. `"nomad+powell"` then performs one Powell hot start from the NOMAD solution and retains the better of the direct NOMAD and polished solutions. This direct joint-search route follows the polynomial-adaptive cross-validation rationale of Hall and Racine (2015). When `bernstein.basis` is not explicitly supplied, the automatic search route defaults to `bernstein.basis=TRUE` for numerical stability; explicit `bernstein.basis=FALSE` is honored but can be poorly conditioned at higher degrees. NOMAD multistarts are initialized more conservatively than the full degree search box: start 1 is the user-supplied degree/bandwidth vector when provided and otherwise a clipped degree-one vector, while later starts are reproducible random draws from a reduced degree proposal box whose candidates are screened using `dim_basis()`. This heuristic is used only to obtain feasible, numerically safer, and quicker initial evaluations; it does not restrict the admissible degree region searched by NOMAD. The direct NOMAD backend is provided by the suggested package `crs`, so install `crs` before using `search.engine="nomad"`, `"nomad+powell"`, `nomad=TRUE`, or `nomad="auto"`.

The use of compactly supported kernels or the occurrence of small bandwidths during cross-validation can lead to numerical problems for the local linear estimator when computing the locally weighted least squares solution. To overcome this problem we rely on a form or 'ridging' proposed by Cheng, Hall, and Titterington (1997), modified so that we solve the problem pointwise rather than globally (i.e. only when it is needed).

The optimizer invoked for search is Powell's conjugate direction method which requires the setting of (non-random) initial values and search directions for bandwidths, and, when restarting, random values for successive invocations. Bandwidths for `numeric` variables are scaled by robust measures of spread, the sample size, and the number of `numeric` variables where appropriate. Two sets of parameters for bandwidths for `numeric` can be modified, those for initial values for the parameters themselves, and those for the directions taken (Powell's algorithm does not involve explicit computation of the function's gradient). The default values are set by considering search performance for a variety of difficult test cases and simulated cases. We highly recommend restarting search a large number of times to avoid the presence of local minima (achieved by modifying `nmulti`). Further refinement for difficult cases can be achieved by modifying these sets of parameters. However, these parameters are intended more for the authors of the package to enable 'tuning' for various methods rather than for the user themselves.

Setting `nomad=TRUE` is a convenience preset for this automatic LP route, not a generic optimizer alias. For regression it expands any missing values to the equivalent long-form call

```
npregbw(...,
```

```
regtype = "lp",
search.engine = "nomad+powell",
degree.select = "coordinate",
bernstein.basis = TRUE,
degree.min = 0L,
degree.max = 10L,
degree.verify = FALSE,
bwtype = "fixed")
```

Compatible explicit tuning arguments are respected. Incompatible explicit settings fail fast so the shortcut never silently changes user-selected semantics. The character value `nomad="auto"` applies the same LP shortcut but leaves `search.engine` and `degree.select` eligible for data-driven resolution when both were not supplied explicitly: scalar continuous degree searches use the exhaustive degree-grid route, while higher-dimensional degree searches keep the NOMAD/Powell route. Explicit `search.engine` or `degree.select` choices are honored. When the direct NOMAD route is active, `nmulti` controls the package-level outer restart count while `nomad.nmulti` controls the inner native **crs** NOMAD API multistart count used within each outer restart. The default `nomad.nmulti=0L` preserves the current single-start inner NOMAD behavior. Advanced native NOMAD parameters may be supplied through the explicit `nomad.opts` argument; invalid or unsupported parameters are rejected by NOMAD rather than silently ignored. Routes that evaluate R callbacks require serial callback evaluation and reject parallel callback settings such as `NB_THREADS_PARALLEL_EVAL > 1`.

Value

`npregbw` returns a `rbandwidth` object, with the following components:

`bw`	bandwidth(s), scale factor(s) or nearest neighbours for the data, `xdat`
`fval`	objective function value at minimum

if `bwtype` is set to `fixed`, an object containing bandwidths (or scale factors if `bwscaling = TRUE`) is returned. If it is set to `generalized_nn` or `adaptive_nn`, then instead the kth nearest neighbors are returned for the continuous variables while the discrete kernel bandwidths are returned for the discrete variables. Bandwidths are stored under the component name `bw`, with each element i corresponding to column i of input data `xdat`.

The functions `predict`, `summary`, and `plot` support objects of this class.

Book And Method Pointers

`npregbw` selects bandwidths and, when requested, local-polynomial degree/search metadata for estimating the conditional mean $m(x) = E[Y \mid X = x]$. For the estimator target and local-polynomial interpretation, see `npreg`.

For book-length derivations, see Li and Racine (2007), Chapter 2 *Regression*, especially Sections 2.2, 2.4, and 2.5, and Chapter 4 *Kernel Estimation with Mixed Data*, especially Sections 4.2 and 4.4. The later workflow treatment is Racine (2019), Chapter 6 *Conditional Mean Function Estimation*.

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

Caution: multivariate data-driven bandwidth selection methods are, by their nature, *computationally intensive*. Virtually all methods require dropping the ith observation from the data set, computing an object, repeating this for all observations in the sample, then averaging each of these leave-one-out estimates for a *given* value of the bandwidth vector, and only then repeating this a large number of times in order to conduct multivariate numerical minimization/maximization. Furthermore, due to the potential for local minima/maxima, *restarting this procedure a large number of times may often be necessary*. This can be frustrating for users possessing large datasets. For exploratory purposes, you may wish to override the default search tolerances, say, setting ftol=.01 and tol=.01 and conduct multistarting (the default is to restart min(2, ncol(xdat)) times) as is done for a number of examples. Once the procedure terminates, you can restart search with default tolerances using those bandwidths obtained from the less rigorous search (i.e., set `bws=bw` on subsequent calls to this routine where `bw` is the initial bandwidth object). A version of this package using the `Rmpi` wrapper is under development that allows one to deploy this software in a clustered computing environment to facilitate computation involving large datasets.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

Cheng, M.-Y. and P. Hall and D.M. Titterington (1997), "On the shrinkage of local linear curve estimators," Statistics and Computing, 7, 11-17.

Fan, J. and I. Gijbels (1996), *Local Polynomial Modelling and Its Applications,* Chapman and Hall.

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

Hall, P. and Q. Li and J.S. Racine (2007), "Nonparametric estimation of regression functions in the presence of irrelevant regressors," The Review of Economics and Statistics, 89, 784-789.

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.

Li, Q. and J.S. Racine (2004), "Cross-validated local linear nonparametric regression," Statistica Sinica, 14, 485-512.

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

Racine, J.S. and Q. Li (2004), "Nonparametric estimation of regression functions with both categorical and continuous data," Journal of Econometrics, 119, 99-130.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

np.kernels, np.options, plot, plot.np npreg

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").


force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

set.seed(42)

n <- 250

x <- runif(n)
z1 <- rbinom(n,1,.5)
z2 <- rbinom(n,1,.5)
y <- cos(2*pi*x) + z1 + rnorm(n,sd=.25)
z1 <- factor(z1)
z2 <- factor(z2)

bw <- npregbw(y~x+z1+z2,
              regtype="lc",
              bwmethod="cv.ls",
              nmulti=1)

summary(bw)

npRmpi.quit()
## npRmpi.quit(force=TRUE)
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}
```

```
## End(Not run)
```

---

npreghat *Nonparametric Regression Hat Operator*

Description

Constructs nonparametric regression hat operators for npreg-compatible bandwidth objects. The returned operator $H^{(s)}$ maps responses to fitted values or derivative estimates via $H^{(s)}y$.

Usage

```
npreghat(bws, ...)

## S3 method for class 'formula'
npreghat(bws,
         data = NULL,
         newdata = NULL,
         ...)


## S3 method for class 'rbandwidth'
npreghat(bws,
         txdat = stop("training data 'txdat' missing"),
         exdat, y = NULL,
         output = c("matrix", "apply", "constraint"),
         basis = NULL,
         bernstein.basis = NULL,
         degree = NULL,
         deriv = NULL,
         leave.one.out = FALSE,
         ridge = 0,
         s = NULL,
         ...)

## S3 method for class 'npregression'
npreghat(bws, txdat, y, ...)

## S3 method for class 'npreghat'
predict(object,
        newdata = NULL,
        y = NULL,
        output = c("matrix", "apply", "constraint"),
        s = attr(object, "s"),
        leave.one.out = attr(object, "leave.one.out"),
        deriv = NULL,
        ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the fitted bandwidth object, formula/data interface, training data, and evaluation data.

`bws`	An object of class `rbandwidth` or `npregression`.
`data`	A data frame used with the formula interface.
`exdat`	Optional evaluation predictors.
`newdata`	Optional evaluation data for formula and predict methods.
`txdat`	Training predictors.

Local-Polynomial Controls: These arguments control local-polynomial basis, degree, derivatives, leave-one-out behavior, and ridge stabilization.

`basis`	Local polynomial basis: `"glp"`, `"additive"`, or `"tensor"`.
`bernstein.basis`	Logical; use Bernstein basis for LP terms.
`degree`	Optional local polynomial degree vector override (LP path).
`deriv`	Convenience alias for `s`.
`leave.one.out`	Logical; if `TRUE`, compute in-sample leave-one-out hat weights. This cannot be combined with explicit `exdat`/`newdata`.
`ridge`	Base diagonal regularization used when local systems are ill-conditioned. The ridge sequence starts at `0` (no regularization) and then increments by `1/n.train` as needed for stable solves.
`s`	Derivative multi-index over continuous predictors.

Method Objects: This argument identifies a fitted hat-operator object supplied to an S3 method.

`object`	An object returned by `npreghat`.

Operator Output: These arguments control whether the operator is returned as a matrix, applied directly, or returned as a quadratic-programming constraint design matrix.

`output`	Either `"matrix"` for the hat matrix, `"apply"` for direct application to `y`, or `"constraint"` for the row-weighted transpose `t(H) * y`.
`y`	Optional response vector or matrix for apply mode. For `output = "constraint"`, `y` is required and must be a vector or one-column object.

Additional Arguments: Further arguments are passed to methods.

`...`	Additional arguments passed to methods.

Details

For `output = "matrix"`, the return value is a matrix with class `c("npreghat", "matrix")` so it can be used directly in matrix products, e.g. `H %*% y`. Attributes on the matrix store metadata used by `predict.npreghat`.

For `output = "apply"`, the function returns `H^{(s)} y` directly and accepts matrix right-hand sides for one-shot bootstrap-style calculations.

For `output = "constraint"`, the function returns `t(H^{(s)}) * y`, the row-weighted transpose commonly used as the design matrix in shape-constrained quadratic-programming examples. This is a convenience route that is exactly equivalent to obtaining `H` with `output = "matrix"` and then computing `t(H) * y`; it does not solve a constrained estimation problem.

Value

Either a hat matrix (class `"npreghat"`) or the applied result `H^{(s)} y`, or the constraint design matrix `t(H^{(s)}) * y`, depending on `output`.

Examples

```
## Not run:
npRmpi.init(nslaves = 1)
data(cps71)
bw <- npregbw(xdat = cps71$age, ydat = cps71$logwage,
              regtype = "ll", bandwidth.compute = FALSE, bws = 1.0)
H <- npreghat(bws = bw, txdat = data.frame(age = cps71$age))
H.fitted <- H
A <- npreghat(bws = bw, txdat = data.frame(age = cps71$age),
              y = cps71$logwage, output = "constraint")
all.equal(A, t(H) * cps71$logwage)
ghat <- npreg(bws = bw)
head(cbind(fitted(ghat), H.fitted), n = 2L)

npRmpi.quit()

## End(Not run)
```

---

npregiv — *Nonparametric Instrumental Regression*

Description

`npregiv` 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 and exogenous variables X and instruments W. The function φ is the solution of an ill-posed inverse problem.

When `method="Tikhonov"`, `npregiv` uses the approach of Darolles, Fan, Florens and Renault (2011) modified for local polynomial kernel regression of any order (Darolles et al use local constant kernel weighting which corresponds to setting `p=0`; see below for details). When `method="Landweber-Fridman"`,

npregiv uses the approach of Horowitz (2011) again using local polynomial kernel regression (Horowitz uses B-spline weighting).

Usage

```
npregiv(y,
        z,
        w,
        x = NULL,
        zeval = NULL,
        xeval = NULL,
        alpha = NULL,
        alpha.iter = NULL,
        alpha.max = 1e-01,
        alpha.min = 1e-10,
        alpha.tol = .Machine$double.eps^0.25,
        bw = NULL,
        constant = 0.5,
        iterate.diff.tol = 1.0e-08,
        iterate.max = 1000,
        iterate.Tikhonov = TRUE,
        iterate.Tikhonov.num = 1,
        method = c("Landweber-Fridman","Tikhonov"),
        nmulti = NULL,
        optim.abstol = .Machine$double.eps,
        optim.maxattempts = 10,
        optim.maxit = 500,
        optim.method = c("Nelder-Mead", "BFGS", "CG"),
        optim.reltol = sqrt(.Machine$double.eps),
        p = 1,
        penalize.iteration = TRUE,
        random.seed = 42,
        return.weights.phi = FALSE,
        return.weights.phi.deriv.1 = FALSE,
        return.weights.phi.deriv.2 = FALSE,
        smooth.residuals = TRUE,
        start.from = c("Eyz","EEywz"),
        starting.values  = NULL,
        stop.on.increase = TRUE,
        ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the response, endogenous variables, instruments, exogenous covariates, and evaluation data.

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 regressors. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.

xeval — an r-variate data frame of exogenous regressors on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by x.

y — a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of z.

z — a p-variate data frame of endogenous regressors. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.

zeval — a p-variate data frame of endogenous regressors 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 the Landweber-Fridman iteration path.

constant — the constant to use when using method="Landweber-Fridman".

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

iterate.max — an integer indicating the maximum number of iterations permitted before termination occurs when using method="Landweber-Fridman".

iterate.Tikhonov — a logical value indicating whether to use iterated Tikhonov (one iteration) or not when using method="Tikhonov".

iterate.Tikhonov.num — an integer indicating the number of iterations to conduct when using method="Tikhonov".

method — the regularization method employed (defaults to "Landweber-Fridman", see Horowitz (2011); see Darolles, Fan, Florens and Renault (2011) for details for "Tikhonov").

nmulti — integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points.

Optimization Controls: These arguments control numerical optimization for the inverse problem.

optim.abstol — the absolute convergence tolerance used by optim. Only useful for non-negative functions, as a tolerance for reaching zero. Defaults to .Machine$double.eps.

optim.maxattempts — maximum number of attempts taken trying to achieve successful convergence in optim. Defaults to 100.

optim.maxit — maximum number of iterations used by optim. Defaults to 500.

optim.method — method used by optim for minimization of the objective function. See ?optim for references. Defaults to "Nelder-Mead".

the default method is an implementation of that of Nelder and Mead (1965), that uses only function values and is robust but relatively slow. It will work reasonably well for non-differentiable functions.

method "BFGS" is quasi-Newton method (also known as a variable metric algorithm), specifically that published simultaneously in 1970 by Broyden, Fletcher, Goldfarb and Shanno. This uses function values and gradients to build up a picture of the surface to be optimized.

method "CG" is a conjugate gradients method based on that by Fletcher and Reeves (1964) (but with the option of Polak-Ribiere or Beale-Sorenson updates). Conjugate gradient methods will generally be more fragile than the BFGS method, but as they do not store a matrix they may be successful in much larger optimization problems.

`optim.reltol` relative convergence tolerance used by `optim`. The algorithm stops if it is unable to reduce the value by a factor of 'reltol * (abs(val) + reltol)' at a step. Defaults to `sqrt(.Machine$double.eps)`, typically about `1e-8`.

`p` the order of the local polynomial regression (defaults to `p=1`, i.e. local linear).

Returned Weights And Smooth Residuals: These arguments control returned kernel weights, starting values, residual smoothing, and iteration stopping behavior.

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

`random.seed` an integer used to seed R's random number generator. This ensures replicability of the numerical search. Defaults to 42.

`return.weights.phi`
a logical value (defaults to `FALSE`) indicating whether to return the weight matrix which when postmultiplied by the response y delivers the instrumental regression

`return.weights.phi.deriv.1`
a logical value (defaults to `FALSE`) indicating whether to return the weight matrix which when postmultiplied by the response y delivers the first partial derivative of the instrumental regression with respect to z

`return.weights.phi.deriv.2`
a logical value (defaults to `FALSE`) indicating whether to return the weight matrix which when postmultiplied by the response y delivers the second partial derivative of the instrumental regression with respect to z

`smooth.residuals`
a logical value indicating whether to optimize bandwidths for the regression of $(y - \varphi(z))$ on w (defaults to `TRUE`) 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`
a value indicating whether to commence Landweber-Fridman assuming $\varphi_{-1} = starting.values$ (proper Landweber-Fridman) or instead begin from $E(y|z)$ (defaults to `NULL`, 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)

Tikhonov Regularization Controls: These arguments control Tikhonov regularization and its bandwidth.

`alpha`
: a numeric scalar that, if supplied, is used rather than numerically solving for `alpha`, when using `method="Tikhonov"`.

`alpha.iter`
: a numeric scalar that, if supplied, is used for iterated Tikhonov rather than numerically solving for `alpha`, when using `method="Tikhonov"`.

`alpha.max`
: maximum of search range for α, the Tikhonov regularization parameter, when using `method="Tikhonov"`.

`alpha.min`
: minimum of search range for α, the Tikhonov regularization parameter, when using `method="Tikhonov"`.

`alpha.tol`
: the search tolerance for `optimize` when solving for α, the Tikhonov regularization parameter, when using `method="Tikhonov"`.

`bw`
: an object which, if provided, contains bandwidths and parameters (obtained from a previous invocation of `npregiv`) required to re-compute the estimator without having to re-run cross-validation and/or numerical optimization which is particularly costly in this setting (see details below for an illustration of its use)

Additional Arguments: Further arguments are passed to lower-level kernel-sum and estimation routines.

`...`
: additional arguments supplied to `npksum`.

Details

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, `plot` for plotting options, and `npRmpi.init` for interactive/cluster MPI startup. See `npRmpi.init` details for performance tradeoffs (message passing/startup mode) and the `inst/Rprofile` manual-broadcast template.

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 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.

`method="Landweber-Fridman"` uses an optimal stopping rule based upon $||E(y|w)-E(\varphi_k(z,x)|w)||^2$. 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=min(2,p)` for the first iteration, where `p` is the dimension of the current smoothing problem.

Note that for subsequent Landweber-Fridman iterations, a "warm start" strategy is employed. The optimal bandwidths 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 bandwidths 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 `min(2,p)` for the first iteration and to 1 for all subsequent iterations. This strategy provides a balance between computational efficiency and robustness, allowing the numerical optimizer to refine the structural bandwidths 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.

The option `bw=` would be useful, say, when bootstrapping is necessary. Note that when passing `bw`, it must be obtained from a previous invocation of `npregiv`. For instance, if `model.iv` was obtained from an invocation of `npregiv` with `method="Landweber-Fridman"`, then the following needs to be fed to the subsequent invocation of `npregiv`:

```
model.iv <- npregiv(\dots)

bw <- NULL
bw$bw.E.y.w <- model.iv$bw.E.y.w
bw$bw.E.y.z <- model.iv$bw.E.y.z
bw$bw.resid.w <- model.iv$bw.resid.w
bw$bw.resid.fitted.w.z <- model.iv$bw.resid.fitted.w.z
bw$norm.index <- model.iv$norm.index

foo <- npregiv(\dots,bw=bw)
```

If, on the other hand `model.iv` was obtained from an invocation of `npregiv` with `method="Tikhonov"`, then the following needs to be fed to the subsequent invocation of `npregiv`:

```
model.iv <- npregiv(\dots)

bw <- NULL
bw$alpha <- model.iv$alpha
bw$alpha.iter <- model.iv$alpha.iter
bw$bw.E.y.w <- model.iv$bw.E.y.w
bw$bw.E.E.y.w.z <- model.iv$bw.E.E.y.w.z
bw$bw.E.phi.w <- model.iv$bw.E.phi.w
bw$bw.E.E.phi.w.z <- model.iv$bw.E.E.phi.w.z

foo <- npregiv(\dots,bw=bw)
```

Or, if `model.iv` was obtained from an invocation of `npregiv` with either `method="Landweber-Fridman"` or `method="Tikhonov"`, then the following would also work:

```
model.iv <- npregiv(\dots)

foo <- npregiv(\dots,bw=model.iv)
```

When exogenous predictors x (xeval) are passed, they are appended to both the endogenous predictors z and the instruments w as additional columns. If this is not desired, one can manually append the exogenous variables to z (or w) prior to passing z (or w), and then they will only appear among the z or w as desired.

Value

npregiv returns a npregiv object. The generic functions print, summary, and plot support objects of this type.

npregiv returns a list with components phi, phi.mat and either alpha when method="Tikhonov" or norm.index, norm.stop and convergence when method="Landweber-Fridman", among others.

In addition, if any of return.weights.* are invoked (*=1,2), then phi.weights and phi.deriv.*.weights return weight matrices for computing the instrumental regression and its partial derivatives. Note that these weights, post multiplied by the response vector y, will deliver the estimates returned in phi, phi.deriv.1, and phi.deriv.2 (the latter only being produced when p is 2 or greater). When invoked with evaluation data, similar matrices are returned but named phi.eval.weights and phi.deriv.eval.*.weights. These weights can be used for constrained estimation, among others.

When method="Landweber-Fridman" is invoked, bandwidth objects are returned in bw.E.y.w (scalar/vector), bw.E.y.z (scalar/vector), and bw.resid.w (matrix) and bw.resid.fitted.w.z, the latter matrices containing bandwidths for each iteration stored as rows. When method="Tikhonov" is invoked, bandwidth objects are returned in bw.E.y.w, bw.E.E.y.w.z, and bw.E.phi.w and bw.E.E.phi.w.z.

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.

Feve, 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 and S. Centorrino (2018), "Nonparametric instrumental derivatives," Journal of Nonparametric Statistics, 30 (2), 368-391.

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.

Li, Q. and J.S. Racine (2004), "Cross-validated Local Linear Nonparametric Regression," Statistica Sinica, 14, 485-512.

See Also

np.kernels, np.options, plot npregivderiv,npreg

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

# ## This illustration was made possible by Samuele Centorrino
```

```
# ## <samuele.centorrino@univ-tlse1.fr>
#
# set.seed(42)
# n <- 1500
#
# ## The DGP is as follows:
#
# ## 1) y = phi(z) + u
#
# ## 2) E(u|z) != 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 + eps)
#
# 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(-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)
# ivdata <- ivdata[order(ivdata$z),]
# rm(y,z,w)
#
# model.iv <- with(ivdata, npregiv(y=y, z=z, w=w))
# phi.iv <- model.iv$phi
#
# ## Now the non-iv local linear estimator of E(y|z)
```

```
#
# ll.mean <- with(ivdata, fitted(npreg(y~z, regtype="ll")))
#
# ## For the plots, 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)
#
# trim <- 0.0025
#
# curve(phi,min(z),max(z),
#       xlim=quantile(z,c(trim,1-trim)),
#       ylim=quantile(y,c(trim,1-trim)),
#       ylab="Y",
#       xlab="Z",
#       main="Nonparametric Instrumental Kernel Regression",
#       lwd=2,lty=1)
#
# points(z,y,type="p",cex=.25,col="grey")
#
# lines(z,phi.iv,col="blue",lwd=2,lty=2)
#
# lines(z,ll.mean,col="red",lwd=2,lty=4)
#
# legend(quantile(z,trim),quantile(y,1-trim),
#        c(expression(paste(varphi(z))),
#          expression(paste("Nonparametric ",hat(varphi)(z))),
#          "Nonparametric E(y|z)"),
#        lty=c(1,2,4),
#        col=c("black","blue","red"),
#        lwd=c(2,2,2))
#
## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

npregivderiv *Nonparametric Instrumental Derivatives*

Description

npregivderiv uses the approach of Florens, Racine and Centorrino (2018) 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

```
npregivderiv(y,
             z,
             w,
             x = NULL,
             zeval = NULL,
             weval = NULL,
             xeval = NULL,
             constant = 0.5,
             iterate.break = TRUE,
             iterate.max = 1000,
             nmulti = NULL,
             random.seed = 42,
             smooth.residuals = TRUE,
             start.from = c("Eyz","EEywz"),
             starting.values = NULL,
             stop.on.increase = TRUE,
             ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the response, endogenous variables, instruments, exogenous covariates, and evaluation data.

w	a q-variate data frame of instruments. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.
weval	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 w.
x	an r-variate data frame of exogenous regressors. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.
xeval	an r-variate data frame of exogenous regressors on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by x.

`y` a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of `z`.

`z` a p-variate data frame of endogenous regressors. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.

`zeval` a p-variate data frame of endogenous regressors on which the regression will be estimated (evaluation data). By default, evaluation takes place on the data provided by `z`.

Iteration Controls: These arguments control derivative iteration and reproducibility settings.

`constant` the constant to use for Landweber-Fridman iteration.

`iterate.break` a logical value indicating whether to compute all objects up to `iterate.max` or to break when a potential optimum arises (useful for inspecting full stopping rule profile up to `iterate.max`)

`iterate.max` an integer indicating the maximum number of iterations permitted before termination occurs for Landweber-Fridman iteration.

`nmulti` integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points.

`random.seed` an integer used to seed R's random number generator. This ensures replicability of the numerical search. Defaults to 42.

Residual Smoothing And Starting Values: These arguments control residual smoothing and the initial derivative path.

`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 Landweber-Fridman 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`
a value indicating whether to commence Landweber-Fridman assuming $\varphi'_{-1} = starting.values$ (proper Landweber-Fridman) or instead begin from $E(y|z)$ (defaults to `NULL`, 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).

Additional Arguments: Further arguments are passed to `npreg` and `npksum`.

`...` additional arguments supplied to `npreg` and `npksum`.

Details

Documentation guide: see np.kernels for kernels, np.options for global options, plot for plotting options, and npRmpi.init for interactive/cluster MPI startup. See npRmpi.init details for performance tradeoffs (message passing/startup mode) and the inst/Rprofile manual-broadcast template.

Note that Landweber-Fridman iteration presumes that $\varphi_{-1} = 0$, and so for derivative estimation we commence iterating from a model having derivatives all equal to zero. Given this starting point it may require a fairly large number of iterations in order to converge. Other perhaps more reasonable starting values might present themselves. When start.phi.zero is set to FALSE iteration will commence instead using derivatives from the conditional mean model $E(y|z)$. Should the default iteration terminate quickly or you are concerned about your results, it would be prudent to verify that this alternative starting value produces the same result. Also, check the norm.stop vector for any anomalies (such as the error criterion increasing immediately).

Landweber-Fridman iteration uses an optimal stopping rule based upon $||E(y|w)-E(\varphi_k(z,x)|w)||^2$. 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=min(2,p) for the first iteration, where p is the dimension of the current smoothing problem.

Note that for subsequent Landweber-Fridman iterations, a "warm start" strategy is employed. The optimal bandwidths 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 bandwidths 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 min(2,p) for the first iteration and to 1 for all subsequent iterations. This strategy provides a balance between computational efficiency and robustness, allowing the numerical optimizer to refine the structural bandwidths as the residuals evolve incrementally while still guarding against local optima.

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.

Value

npregivderiv returns a npregivderiv object. The generic functions print, summary, and plot support objects of this type.

npregivderiv returns a list with components phi.prime, phi, num.iterations, norm.stop and convergence.

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.

Feve, 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 and S. Centorrino (2018), "Nonparametric instrumental derivatives," Journal of Nonparametric Statistics, 30 (2), 368-391.

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.

Li, Q. and J.S. Racine (2004), "Cross-validated Local Linear Nonparametric Regression," Statistica Sinica, 14, 485-512.

See Also

np.kernels, np.options, plot npregiv,npreg

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
```

```
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

# ## This illustration was made possible by Samuele Centorrino
# ## <samuele.centorrino@univ-tlse1.fr>
#
# set.seed(42)
# n <- 1500
#
# ## 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) != 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 + eps)
#
# 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(-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$z),]
# rm(y,z,w,u,v)
#
# model.ivderiv <- with(ivdata, npregivderiv(y=y, z=z, w=w))
#
# ylim <-c(quantile(model.ivderiv$phi.prime,trim),
#          quantile(model.ivderiv$phi.prime,1-trim))
#
# plot(z,model.ivderiv$phi.prime,
#      xlim=quantile(z,c(trim,1-trim)),
#      main="",
#      ylim=ylim,
#      xlab="Z",
#      ylab="Derivative",
#      type="l",
#      lwd=2)
# rug(z)
#
## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

npRmpi *Parallel Nonparametric Kernel Smoothing Methods for Mixed Data Types*

Description

This package provides a variety of nonparametric and semiparametric kernel methods that seamlessly handle a mix of continuous, unordered, and ordered factor data types (unordered and ordered factors are often referred to as 'nominal' and 'ordinal' categorical variables respectively). A getting-started vignette containing a short introduction to the **npRmpi** package can be accessed via `vignette("npRmpi_getting_started", package = "npRmpi")`.

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

Bandwidth selection is a key aspect of sound nonparametric and semiparametric kernel estimation. `npRmpi` is designed from the ground up to make bandwidth selection the focus of attention. To this end, one typically begins by creating a 'bandwidth object' which embodies all aspects of the method, including specific kernel functions, data names, data types, and the like. One then passes these bandwidth objects to other functions, and those functions can grab the specifics from the bandwidth object thereby removing potential inconsistencies and unnecessary repetition. Furthermore, many functions such as `plot` (via class-specific S3 methods) can work with the bandwidth object directly without having to do the subsequent companion function evaluation.

The user may also combine these steps. If the first step (bandwidth selection) is not performed explicitly then the second step will automatically call the omitted first-step bandwidth selector using defaults unless otherwise specified, and the bandwidth object can be retrieved retroactively if so desired via `objectname$bws`. Furthermore, options for bandwidth selection will be passed directly to the bandwidth selector function. Note that the combined approach would not be a wise choice for certain applications such as when bootstrapping (as it would involve unnecessary computation since the bandwidths would properly be those for the original sample and not the bootstrap resamples) or when conducting quantile regression (as it would involve unnecessary computation when different quantiles are computed from the same conditional cumulative distribution estimate).

There are two ways in which you can interact with functions in `npRmpi`, either i) using data frames, or ii) using a formula interface, where appropriate.

To some, it may be natural to use the data frame interface. The R `data.frame` function preserves a variable's type once it has been cast (unlike `cbind`, which we avoid for this reason). If you find this most natural for your project, you first create a data frame casting data according to their type (i.e., one of continuous (default, `numeric`), `factor`, `ordered`). Then you would simply pass this data frame to the appropriate `npRmpi` function, for example `npudensbw(dat=data)`.

To others, however, it may be natural to use the formula interface that is used for the regression examples, among others. For nonparametric regression functions such as `npreg`, you would proceed as you would using `lm` (e.g., `bw <- npregbw(y~factor(x1)+x2)`) except that you would of course not need to specify, e.g., polynomials in variables, interaction terms, or create a number of dummy variables for a factor. Every function in npRmpi supports both interfaces, where appropriate.

Note that if your factor is in fact a character string such as, say, `X` being either `"MALE"` or `"FEMALE"`, npRmpi will handle this directly, i.e., there is no need to map the string values into unique integers such as (0,1). Once the user casts a variable as a particular data type (i.e., `factor`, `ordered`, or continuous (default, `numeric`)), all subsequent methods automatically detect the type and use the appropriate kernel function and method where appropriate.

All estimation methods are fully multivariate, i.e., there are no limitations on the number of variables one can model (or number of observations for that matter). Execution time for most routines is, however, exponentially increasing in the number of observations and increases with the number of variables involved.

Nonparametric methods include unconditional density (distribution), conditional density (distribution), regression, mode, and quantile estimators along with gradients where appropriate, while semiparametric methods include single index, partially linear, and smooth (i.e., varying) coefficient models.

A number of tests are included such as consistent specification tests for parametric regression and quantile regression models along with tests of significance for nonparametric regression.

A variety of bootstrap methods for computing standard errors, nonparametric confidence bounds, and bias-corrected bounds are implemented.

A variety of bandwidth methods are implemented including fixed, nearest-neighbor, and adaptive nearest-neighbor.

A variety of data-driven methods of bandwidth selection are implemented, while the user can specify their own bandwidths should they so choose (either a raw bandwidth or scaling factor).

A flexible plotting utility, via class-specific S3 `plot` methods, facilitates graphing of multivariate objects. An example for creating postscript graphs and pulling this into a LaTeX document is provided.

The function `npksum` allows users to create or implement their own kernel estimators or tests should they so desire.

The underlying functions are written in C for computational efficiency. Despite this, due to their nature, data-driven bandwidth selection methods involving multivariate numerical search can be time-consuming, particularly for large datasets. The **npRmpi** package provides the MPI-aware companion to **np**, extending the same mixed-data kernel methodology to clustered computing environments while preserving the familiar estimator surface after MPI initialization.

To cite the `npRmpi` package, type `citation("npRmpi")` from within `R` for details.

Details

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, `plot` for plotting options, and `npRmpi.init` for interactive/cluster MPI startup. See `npRmpi.init` details for performance tradeoffs (message passing/startup mode) and the `inst/Rprofile` manual-broadcast template. For interactive and cluster batch workflows, see `npRmpi.init`. The kernel methods in `npRmpi` employ the so-called 'generalized product kernels' found in Hall, Racine, and Li (2004), Li, Lin, and Racine (2013), Li, Ouyang, and Racine (2013), Li and Racine (2003), Li and Racine (2004), Li and Racine (2007), Li and Racine (2010), Ouyang, Li, and Racine (2006), and Racine and Li (2004), among others. For details on a particular method, kindly refer to the original references listed above.

We briefly describe the particulars of various univariate kernels used to generate the generalized product kernels that underlie the kernel estimators implemented in the `npRmpi` package. In a nutshell, the generalized kernel functions that underlie the kernel estimators in `npRmpi` are formed by taking the product of univariate kernels such as those listed below. When you cast your data as a particular type (continuous, factor, or ordered factor) in a data frame or formula, the routines will automatically recognize the type of variable being modelled and use the appropriate kernel type for each variable in the resulting estimator.

Second Order Gaussian (x is continuous) $k(z) = \exp(-z^2/2)/\sqrt{2\pi}$ where $z = (x_i - x)/h$, and $h > 0$.

Second Order Truncated Gaussian (x is continuous) $k(z) = (\exp(-z^2/2) - \exp(-b^2/2))/(\mathrm{erf}(b/\sqrt{2})\sqrt{2\pi} - 2b\exp(-b^2/2))$ where $z = (x_i - x)/h$, $b > 0$, $|z| \leq b$ and $h > 0$.

See `nptgauss` for details on modifying b.

Second Order Epanechnikov (x is continuous) $k(z) = 3\left(1 - z^2/5\right)/(4\sqrt{5})$ if $z^2 < 5$, 0 otherwise, where $z = (x_i - x)/h$, and $h > 0$.

Uniform (x is continuous) $k(z) = 1/2$ if $|z| < 1$, 0 otherwise, where $z = (x_i - x)/h$, and $h > 0$.

Aitchison and Aitken (x is a (discrete) factor) $l(x_i, x, \lambda) = 1 - \lambda$ if $x_i = x$, and $\lambda/(c-1)$ if $x_i \neq x$, where c is the number of (discrete) outcomes assumed by the factor x.

Note that λ must lie between 0 and $(c-1)/c$.

Wang and van Ryzin (x is a (discrete) ordered factor) $l(x_i, x, \lambda) = 1 - \lambda$ if $|x_i - x| = 0$, and $((1-\lambda)/2)\lambda^{|x_i - x|}$ if $|x_i - x| \geq 1$.

Note that λ must lie between 0 and 1.

Li and Racine (x is a (discrete) factor) $l(x_i, x, \lambda) = 1$ if $x_i = x$, and λ if $x_i \neq x$.

Note that λ must lie between 0 and 1.

Li and Racine Normalised for Unconditional Objects (x is a (discrete) factor) $l(x_i, x, \lambda) = 1/(1 + (c-1)\lambda)$ if $x_i = x$, and $\lambda/(1 + (c-1)\lambda)$ if $x_i \neq x$.

Note that λ must lie between 0 and 1.

Li and Racine (x is a (discrete) ordered factor) $l(x_i, x, \lambda) = 1$ if $|x_i - x| = 0$, and $\lambda^{|x_i - x|}$ if $|x_i - x| \geq 1$.

Note that λ must lie between 0 and 1.

Li and Racine Normalised for Unconditional Objects (x is a (discrete) ordered factor) $l(x_i, x, \lambda) = (1-\lambda)/(1+\lambda)$ if $|x_i - x| = 0$, and $(1-\lambda)/(1+\lambda)\lambda^{|x_i - x|}$ if $|x_i - x| \geq 1$.

Note that λ must lie between 0 and 1.

Racine, Li, and Yan (x is a (discrete) ordered factor) $l(x_i, x, \lambda) = \lambda^{|x_i - x|} / \sum_{z \in D} \lambda^{|x_i - z|}$, where D is the ordered support.

Note that λ must lie between 0 and 1.

So, if you had two variables, x_{i1} and x_{i2}, and x_{i1} was continuous while x_{i2} was, say, binary (0/1), and you created a data frame of the form `X <- data.frame(x1,factor(x2))`, then the kernel function used by `npRmpi` would be $K(\cdot) = k(\cdot) \times l(\cdot)$ where the particular kernel functions $k(\cdot)$ and $l(\cdot)$ would be, say, the second order Gaussian (`ckertype="gaussian"`) and Aitchison and Aitken (`ukertype="aitchisonaitken"`) kernels by default, respectively.

Note that higher order continuous kernels (i.e., fourth, sixth, and eighth order) are derived from the second order kernels given above (see Li and Racine (2007) for details).

For continuous kernels, one can optionally enforce finite support normalization via user-supplied bounds. When finite lower/upper bounds are supplied (e.g., `ckerbound="fixed"` with `ckerlb` and `ckerub`), continuous kernels are normalized on $[a, b]$ using the corresponding kernel CDF in the denominator. Setting infinite bounds recovers the standard unbounded kernel. This boundary-adaptive normalization is especially useful for unconditional density/distribution estimation on bounded supports.

For particulars on any given method, kindly see the references listed for the method in question.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

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

We are grateful to John Fox and Achim Zeleis for their valuable input and encouragement. We would like to gratefully acknowledge support from the Natural Sciences and Engineering Research Council of Canada (NSERC:www.nserc.ca), the Social Sciences and Humanities Research Council of Canada (SSHRC:www.sshrc.ca), and the Shared Hierarchical Academic Research Computing Network (SHARCNET:www.sharcnet.ca)

References

Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

Hall, P. and J.S. Racine and Q. Li (2004), "Cross-validation and the estimation of conditional probability densities," Journal of the American Statistical Association, 99, 1015-1026.

Li, Q. and J. Lin and J.S. Racine (2013), "Optimal bandwidth selection for nonparametric conditional distribution and quantile functions", Journal of Business and Economic Statistics, 31, 57-65.

Li, Q. and D. Ouyang and J.S. Racine (2013), "Categorical Semiparametric Varying-Coefficient Models," Journal of Applied Econometrics, 28, 551-589.

Li, Q. and J.S. Racine (2003), "Nonparametric estimation of distributions with categorical and continuous data," Journal of Multivariate Analysis, 86, 266-292.

Li, Q. and J.S. Racine (2004), "Cross-validated local linear nonparametric regression," Statistica Sinica, 14, 485-512.

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

Li, Q. and J.S. Racine (2010), "Smooth varying-coefficient estimation and inference for qualitative and quantitative data," Econometric Theory, 26, 1-31.

Ouyang, D. and Q. Li and J.S. Racine (2006), "Cross-validation and the estimation of probability distributions with categorical data," Journal of Nonparametric Statistics, 18, 69-100.

Racine, J.S. and Q. Li (2004), "Nonparametric estimation of regression functions with both categorical and continuous data," Journal of Econometrics, 119, 99-130.

Racine, J.S. and Q. Li and Q. Wang (2024), "Boundary-Adaptive Kernel Density Estimation: The Case of (Near) Uniform Density," Journal of Nonparametric Statistics, 36(1), 146-164.

Racine, J.S., Q. Li, and K.X. Yan (2020), "Kernel Smoothed Probability Mass Functions for Ordered Datatypes," Journal of Nonparametric Statistics, 32(3), 563-586.

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

Scott, D.W. (1992), *Multivariate Density Estimation: Theory, Practice and Visualization,* New York: Wiley.

Silverman, B.W. (1986), *Density Estimation,* London: Chapman and Hall.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

np.kernels, np.options, plot np.options

---

npRmpi.init *Init/Quit Helpers for Session and Attach npRmpi Workflows*

Description

Convenience helpers for the two ordinary **npRmpi** startup routes: session mode (the "spawn" code path) and attach mode (mode="attach"). These helpers are the recommended entry points for routine interactive use and for mpiexec-launched scripts that attach to an already-running MPI world.

Usage

```
npRmpi.init(...,
            nslaves = 1,
            comm = 1,
            mode = c("auto", "spawn", "attach"),
            autodispatch = TRUE,
            autodispatch.verify.options = FALSE,
            autodispatch.option.sync = c("onchange", "always", "never"),
            np.messages = NULL,
            nonblock = TRUE,
            sleep = 0.1,
            quiet = FALSE)

npRmpi.quit(force = FALSE,
            dellog = TRUE,
            comm = 1,
            mode = c("auto", "spawn", "attach"))

npRmpi.session.info(comm = 1)
```

Arguments

Session And Worker Controls: MPI startup, shutdown, worker-count, and worker-status controls.

nslaves	Number of slaves to spawn for interactive execution (mode="spawn"); must be at least 1. For serial workflows, use package **np**.
comm	Communicator used for the master+slaves pool (defaults to 1).
mode	Startup/stop mode. "spawn" starts slaves from rank 0 and is the session-mode code path. "attach" attaches to an already-launched MPI world (for example started with mpiexec -n ...) without spawning. "auto" selects "attach" when mpi.comm.size(0) > 1, otherwise "spawn".

autodispatch Logical; if non-NULL, sets options(npRmpi.autodispatch=...) inside npRmpi.init().

autodispatch.verify.options
Logical; when TRUE, verify selected synchronized options across ranks on each dispatch sync event.

autodispatch.option.sync
Option synchronization policy for auto-dispatch: "onchange" (default), "always", or "never".

np.messages Logical; if non-NULL, sets options(np.messages=...) inside npRmpi.init().

nonblock Logical passed to the internal attach-mode worker loop receive path.

sleep Polling sleep interval (seconds) for nonblocking attach-mode worker-loop receives.

quiet Logical; suppress host-info printing when FALSE.

force Logical; when TRUE, force a hard shutdown of slave daemons.

dellog Logical; when TRUE, remove slave log files (if applicable).

... Additional arguments passed to mpi.spawn.Rslaves().

Details

Documentation guide: see np.kernels for kernels, np.options for global options, plot for plotting options, and npRmpi.init for interactive/cluster MPI startup.

npRmpi.init() and npRmpi.quit() are the ordinary entry points for two workflows:

1. Session mode (mode="spawn", nslaves>=1): a single R process spawns workers and then uses ordinary **npRmpi** calls.
2. Batch/attach mode (mode="attach"): attaches to a pre-launched MPI world and runs worker-loop coordination internally.

Profile/manual-broadcast mode is a separate advanced route. It does not call npRmpi.init(). Instead, ranks are started under mpiexec with inst/Rprofile (or an explicit R_PROFILE_USER path), and the script then uses mpi.bcast.cmd(np.mpi.initialize(), caller.execute=TRUE) plus explicit mpi.bcast.* calls. See np.mpi.initialize, inst/Rprofile, and the demo run guide for that route.

Workflow quick-start guidance:

1. **Session mode** (mode="spawn"): recommended first on platforms where spawning is supported. Typical launch: Rscript foo.R or R CMD BATCH --no-save foo.R, then call npRmpi.init(nslaves=...) near the top of the script and npRmpi.quit() at the end.
2. **Attach mode** (mode="attach"): launch under mpiexec and call npRmpi.init(mode="attach") inside the script. Typical launch: mpiexec -env R_PROFILE_USER '' -env R_PROFILE '' -n <ranks> Rscript --no-save foo.R (or R CMD BATCH --no-save). Clearing startup-profile variables is intentional here because profile/manual-broadcast startup belongs to a different route.
3. **Profile/manual-broadcast mode**: launch under mpiexec with inst/Rprofile and explicit mpi.bcast.* calls. Use R CMD BATCH --no-save (not --vanilla) and provide exactly one startup profile source.

Performance note. Wall-clock can differ across workflows even for identical statistical output. The main drivers are MPI message passing and startup/teardown behavior:

1. Profile/manual-broadcast mode often has the lowest messaging overhead for small/moderate jobs because startup and broadcasts are explicit.
2. Using `R CMD BATCH --no-save` with `npRmpi.init(mode="spawn")` is simpler to use but may pay additional broadcast/setup overhead, especially when many slaves are used on small `n`.
3. As `n` grows, compute usually dominates fixed messaging costs and relative penalties commonly shrink.
4. Worker-count changes can also change summation or reduction order at numerical roundoff scale. Use tolerance-based comparisons for objective values or selected bandwidths across worker counts unless the run is a controlled single-process reproducibility check.
5. In session/attach mode with auto-dispatch enabled, lightweight calls (especially post-bandwidth `npreg(bws=...)` and `predict(...)`) can be slower due to command marshalling/serialization overhead. A practical pattern is: keep auto-dispatch enabled for bandwidth selection, then set `options(npRmpi.autodispatch=FALSE)` for post-`bw` fit/predict. Manual-broadcast/profile mode often behaves closer to this low-overhead pattern.

Template startup profile for profile/manual-broadcast workflows is provided at `inst/Rprofile`. Copy it to the job working directory (or set `R_PROFILE_USER` to that file) when using `mpiexec -n ... R CMD BATCH --no-save ...` with explicit `mpi.bcast.cmd()` and `mpi.bcast.Robj2slave()` calls. Do not use `R CMD BATCH --vanilla` for this route, because `--vanilla` disables reading startup profiles and the manual-broadcast worker loop will not be initialized from `.Rprofile`. Also avoid setting both `R_PROFILE` and `R_PROFILE_USER` to the same file; this is treated as a startup misconfiguration and fails fast.

Minimal comparison script (three patterns) follows:

```
## CASE 1: user-friendly (single R process; spawn mode)
## run: R CMD BATCH --no-save script_spawn.R
library(npRmpi); library(MASS)
npRmpi.init(nslaves=5)  # autodispatch=TRUE by default
set.seed(42); n <- 5000
rho <- 0.25; mu <- c(0,0); Sigma <- matrix(c(1,rho,rho,1),2,2)
dat <- mvrnorm(n=n, mu, Sigma); mydat <- data.frame(x=dat[,2], y=dat[,1])
bw <- npcdensbw(y~x, bwmethod="cv.ml", data=mydat)
fit <- npcdens(bws=bw)
npRmpi.quit()

## CASE 2: user-friendly under mpiexec (attach mode; no manual bcast calls)
## run: mpiexec -n 6 R CMD BATCH --no-save script_attach_auto.R
library(npRmpi); library(MASS)
is.master <- isTRUE(npRmpi.init(mode="attach"))
if (is.master) {
  set.seed(42); n <- 5000
  rho <- 0.25; mu <- c(0,0); Sigma <- matrix(c(1,rho,rho,1),2,2)
  dat <- mvrnorm(n=n, mu, Sigma); mydat <- data.frame(x=dat[,2], y=dat[,1])
  bw <- npcdensbw(y~x, bwmethod="cv.ml", data=mydat)
  fit <- npcdens(bws=bw)
```

```
  npRmpi.quit(mode="attach")
  mpi.quit()  # explicit master finalize for clean mpiexec exit
}

## CASE 3: performance-oriented profile/manual-broadcast mode
## run: mpiexec -env R_PROFILE_USER ../inst/Rprofile -env R_PROFILE '' \
##              -n 6 R CMD BATCH --no-save script_attach_manual.R
## requires: inst/Rprofile (or R_PROFILE_USER set to that file)
## do not use: R CMD BATCH --vanilla (skips .Rprofile)
mpi.bcast.cmd(np.mpi.initialize(), caller.execute=TRUE)
mpi.bcast.cmd(library(MASS), caller.execute=TRUE)
mpi.bcast.cmd(set.seed(42), caller.execute=TRUE)
n <- 5000
rho <- 0.25; mu <- c(0,0); Sigma <- matrix(c(1,rho,rho,1),2,2)
dat <- mvrnorm(n=n, mu, Sigma); mydat <- data.frame(x=dat[,2], y=dat[,1])
mpi.bcast.Robj2slave(mydat)
t <- system.time(mpi.bcast.cmd(bw <- npcdensbw(y~x, bwmethod="cv.ml", data=mydat),
                               caller.execute=TRUE))
t <- t + system.time(mpi.bcast.cmd(fit <- npcdens(bws=bw), caller.execute=TRUE))
cat("Elapsed time =", t[3], "\n")
mpi.bcast.cmd(mpi.quit(), caller.execute=TRUE)
```

`npRmpi.quit()` is idempotent: if no slaves are running it returns silently. When `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), `force=FALSE` performs a soft-close to keep daemons alive for reuse within the session; use `force=TRUE` to actually shut down the slaves. In `mode="attach"`, `npRmpi.quit()` signals worker ranks to exit their loop and returns on rank 0 without forcing an R quit on the master process. In profile/manual-broadcast mode, termination is handled explicitly in the script via broadcasted `mpi.quit()` calls rather than via `npRmpi.quit()`.

If an MPI-fanned bootstrap dispatch fails or times out, the current MPI session may no longer be usable. Restart R before performing further MPI-backed computations.

Advanced diagnostic option: setting environment variable `NP_RMPI_SKIP_INIT` to a non-empty value before loading **npRmpi** skips MPI initialization in `.onLoad`. This is intended for development/debug workflows only, and disables normal MPI session startup until a standard initialization path is used.

For stability, avoid attaching **Rmpi** directly before calling `npRmpi.init()`. If **Rmpi** is attached, `npRmpi.init()` fails fast with an actionable error message.

`npRmpi.session.info()` prints and returns a list of useful version, platform, and MPI/communicator details to aid reproducibility and bug reports.

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## Start once, run many examples, then stop.
npRmpi.init(nslaves=1)

## ... run np* calls here ...
```

```
## Soft-stop (may keep daemons alive for reuse)
npRmpi.quit()

## Hard-stop (actually shuts down slaves)
## npRmpi.quit(force=TRUE)

## Batch/cluster style (under mpiexec):
## mpiexec -n 128 Rscript foo.R
## inside foo.R:
## npRmpi.init(mode="attach")
## ... np* calls ...
## npRmpi.quit(mode="attach")
## mpi.quit()
##
## Profile/manual-broadcast mode is separate:
## start ranks with inst/Rprofile, then use
## mpi.bcast.cmd(np.mpi.initialize(), caller.execute=TRUE)
## and explicit mpi.bcast.* calls.

## End(Not run)
```

---

npscoef *Smooth Coefficient Kernel Regression*

Description

npscoef computes a kernel regression estimate of a one (1) dimensional dependent variable on p-variate explanatory data, using the model $Y_i = W_i'\gamma(Z_i) + u_i$ where $W_i' = (1, X_i')$, given a set of evaluation points, training points (consisting of explanatory data and dependent data), and a bandwidth specification. A bandwidth specification can be a scbandwidth object, or a bandwidth vector, bandwidth type and kernel type.

Usage

```
npscoef(bws, ...)

## S3 method for class 'formula'
npscoef(bws,
        data = NULL,
        newdata = NULL,
        y.eval = FALSE,
        ...)


## Default S3 method:
npscoef(bws,
        txdat,
        tydat,
        tzdat,
```

```
        nomad = FALSE,
        ...)

## S3 method for class 'scbandwidth'
npscoef(bws,
        txdat = stop("training data 'txdat' missing"),
        tydat = stop("training data 'tydat' missing"),
        tzdat = NULL,
        exdat,
        eydat,
        ezdat,
        betas = FALSE,
        errors = TRUE,
        iterate = TRUE,
        leave.one.out = FALSE,
        maxiter = 100,
        residuals = FALSE,
        tol = .Machine$double.eps,
        ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and smooth-coefficient training data.

`bws` a bandwidth specification. This can be set as a `scbandwidth` object returned from an invocation of `npscoefbw`, or as a vector of bandwidths, with each element i corresponding to the bandwidth for column i in `tzdat`. If specified as a vector additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, training data, and so on.

`data` an optional data frame, list or environment (or object coercible to a data frame by `as.data.frame`) containing the variables in the model. If not found in data, the variables are taken from `environment(bws)`, typically the environment from which `npscoefbw` was called.

`txdat` a p-variate data frame of explanatory data (training data), which, by default, populates the columns 2 through $p+1$ of W in the model equation, and in the absence of `zdat`, will also correspond to Z from the model equation. Defaults to the training data used to compute the bandwidth object.

`tydat` a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of `txdat`. Defaults to the training data used to compute the bandwidth object.

`tzdat` an optionally specified q-variate data frame of explanatory data (training data), which corresponds to Z in the model equation. Defaults to the training data used to compute the bandwidth object.

Local-Polynomial Degree And Bandwidth Search: This argument controls the recommended automatic local-polynomial NOMAD route, which jointly selects continuous polynomial degree and bandwidths when these are computed inside `npscoef`.

nomad — logical shortcut passed through to npscoefbw when bandwidths are computed inside npscoef. When TRUE, the smooth-coefficient bandwidth route fills any missing values among regtype, search.engine, degree.select, bernstein.basis, degree.min, degree.max, degree.verify, and bwtype with the recommended automatic local-polynomial degree-and-bandwidth NOMAD preset documented in npscoefbw. Additional NOMAD tuning arguments such as nomad.nmulti may also be supplied through ...; nmulti remains the outer restart count while nomad.nmulti controls inner native **crs** NOMAD API multistarts within each outer restart. After fitting, inspect fitbwsnomad.shortcut on the returned object fit to see the normalized shortcut metadata.

Evaluation Data And Returned Quantities: These arguments control where the smooth-coefficient fit is evaluated and which evaluation quantities are returned.

exdat — a p-variate data frame of points on which the regression will be estimated (evaluation data).By default, evaluation takes place on the data provided by txdat.

eydat — a one (1) dimensional numeric or integer vector of the true values of the dependent variable. Optional, and used only to calculate the true errors.

ezdat — an optionally specified q-variate data frame of points on which the regression will be estimated (evaluation data), which corresponds to Z in the model equation. Defaults to be the same as txdat.

newdata — An optional data frame in which to look for evaluation data. If omitted, the training data are used.

y.eval — If newdata contains dependent data and y.eval = TRUE, npRmpi will compute goodness of fit statistics on these data and return them. Defaults to FALSE.

Fitted Quantities And Backfitting: These arguments control returned coefficient estimates, errors, residuals, and iterative backfitting.

betas — a logical value indicating whether or not estimates of the components of γ should be returned in the smoothcoefficient object along with the regression estimates. Defaults to FALSE.

errors — a logical value indicating whether or not asymptotic standard errors should be computed and returned in the resulting smoothcoefficient object. Defaults to TRUE.

iterate — a logical value indicating whether or not backfitted estimates should be iterated for self-consistency. Defaults to TRUE.

leave.one.out — a logical value to specify whether or not to compute the leave one out estimates. Will not work if e[xyz]dat is specified. Defaults to FALSE.

maxiter — integer specifying the maximum number of times to iterate the backfitted estimates while attempting to make the backfitted estimates converge to the desired tolerance. Defaults to 100.

residuals — a logical value indicating that you want residuals computed and returned in the resulting smoothcoefficient object. Defaults to FALSE.

tol — desired tolerance on the relative convergence of backfit estimates. Defaults to .Machine$double.eps.

Additional Arguments: Further arguments are passed to the bandwidth-selection counterpart when bandwidths are not supplied.

`...` additional arguments supplied to specify the regression type, bandwidth type, kernel types, selection methods, and so on. To do this, you may specify any of `bwscaling`, `bwtype` (one of `fixed`, `generalized_nn`, `adaptive_nn`), `ckertype`, `ckerorder`, as described in `npscoefbw`.

Value

`npscoef` returns a `smoothcoefficient` object. The generic functions `fitted`, `residuals`, `coef`, `se`, and `predict`, extract (or generate) estimated values, residuals, coefficients, bootstrapped standard errors on estimates, and predictions, respectively, from the returned object. Furthermore, the functions `summary` and `plot` support objects of this type. The returned object has the following components:

`eval`	evaluation points
`mean`	estimation of the regression function (conditional mean) at the evaluation points
`merr`	if `errors = TRUE`, standard errors of the regression estimates
`beta`	if `betas = TRUE`, estimates of the coefficients γ at the evaluation points
`grad`	estimated derivatives of the conditional mean with respect to the regressors in `xdat`; these correspond to the non-intercept smooth coefficient estimates at each evaluation point
`gerr`	if `errors = TRUE`, asymptotic standard errors for `grad`
`resid`	if `residuals = TRUE`, in-sample or out-of-sample residuals where appropriate (or possible)
`R2`	coefficient of determination (Doksum and Samarov (1995))
`MSE`	mean squared error
`MAE`	mean absolute error
`MAPE`	mean absolute percentage error
`CORR`	absolute value of Pearson's correlation coefficient
`SIGN`	fraction of observations where fitted and observed values agree in sign

Book And Method Pointers

The smooth-coefficient model lets slopes vary with conditioning variables, typically written $Y = X'\beta(Z)+\epsilon$. The functions estimate the coefficient functions $\beta_j(z)$ using mixed-data kernel smoothing over Z, with the reported fitted values obtained by combining the estimated coefficient functions with the corresponding columns of X.

For book-length derivations, see Li and Racine (2007), Chapter 9 *Additive and Smooth (Varying) Coefficient Semiparametric Models*, especially Sections 9.3-9.3.4, and Racine (2019), Chapter 8 *Semiparametric Conditional Mean Function Estimation*, especially the varying-coefficient material.

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

When reusing a bandwidth object without explicit training or evaluation data, the required data are recovered from object-retained state or from the stored call environment when available. This is a convenience for ordinary same-session use, not a guarantee that a saved object is self-contained. For reproducible replay after `saveRDS()` and `readRDS()` in a fresh R session, supply the required data arguments explicitly.

For practitioners who want the recommended automatic local-polynomial degree-and-bandwidth NOMAD route without spelling out all LP tuning arguments, `npscoef(..., nomad=TRUE)` and `npscoefbw(..., nomad=TRUE)` expand missing settings to the same documented preset. Explicit incompatible settings fail fast rather than being silently rewritten.

For plotting options for fitted smooth-coefficient objects and their bandwidth objects, see `plot.np`.

Support for backfitted bandwidths is experimental and is limited in functionality. The code does not support asymptotic standard errors or out of sample estimates with backfitting.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

Cai Z. (2007), "Trending time-varying coefficient time series models with serially correlated errors," Journal of Econometrics, 136, 163-188.

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.

Hastie, T. and R. Tibshirani (1993), "Varying-coefficient models," Journal of the Royal Statistical Society, B 55, 757-796.

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

Li, Q. and J.S. Racine (2010), "Smooth varying-coefficient estimation and inference for qualitative and quantitative data," Econometric Theory, 26, 1-31.

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

Li, Q. and D. Ouyang and J.S. Racine (2013), "Categorical semiparametric varying-coefficient models," Journal of Applied Econometrics, 28, 551-589.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

`npRmpi.init`, `np.kernels`, `np.options`, `plot`, `plot.np`, `bw.nrd`, `bw.SJ`, `hist`, `npudens`, `npudist`, `npudensbw`, `npscoefbw`

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)
set.seed(42)

n <- 500

x <- runif(n)
z <- runif(n, min=-2, max=2)
y <- x*exp(z)*(1.0+rnorm(n,sd = 0.2))

## A smooth coefficient model example

bw <- npscoefbw(y~x|z)

summary(bw)

model <- npscoef(bws=bw, gradients=TRUE)

summary(model)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
```

```
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

npscoefbw *Smooth Coefficient Kernel Regression Bandwidth Selection*

Description

npscoefbw computes a bandwidth object for a smooth coefficient kernel regression estimate of a one (1) dimensional dependent variable on $p + q$-variate explanatory data, using the model $Y_i = W_i'\gamma(Z_i) + u_i$ where $W_i' = (1, X_i')$ given training points (consisting of explanatory data and dependent data), and a bandwidth specification, which can be a rbandwidth object, or a bandwidth vector, bandwidth type and kernel type.

Usage

```
npscoefbw(...)

## S3 method for class 'formula'
npscoefbw(formula,
          data,
          subset,
          na.action,
          call,
          ...)

## Default S3 method:
npscoefbw(xdat = stop("invoked without data 'xdat'"),
          ydat = stop("invoked without data 'ydat'"),
          zdat = NULL,
          bws,
          backfit.iterate,
          backfit.maxiter,
          backfit.tol,
          bandwidth.compute = TRUE,
          basis,
          bernstein.basis,
```

```
            bwmethod,
            bwscaling,
            bwtype,
            ckerbound,
            ckerlb,
            ckerorder,
            ckertype,
            ckerub,
            cv.iterate,
            cv.num.iterations,
            degree,
            degree.select = c("manual", "coordinate", "exhaustive"),
            search.engine = c("nomad+powell", "cell", "nomad"),
            nomad = FALSE,
            nomad.nmulti = 0L,
              degree.min = NULL,
            degree.max = NULL,
            degree.start = NULL,
            degree.restarts = 0L,
            degree.max.cycles = 20L,
            degree.verify = FALSE,
            nmulti,
            nomad.remin = FALSE,
            powell.remin = TRUE,
            okertype,
            optim.abstol,
            optim.maxattempts,
            optim.maxit,
            optim.method,
            optim.reltol,
            random.seed,
            regtype,
            ukertype,
            scale.factor.init.lower = 0.1,
            scale.factor.init.upper = 2.0,
            scale.factor.init = 0.5,
            lbd.init = 0.5,
            hbd.init = 1.5,
            dfac.init = 1.0,
            scale.factor.search.lower = NULL,
            ...,
            nomad.opts = list())

## S3 method for class 'scbandwidth'
npscoefbw(xdat = stop("invoked without data 'xdat'"),
          ydat = stop("invoked without data 'ydat'"),
          zdat = NULL,
          bws,
```

```
          backfit.iterate = FALSE,
          backfit.maxiter = 100,
          backfit.tol = .Machine$double.eps,
          bandwidth.compute = TRUE,
          cv.iterate = FALSE,
          cv.num.iterations = 1,
          nmulti,
          optim.abstol = .Machine$double.eps,
          optim.maxattempts = 10,
          optim.maxit = 500,
          optim.method = c("Nelder-Mead", "BFGS", "CG"),
          optim.reltol = sqrt(.Machine$double.eps),
          random.seed = 42,
          scale.factor.init.lower = 0.1,
          scale.factor.init.upper = 2.0,
          scale.factor.init = 0.5,
          lbd.init = 0.5,
          hbd.init = 1.5,
          dfac.init = 1.0,
          scale.factor.search.lower = NULL,
          ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the smooth-coefficient data, formula interface, and whether bandwidths are supplied or computed.

`bandwidth.compute`
: a logical value which specifies whether to do a numerical search for bandwidths or not. If set to `FALSE`, a `scbandwidth` object will be returned with bandwidths set to those specified in `bws`. Defaults to `TRUE`.

`bws`
: a bandwidth specification. This can be set as a `scbandwidth` object returned from a previous invocation, or as a vector of bandwidths, with each element i corresponding to the bandwidth for column i in `xdat`. In either case, the bandwidth supplied will serve as a starting point in the numerical search for optimal bandwidths. If specified as a vector, then additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, selection methods, and so on. This can be left unset.

`call`
: the original function call. This is passed internally by `npRmpi` when a bandwidth search has been implied by a call to another function. It is not recommended that the user set this.

`data`
: an optional data frame, list or environment (or object coercible to a data frame by `as.data.frame`) containing the variables in the model. If not found in data, the variables are taken from `environment(formula)`, typically the environment from which the function is called.

`formula`
: a symbolic description of variables on which bandwidth selection is to be performed. The details of constructing a formula are described below.

`na.action` a function which indicates what should happen when the data contain NAs. The default is set by the `na.action` setting of options, and is `na.fail` if that is unset. The (recommended) default is `na.omit`.

`subset` an optional vector specifying a subset of observations to be used in the fitting process.

`xdat` a p-variate data frame of explanatory data (training data), which, by default, populates the columns 2 through $p + 1$ of W in the model equation, and in the absence of `zdat`, will also correspond to Z from the model equation.

`ydat` a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of `xdat`.

`zdat` an optionally specified q-variate data frame of explanatory data (training data), which corresponds to Z in the model equation. Defaults to be the same as `xdat`.

Automatic Degree Search Controls: These arguments control automatic local-polynomial degree search.

`degree.max` optional scalar or integer vector giving upper bounds for automatic degree search when `degree.select != "manual"`.

`degree.max.cycles` positive integer giving the maximum number of coordinate-search sweeps over the degree vector. Ignored for `"manual"` and `"exhaustive"` degree selection.

`degree.min` optional scalar or integer vector giving lower bounds for automatic degree search when `degree.select != "manual"`.

`degree.restarts` non-negative integer giving the number of additional deterministic coordinate-search restarts. Ignored for `"manual"` and `"exhaustive"` degree selection.

`degree.select` character string controlling local-polynomial degree handling when `regtype="lp"`. `"manual"` (default) treats `degree` as fixed. `"coordinate"` performs cached coordinate-wise search over admissible degree vectors for the continuous `z` variables. `"exhaustive"` evaluates the full admissible degree grid when `search.engine="cell"`. For NOMAD-based search engines, any non-`"manual"` value requests direct joint search over degree and bandwidth coordinates.

`degree.start` optional starting degree vector for automatic coordinate search. If omitted, the search starts from the degree-zero local-constant baseline on the continuous `z` variables.

`degree.verify` logical value indicating whether a coordinate-search solution should be exhaustively verified over the admissible degree grid after the heuristic phase completes. Available only for `search.engine="cell"`.

Backfitting Controls: These controls tune the optional smooth-coefficient backfitting iterations.

`backfit.iterate` boolean value specifying whether or not to iterate evaluations of the smooth coefficient estimator, for extra accuracy, during the cross-validated backfitting procedure. Defaults to `FALSE`.

`backfit.maxiter`
: integer specifying the maximum number of times to iterate the evaluation of the smooth coefficient estimator in the attempt to obtain the desired accuracy. Defaults to `100`.

`backfit.tol`
: tolerance to determine convergence of iterated evaluations of the smooth coefficient estimator. Defaults to `.Machine$double.eps`.

Bandwidth Criterion And Representation: These arguments choose the selection criterion and the way continuous bandwidths are represented.

`bwmethod`
: which method was used to select bandwidths. `cv.ls` specifies least-squares cross-validation, which is all that is currently supported. Defaults to `cv.ls`.

`bwscaling`
: a logical value that when set to `TRUE` the supplied bandwidths are interpreted as 'scale factors' (c_j), otherwise when the value is `FALSE` they are interpreted as 'raw bandwidths' (h_j for continuous data types, λ_j for discrete data types). For continuous data types, c_j and h_j are related by the formula $h_j = c_j \sigma_j n^{-1/(2P+l)}$, where σ_j is an adaptive measure of spread of continuous variable j defined as min(standard deviation, mean absolute deviation, interquartile range/1.349), n the number of observations, P the order of the kernel, and l the number of continuous variables. For discrete data types, c_j and h_j are related by the formula $h_j = c_j n^{-2/(2P+l)}$, where here j denotes discrete variable j. Defaults to `FALSE`.

`bwtype`
: character string used for the continuous variable bandwidth type, specifying the type of bandwidth provided. Defaults to `fixed`. Option summary:
`fixed`: fixed bandwidths or scale factors
`generalized_nn`: generalized nearest neighbors
`adaptive_nn`: adaptive nearest neighbors

Categorical Search Initialization: These controls set categorical search starts.

`dfac.init`
: deterministic fixed-bandwidth start factor for ordered and unordered categorical coordinates. Used only when `bwtype="fixed"`. Defaults to `1.0`. Values must not exceed `2`.

`hbd.init`
: upper bound for random fixed-bandwidth start factors for ordered and unordered categorical coordinates. Used only when `bwtype="fixed"`. Defaults to `1.5`. Must be greater than or equal to `lbd.init` and must not exceed `2`.

`lbd.init`
: lower bound for random fixed-bandwidth start factors for ordered and unordered categorical coordinates. Used only when `bwtype="fixed"`. Defaults to `0.5`. Values must not exceed `2` so that categorical fixed starts remain within lawful bandwidth bounds.

Continuous Kernel Support Controls: These controls choose and parameterize bounded support for continuous kernels.

`ckerbound`
: character string controlling continuous-kernel support handling. Can be set as `none` (default kernel on full support), `range` (use sample min/max), or `fixed` (use `ckerlb`/`ckerub`). The bounded-kernel route reuses the selected continuous kernel and renormalizes it on the chosen support; see `np.kernels`.

`ckerlb` numeric scalar/vector of lower bounds for continuous variables used when `ckerbound="fixed"`. Must satisfy lower-bound validity for each continuous variable (e.g., `<= min(variable)`). Use `-Inf` for unbounded below. See `np.kernels` for bounded-kernel normalization details.

`ckerub` numeric scalar/vector of upper bounds for continuous variables used when `ckerbound="fixed"`. Must satisfy upper-bound validity for each continuous variable (e.g., `>= max(variable)`). Use `Inf` for unbounded above. See `np.kernels` for bounded-kernel normalization details.

Continuous Scale-Factor Search Initialization: These controls define deterministic and random continuous scale-factor starts and the lower admissibility floor for fixed-bandwidth search.

`scale.factor.init`
deterministic initial scale factor for continuous fixed-bandwidth search. Defaults to `0.5`. The value supplied by the user is not rewritten, but the effective first start passed to the optimizer is `max(scale.factor.init, scale.factor.search.lower)`. See Details.

`scale.factor.init.lower`
lower endpoint for random continuous scale-factor starts. Defaults to `0.1`. The value supplied by the user is not rewritten, but the effective random-start lower endpoint is `max(scale.factor.init.lower, scale.factor.search.lower)`. See Details.

`scale.factor.init.upper`
upper endpoint for random continuous scale-factor starts. Defaults to `2.0`. It must be greater than or equal to the effective lower endpoint, `max(scale.factor.init.lower, scale.factor.search.lower)`; otherwise bandwidth search errors rather than silently expanding the interval. See Details.

`scale.factor.search.lower`
optional nonnegative scalar giving the hard lower admissibility bound for continuous fixed-bandwidth search candidates. Defaults to `NULL`. If `NULL`, an existing bandwidth object's stored value is inherited when available; otherwise the package default `0.1` is used. This floor applies to computed/search bandwidth candidates and to effective search starts only. It does not rewrite explicit bandwidths supplied for storage with `bandwidth.compute = FALSE`. Final fixed-bandwidth search candidates must also have a finite valid raw objective value.

Cross-Validation Iteration Controls: These controls tune iterative cross-validation behavior.

`cv.iterate` boolean value specifying whether or not to perform iterative, cross-validated backfitting on the data. See details for limitations of the backfitting procedure. Defaults to `FALSE`.

`cv.num.iterations`
integer specifying the number of times to iterate the backfitting process over all covariates. Defaults to `1`.

Kernel Type Controls: These controls choose continuous, unordered, and ordered kernels.

`ckerorder` numeric value specifying kernel order (one of `(2,4,6,8)`). Kernel order specified along with a `uniform` continuous kernel type will be ignored. Defaults to `2`.

ckertype character string used to specify the continuous kernel type. Can be set as gaussian, epanechnikov, or uniform. Defaults to gaussian.

okertype character string used to specify the ordered categorical kernel type. Can be set as wangvanryzin, liracine, or racineliyan. Defaults to liracine.

ukertype character string used to specify the unordered categorical kernel type. Can be set as aitchisonaitken or liracine. Defaults to aitchisonaitken.

Local-Polynomial Model Specification: These arguments control the local-polynomial estimator, basis, and fixed degree specification.

basis for regtype="lp", the polynomial basis family used for local polynomial fitting. Options are "glp" (default), "additive", and "tensor".

bernstein.basis for regtype="lp", logical flag selecting Bernstein-basis representation where supported. When automatic degree search is requested and bernstein.basis is not explicitly supplied, the search route defaults to TRUE for numerical stability. Explicit bernstein.basis=FALSE is honored, but raw-polynomial search can be poorly conditioned at higher degrees.

degree for regtype="lp", polynomial degree specification for each continuous z variable.

regtype a character string specifying local smoothing type for the z surface. Options are "lc" (default), "ll", and "lp".

NOMAD Search Controls: These arguments control the optional NOMAD direct-search route for local-polynomial degree and bandwidth search.

nomad logical or character shortcut for the recommended automatic local-polynomial NOMAD route. When TRUE, any missing values among regtype, search.engine, degree.select, bernstein.basis, degree.min, degree.max, degree.verify, and bwtype are filled with regtype="lp", search.engine="nomad+powell", degree.select="coordinate", bernstein.basis=TRUE, degree.min=0L, degree.max=10L, degree.verify=FALSE, and bwtype="fixed". Explicit incompatible settings error immediately; in particular, explicit nomad=TRUE currently requires regtype="lp", bwtype="fixed", automatic degree search, no explicit degree, and search.engine %in% c("nomad", "nomad+powell"). This shortcut does not change the meaning of nmulti or nomad.nmulti: nmulti remains the outer restart count, while nomad.nmulti controls inner native **crs** NOMAD API multistarts within each outer restart. Returned bandwidth objects retain this normalized preset metadata in bw$nomad.shortcut for a returned object bw; when available, nomad.time records the direct-search component. The legacy field powell.time records the subsequent family-native R optim refinement selected by optim.method.

nomad.nmulti non-negative integer controlling the inner the native **crs** NOMAD C API multistart count used within each outer NOMAD restart when regtype="lp" and automatic degree search uses search.engine="nomad" or "nomad+powell". Defaults to 0L, which preserves the current one-start-per- restart behavior. This does not replace nmulti: nmulti controls outer restarts, while nomad.nmulti controls inner NOMAD multistarts within each outer restart.

`nomad.opts` optional named list of native NOMAD parameters forwarded to the **crs** NOMAD C API for NOMAD-backed bandwidth searches. `NULL` is treated as no extra NOMAD options. Non-list values and invalid or unsupported parameters are rejected before solver entry rather than silently ignored. Routes that evaluate R callbacks require serial callback evaluation and reject parallel callback settings such as `NB_THREADS_PARALLEL_EVAL > 1`.

`nomad.remin` logical flag controlling the optional second NOMAD hot start. When `TRUE`, NOMAD is restarted once from the best full candidate found, including both bandwidth and degree coordinates. Defaults to `FALSE`; current simulation evidence favors the one-pass NOMAD default for routine use, while leaving this switch available for sensitivity checks.

`search.engine` character string controlling the automatic local-polynomial search backend when `regtype="lp"` and `degree.select != "manual"`. The compatibility value `"nomad+powell"` (default) performs direct joint search over the `zdat`-side continuous bandwidth coordinates and degree vector using the native **crs** NOMAD C API, then applies one family-native R `optim` hot start from the NOMAD solution using `optim.method`. The value is retained for API compatibility; `"nomad"` omits this R-optimizer refinement. `"cell"` profiles the criterion over the admissible degree grid using repeated fixed-degree bandwidth solves. NOMAD-based search supports `bwtype="fixed"`, `bwtype="generalized_nn"`, and `bwtype="adaptive_nn"`; it currently requires `degree.verify=FALSE` and the suggested package **crs** to be installed.

Numerical Search Controls: These controls set search restart behavior.

`nmulti` integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points. Defaults to `min(2,ncol(xdat))`.

`powell.remin` compatibility argument shared with bandwidth families that use the Powell direction-set solver. Smooth-coefficient fixed-degree and post-NOMAD refinement use R `optim` with `optim.method`; consequently, this argument does not alter their refinement behavior.

Optimization Controls: These arguments control outer optimization behavior for the semiparametric search.

`optim.abstol` the absolute convergence tolerance used by `optim`. Only useful for non-negative functions, as a tolerance for reaching zero. Defaults to `.Machine$double.eps`.

`optim.maxattempts`
maximum number of attempts taken trying to achieve successful convergence in `optim`. Defaults to `100`.

`optim.maxit` maximum number of iterations used by `optim`. Defaults to `500`.

`optim.method` method used by `optim` for minimization of the objective function. See `?optim` for references. Defaults to `"Nelder-Mead"`.

the default method is an implementation of that of Nelder and Mead (1965), that uses only function values and is robust but relatively slow. It will work reasonably well for non-differentiable functions.

method `"BFGS"` is a quasi-Newton method (also known as a variable metric algorithm), specifically that published simultaneously in 1970 by Broyden, Fletcher,

Goldfarb and Shanno. This uses function values and gradients to build up a picture of the surface to be optimized.

method "CG" is a conjugate gradients method based on that by Fletcher and Reeves (1964) (but with the option of Polak-Ribiere or Beale-Sorenson updates). Conjugate gradient methods will generally be more fragile than the BFGS method, but as they do not store a matrix they may be successful in much larger optimization problems.

`optim.reltol` relative convergence tolerance used by `optim`. The algorithm stops if it is unable to reduce the value by a factor of 'reltol * (abs(val) + reltol)' at a step. Defaults to `sqrt(.Machine$double.eps)`, typically about `1e-8`.

`random.seed` an integer used to seed R's random number generator. This ensures replicability of the numerical search. Defaults to 42.

Additional Arguments: These arguments collect remaining controls passed through S3 methods.

`...` additional arguments supplied to specify the regression type, bandwidth type, kernel types, selection methods, and so on, detailed below.

Details

The `scale.factor.*` controls are dimensionless search controls. The package converts scale factors to bandwidths using the estimator-specific scaling encoded in the bandwidth object, including kernel order and the number of continuous variables relevant for the estimator. Users should not pre-multiply these controls by sample-size or standard-deviation factors.

`scale.factor.init` controls the deterministic first search start when that control is exposed. `scale.factor.init.lower` and `scale.factor.init.upper` define the random multistart interval when exposed. `scale.factor.search.lower` is the lower admissibility bound for continuous fixed-bandwidth search candidates. The effective first start is `max(scale.factor.init, scale.factor.search.lower)` when both controls are present, and the effective random-start lower endpoint is `max(scale.factor.init.lower, scale.factor.search.lower)`. `scale.factor.init.upper` must be at least that effective lower endpoint; the package errors rather than silently expanding the user's interval.

When `scale.factor.search.lower` is `NULL`, an existing bandwidth object's stored floor is inherited when available; otherwise the package default `0.1` is used. Explicit bandwidths supplied for storage with `bandwidth.compute = FALSE` are not rewritten by the search floor.

Categorical search-start controls such as `dfac.init`, `lbd.init`, and `hbd.init` have separate semantics and are not affected by `scale.factor.search.lower`.

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, `plot`, `plot.np` for plotting options, and `npRmpi.init` for interactive/cluster MPI startup. See `npRmpi.init` details for performance tradeoffs (message passing/startup mode) and the `inst/Rprofile` manual-broadcast template.

For S3 plotting help, see `plot.np`. You can list available plot methods with `methods("plot")`.

`npscoefbw` implements a variety of methods for semiparametric regression on multivariate ($p + q$-variate) explanatory data defined over a set of possibly continuous data. The approach is based on Li and Racine (2003) who employ 'generalized product kernels' that admit a mix of continuous and discrete data types.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i, when estimating the density at the point x. Generalized nearest-neighbor bandwidths change with the point at which the density is estimated, x. Fixed bandwidths are constant over the support of x.

npscoefbw may be invoked *either* with a formula-like symbolic description of variables on which bandwidth selection is to be performed *or* through a simpler interface whereby data is passed directly to the function via the xdat, ydat, and zdat parameters. Use of these two interfaces is **mutually exclusive**.

Data contained in the data frame xdat may be continuous and in zdat may be of mixed type. Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see npRmpi for details).

Data for which bandwidths are to be estimated may be specified symbolically. A typical description has the form dependent data ~ parametric explanatory data | nonparametric explanatory data, where dependent data is a univariate response, and parametric explanatory data and nonparametric explanatory data are both series of variables specified by name, separated by the separation character '+'. For example, y1 ~ x1 + x2 | z1 specifies that the bandwidth object for the smooth coefficient model with response y1, linear parametric regressors x1 and x2, and nonparametric regressor (that is, the slope-changing variable) z1 is to be estimated. See below for further examples. In the case where the nonparametric (slope-changing) variable is not specified, it is assumed to be the same as the parametric variable.

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth order Gaussian and Epanechnikov kernels, and the uniform kernel. Unordered discrete data types use a variation on Aitchison and Aitken's (1976) kernel, while ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

Setting nomad=TRUE is a convenience preset for this automatic LP route, not a generic optimizer alias. For smooth coefficient regression it expands any missing values to the equivalent long-form call

```
npscoefbw(...,
          regtype = "lp",
          search.engine = "nomad+powell",
          degree.select = "coordinate",
          bernstein.basis = TRUE,
          degree.min = 0L,
          degree.max = 10L,
          degree.verify = FALSE,
          bwtype = "fixed")
```

Compatible explicit tuning arguments are respected. Incompatible explicit settings fail fast so the shortcut never silently changes user-selected semantics. The character value nomad="auto" applies the same LP shortcut but leaves search.engine and degree.select eligible for data-driven resolution when both were not supplied explicitly: scalar continuous degree searches use the exhaustive degree-grid route, while higher-dimensional degree searches keep the compatibility "nomad+powell" route. Explicit search.engine or degree.select choices are honored.

When regtype="lp" and degree.select != "manual", npscoefbw can jointly determine the zdat-side local polynomial degree vector together with the associated bandwidth coordinates. With search.engine="cell", the criterion is profiled over the admissible degree grid using cached coordinate-wise or exhaustive search together with repeated fixed-degree bandwidth solves. With search.engine="nomad" or "nomad+powell", the criterion is optimized directly over the joint degree/bandwidth space using the native **crs** NOMAD C API. For this family, the compatibility value "nomad+powell" then performs one family-native R optim hot start using optim.method and keeps the better of the direct NOMAD and refined answers. This polynomial-adaptive joint-search route is motivated by Hall and Racine (2015). When bernstein.basis is not explicitly supplied, the automatic search route defaults to bernstein.basis=TRUE for numerical stability.

Value

if bwtype is set to fixed, an object containing bandwidths (or scale factors if bwscaling = TRUE) is returned. If it is set to generalized_nn or adaptive_nn, then instead the kth nearest neighbors are returned for the continuous variables while the discrete kernel bandwidths are returned for the discrete variables. Bandwidths are stored in a vector under the component name bw. Backfitted bandwidths are stored under the component name bw.fitted.

The functions predict, summary, and plot support objects of this class.

Book And Method Pointers

npscoefbw selects bandwidths for the smooth-coefficient model $Y = X'\beta(Z) + \epsilon$. The selected bandwidths control the mixed-data smoothing over Z used to estimate the coefficient functions $\beta_j(z)$.

For book-length derivations, see Li and Racine (2007), Chapter 9 *Additive and Smooth (Varying) Coefficient Semiparametric Models*, especially Sections 9.3-9.3.4, and Racine (2019), Chapter 8 *Semiparametric Conditional Mean Function Estimation*.

Usage Issues

If you are using data of mixed types, then it is advisable to use the data.frame function to construct your input data and not cbind, since cbind will typically not work as intended on mixed data types and will coerce the data to the same type.

Caution: multivariate data-driven bandwidth selection methods are, by their nature, *computationally intensive*. Virtually all methods require dropping the ith observation from the data set, computing an object, repeating this for all observations in the sample, then averaging each of these leave-one-out estimates for a *given* value of the bandwidth vector, and only then repeating this a large number of times in order to conduct multivariate numerical minimization/maximization. Furthermore, due to the potential for local minima/maxima, *restarting this procedure a large number of times may often be necessary*. This can be frustrating for users possessing large datasets. For exploratory purposes, you may wish to override the default search tolerances, say, setting optim.reltol=.1 and conduct multistarting (the default is to restart min(2,ncol(zdat)) times). Once the procedure terminates, you can restart search with default tolerances using those bandwidths obtained from the less rigorous search (i.e., set bws=bw on subsequent calls to this routine where bw is the initial bandwidth object). A version of this package using the Rmpi wrapper is under development that allows one to deploy this software in a clustered computing environment to facilitate computation involving large datasets.

Support for backfitted bandwidths is experimental and is limited in functionality. The code does not support asymptotic standard errors or out of sample estimates with backfitting.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

Cai Z. (2007), "Trending time-varying coefficient time series models with serially correlated errors," Journal of Econometrics, 136, 163-188.

Hastie, T. and R. Tibshirani (1993), "Varying-coefficient models," Journal of the Royal Statistical Society, B 55, 757-796.

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

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

Li, Q. and J.S. Racine (2010), "Smooth varying-coefficient estimation and inference for qualitative and quantitative data," Econometric Theory, 26, 1-31.

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

Li, Q. and D. Ouyang and J.S. Racine (2013), "Categorical semiparametric varying-coefficient models," Journal of Applied Econometrics, 28, 551-589.

Li, A. and Q. Li and J.S. Racine (under revision), "Boundary Adjusted, Polynomial Adaptive, Nonparametric Kernel Conditional Density Estimation," Econometric Reviews.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

np.kernels, np.options, plot, plot.np npregbw, npreg

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
```

```
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)
set.seed(42)

n <- 500

x <- runif(n)
z <- runif(n, min=-2, max=2)
y <- x*exp(z)*(1.0+rnorm(n,sd = 0.2))

## A smooth coefficient model example

bw <- npscoefbw(y~x|z)

summary(bw)

model <- npscoef(bws=bw, gradients=TRUE)

summary(model)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

npsdeptest *Kernel Consistent Serial Dependence Test for Univariate Nonlinear Processes*

Description

npsdeptest implements the consistent metric entropy test of nonlinear serial dependence as described in Granger, Maasoumi and Racine (2004).

Usage

```
npsdeptest(data = NULL,
           lag.num = 1,
           method = c("integration","summation"),
           bootstrap = TRUE,
           boot.num = 399,
           random.seed = 42)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the data series, lag count, and statistic variant.

data	a vector containing the variable that can be of type numeric or ts.
lag.num	an integer value specifying the maximum number of lags to use. Defaults to 1.
method	a character string used to specify whether to compute the integral version or the summation version of the statistic. Can be set as integration or summation (see below for details). Defaults to integration.

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

boot.num	an integer value specifying the number of bootstrap replications to use. Defaults to 399.
bootstrap	a logical value which specifies whether to conduct the bootstrap test or not. If set to FALSE, only the statistic will be computed. Defaults to TRUE.
random.seed	an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.

Details

Documentation guide: see np.kernels for kernels, np.options for global options, plot for plotting options, and npRmpi.init for interactive/cluster MPI startup. See npRmpi.init details for performance tradeoffs (message passing/startup mode) and the inst/Rprofile manual-broadcast template.

npsdeptest computes the nonparametric metric entropy (normalized Hellinger of Granger, Maasoumi and Racine (2004)) for testing for nonlinear serial dependence, $D[f(y_t, \hat{y}_{t-k}), f(y_t) \times$

$f(\hat{y}_{t-k})]$. Default bandwidths are of the Kullback-Leibler variety obtained via likelihood cross-validation.

The test may be applied to a raw data series or to residuals of user estimated models.

The summation version of this statistic may be numerically unstable when `data` is sparse (the summation version involves division of densities while the integration version involves differences). Warning messages are produced should this occur ('integration recommended') and should be heeded.

Value

`npsdeptest` returns an object of type `deptest` with the following components

`Srho`	the statistic vector `Srho`
`Srho.cumulant`	the cumulant statistic vector `Srho.cumulant`
`Srho.bootstrap.mat`	contains the bootstrap replications of `Srho`
`Srho.cumulant.bootstrap.mat`	contains the bootstrap replications of `Srho.cumulant`
`P`	the P-value vector of the Srho statistic vector
`P.cumulant`	the P-value vector of the cumulant Srho statistic vector
`bootstrap`	a logical value indicating whether bootstrapping was performed
`boot.num`	number of bootstrap replications
`lag.num`	the number of lags
`bw.y`	the numeric vector of bandwidths for `data` marginal density at lag `num.lag`
`bw.y.lag`	the numeric vector of bandwidths for lagged `data` marginal density at lag `num.lag`
`bw.joint`	the numeric matrix of bandwidths for `data` and lagged `data` joint density at lag `num.lag`

`summary` supports object of type `deptest`.

Book And Method Pointers

`npsdeptest` applies the dependence-test idea to a time series by comparing the joint law of current and lagged values with the product of their marginal laws. The lag and bootstrap choices determine the serial-dependence alternatives being probed and the null calibration.

For book-length background, see Li and Racine (2007), Chapter 13 *Nonsmoothing Tests*, especially Sections 13.4 and 13.5, and Chapter 12 *Model Specification Tests* for related distributional tests. For entropy and density context, see Racine (2019), Chapter 2.

Usage Issues

The `integration` version of the statistic uses multidimensional numerical methods from the **cubature** package. See **adaptIntegrate** for details. The `integration` version of the statistic will be substantially slower than the `summation` version, however, it will likely be both more accurate and powerful.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Granger, C.W. and E. Maasoumi and J.S. Racine (2004), "A dependence metric for possibly non-linear processes", Journal of Time Series Analysis, 25, 649-669.

See Also

np.kernels, np.options, plot npdeptest,npdeneqtest,npsymtest,npunitest

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

set.seed(42)

ar.series <- function(phi,epsilon) {
  n <- length(epsilon)
  series <- numeric(n)
  series[1] <- epsilon[1]/(1-phi)
  for(i in 2:n) {
    series[i] <- phi*series[i-1] + epsilon[i]
  }
  return(series)
}
```

```
n <- 100

yt <- ar.series(0.95,rnorm(n))

output <- npsdeptest(yt,
                     lag.num=2,
                     boot.num=29,
                     method="summation")

summary(output)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

npseed *Set Random Seed*

Description

npseed is a function which sets the random seed in the npRmpi C backend, resetting the random number generator.

Usage

```
npseed(seed)
```

Arguments

Seed Input: Seed value used to reset the package C backend random number generator.

seed	an integer seed for the random number generator.

Details

Documentation guide: see np.kernels for kernels, np.options for global options, plot for plotting options, and npRmpi.init for interactive/cluster MPI startup. See npRmpi.init details for performance tradeoffs (message passing/startup mode) and the inst/Rprofile manual-broadcast template.

npseed provides an interface for setting the random seed (and resetting the random number generator) used by npRmpi. The random number generator is used during the bandwidth search procedure to set the search starting point, and in subsequent searches when using multistarting, to avoid being trapped in local minima if the objective function is not globally concave.

Calling npseed will only affect the numerical search if it is performed by the C backend. The affected functions include: npudensbw, npcdensbw, npregbw, npplregbw, npqreg, npcmstest (via npregbw), npqcmstest (via npregbw), npsigtest (via npregbw).

Value

None.

Note

This method currently only supports objects from the npRmpi library.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

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

See Also

np.kernels, np.options, plot set.seed

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.
```

```
force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

npseed(712)

x <- runif(10)
y <- x + rnorm(10, sd = 0.1)

bw <- npregbw(y~x)

summary(bw)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

npsemihat *Experimental Hat Operators for Semiparametric Estimators*

Description

Constructs hat operators for semiparametric estimators so that fitted values or bootstrap draws can be computed by matrix application in one step. These interfaces are currently experimental.

Usage

```
npindexhat(bws,
           txdat = stop("training data 'txdat' missing"),
           exdat = txdat,
           y = NULL,
           output = c("matrix", "apply", "constraint"),
           s = 0L,
           fd.step = NULL,
           ...)

npplreghat(bws,
           txdat = stop("training data 'txdat' missing"),
           tzdat = stop("training data 'tzdat' missing"),
           exdat = txdat,
           ezdat = tzdat,
           y = NULL,
           output = c("apply", "matrix", "constraint"),
           ...)

npscoefhat(bws,
           txdat = stop("training data 'txdat' missing"),
           tzdat = NULL,
           exdat = txdat,
           ezdat = tzdat,
           y = NULL,
           output = c("matrix", "apply", "constraint"),
           ridge = 0,
           iterate = FALSE,
           leave.one.out = FALSE,
           ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the fitted bandwidth object, training data, and evaluation data.

`bws`	A fitted bandwidth object. `npindexhat()` requires class `sibandwidth`, `npplreghat()` requires class `plbandwidth`, and `npscoefhat()` requires class `scbandwidth`.
`exdat`	Evaluation `x` data.
`txdat`	Training `x` data.

Operator Output And Derivatives: These arguments control operator output, derivative selection, finite-difference compatibility, and apply-mode right-hand sides.

`fd.step`	Compatibility argument for `npindexhat(..., s=1)`. If supplied it must be a positive finite scalar, but the current `s=1` route uses the canonical exact derivative operator rather than finite-differencing the fit operator.

`output` Either `"matrix"` for the hat matrix, `"apply"` for direct application to `y`, or `"constraint"` for the row-weighted transpose `t(H) * y`. Note that `npplreghat()` retains `"apply"` as its default for historical compatibility.

`s` For `npindexhat`, `s=0` returns fit operator and `s=1` returns index-derivative operator.

`y` Optional response vector or matrix for apply mode. For `output = "constraint"`, `y` is required and must be a vector or one-column object.

Partially Linear And Smooth-Coefficient Data: These arguments supply the additional `z` data used by partially linear and smooth-coefficient models.

`ezdat` Evaluation `z` data.

`tzdat` Training `z` data for partially linear and smooth-coefficient models.

Smooth-Coefficient Controls: These arguments control smooth-coefficient iteration, leave-one-out behavior, and ridge stabilization.

`iterate` Logical; `npscoefhat()` currently supports `iterate = FALSE` only.

`leave.one.out` Logical; leave-one-out kernel weights for `npscoefhat`. This currently requires evaluation `z` data to match training `z` data.

`ridge` Base ridge term for local linear solves in `npscoefhat`; must be a non-negative finite scalar. The ridge sequence starts at `0` (no regularization) and then increments by `1/n.train` as needed for stable solves.

Additional Arguments: Reserved for future extensions.

`...` Reserved for future extensions.

Details

These operators are intended for fixed-X workflows such as one-shot wild bootstrap calculations where many response draws are projected through the same operator. The implementation is intentionally conservative: class and scalar argument contracts are validated explicitly, and unsupported iterative `npscoefhat()` paths fail fast. `npscoefhat()` inherits `regtype`/LP-basis controls from the supplied `scbandwidth` object. For non-fixed `npscoef` bootstrap plotting, these operators can support a frozen approximation, but they do not remove the need to recompute the local smooth-coefficient vector itself for each resample: the local weighted systems depend on the resample weights/counts at each evaluation point, so unlike `npplreg` there is no single global coefficient vector to update once per draw.

Method-specific argument map: `npindexhat()` uses `s`; `fd.step` is accepted for compatibility but the current `s=1` route uses the canonical exact derivative operator; `npplreghat()` and `npscoefhat()` use `tzdat`/`ezdat`; `npscoefhat()` additionally uses `ridge`, `iterate`, and `leave.one.out`.

For `output = "constraint"`, the selected operator is first defined exactly as in `output = "matrix"` and the return value is `t(H) * y`. This is a convenience route for constrained quadratic-programming workflows; it does not solve a constrained estimation problem and it does not change the meaning of `output = "apply"`.

Value

If `output = "matrix"`, returns a hat matrix H. If `output = "apply"`, returns Hy (or HY for matrix right-hand-side input). If `output = "constraint"`, returns the constraint design matrix $\mathrm{diag}(y)H'$ as represented in R by `t(H) * y`.

Examples

```
## Not run:
npRmpi.init(nslaves = 1)
set.seed(42)
n <- 100
x <- runif(n)
z <- runif(n)
y <- sin(2*pi*x) + 0.5 * z + rnorm(n, sd = 0.1)

tx <- data.frame(x = x)
tz <- data.frame(z = z)

ibw <- npindexbw(xdat = data.frame(x, x2 = x^2), ydat = y,
                 bws = c(0.5, 1.0, 1.0), bandwidth.compute = FALSE)
iH <- npindexhat(bws = ibw, txdat = data.frame(x, x2 = x^2), output = "matrix")
iH.fitted <- iH
iA <- npindexhat(bws = ibw, txdat = data.frame(x, x2 = x^2),
                 y = y, output = "constraint")
all.equal(iA, t(iH) * y)
ifit <- npindex(bws = ibw, txdat = data.frame(x, x2 = x^2), tydat = y)
head(cbind(fitted(ifit), iH.fitted), n = 2L)

pbw <- npplregbw(xdat = tx, zdat = tz, ydat = y,
                 bws = matrix(c(0.2, 0.2), nrow = 2L, ncol = 1L),
                 bandwidth.compute = FALSE)
pH <- npplreghat(bws = pbw, txdat = tx, tzdat = tz, output = "matrix")
pH.fitted <- pH
pfit <- npplreg(bws = pbw, txdat = tx, tydat = y, tzdat = tz)
head(cbind(fitted(pfit), pH.fitted), n = 2L)

sbw <- npscoefbw(xdat = tx, zdat = tz, ydat = y,
                 bws = 0.2, bandwidth.compute = FALSE)
sH <- npscoefhat(bws = sbw, txdat = tx, tzdat = tz,
                 output = "matrix", iterate = FALSE)
sH.fitted <- sH
sfit <- npscoef(bws = sbw, txdat = tx, tydat = y, tzdat = tz,
                iterate = FALSE)
head(cbind(fitted(sfit), sH.fitted), n = 2L)

npRmpi.quit()

## End(Not run)
```

npsigtest *Kernel Regression Significance Test with Mixed Data Types*

Description

npsigtest implements a consistent test of significance of an explanatory variable(s) in a nonparametric regression setting that is analogous to a simple t-test (F-test) in a parametric regression setting. The test is based on Racine, Hart, and Li (2006) and Racine (1997).

Usage

```
npsigtest(bws,
          ...)

## S3 method for class 'formula'
npsigtest(bws,
          data = NULL,
          ...)

## S3 method for class 'npregression'
npsigtest(bws,
          ...)

## Default S3 method:
npsigtest(bws,
          xdat,
          ydat,
          ...)

## S3 method for class 'rbandwidth'
npsigtest(bws,
          xdat = stop("data xdat missing"),
          ydat = stop("data ydat missing"),
          boot.num = 399,
          boot.method = c("iid","wild","wild-rademacher","pairwise"),
          boot.type = c("I","II"),
          pivot=TRUE,
          joint=FALSE,
          index = seq_len(ncol(xdat)),
          random.seed = 42,
          ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth object, formula/data interface, and explicit data inputs.

bws — a bandwidth specification. This can be set as a rbandwidth object returned from a previous invocation, or as a vector of bandwidths, with each element i corresponding to the bandwidth for column i in xdat. In either case, the bandwidth supplied will serve as a starting point in the numerical search for optimal bandwidths when using boot.type="II". If specified as a vector, then additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, selection methods, and so on.

data — an optional data frame, list or environment (or object coercible to a data frame by as.data.frame) containing the variables in the model. If not found in data, the variables are taken from environment(bws), typically the environment from which npregbw was called.

xdat — a p-variate data frame of explanatory data (training data) used to calculate the regression estimators.

ydat — a one (1) dimensional numeric or integer vector of dependent data, each element i corresponding to each observation (row) i of xdat.

Bootstrap And Test Controls: These arguments control bootstrap execution, tested effects, and reproducibility settings.

boot.method — a character string used to specify the bootstrap method for determining the null distribution. pairwise resamples pairwise, while the remaining methods use a residual bootstrap procedure. iid will generate independent identically distributed draws. wild will use a wild bootstrap. wild-rademacher will use a wild bootstrap with Rademacher variables. Defaults to iid.

boot.num — an integer value specifying the number of bootstrap replications to use. Defaults to 399.

boot.type — a character string specifying whether to use a 'Bootstrap I' or 'Bootstrap II' method (see Racine, Hart, and Li (2006) for details). The 'Bootstrap II' method re-runs cross-validation for each bootstrap replication and uses the new cross-validated bandwidth for variable i and the original ones for the remaining variables. Defaults to boot.type="I".

index — a vector of indices for the columns of xdat for which the test of significance is to be conducted. Defaults to (1,2,...,p) where p is the number of columns in xdat.

joint — a logical value which specifies whether to conduct a joint test or individual test. This is to be used in conjunction with index where index contains two or more integers corresponding to the variables being tested, where the integers correspond to the variables in the order in which they appear among the set of explanatory variables in the function call to npreg/npregbw. Defaults to FALSE.

pivot — a logical value which specifies whether to bootstrap a pivotal statistic or not (pivoting is achieved by dividing gradient estimates by their asymptotic standard errors). Defaults to TRUE.

random.seed — an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.

Additional Arguments: Further arguments are passed to the bandwidth-selection routines used by the test.

... additional arguments supplied to specify the bandwidth type, kernel types, selection methods, and so on, detailed below.

Details

Documentation guide: see np.kernels for kernels, np.options for global options, plot for plotting options, and npRmpi.init for interactive/cluster MPI startup. See npRmpi.init details for performance tradeoffs (message passing/startup mode) and the inst/Rprofile manual-broadcast template.

npsigtest implements a variety of methods for computing the null distribution of the test statistic and allows the user to investigate the impact of a variety of default settings including whether or not to pivot the statistic (pivot), whether pairwise or residual resampling is to be used (boot.method), and whether or not to recompute the bandwidths for the variables being tested (boot.type), among others.

Defaults are chosen so as to provide reasonable behaviour in a broad range of settings and this involves a trade-off between computational expense and finite-sample performance. However, the default boot.type="I", though computationally expedient, can deliver a test that can be slightly over-sized in small sample settings (e.g. at the 5% level the test might reject 8% of the time for samples of size $n = 100$ for some data generating processes). If the default setting (boot.type="I") delivers a P-value that is in the neighborhood (i.e. slightly smaller) of any classical level (e.g. 0.05) and you only have a modest amount of data, it might be prudent to re-run the test using the more computationally intensive boot.type="II" setting to confirm the original result. Note also that boot.method="pairwise" is not recommended for the multivariate local linear estimator due to substantial size distortions that may arise in certain cases.

For npRmpi, npsigtest supports direct execution when autodispatch is enabled (default via npRmpi.init()). In that mode, users can call npregbw, npreg, and npsigtest directly without wrapping calls in mpi.bcast.cmd(...).

Value

npsigtest returns an object of type sigtest. summary supports sigtest objects. It has the following components:

In	the vector of statistics In
P	the vector of P-values for each statistic in In
In.bootstrap	contains a matrix of the bootstrap replications of the vector In, each column corresponding to replications associated with explanatory variables in xdat indexed by index (e.g., if you selected index = c(1,4) then In.bootstrap will have two columns, the first being the bootstrap replications of In associated with variable 1, the second with variable 4).

Book And Method Pointers

npsigtest assesses predictor relevance in nonparametric regression by comparing the fitted unrestricted regression with restrictions implied by excluding or smoothing over selected regressors. Bootstrap choices trade computational cost against finite-sample behavior.

For book-length background, see Li and Racine (2007), Chapter 12 *Model Specification Tests*, especially Sections 12.3 and 12.3.5, and Chapter 13 *Nonsmoothing Tests*, especially Section 13.3 where

relevant. For the underlying conditional-mean setting, see Racine (2019), Chapter 6 *Conditional Mean Function Estimation*.

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

When reusing a bandwidth object without explicit training or evaluation data, the required data are recovered from object-retained state or from the stored call environment when available. This is a convenience for ordinary same-session use, not a guarantee that a saved object is self-contained. For reproducible replay after `saveRDS()` and `readRDS()` in a fresh R session, supply the required data arguments explicitly.

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`. A version of this package using the `Rmpi` wrapper is under development that allows one to deploy this software in a clustered computing environment to facilitate computation involving large datasets.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

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

Racine, J.S., J. Hart, and Q. Li (2006), "Testing the significance of categorical predictor variables in nonparametric regression models," Econometric Reviews, 25, 523-544.

Racine, J.S. (1997), "Consistent significance testing for nonparametric regression," Journal of Business and Economic Statistics 15, 369-379.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

`npRmpi.init`

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
```

```
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

set.seed(42)

## Significance testing with z irrelevant

n <- 250

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

model <- npreg(y~z+x1+x2,
               regtype="ll",
               bwmethod="cv.aic")

output <- npsigtest(model,boot.num=29)

summary(output)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.
```

```
npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

npsymtest *Kernel Consistent Density Asymmetry Test with Mixed Data Types*

Description

npsymtest implements the consistent metric entropy test of asymmetry as described in Maasoumi and Racine (2009).

Usage

```
npsymtest(data = NULL,
          method = c("integration","summation"),
          boot.num = 399,
          bw = NULL,
          boot.method = c("iid", "geom"),
          random.seed = 42,
          ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the data, statistic variant, and any supplied bandwidth.

bw	a numeric (scalar) bandwidth. Defaults to plug-in (see details below).
data	a vector containing the variable.
method	a character string used to specify whether to compute the integral version or the summation version of the statistic. Can be set as integration or summation (see below for details). Defaults to integration.

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

boot.method	a character string used to specify the bootstrap method. Can be set as iid or geom (see below for details). Defaults to iid.
boot.num	an integer value specifying the number of bootstrap replications to use. Defaults to 399.
random.seed	an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.

Additional Arguments: Further arguments are passed to the bandwidth-selection routines used by the test.

... additional arguments supplied to specify the bandwidth type, kernel types, and so on. This is used since we specify bw as a numeric scalar and not a bandwidth object, and is of interest if you do not desire the default behaviours. To change the defaults, you may specify any of bwscaling, bwtype, ckertype, ckerorder, ukertype, okertype.

Details

Documentation guide: see np.kernels for kernels, np.options for global options, plot for plotting options, and npRmpi.init for interactive/cluster MPI startup. See npRmpi.init details for performance tradeoffs (message passing/startup mode) and the inst/Rprofile manual-broadcast template.

npsymtest computes the nonparametric metric entropy (normalized Hellinger of Granger, Maasoumi and Racine (2004)) for testing symmetry using the densities/probabilities of the data and the rotated data, $D[f(y), f(\tilde{y})]$. See Maasoumi and Racine (2009) for details. Default bandwidths are of the plug-in variety (bw.SJ for continuous variables and direct plug-in for discrete variables).

For bootstrapping the null distribution of the statistic, iid conducts simple random resampling, while geom conducts Politis and Romano's (1994) stationary bootstrap using automatic block length selection via the b.star function in the npRmpi package. See the **boot** package for details.

The summation version of this statistic may be numerically unstable when y is sparse (the summation version involves division of densities while the integration version involves differences). Warning messages are produced should this occur ('integration recommended') and should be heeded.

Value

npsymtest returns an object of type symtest with the following components

Srho	the statistic Srho
Srho.bootstrap	contains the bootstrap replications of Srho
P	the P-value of the statistic
boot.num	number of bootstrap replications
data.rotate	the rotated data series
bw	the numeric (scalar) bandwidth

summary supports object of type symtest.

Book And Method Pointers

npsymtest tests symmetry by comparing the density or probability function of the observed data with that of the data reflected around the relevant center. The same entropy-distance logic used by the univariate density tests underlies the statistic.

For book-length background, see Racine (2019), Chapter 2 *Continuous Density and Cumulative Distribution Functions* and Li and Racine (2007), Chapter 13 *Nonsmoothing Tests* and Chapter 12 *Model Specification Tests* where related density-feature tests are treated.

Usage Issues

When using data of type factor it is crucial that the variable not be an alphabetic character string (i.e. the factor must be integer-valued). The rotation is conducted about the median after conversion to type numeric which is then converted back to type factor. Failure to do so will have unpredictable results. See the example below for proper usage.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Granger, C.W. and E. Maasoumi and J.S. Racine (2004), "A dependence metric for possibly nonlinear processes", Journal of Time Series Analysis, 25, 649-669.

Maasoumi, E. and J.S. Racine (2009), "A robust entropy-based test of asymmetry for discrete and continuous processes," Econometric Reviews, 28, 246-261.

Politis, D.N. and J.P. Romano (1994), "The stationary bootstrap," Journal of the American Statistical Association, 89, 1303-1313.

See Also

np.kernels, np.options, plot npdeneqtest,npdeptest,npsdeptest,npunitest

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
```

```
npRmpi.init(nslaves=1)

set.seed(42)

## A function to create a time series

ar.series <- function(phi,epsilon) {
  n <- length(epsilon)
  series <- numeric(n)
  series[1] <- epsilon[1]/(1-phi)
  for(i in 2:n) {
    series[i] <- phi*series[i-1] + epsilon[i]
  }
  return(series)
}

n <- 250

## Stationary persistent symmetric time-series

yt <- ar.series(0.5,rnorm(n))

## A simple example of the test for symmetry

output <- npsymtest(yt,
                    boot.num=29,
                    boot.method="geom",
                    method="summation")

summary(output)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

nptgauss *Truncated Second-order Gaussian Kernels*

Description

nptgauss provides an interface for setting the truncation radius of the truncated second-order Gaussian kernel used by **npRmpi**.

Usage

```
nptgauss(b)
```

Arguments

Kernel Truncation: Truncation radius for the truncated Gaussian kernel helper.

b	Truncation radius of the kernel.

Details

Documentation guide: see np.kernels for kernels, np.options for global options, plot for plotting options, and npRmpi.init for interactive/cluster MPI startup. See npRmpi.init details for performance tradeoffs (message passing/startup mode) and the inst/Rprofile manual-broadcast template. For interactive and cluster batch workflows, see npRmpi.init.

nptgauss allows one to set the truncation radius of the truncated Gaussian kernel used by **npRmpi**, which defaults to 3. It automatically computes the constants describing the truncated gaussian kernel for the user.

We define the truncated gaussion kernel on the interval $[-b, b]$ as:

$$K = \frac{\alpha}{\sqrt{2\pi}} \left(e^{-z^2/2} - e^{-b^2/2} \right)$$

The constant α is computed as:

$$\alpha = \left[\int_{-b}^{b} \frac{1}{\sqrt{2\pi}} \left(e^{-z^2/2} - e^{-b^2/2} \right) \right]^{-1}$$

Given these definitions, the derivative kernel is simply:

$$K' = (-z) \frac{\alpha}{\sqrt{2\pi}} e^{-z^2/2}$$

The CDF kernel is:

$$G = \frac{\alpha}{2} \mathrm{erf}(z/\sqrt{2}) + \frac{1}{2} - c_0 z$$

The convolution kernel on $[-2b, 0]$ has the general form:

$$H_- = a_0 \,\mathrm{erf}(z/2 + b) e^{-z^2/4} + a_1 z + a_2 \,\mathrm{erf}((z + b)/\sqrt{2}) - c_0$$

and on $[0, 2b]$ it is:

$$H_+ = -a_0 \operatorname{erf}(z/2 - b)e^{-z^2/4} - a_1 z - a_2 \operatorname{erf}((z-b)/\sqrt{2}) - c_0$$

where a_0 is determined by the normalisation condition on H, a_2 is determined by considering the value of the kernel at $z = 0$ and a_1 is determined by the requirement that $H = 0$ at $[-2b, 2b]$.

Value

No return value, called for side effects (sets kernel constants in the **npRmpi** C backend).

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

See Also

npRmpi.init for MPI startup and workflow guidance.

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The default kernel, a gaussian truncated at +- 3
nptgauss(b = 3.0)

## End(Not run)
```

---

npudens *Kernel Density Estimation with Mixed Data Types*

Description

npudens computes kernel unconditional density estimates on evaluation data, given a set of training data and a bandwidth specification (a bandwidth object or a bandwidth vector, bandwidth type, and kernel type) using the method of Li and Racine (2003).

Usage

```
npudens(bws,
        ...)

## S3 method for class 'formula'
npudens(bws,
        data = NULL,
        newdata = NULL,
        ...)
```

```
## S3 method for class 'bandwidth'
npudens(bws,
        tdat = stop("invoked without training data 'tdat'"),
        edat,
        ...)

## Default S3 method:
npudens(bws,
        tdat,
        ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and training data.

`bws` a bandwidth specification. This can be set as a `bandwidth` object returned from an invocation of `npudensbw`, or as a p-vector of bandwidths, with an element for each variable in the training data. If specified as a vector, then additional arguments will need to be supplied as necessary to change them from the defaults to specify the bandwidth type, kernel types, training data, and so on.

`data` an optional data frame, list or environment (or object coercible to a data frame by `as.data.frame`) containing the variables in the model. If not found in data, the variables are taken from `environment(bws)`, typically the environment from which `npudensbw` was called.

`tdat` a p-variate data frame of sample realizations (training data) used to estimate the density. Defaults to the corresponding training data recoverable from the bandwidth object in the current session.

Evaluation Data: These arguments control where the fitted density is evaluated.

`edat` a p-variate data frame of density evaluation points. By default, evaluation takes place on the data provided by `tdat`.

`newdata` An optional data frame in which to look for evaluation data. If omitted, the training data are used.

Additional Arguments: Further arguments are passed to `npudensbw` when bandwidths are computed internally, or used to interpret a numeric `bws` vector.

`...` additional arguments supplied to `npudensbw` when `npudens` computes bandwidths internally, or arguments needed to interpret a numeric `bws` vector. This is where bandwidth selection controls such as `bwmethod`, `bwtype`, `bwscaling`, kernel/support controls such as `ckertype`, `ckerorder`, and `ckerbound`, categorical kernel controls such as `ukertype` and `okertype`, and search controls such as `nmulti` and `scale.factor.search.lower` are supplied. See `npudensbw` for the complete bandwidth-selection argument surface.

Details

Documentation guide: see npudensbw for bandwidth selection and search controls, np.kernels for kernels, np.options for global options, plot, plot.np for plotting options, and npRmpi.init for interactive/cluster MPI startup. See npRmpi.init details for performance tradeoffs (message passing/startup mode) and the inst/Rprofile manual-broadcast template.

When bws is omitted, the formula and default methods call npudensbw first and pass bandwidth-selection arguments from ... to that call. When bws is already a bandwidth object, npudens estimates with the stored bandwidth metadata in that object.

Argument groups for bandwidth selection are documented on npudensbw. The most common workflow is to choose data and bandwidth inputs first, then bandwidth criterion and representation, then kernel/support controls and numerical search controls if defaults need to be changed.

For S3 plotting help, see plot.np. You can list available plot methods with methods("plot").

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

```
Usage 1: first compute the bandwidth object via npudensbw and then
compute the density:

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.
npRmpi.init(nslaves=1)
bw <- npudensbw(formula = ~y, data = mydat)
fhat <- npudens(bw)
npRmpi.quit()

Usage 2: alternatively, compute the bandwidth object indirectly:

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.
npRmpi.init(nslaves=1)
fhat <- npudens(formula = ~y, data = mydat)
npRmpi.quit()

Usage 3: modify the default kernel and order:

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.
npRmpi.init(nslaves=1)
fhat <- npudens(formula = ~y, data = mydat, ckertype="epanechnikov", ckerorder=4)
npRmpi.quit()
```

```
Usage 4: use the data frame interface rather than the formula
interface:

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.
npRmpi.init(nslaves=1)
fhat <- npudens(tdat = y, ckertype="epanechnikov", ckerorder=4)
npRmpi.quit()
```

npudens implements a variety of methods for estimating multivariate density functions (p-variate) defined over a set of possibly continuous and/or discrete (unordered, ordered) data. The approach is based on Li and Racine (2003) who employ 'generalized product kernels' that admit a mix of continuous and discrete data types.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i, when estimating the density at the point x. Generalized nearest-neighbor bandwidths change with the point at which the density is estimated, x. Fixed bandwidths are constant over the support of x.

Data contained in the data frame tdat (and also edat) may be a mix of continuous (default), unordered discrete (to be specified in the data frame tdat using the factor command), and ordered discrete (to be specified in the data frame tdat using the ordered command). Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see npRmpi for details).

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth order Gaussian and Epanechnikov kernels, and the uniform kernel. Unordered discrete data types use a variation on Aitchison and Aitken's (1976) kernel, while ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

Value

npudens returns a npdensity object. The generic accessor functions fitted, and se, extract estimated values and asymptotic standard errors on estimates, respectively, from the returned object. Furthermore, the functions predict, summary and plot support objects of both classes. The returned objects have the following components:

eval	the evaluation points.
dens	estimation of the density at the evaluation points
derr	standard errors of the density estimates
log_likelihood	log likelihood of the density estimates

Book And Method Pointers

The unconditional density target is $f(x)$. In the continuous fixed-bandwidth case the estimator has the usual kernel form $\hat{f}(x) = n^{-1} \sum_i K_h(X_i, x)$; with mixed data, npudens uses generalized

product kernels so that the continuous, unordered, and ordered components each contribute their appropriate kernel factor. Nearest-neighbor bandwidth types change how the continuous bandwidth is indexed, not the statistical target.

For book-length derivations, see Li and Racine (2007), Chapter 1 *Density Estimation*, especially Sections 1.1, 1.3, 1.6, 1.8, and 1.9, and Chapter 4 *Kernel Estimation with Mixed Data*. The later workflow treatment is Racine (2019), Chapter 2 *Continuous Density and Cumulative Distribution Functions* and Chapter 3 *Mixed-Data Probability Density and Cumulative Distribution Functions*.

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

When reusing a bandwidth object without explicit training or evaluation data, the required data are recovered from object-retained state or from the stored call environment when available. This is a convenience for ordinary same-session use, not a guarantee that a saved object is self-contained. For reproducible replay after `saveRDS()` and `readRDS()` in a fresh R session, supply the required data arguments explicitly.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), " Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

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

Li, Q. and J.S. Racine (2003), "Nonparametric estimation of distributions with categorical and continuous data," Journal of Multivariate Analysis, 86, 266-292.

Ouyang, D. and Q. Li and J.S. Racine (2006), "Cross-validation and the estimation of probability distributions with categorical data," Journal of Nonparametric Statistics, 18, 69-100.

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

Scott, D.W. (1992), *Multivariate Density Estimation: Theory, Practice and Visualization,* New York: Wiley.

Silverman, B.W. (1986), *Density Estimation,* London: Chapman and Hall.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

`np.kernels`, `np.options`, `plot`, `plot.np` `npudensbw` , `density`

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

data("Italy")

bw <- npudensbw(formula=~year+gdp, data=Italy)

fhat <- npudens(bws=bw)

summary(fhat)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
```

```
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

npudensbw *Kernel Density Bandwidth Selection with Mixed Data Types*

Description

npudensbw computes a bandwidth object for a p-variate kernel unconditional density estimator defined over mixed continuous and discrete (unordered, ordered) data using either the normal reference rule-of-thumb, likelihood cross-validation, or least-squares cross validation using the method of Li and Racine (2003).

Usage

```
npudensbw(...)

## S3 method for class 'formula'
npudensbw(formula,
          data,
          subset,
          na.action,
          call,
          ...)

## S3 method for class 'bandwidth'
npudensbw(dat = stop("invoked without input data 'dat'"),
          bws,
          bandwidth.compute = TRUE,
          cfac.dir = 2.5*(3.0-sqrt(5)),
          scale.factor.init = 0.5,
          dfac.dir = 0.25*(3.0-sqrt(5)),
          dfac.init = 0.375,
          dfc.dir = 3,
          ftol = 1.490116e-07,
          scale.factor.init.upper = 2.0,
          hbd.dir = 1,
          hbd.init = 0.9,
          initc.dir = 1.0,
          initd.dir = 1.0,
          invalid.penalty = c("baseline","dbmax"),
          itmax = 10000,
          lbc.dir = 0.5,
          scale.factor.init.lower = 0.1,
          lbd.dir = 0.1,
```

```
            lbd.init = 0.1,
            nmulti,
            penalty.multiplier = 10,
            powell.remin = TRUE,
            bwsolver = c("powell", "mads", "mads+powell"),
            scale.init.categorical.sample = FALSE,
            scale.factor.search.lower = NULL,
            small = 1.490116e-05,
            tol = 1.490116e-04,
            transform.bounds = FALSE,
            eval.only = FALSE,
            ...,
            nomad.opts = list())

## Default S3 method:
npudensbw(dat = stop("invoked without input data 'dat'"),
            bws,
            bandwidth.compute = TRUE,
            bwmethod,
            bwscaling,
            bwtype,
            cfac.dir,
            scale.factor.init,
            ckerbound,
            ckerlb,
            ckerorder,
            ckertype,
            ckerub,
            dfac.dir,
            dfac.init,
            dfc.dir,
            ftol,
            scale.factor.init.upper,
            hbd.dir,
            hbd.init,
            initc.dir,
            initd.dir,
            invalid.penalty,
            itmax,
            lbc.dir,
            scale.factor.init.lower,
            lbd.dir,
            lbd.init,
            nmulti,
            okertype,
            penalty.multiplier,
            powell.remin,
            bwsolver = c("powell", "mads", "mads+powell"),
```

```
scale.init.categorical.sample,
scale.factor.search.lower = NULL,
small,
tol,
transform.bounds,
ukertype,
...,
nomad.opts = list())
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the data, formula interface, and whether bandwidths are supplied or computed.

`bandwidth.compute`
: a logical value which specifies whether to do a numerical search for bandwidths or not. If set to `FALSE`, a `bandwidth` object will be returned with bandwidths set to those specified in `bws`. Defaults to `TRUE`.

`bws`
: a bandwidth specification. This can be set as a bandwidth object returned from a previous invocation, or as a vector of bandwidths, with each element i corresponding to the bandwidth for column i in `dat`. In either case, the bandwidth supplied will serve as a starting point in the numerical search for optimal bandwidths. If specified as a vector, then additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, selection methods, and so on. This can be left unset.

`call`
: the original function call. This is passed internally by `npRmpi` when a bandwidth search has been implied by a call to another function. It is not recommended that the user set this.

`dat`
: a p-variate data frame on which bandwidth selection will be performed. The data types may be continuous, discrete (unordered and ordered factors), or some combination thereof.

`data`
: an optional data frame, list or environment (or object coercible to a data frame by `as.data.frame`) containing the variables in the model. If not found in data, the variables are taken from `environment(formula)`, typically the environment from which the function is called.

`formula`
: a symbolic description of variables on which bandwidth selection is to be performed. The details of constructing a formula are described below.

`na.action`
: a function which indicates what should happen when the data contain `NA`s. The default is set by the `na.action` setting of options, and is `na.fail` if that is unset. The (recommended) default is `na.omit`.

`subset`
: an optional vector specifying a subset of observations to be used in the fitting process.

Bandwidth Criterion And Representation: These arguments choose the selection criterion and the way continuous bandwidths are represented.

`bwmethod` a character string specifying the bandwidth selection method. `cv.ml` specifies likelihood cross-validation, `cv.ls` specifies least-squares cross-validation, and `normal-reference` just computes the 'rule-of-thumb' bandwidth h_j using the standard formula $h_j = 1.06\sigma_j n^{-1/(2P+l)}$, where σ_j is an adaptive measure of spread of the jth continuous variable defined as min(standard deviation, mean absolute deviation/1.4826, interquartile range/1.349), n the number of observations, P the order of the kernel, and l the number of continuous variables. Note that when there exist factors and the normal-reference rule is used, there is zero smoothing of the factors, corresponding to exact cell-indicator kernels for categorical coordinates. Defaults to `cv.ml`.

`bwscaling` a logical value that when set to `TRUE` the supplied bandwidths are interpreted as 'scale factors' (c_j), otherwise when the value is `FALSE` they are interpreted as 'raw bandwidths' (h_j for continuous data types, λ_j for discrete data types). For continuous data types, c_j and h_j are related by the formula $h_j = c_j\sigma_j n^{-1/(2P+l)}$, where σ_j is an adaptive measure of spread of the jth continuous variable defined as min(standard deviation, mean absolute deviation/1.4826, interquartile range/1.349), n the number of observations, P the order of the kernel, and l the number of continuous variables. For discrete data types, c_j and h_j are related by the formula $h_j = c_j n^{-2/(2P+l)}$, where here j denotes discrete variable j. Defaults to `FALSE`.

`bwtype` character string used for the continuous variable bandwidth type, specifying the type of bandwidth to compute and return in the `bandwidth` object. Defaults to `fixed`. Option summary:
`fixed`: compute fixed bandwidths
`generalized_nn`: compute generalized nearest neighbors
`adaptive_nn`: compute adaptive nearest neighbors

Categorical Search Initialization: These controls set categorical search starts and categorical direction-set initialization.

`dfac.dir` stretch factor for direction set search for Powell's algorithm for categorical variables. See Details

`dfac.init` non-random initial values for scale factors for categorical variables for Powell's algorithm. See Details

`hbd.dir` upper bound for direction set search for Powell's algorithm for categorical variables. See Details

`hbd.init` upper bound for scale factors for categorical variables for Powell's algorithm. See Details

`initd.dir` initial non-random values for direction set search for Powell's algorithm for categorical variables. See Details

`lbd.dir` lower bound for direction set search for Powell's algorithm for categorical variables. See Details

`lbd.init` lower bound for scale factors for categorical variables for Powell's algorithm. See Details

`scale.init.categorical.sample`
a logical value that when set to `TRUE` scales `lbd.dir`, `hbd.dir`, `dfac.dir`, and `initd.dir` by $n^{-2/(2P+l)}$, n the number of observations, P the order of the kernel, and l the number of `numeric` variables. See Details

Continuous Direction-Set Search Controls: These controls set Powell direction-set initialization for continuous variables.

`cfac.dir` stretch factor for direction set search for Powell's algorithm for `numeric` variables. See Details

`dfc.dir` chi-square degrees of freedom for direction set search for Powell's algorithm for `numeric` variables. See Details

`initc.dir` initial non-random values for direction set search for Powell's algorithm for `numeric` variables. See Details

`lbc.dir` lower bound for direction set search for Powell's algorithm for `numeric` variables. See Details

Continuous Kernel Support Controls: These controls choose and parameterize bounded support for continuous kernels.

`ckerbound` character string controlling continuous-kernel support handling. Can be set as `none` (default kernel on full support), `range` (use sample min/max), or `fixed` (use `ckerlb`/`ckerub`). The bounded-kernel route reuses the selected continuous kernel and renormalizes it on the chosen support; see `np.kernels`.

`ckerlb` numeric scalar/vector of lower bounds for continuous variables used when `ckerbound="fixed"`. Must satisfy lower-bound validity for each continuous variable (e.g., `<= min(variable)`). Use `-Inf` for unbounded below. See `np.kernels` for bounded-kernel normalization details.

`ckerub` numeric scalar/vector of upper bounds for continuous variables used when `ckerbound="fixed"`. Must satisfy upper-bound validity for each continuous variable (e.g., `>= max(variable)`). Use `Inf` for unbounded above. See `np.kernels` for bounded-kernel normalization details.

Continuous Scale-Factor Search Initialization: These controls define deterministic and random continuous scale-factor starts and the lower admissibility floor for fixed-bandwidth search.

`scale.factor.init`
deterministic initial scale factor for continuous fixed-bandwidth search. Defaults to `0.5`. The value supplied by the user is not rewritten, but the effective first start passed to the optimizer is `max(scale.factor.init, scale.factor.search.lower)`. See Details.

`scale.factor.init.lower`
lower endpoint for random continuous scale-factor starts. Defaults to `0.1`. The value supplied by the user is not rewritten, but the effective random-start lower endpoint is `max(scale.factor.init.lower, scale.factor.search.lower)`. See Details.

`scale.factor.init.upper`
upper endpoint for random continuous scale-factor starts. Defaults to `2.0`. It must be greater than or equal to the effective lower endpoint, `max(scale.factor.init.lower, scale.factor.search.lower)`; otherwise bandwidth search errors rather than silently expanding the interval. See Details.

scale.factor.search.lower
: optional nonnegative scalar giving the hard lower admissibility bound for continuous fixed-bandwidth search candidates. Defaults to NULL. If NULL, an existing bandwidth object's stored value is inherited when available; otherwise the package default 0.1 is used. This floor applies to computed/search bandwidth candidates and to effective search starts only. It does not rewrite explicit bandwidths supplied for storage with bandwidth.compute = FALSE. Final fixed-bandwidth search candidates must also have a finite valid raw objective value.

Kernel Type Controls: These controls choose continuous, unordered, and ordered kernels.

ckerorder
: numeric value specifying kernel order (one of (2,4,6,8)). Kernel order specified along with a uniform continuous kernel type will be ignored. Defaults to 2.

ckertype
: character string used to specify the continuous kernel type. Can be set as gaussian, epanechnikov, or uniform. Defaults to gaussian.

okertype
: character string used to specify the ordered categorical kernel type. Can be set as wangvanryzin, liracine, or racineliyan. Defaults to liracine.

ukertype
: character string used to specify the unordered categorical kernel type. Can be set as aitchisonaitken or liracine. Defaults to aitchisonaitken.

Numerical Search And Tolerance Controls: These controls set optimizer tolerances, restart behavior, invalid-candidate penalties, and bounded search transformations.

ftol
: fractional tolerance on the value of the cross-validation function evaluated at located minima (of order the machine precision or perhaps slightly larger so as not to be diddled by roundoff). Defaults to 1.490116e-07 (1.0e+01*sqrt(.Machine$double.eps)).

invalid.penalty
: a character string specifying the penalty used when the optimizer encounters invalid bandwidths. "baseline" returns a finite penalty based on a baseline objective; "dbmax" returns DBL_MAX. Defaults to "baseline".

itmax
: integer number of iterations before failure in the numerical optimization routine. Defaults to 10000.

nmulti
: integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points.

penalty.multiplier
: a numeric multiplier applied to the baseline penalty when invalid.penalty="baseline". Defaults to 10.

powell.remin
: logical flag controlling Powell restart-from-minimum behavior. When TRUE, the Powell-style search restarts from the located minimum for a minor gain in accuracy. Defaults to TRUE.

bwsolver
: bandwidth optimizer for fixed-degree searches. The default "powell" preserves the historical Powell route. "mads" uses the native **crs** NOMAD C API over the bandwidth coordinates, and "mads+powell" applies one Powell hot start from the MADS solution. Advanced native NOMAD parameters may be supplied through the explicit nomad.opts argument; invalid or unsupported parameters are rejected by NOMAD rather than silently ignored.

`nomad.opts` optional named list of native NOMAD parameters forwarded to the **crs** NOMAD C API for MADS/NOMAD-backed bandwidth searches. `NULL` is treated as no extra NOMAD options. Non-list values and unsupported native options are rejected before solver entry.

`small` a small number used to bracket a minimum (it is hopeless to ask for a bracketing interval of width less than sqrt(epsilon) times its central value, a fractional width of only about 10-04 (single precision) or 3x10-8 (double precision)). Defaults to `small = 1.490116e-05 (1.0e+03*sqrt(.Machine$double.eps))`.

`tol` tolerance on the position of located minima of the cross-validation function (tol should generally be no smaller than the square root of your machine's floating point precision). Defaults to `1.490116e-04 (1.0e+04*sqrt(.Machine$double.eps))`.

`transform.bounds`
a logical value that when set to `TRUE` applies an internal transformation that maps the unconstrained search to the feasible bandwidth domain. Defaults to `FALSE`.

`eval.only` internal logical flag used by solver wrappers to evaluate an existing bandwidth candidate without running a bandwidth search. Ordinary users should leave this at its default `FALSE`.

Additional Arguments: These arguments collect remaining controls passed through S3 methods.

`...` additional arguments supplied to specify the bandwidth type, kernel types, selection methods, and so on, detailed below.

Details

The `scale.factor.*` controls are dimensionless search controls. The package converts scale factors to bandwidths using the estimator-specific scaling encoded in the bandwidth object, including kernel order and the number of continuous variables relevant for the estimator. Users should not pre-multiply these controls by sample-size or standard-deviation factors.

`scale.factor.init` controls the deterministic first search start. `scale.factor.init.lower` and `scale.factor.init.upper` define the random multistart interval. `scale.factor.search.lower` is the lower admissibility bound for continuous fixed-bandwidth search candidates. The effective first start is `max(scale.factor.init, scale.factor.search.lower)`, and the effective random-start lower endpoint is `max(scale.factor.init.lower, scale.factor.search.lower)`. `scale.factor.init.upper` must be at least that effective lower endpoint; the package errors rather than silently expanding the user's interval.

When `scale.factor.search.lower` is `NULL`, an existing bandwidth object's stored floor is inherited when available; otherwise the package default `0.1` is used. Explicit bandwidths supplied for storage with `bandwidth.compute = FALSE` are not rewritten by the search floor.

Categorical search-start controls such as `dfac.init`, `lbd.init`, and `hbd.init` have separate semantics and are not affected by `scale.factor.search.lower`.

For `bwmethod="cv.ml"`, the ordinary contribution is based on $\log(\hat{f}_i)$. Compact-support kernels can yield zero leave-one-out density estimates, and higher-order kernels can yield negative estimates. In those cases the package optimizes a guarded CVML criterion rather than the literal log likelihood. Let ϵ denote the positive machine floor used for the guard. Positive fitted values use the ordinary branch $-\log(\hat{f}_i)$. Near-zero fitted values use $-\log(\epsilon)$, and negative fitted values use the

smooth penalty $\log(-\hat{f}_i) - 2\log(\epsilon)$. The guarded count reported in bandwidth summaries records objective evaluations in which this guard was used. The `bwmethod="cv.ls"` criterion remains well defined for negative kernel density estimates and is not modified by this CVML guard.

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, `plot`, `plot.np` for plotting options, and `npRmpi.init` for interactive/cluster MPI startup. See `npRmpi.init` details for performance tradeoffs (message passing/startup mode) and the `inst/Rprofile` manual-broadcast template.

The bandwidth-selection argument surface is easiest to read by decision group: data and existing bandwidth inputs; bandwidth criterion and representation; continuous kernel and support controls beginning with `cker*`; categorical kernel controls `ukertype` and `okertype`; and numerical search initialization, tolerances, and feasibility controls. Users who call `npudens` without a bandwidth object can pass these same bandwidth-selection controls through that function's `....`

For S3 plotting help, see `plot.np`. You can list available plot methods with `methods("plot")`.

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

```
Usage 1: compute a bandwidth object using the formula interface:

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.
npRmpi.init(nslaves=1)
bw <- npudensbw(formula = ~y, data = mydat)
npRmpi.quit()

Usage 2: compute a bandwidth object using the data frame interface
and change the default kernel and order:

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.
npRmpi.init(nslaves=1)
fhat <- npudensbw(tdat = y, ckertype="epanechnikov", ckerorder=4)
npRmpi.quit()
```

`npudensbw` implements a variety of methods for choosing bandwidths for multivariate (p-variate) distributions defined over a set of possibly continuous and/or discrete (unordered, ordered) data. The approach is based on Li and Racine (2003) who employ 'generalized product kernels' that admit a mix of continuous and discrete data types.

The cross-validation methods employ multivariate numerical search algorithms (direction set (Powell's) methods in multidimensions).

Bandwidths can (and will) differ for each variable which is, of course, desirable.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i, when estimating the density at the point x. Generalized nearest-neighbor bandwidths change with the point at which the density is estimated, x. Fixed bandwidths are constant over the support of x.

npudensbw may be invoked *either* with a formula-like symbolic description of variables on which bandwidth selection is to be performed *or* through a simpler interface whereby data is passed directly to the function via the dat parameter. Use of these two interfaces is **mutually exclusive**.

Data contained in the data frame dat may be a mix of continuous (default), unordered discrete (to be specified in the data frame dat using factor), and ordered discrete (to be specified in the data frame dat using ordered). Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see npRmpi for details).

Data for which bandwidths are to be estimated may be specified symbolically. A typical description has the form ~ data, where data is a series of variables specified by name, separated by the separation character '+'. For example, ~ x + y specifies that the bandwidths for the joint distribution of variables x and y are to be estimated. See below for further examples.

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth order Gaussian and Epanechnikov kernels, and the uniform kernel. Unordered discrete data types use a variation on Aitchison and Aitken's (1976) kernel, while ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

The optimizer invoked for search is Powell's conjugate direction method which requires the setting of (non-random) initial values and search directions for bandwidths, and, when restarting, random values for successive invocations. Bandwidths for numeric variables are scaled by robust measures of spread, the sample size, and the number of numeric variables where appropriate. Two sets of parameters for bandwidths for numeric can be modified, those for initial values for the parameters themselves, and those for the directions taken (Powell's algorithm does not involve explicit computation of the function's gradient). The default values are set by considering search performance for a variety of difficult test cases and simulated cases. We highly recommend restarting search a large number of times to avoid the presence of local minima (achieved by modifying nmulti). Further refinement for difficult cases can be achieved by modifying these sets of parameters. However, these parameters are intended more for the authors of the package to enable 'tuning' for various methods rather than for the user themselves.

Value

npudensbw returns a bandwidth object, with the following components:

bw	bandwidth(s), scale factor(s) or nearest neighbours for the data, dat
fval	objective function value at minimum

if bwtype is set to fixed, an object containing bandwidths, of class bandwidth (or scale factors if bwscaling = TRUE) is returned. If it is set to generalized_nn or adaptive_nn, then instead the kth nearest neighbors are returned for the continuous variables while the discrete kernel bandwidths are returned for the discrete variables. Bandwidths are stored under the component name bw, with each element i corresponding to column i of input data dat.

The functions predict, summary and plot support objects of type bandwidth.

Book And Method Pointers

npudensbw selects bandwidths for the unconditional density target $f(x)$ estimated by npudens. The continuous scale factors and discrete smoothing parameters are those used in the generalized product-kernel density estimator.

For book-length derivations, see Li and Racine (2007), Chapter 1 *Density Estimation*, especially Sections 1.3, 1.6, 1.8, and 1.9, and Chapter 4 *Kernel Estimation with Mixed Data*. The later workflow treatment is Racine (2019), Chapters 2 and 3.

Usage Issues

If you are using data of mixed types, then it is advisable to use the data.frame function to construct your input data and not cbind, since cbind will typically not work as intended on mixed data types and will coerce the data to the same type.

Caution: multivariate data-driven bandwidth selection methods are, by their nature, *computationally intensive*. Virtually all methods require dropping the ith observation from the data set, computing an object, repeating this for all observations in the sample, then averaging each of these leave-one-out estimates for a *given* value of the bandwidth vector, and only then repeating this a large number of times in order to conduct multivariate numerical minimization/maximization. Furthermore, due to the potential for local minima/maxima, *restarting this procedure a large number of times may often be necessary*. This can be frustrating for users possessing large datasets. For exploratory purposes, you may wish to override the default search tolerances, say, setting ftol=.01 and tol=.01 and conduct multistarting (the default is to restart min(2, ncol(dat)) times) as is done for a number of examples. Once the procedure terminates, you can restart search with default tolerances using those bandwidths obtained from the less rigorous search (i.e., set bws=bw on subsequent calls to this routine where bw is the initial bandwidth object). A version of this package using the Rmpi wrapper is under development that allows one to deploy this software in a clustered computing environment to facilitate computation involving large datasets.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and , C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

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

Li, Q. and J.S. Racine (2003), "Nonparametric estimation of distributions with categorical and continuous data," Journal of Multivariate Analysis, 86, 266-292.

Ouyang, D. and Q. Li and J.S. Racine (2006), "Cross-validation and the estimation of probability distributions with categorical data," Journal of Nonparametric Statistics, 18, 69-100.

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

Scott, D.W. (1992), *Multivariate Density Estimation. Theory, Practice and Visualization,* New York: Wiley.

Silverman, B.W. (1986), *Density Estimation,* London: Chapman and Hall.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

np.kernels, np.options, plot, plot.np bw.nrd, bw.SJ, hist, npudens, npudist

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

data("Italy")

bw <- npudensbw(formula=~year+gdp, data=Italy)

summary(bw)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
```

```
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

npudenshat *Unconditional Density Hat Operator*

Description

Constructs the unconditional density hat operator associated with npudens bandwidth objects. The returned operator maps a right-hand side y to Hy; with y = 1 this reproduces the fitted unconditional density.

Usage

```
npudenshat(bws,
           tdat = stop("training data 'tdat' missing"),
           edat,
           y = NULL,
           output = c("matrix", "apply"))
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the fitted bandwidth object, training data, and evaluation data.

bws	A fitted unconditional density bandwidth object of class "bandwidth".
edat	Optional evaluation data. If omitted, the operator is built on the training data.
tdat	Training data used to construct the operator.

Operator Output: These arguments control whether the operator is returned as a matrix or applied directly.

output	Either "matrix" for the hat matrix or "apply" for direct application to y.
y	Optional right-hand side vector or matrix with one row per training observation.

Details

For output = "matrix", the return value is a matrix with class c("npudenshat", "matrix") and attributes storing the bandwidth object, training data, evaluation data, and call metadata.

For output = "apply", the function returns H y directly. Matrix right-hand sides are applied columnwise.

This helper is intended for object-fed repeated evaluation once a bandwidth object has already been constructed. It does not perform bandwidth selection.

Value

Either a hat matrix of class "npudenshat" or the applied result H y, depending on output.

Examples

```
## Not run:
npRmpi.init(nslaves = 1)
data(cps71)
tx <- data.frame(age = cps71$age)

bw <- npudensbw(dat = tx, bwtype = "fixed",
                bandwidth.compute = FALSE, bws = 1.0)

H <- npudenshat(bws = bw, tdat = tx)
dens.hat <- npudenshat(bws = bw, tdat = tx,
                       y = rep(1, nrow(tx)),
                       output = "apply")
dens.core <- fitted(npudens(bws = bw, tdat = tx))

head(cbind(dens.core, dens.hat), n = 2L)

npRmpi.quit()

## End(Not run)
```

---

npudist *Kernel Distribution Estimation with Mixed Data Types*

Description

npudist computes kernel unconditional cumulative distribution estimates on evaluation data, given a set of training data and a bandwidth specification (a dbandwidth object or a bandwidth vector, bandwidth type, and kernel type) using the method of Li, Li and Racine (2017).

Usage

```
npudist(bws, ...)

## S3 method for class 'formula'
npudist(bws,
        data = NULL,
        newdata = NULL,
        ...)

## S3 method for class 'dbandwidth'
npudist(bws,
        tdat = stop("invoked without training data 'tdat'"),
        edat,
```

```
        ...)

## Default S3 method:
npudist(bws,
        tdat,
        ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the bandwidth specification, formula/data interface, and training data.

`bws` a `dbandwidth` specification. This can be set as a `dbandwidth` object returned from an invocation of `npudistbw`, or as a p-vector of bandwidths, with an element for each variable in the training data. If specified as a vector, then additional arguments will need to be supplied as necessary to change them from the defaults to specify the bandwidth type, kernel types, training data, and so on.

`data` an optional data frame, list or environment (or object coercible to a data frame by `as.data.frame`) containing the variables in the model. If not found in data, the variables are taken from `environment(bws)`, typically the environment from which `npudistbw` was called.

`tdat` a p-variate data frame of sample realizations (training data) used to estimate the cumulative distribution. Defaults to the corresponding training data recoverable from the bandwidth object in the current session.

Evaluation Data: These arguments control where the fitted cumulative distribution is evaluated.

`edat` a p-variate data frame of cumulative distribution evaluation points. By default, evaluation takes place on the data provided by `tdat`.

`newdata` An optional data frame in which to look for evaluation data. If omitted, the training data are used.

Additional Arguments: Further arguments are passed to `npudistbw` when bandwidths are computed internally, or used to interpret a numeric `bws` vector.

`...` additional arguments supplied to `npudistbw` when `npudist` computes bandwidths internally, or arguments needed to interpret a numeric `bws` vector. This is where bandwidth selection controls such as `bwmethod`, `bwtype`, `bwscaling`, kernel/support controls such as `ckertype`, `ckerorder`, and `ckerbound`, categorical kernel controls such as `okertype`, and search controls such as `nmulti`, `ngrid`, and `scale.factor.search.lower` are supplied. See `npudistbw` for the complete bandwidth-selection argument surface.

Details

Documentation guide: see `npudistbw` for bandwidth selection and search controls, `np.kernels` for kernels, `np.options` for global options, `plot`, `plot.np` for plotting options, and `npRmpi.init` for interactive/cluster MPI startup. See `npRmpi.init` details for performance tradeoffs (message passing/startup mode) and the `inst/Rprofile` manual-broadcast template.

When bws is omitted, the formula and default methods call npudistbw first and pass bandwidth-selection arguments from ... to that call. When bws is already a dbandwidth object, npudist estimates with the stored bandwidth metadata in that object.

With finite continuous bounds, npudist uses the observation-centered truncated-kernel CDF documented in np.kernels. It is exactly zero and one at the declared lower and upper endpoints and has the linear uniform-CDF limit as the bandwidth diverges.

Argument groups for bandwidth selection are documented on npudistbw. The most common workflow is to choose data and bandwidth inputs first, then bandwidth criterion and representation, then kernel/support controls, distribution-specific integral/grid controls, and numerical search controls if defaults need to be changed.

For S3 plotting help, see plot.np. You can list available plot methods with methods("plot").

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

```
Usage 1: first compute the bandwidth object via npudistbw and then
compute the cumulative distribution:

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.
npRmpi.init(nslaves=1)
bw <- npudistbw(formula = ~y, data = mydat)
Fhat <- npudist(bw)
npRmpi.quit()

Usage 2: alternatively, compute the bandwidth object indirectly:

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.
npRmpi.init(nslaves=1)
Fhat <- npudist(formula = ~y, data = mydat)
npRmpi.quit()

Usage 3: modify the default kernel and order:

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.
npRmpi.init(nslaves=1)
Fhat <- npudist(formula = ~y, data = mydat, ckertype="epanechnikov", ckerorder=4)
npRmpi.quit()

Usage 4: use the data frame interface rather than the formula
```

```
interface:

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.
npRmpi.init(nslaves=1)
Fhat <- npudist(tdat = y, ckertype="epanechnikov", ckerorder=4)
npRmpi.quit()
```

npudist implements a variety of methods for estimating multivariate cumulative distributions (p-variate) defined over a set of possibly continuous and/or discrete (ordered) data. The approach is based on Li and Racine (2003) who employ 'generalized product kernels' that admit a mix of continuous and discrete data types.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i, when estimating the cumulative distribution at the point x. Generalized nearest-neighbor bandwidths change with the point at which the cumulative distribution is estimated, x. Fixed bandwidths are constant over the support of x.

Data contained in the data frame tdat (and also edat) may be a mix of continuous (default) and ordered discrete (to be specified in the data frame tdat using the ordered command). Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see npRmpi for details).

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth-order Gaussian and Epanechnikov kernels, and the uniform kernel. Ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

Value

npudist returns a npdistribution object. The generic accessor functions fitted and se extract estimated values and asymptotic standard errors on estimates, respectively, from the returned object. Furthermore, the functions predict, summary and plot support objects of both classes. The returned objects have the following components:

eval	the evaluation points.
dist	estimate of the cumulative distribution at the evaluation points
derr	standard errors of the cumulative distribution estimates

Book And Method Pointers

The unconditional distribution target is $F(x) = \Pr(X \leq x)$. The estimator replaces the density kernel factors by integrated distribution-kernel factors where appropriate, while retaining the same mixed-data product-kernel logic for continuous and ordered components. The returned standard errors are asymptotic standard errors for the estimated cumulative distribution values.

For book-length derivations, see Li and Racine (2007), Chapter 1 *Density Estimation*, especially Sections 1.4 and 1.5 for CDF estimation and bandwidth selection, together with Chapter 4 *Kernel*

Estimation with Mixed Data. The later workflow treatment is Racine (2019), Chapter 2 *Continuous Density and Cumulative Distribution Functions* and Chapter 3 *Mixed-Data Probability Density and Cumulative Distribution Functions*.

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

When reusing a bandwidth object without explicit training or evaluation data, the required data are recovered from object-retained state or from the stored call environment when available. This is a convenience for ordinary same-session use, not a guarantee that a saved object is self-contained. For reproducible replay after `saveRDS()` and `readRDS()` in a fresh R session, supply the required data arguments explicitly.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), " Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

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

Li, Q. and J.S. Racine (2003), "Nonparametric estimation of distributions with categorical and continuous data," Journal of Multivariate Analysis, 86, 266-292.

Li, C. and H. Li and J.S. Racine (2017), "Cross-Validated Mixed Datatype Bandwidth Selection for Nonparametric Cumulative Distribution/Survivor Functions," Econometric Reviews, **36**, 970-987.

Ouyang, D. and Q. Li and J.S. Racine (2006), "Cross-validation and the estimation of probability distributions with categorical data," Journal of Nonparametric Statistics, 18, 69-100.

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

Scott, D.W. (1992), *Multivariate Density Estimation. Theory, Practice and Visualization,* New York: Wiley.

Silverman, B.W. (1986), *Density Estimation,* London: Chapman and Hall.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

`np.kernels`, `np.options`, `plot`, `plot.np` `npudistbw` , `density`

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

data("Italy")

bw <- npudistbw(formula=~ordered(year)+gdp,
                data=Italy)

Fhat <- npudist(bws=bw)

summary(Fhat)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
```

```
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

npudistbw *Kernel Distribution Bandwidth Selection with Mixed Data Types*

Description

npudistbw computes a bandwidth object for a p-variate kernel cumulative distribution estimator defined over mixed continuous and discrete (ordered) data using either the normal reference rule-of-thumb or least-squares cross validation using the method of Li, Li and Racine (2017).

Usage

```
npudistbw(...)

## S3 method for class 'formula'
npudistbw(formula,
          data, subset,
          na.action,
          call,
          gdata = NULL,
          ...)

## S3 method for class 'dbandwidth'
npudistbw(dat = stop("invoked without input data 'dat'"),
          bws,
          gdat = NULL,
          bandwidth.compute = TRUE,
          cfac.dir = 2.5*(3.0-sqrt(5)),
          scale.factor.init = 0.5,
          dfac.dir = 0.25*(3.0-sqrt(5)),
          dfac.init = 0.375,
          dfc.dir = 3,
          do.full.integral = FALSE,
          ftol = 1.490116e-07,
          scale.factor.init.upper = 2.0,
          hbd.dir = 1,
          hbd.init = 0.9,
          initc.dir = 1.0,
          initd.dir = 1.0,
          invalid.penalty = c("baseline","dbmax"),
          itmax = 10000,
          lbc.dir = 0.5,
```

```
            scale.factor.init.lower = 0.1,
            lbd.dir = 0.1,
            lbd.init = 0.1,
            memfac = 500.0,
            ngrid = 100,
            nmulti,
            penalty.multiplier = 10,
            powell.remin = TRUE,
            bwsolver = c("powell", "mads", "mads+powell"),
            scale.init.categorical.sample = FALSE,
            scale.factor.search.lower = NULL,
            small = 1.490116e-05,
            tol = 1.490116e-04,
            transform.bounds = FALSE,
            eval.only = FALSE,
            ...,
            nomad.opts = list())

## Default S3 method:
npudistbw(dat = stop("invoked without input data 'dat'"),
            bws,
            gdat,
            bandwidth.compute = TRUE,
            bwmethod,
            bwscaling,
            bwtype,
            cfac.dir,
            scale.factor.init,
            ckerbound,
            ckerlb,
            ckerorder,
            ckertype,
            ckerub,
            dfac.dir,
            dfac.init,
            dfc.dir,
            do.full.integral,
            ftol,
            scale.factor.init.upper,
            hbd.dir,
            hbd.init,
            initc.dir,
            initd.dir,
            invalid.penalty,
            itmax,
            lbc.dir,
            scale.factor.init.lower,
            lbd.dir,
```

```
          lbd.init,
          memfac,
          ngrid,
          nmulti,
          okertype,
          penalty.multiplier,
          powell.remin,
          bwsolver = c("powell", "mads", "mads+powell"),
          scale.init.categorical.sample,
          scale.factor.search.lower = NULL,
          small,
          tol,
          transform.bounds,
          ...,
          nomad.opts = list())
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the data, formula interface, optional integration grid, and whether bandwidths are supplied or computed.

`bandwidth.compute`
: a logical value which specifies whether to do a numerical search for bandwidths or not. If set to `FALSE`, a `bandwidth` object will be returned with bandwidths set to those specified in `bws`. Defaults to `TRUE`.

`bws`
: a bandwidth specification. This can be set as a bandwidth object returned from a previous invocation, or as a vector of bandwidths, with each element i corresponding to the bandwidth for column i in `dat`. In either case, the bandwidth supplied will serve as a starting point in the numerical search for optimal bandwidths. If specified as a vector, then additional arguments will need to be supplied as necessary to specify the bandwidth type, kernel types, selection methods, and so on. This can be left unset.

`call`
: the original function call. This is passed internally by `npRmpi` when a bandwidth search has been implied by a call to another function. It is not recommended that the user set this.

`dat`
: a p-variate data frame on which bandwidth selection will be performed. The data types may be continuous, discrete (ordered factors), or some combination thereof.

`data`
: an optional data frame, list or environment (or object coercible to a data frame by `as.data.frame`) containing the variables in the model. If not found in data, the variables are taken from `environment(formula)`, typically the environment from which the function is called.

`formula`
: a symbolic description of variables on which bandwidth selection is to be performed. The details of constructing a formula are described below.

`gdat`
: a grid of data on which the indicator function for least-squares cross-validation is to be computed (can be the sample or a grid of quantiles).

gdata — a grid of data on which the indicator function for least-squares cross-validation is to be computed (can be the sample or a grid of quantiles).

na.action — a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting of options, and is na.fail if that is unset. The (recommended) default is na.omit.

subset — an optional vector specifying a subset of observations to be used in the fitting process.

Bandwidth Criterion And Representation: These arguments choose the selection criterion and the way continuous bandwidths are represented.

bwmethod — a character string specifying the bandwidth selection method. cv.cdf specifies least-squares cross-validation for cumulative distribution functions (Li, Li and Racine (2017)), and normal-reference just computes the 'rule-of-thumb' bandwidth h_j using the standard formula $h_j = 1.587\sigma_j n^{-1/(P+l)}$, where σ_j is an adaptive measure of spread of the jth continuous variable defined as min(standard deviation, mean absolute deviation/1.4826, interquartile range/1.349), n the number of observations, P the order of the kernel, and l the number of continuous variables. Note that when there exist factors and the normal-reference rule is used, there is zero smoothing of the factors. Defaults to cv.cdf.

bwscaling — a logical value that when set to TRUE the supplied bandwidths are interpreted as 'scale factors' (c_j), otherwise when the value is FALSE they are interpreted as 'raw bandwidths' (h_j for continuous data types, λ_j for discrete data types). For continuous data types, c_j and h_j are related by the formula $h_j = c_j\sigma_j n^{-1/(P+l)}$, where σ_j is an adaptive measure of spread of the jth continuous variable defined as min(standard deviation, mean absolute deviation/1.4826, interquartile range/1.349), n the number of observations, P the order of the kernel, and l the number of continuous variables. For discrete data types, c_j and h_j are related by the formula $h_j = c_j n^{-2/(P+l)}$, where here j denotes discrete variable j. Defaults to FALSE.

bwtype — character string used for the continuous variable bandwidth type, specifying the type of bandwidth to compute and return in the bandwidth object. Defaults to fixed. Option summary:
fixed: compute fixed bandwidths
generalized_nn: compute generalized nearest neighbors
adaptive_nn: compute adaptive nearest neighbors

Categorical Search Initialization: These controls set categorical search starts and categorical direction-set initialization.

dfac.dir — stretch factor for direction set search for Powell's algorithm for categorical variables. See Details

dfac.init — non-random initial values for scale factors for categorical variables for Powell's algorithm. See Details

hbd.dir — upper bound for direction set search for Powell's algorithm for categorical variables. See Details

hbd.init — upper bound for scale factors for categorical variables for Powell's algorithm. See Details

`initd.dir` initial non-random values for direction set search for Powell's algorithm for categorical variables. See Details

`lbd.dir` lower bound for direction set search for Powell's algorithm for categorical variables. See Details

`lbd.init` lower bound for scale factors for categorical variables for Powell's algorithm. See Details

`scale.init.categorical.sample` a logical value that when set to `TRUE` scales `lbd.dir`, `hbd.dir`, `dfac.dir`, and `initd.dir` by $n^{-2/(2P+l)}$, n the number of observations, P the order of the kernel, and l the number of `numeric` variables. See Details

Continuous Direction-Set Search Controls: These controls set Powell direction-set initialization for continuous variables.

`cfac.dir` stretch factor for direction set search for Powell's algorithm for `numeric` variables. See Details

`dfc.dir` chi-square degrees of freedom for direction set search for Powell's algorithm for `numeric` variables. See Details

`initc.dir` initial non-random values for direction set search for Powell's algorithm for `numeric` variables. See Details

`lbc.dir` lower bound for direction set search for Powell's algorithm for `numeric` variables. See Details

Continuous Kernel Support Controls: These controls choose and parameterize bounded support for continuous kernels.

`ckerbound` character string controlling continuous-kernel support handling. Can be set as `none` (default kernel on full support), `range` (use sample min/max), or `fixed` (use `ckerlb`/`ckerub`). For distribution estimation the selected continuous kernel is integrated after observation-centered truncation and normalization on the chosen support; see `np.kernels`.

`ckerlb` numeric scalar/vector of lower bounds for continuous variables used when `ckerbound="fixed"`. Must satisfy lower-bound validity for each continuous variable (e.g., `<= min(variable)`). Use `-Inf` for unbounded below. See `np.kernels` for bounded-kernel normalization details.

`ckerub` numeric scalar/vector of upper bounds for continuous variables used when `ckerbound="fixed"`. Must satisfy upper-bound validity for each continuous variable (e.g., `>= max(variable)`). Use `Inf` for unbounded above. See `np.kernels` for bounded-kernel normalization details.

Continuous Scale-Factor Search Initialization: These controls define deterministic and random continuous scale-factor starts and the lower admissibility floor for fixed-bandwidth search.

`scale.factor.init` deterministic initial scale factor for continuous fixed-bandwidth search. Defaults to `0.5`. The value supplied by the user is not rewritten, but the effective first start passed to the optimizer is `max(scale.factor.init, scale.factor.search.lower)`. See Details.

`scale.factor.init.lower`
: lower endpoint for random continuous scale-factor starts. Defaults to `0.1`. The value supplied by the user is not rewritten, but the effective random-start lower endpoint is `max(scale.factor.init.lower, scale.factor.search.lower)`. See Details.

`scale.factor.init.upper`
: upper endpoint for random continuous scale-factor starts. Defaults to `2.0`. It must be greater than or equal to the effective lower endpoint, `max(scale.factor.init.lower, scale.factor.search.lower)`; otherwise bandwidth search errors rather than silently expanding the interval. See Details.

`scale.factor.search.lower`
: optional nonnegative scalar giving the hard lower admissibility bound for continuous fixed-bandwidth search candidates. Defaults to `NULL`. If `NULL`, an existing bandwidth object's stored value is inherited when available; otherwise the package default `0.1` is used. This floor applies to computed/search bandwidth candidates and to effective search starts only. It does not rewrite explicit bandwidths supplied for storage with `bandwidth.compute = FALSE`. Final fixed-bandwidth search candidates must also have a finite valid raw objective value.

Distribution Integral And Grid Controls: These controls tune the distribution-function integral and grid calculations.

`do.full.integral`
: a logical value which when set as `TRUE` evaluates the moment-based criterion on the empirical training support. Defaults to `FALSE`, which uses a faster row-aligned marginal-quantile path controlled by `ngrid`. For multivariate data, use `do.full.integral = TRUE` when you want the criterion evaluated on the empirical joint support and can tolerate the additional cost.

`memfac`
: The algorithm to compute the least-squares objective function uses a block-based algorithm to eliminate or minimize redundant kernel evaluations. Due to memory, hardware and software constraints, a maximum block size must be imposed by the algorithm. This block size is roughly equal to memfac*10^5 elements. Empirical tests on modern hardware find that a memfac of 500 performs well. If you experience out of memory errors, or strange behaviour for large data sets (>100k elements) setting memfac to a lower value may fix the problem.

`ngrid`
: integer number of row-aligned marginal-quantile grid points to use when `gdat` is not supplied and `do.full.integral = FALSE`. In multivariate problems this is a one-dimensional path through the marginal quantiles, not a Cartesian product grid. Defaults to `100`.

Kernel Type Controls: These controls choose continuous, unordered, and ordered kernels.

`ckerorder`
: numeric value specifying kernel order (one of `(2,4,6,8)`). Kernel order specified along with a `uniform` continuous kernel type will be ignored. Defaults to `2`.

`ckertype`
: character string used to specify the continuous kernel type. Can be set as `gaussian`, `epanechnikov`, or `uniform`. Defaults to `gaussian`.

okertype character string used to specify the ordered categorical kernel type. Can be set as wangvanryzin, liracine, or racineliyan. Defaults to liracine.

Numerical Search And Tolerance Controls: These controls set optimizer tolerances, restart behavior, invalid-candidate penalties, and bounded search transformations.

ftol fractional tolerance on the value of the cross-validation function evaluated at located minima (of order the machine precision or perhaps slightly larger so as not to be diddled by roundoff). Defaults to 1.490116e-07 (1.0e+01*sqrt(.Machine$double.eps)).

invalid.penalty
a character string specifying the penalty used when the optimizer encounters invalid bandwidths. "baseline" returns a finite penalty based on a baseline objective; "dbmax" returns DBL_MAX. Defaults to "baseline".

itmax integer number of iterations before failure in the numerical optimization routine. Defaults to 10000.

nmulti integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points.

penalty.multiplier
a numeric multiplier applied to the baseline penalty when invalid.penalty="baseline". Defaults to 10.

powell.remin logical flag controlling Powell restart-from-minimum behavior. When TRUE, the Powell-style search restarts from the located minimum for a minor gain in accuracy. Defaults to TRUE.

bwsolver bandwidth optimizer for fixed-degree searches. The default "powell" preserves the historical Powell route. "mads" uses the native **crs** NOMAD C API over the bandwidth coordinates, and "mads+powell" applies one Powell hot start from the MADS solution. Advanced native NOMAD parameters may be supplied through the explicit nomad.opts argument; invalid or unsupported parameters are rejected by NOMAD rather than silently ignored.

nomad.opts optional named list of native NOMAD parameters forwarded to the **crs** NOMAD C API for MADS/NOMAD-backed bandwidth searches. NULL is treated as no extra NOMAD options. Non-list values and unsupported native options are rejected before solver entry.

small a small number used to bracket a minimum (it is hopeless to ask for a bracketing interval of width less than sqrt(epsilon) times its central value, a fractional width of only about 10-04 (single precision) or 3x10-8 (double precision)). Defaults to small = 1.490116e-05 (1.0e+03*sqrt(.Machine$double.eps)).

tol tolerance on the position of located minima of the cross-validation function (tol should generally be no smaller than the square root of your machine's floating point precision). Defaults to 1.490116e-04 (1.0e+04*sqrt(.Machine$double.eps)).

transform.bounds
a logical value that when set to TRUE applies an internal transformation that maps the unconstrained search to the feasible bandwidth domain. Defaults to FALSE.

eval.only internal logical flag used by solver wrappers to evaluate an existing bandwidth candidate without running a bandwidth search. Ordinary users should leave this at its default FALSE.

Additional Arguments: These arguments collect remaining controls passed through S3 methods.

... additional arguments supplied to specify the bandwidth type, kernel types, selection methods, and so on, detailed below.

Details

The `scale.factor.*` controls are dimensionless search controls. The package converts scale factors to bandwidths using the estimator-specific scaling encoded in the bandwidth object, including kernel order and the number of continuous variables relevant for the estimator. Users should not pre-multiply these controls by sample-size or standard-deviation factors.

`scale.factor.init` controls the deterministic first search start. `scale.factor.init.lower` and `scale.factor.init.upper` define the random multistart interval. `scale.factor.search.lower` is the lower admissibility bound for continuous fixed-bandwidth search candidates. The effective first start is `max(scale.factor.init, scale.factor.search.lower)`, and the effective random-start lower endpoint is `max(scale.factor.init.lower, scale.factor.search.lower)`. `scale.factor.init.upper` must be at least that effective lower endpoint; the package errors rather than silently expanding the user's interval.

When `scale.factor.search.lower` is `NULL`, an existing bandwidth object's stored floor is inherited when available; otherwise the package default `0.1` is used. Explicit bandwidths supplied for storage with `bandwidth.compute = FALSE` are not rewritten by the search floor.

Categorical search-start controls such as `dfac.init`, `lbd.init`, and `hbd.init` have separate semantics and are not affected by `scale.factor.search.lower`.

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, `plot`, `plot.np` for plotting options, and `npRmpi.init` for interactive/cluster MPI startup. See `npRmpi.init` details for performance tradeoffs (message passing/startup mode) and the `inst/Rprofile` manual-broadcast template.

The bandwidth-selection argument surface is easiest to read by decision group: data, grid, and existing bandwidth inputs; bandwidth criterion and representation; continuous kernel and support controls beginning with `cker*`; ordered categorical kernel controls such as `okertype`; distribution-specific integral/grid controls such as `gdat`, `gdata`, `do.full.integral`, and `ngrid`; and numerical search initialization, tolerances, and feasibility controls. Users who call `npudist` without a bandwidth object can pass these same bandwidth-selection controls through that function's `....`

For S3 plotting help, see `plot.np`. You can list available plot methods with `methods("plot")`.

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

```
Usage 1: compute a bandwidth object using the formula interface:

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.
npRmpi.init(nslaves=1)
bw <- npudistbw(formula = ~y, data = mydat)
```

```
npRmpi.quit()

Usage 2: compute a bandwidth object using the data frame interface
and change the default kernel and order:

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.
npRmpi.init(nslaves=1)
Fhat <- npudistbw(tdat = y, ckertype="epanechnikov", ckerorder=4)
npRmpi.quit()
```

npudistbw implements a variety of methods for choosing bandwidths for multivariate (p-variate) distributions defined over a set of possibly continuous and/or discrete (ordered) data. The approach is based on Li and Racine (2003) who employ 'generalized product kernels' that admit a mix of continuous and discrete data types.

The cross-validation methods employ multivariate numerical search algorithms (direction set (Powell's) methods in multidimensions).

Bandwidths can (and will) differ for each variable which is, of course, desirable.

Three classes of kernel estimators for the continuous data types are available: fixed, adaptive nearest-neighbor, and generalized nearest-neighbor. Adaptive nearest-neighbor bandwidths change with each sample realization in the set, x_i, when estimating the cumulative distribution at the point x. Generalized nearest-neighbor bandwidths change with the point at which the cumulative distribution is estimated, x. Fixed bandwidths are constant over the support of x.

npudistbw may be invoked *either* with a formula-like symbolic description of variables on which bandwidth selection is to be performed *or* through a simpler interface whereby data is passed directly to the function via the dat parameter. Use of these two interfaces is **mutually exclusive**.

Data contained in the data frame dat may be a mix of continuous (default) and ordered discrete (to be specified in the data frame dat using ordered). Data can be entered in an arbitrary order and data types will be detected automatically by the routine (see npRmpi for details).

Data for which bandwidths are to be estimated may be specified symbolically. A typical description has the form ~ data, where data is a series of variables specified by name, separated by the separation character '+'. For example, ~ x + y specifies that the bandwidths for the joint distribution of variables x and y are to be estimated. See below for further examples.

A variety of kernels may be specified by the user. Kernels implemented for continuous data types include the second, fourth, sixth, and eighth-order Gaussian and Epanechnikov kernels, and the uniform kernel. Ordered data types use a variation of the Wang and van Ryzin (1981) kernel.

The optimizer invoked for search is Powell's conjugate direction method which requires the setting of (non-random) initial values and search directions for bandwidths, and when restarting, random values for successive invocations. Bandwidths for numeric variables are scaled by robust measures of spread, the sample size, and the number of numeric variables where appropriate. Two sets of parameters for bandwidths for numeric can be modified, those for initial values for the parameters

themselves, and those for the directions taken (Powell's algorithm does not involve explicit computation of the function's gradient). The default values are set by considering search performance for a variety of difficult test cases and simulated cases. We highly recommend restarting search a large number of times to avoid the presence of local minima (achieved by modifying `nmulti`). Further refinement for difficult cases can be achieved by modifying these sets of parameters. However, these parameters are intended more for the authors of the package to enable 'tuning' for various methods rather than for the user them self.

Value

`npudistbw` returns a `bandwidth` object with the following components:

`bw`	bandwidth(s), scale factor(s) or nearest neighbours for the data, `dat`
`fval`	objective function value at minimum

if `bwtype` is set to `fixed`, an object containing bandwidths, of class `bandwidth` (or scale factors if `bwscaling = TRUE`) is returned. If it is set to `generalized_nn` or `adaptive_nn`, then instead the kth nearest neighbors are returned for the continuous variables while the discrete kernel bandwidths are returned for the discrete variables. Bandwidths are stored under the component name `bw`, with each element i corresponding to column i of input data `dat`.

The functions `predict`, `summary` and `plot` support objects of type `bandwidth`.

Book And Method Pointers

`npudistbw` selects bandwidths for the unconditional distribution target $F(x) = \Pr(X \leq x)$ estimated by `npudist`. The selected bandwidths control the integrated kernel factors used in the mixed-data CDF estimator.

By default, data-driven bandwidth selection evaluates the CDF criterion on a row-aligned marginal-quantile path of length `ngrid`. This default is fast and useful for routine selection. To evaluate the criterion on the empirical joint support, set `do.full.integral = TRUE`; to use a user-chosen evaluation cloud or grid, supply `gdat`. For multivariate data, a Cartesian product of marginal quantiles can be supplied explicitly through `gdat` when appropriate, but it is not the default because its size grows exponentially with dimension.

Higher-order continuous CDF kernels can yield estimates that are not globally monotone and may fall outside $[0, 1]$ at some evaluation points. Returned values are not silently clamped.

For book-length derivations, see Li and Racine (2007), Chapter 1 *Density Estimation*, especially Sections 1.4 and 1.5, and Chapter 4 *Kernel Estimation with Mixed Data*. The later workflow treatment is Racine (2019), Chapters 2 and 3.

Usage Issues

If you are using data of mixed types, then it is advisable to use the `data.frame` function to construct your input data and not `cbind`, since `cbind` will typically not work as intended on mixed data types and will coerce the data to the same type.

Caution: multivariate data-driven bandwidth selection methods are, by their nature, *computationally intensive*. Virtually all methods require dropping the ith observation from the data set, computing an object, repeating this for all observations in the sample, then averaging each of these leave-one-out estimates for a *given* value of the bandwidth vector, and only then repeating this a large number

of times in order to conduct multivariate numerical minimization/maximization. Furthermore, due to the potential for local minima/maxima, *restarting this procedure a large number of times may often be necessary*. This can be frustrating for users possessing large datasets. For exploratory purposes, you may wish to override the default search tolerances, say, setting ftol=.01 and tol=.01 and conduct multistarting (the default is to restart min(2, ncol(dat)) times) as is done for a number of examples. Once the procedure terminates, you can restart search with default tolerances using those bandwidths obtained from the less rigorous search (i.e., set `bws=bw` on subsequent calls to this routine where `bw` is the initial bandwidth object). A version of this package using the `Rmpi` wrapper is under development that allows one to deploy this software in a clustered computing environment to facilitate computation involving large datasets.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

Bowman, A. and P. Hall and T. Prvan (1998), "Bandwidth selection for the smoothing of distribution functions," Biometrika, 85, 799-808.

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

Li, Q. and J.S. Racine (2003), "Nonparametric estimation of distributions with categorical and continuous data," Journal of Multivariate Analysis, 86, 266-292.

Li, C. and H. Li and J.S. Racine (2017), "Cross-Validated Mixed Datatype Bandwidth Selection for Nonparametric Cumulative Distribution/Survivor Functions," Econometric Reviews, **36**, 970-987.

Ouyang, D. and Q. Li and J.S. Racine (2006), "Cross-validation and the estimation of probability distributions with categorical data," Journal of Nonparametric Statistics, 18, 69-100.

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

Scott, D.W. (1992), *Multivariate Cumulative Distribution Estimation: Theory, Practice and Visualization,* New York: Wiley.

Silverman, B.W. (1986), *Density Estimation,* London: Chapman and Hall.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

`np.kernels`, `np.options`, `plot`, `plot.np` `bw.nrd`, `bw.SJ`, `hist`, `npudist`, `npudist`

Examples

```
## Not run:
## Not run in checks: data-driven CDF bandwidth selection on this dataset is
## computationally intensive and can hang/timeout in some MPI setups.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
```

```
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.
npRmpi.init(nslaves=1)

data("Italy")

bw <- npudistbw(formula=~ordered(year)+gdp,
                data=Italy)

summary(bw)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close

## End(Not run)
```

---

npudisthat *Unconditional Distribution Hat Operator*

Description

Constructs the unconditional distribution hat operator associated with npudist bandwidth objects. The returned operator maps a right-hand side y to Hy; with y = 1 this reproduces the fitted unconditional distribution function.

Usage

```
npudisthat(bws,
           tdat = stop("training data 'tdat' missing"),
```

```
            edat,
            y = NULL,
            output = c("matrix", "apply"))
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the fitted bandwidth object, training data, and evaluation data.

bws	A fitted unconditional distribution bandwidth object of class "dbandwidth".
edat	Optional evaluation data. If omitted, the operator is built on the training data.
tdat	Training data used to construct the operator.

Operator Output: These arguments control whether the operator is returned as a matrix or applied directly.

output	Either "matrix" for the hat matrix or "apply" for direct application to y.
y	Optional right-hand side vector or matrix with one row per training observation.

Details

For output = "matrix", the return value is a matrix with class c("npudisthat", "matrix") and attributes storing the bandwidth object, training data, evaluation data, and call metadata.

For output = "apply", the function returns H y directly. Matrix right-hand sides are applied columnwise.

This helper is intended for object-fed repeated evaluation once a bandwidth object has already been constructed. It does not perform bandwidth selection.

Value

Either a hat matrix of class "npudisthat" or the applied result H y, depending on output.

Examples

```
## Not run:
npRmpi.init(nslaves = 1)
data(cps71)
tx <- data.frame(age = cps71$age)

bw <- npudistbw(dat = tx, bwtype = "fixed",
                bandwidth.compute = FALSE, bws = 1.0)

H <- npudisthat(bws = bw, tdat = tx)
dist.hat <- npudisthat(bws = bw, tdat = tx,
                       y = rep(1, nrow(tx)),
                       output = "apply")
dist.core <- fitted(npudist(bws = bw, tdat = tx))

head(cbind(dist.core, dist.hat), n = 2L)
```

```
npRmpi.quit()

## End(Not run)
```

---

npuniden.boundary *Kernel Bounded Univariate Density Estimation Via Boundary Kernel Functions*

Description

npuniden.boundary computes kernel univariate unconditional density estimates given a vector of continuously distributed training data and, optionally, a bandwidth (otherwise least squares cross-validation is used for its selection). Lower and upper bounds [a,b] can be supplied (default is the empirical support $[\min(X), \max(X)]$) and if a is set to -Inf there is only one bound on the right, while if b is set to Inf there is only one bound on the left. If a is set to -Inf and b to Inf and the Gaussian type 1 kernel function is used, this will deliver the standard unadjusted kernel density estimate.

Usage

```
npuniden.boundary(X = NULL,
                  Y = NULL,
                  h = NULL,
                  a = min(X),
                  b = max(X),
                  bwmethod = c("cv.ls","cv.ml"),
                  cv = c("grid-hybrid","numeric"),
                  grid = NULL,
                  kertype = c("gaussian1","gaussian2",
                              "beta1","beta2",
                              "fb","fbl","fbu",
                              "rigaussian","gamma"),
                  nmulti = 1,
                  proper = FALSE)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify evaluation points, observations, support bounds, and optional bandwidths.

a	an optional lower bound (defaults to lower bound of empirical support $\min(X)$)
b	an optional upper bound (defaults to upper bound of empirical support $\max(X)$)
h	an optional bandwidth (>0)
X	a required numeric vector of training data lying in $[a, b]$
Y	an optional numeric vector of evaluation data lying in $[a, b]$

Bandwidth Search Controls: These arguments control the boundary-corrected bandwidth search.

bwmethod	whether to conduct bandwidth search via least squares cross-validation ("cv.ls") or likelihood cross-validation ("cv.ml")
cv	an optional argument for search (default is likely more reliable in the presence of local maxima)
grid	an optional grid used for the initial grid search when cv="grid-hybrid"
kertype	an optional kernel specification (defaults to "gaussian1")
nmulti	number of multi-starts used when cv="numeric" (defaults to 1 for the univariate boundary problem)

Proper Density Control: This argument controls optional proper-density repair.

proper	an optional logical value indicating whether to enforce proper density and distribution function estimates over the range $[a, b]$

Details

Documentation guide: see np.kernels for kernels, np.options for global options, plot for plotting options, and npRmpi.init for interactive/cluster MPI startup. See npRmpi.init details for performance tradeoffs (message passing/startup mode) and the inst/Rprofile manual-broadcast template. For interactive and cluster batch workflows, see npRmpi.init.

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

```
model <- npuniden.boundary(X,a=-2,b=3)
```

npuniden.boundary implements a variety of methods for estimating a univariate density function defined over a continuous random variable in the presence of bounds via the use of so-called boundary or edge kernel functions.

The kernel functions "beta1" and "beta2" are Chen's (1999) type 1 and 2 kernel functions with biases of $O(h)$, the "gamma" kernel function is from Chen (2000) with a bias of $O(h)$, "rigaussian" is the reciprocal inverse Gaussian kernel function (Scaillet (2004), Igarashi & Kakizawa (2014)) with bias of $O(h)$, and "gaussian1" and "gaussian2" are truncated Gaussian kernel functions with biases of $O(h)$ and $O(h^2)$, respectively. The kernel functions "fb", "fbl" and "fbu" are floating boundary polynomial biweight kernels with biases of $O(h^2)$ (Scott (1992), Page 146). Without exception, these kernel functions are asymmetric in general with shape that changes depending on where the density is being estimated (i.e., how close the estimation point x in $\hat{f}(x)$ is to a boundary). This function is written purely in R, so to see the exact form for each of these kernel functions, simply enter the name of this function in R (i.e., enter npuniden.boundary after loading this package) and scroll up for their definitions.

The kernel functions "gamma", "rigaussian", and "fbl" have support $[a, \infty]$. The kernel function "fbu" has support $[-\infty, b]$. The rest have support on $[a, b]$. Note that the two sided support default values are a=min(X) and b=max(X).

Note that data-driven bandwidth selection is more nuanced in bounded settings, therefore it would be prudent to manually select a bandwidth that is, say, 1/25th of the range of the data and manually inspect the estimate (say h=0.05 when $X \in [0, 1]$). Also, it may be wise to compare the density estimate with that from a histogram with the option breaks=25. Note also that the kernel functions

"gaussian2", "fb", "fbl" and "fbu" can assume negative values leading to potentially negative density estimates, and must be trimmed when conducting likelihood cross-validation which can lead to oversmoothing. Least squares cross-validation is unaffected and appears to be more reliable in such instances hence is the default here.

Scott (1992, Page 149) writes "While boundary kernels can be very useful, there are potentially serious problems with real data. There are an infinite number of boundary kernels reflecting the spectrum of possible design constraints, and these kernels are not interchangeable. Severe artifacts can be introduced by any one of them in inappropriate situations. Very careful examination is required to avoid being victimized by the particular boundary kernel chosen. Artifacts can unfortunately be introduced by the choice of the support interval for the boundary kernel."

Note that since some kernel functions can assume negative values, this can lead to improper density estimates. The estimated distribution function is obtained via numerical integration of the estimated density function and may itself not be proper even when evaluated on the full range of the data $[a, b]$. Setting the option proper=TRUE will render the density and distribution estimates proper over the full range of the data, though this may not in general be a mean square error optimal strategy.

Finally, note that this function is pretty bare-bones relative to other functions in this package. For one, at this time there is no automatic print support so kindly see the examples for illustrations of its use, among other differences.

Value

npuniden.boundary returns the following components:

f	estimated density at the points X
F	estimated distribution at the points X (numeric integral of f)
sd.f	asymptotic standard error of the estimated density at the points X
sd.F	asymptotic standard error of the estimated distribution at the points X
h	bandwidth used
nmulti	number of multi-starts used

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

References

Bouezmarni, T. and Rolin, J.-M. (2003). "Consistency of the beta kernel density function estimator," The Canadian Journal of Statistics / La Revue Canadienne de Statistique, 31(1):89-98.

Chen, S. X. (1999). "Beta kernel estimators for density functions," Computational Statistics & Data Analysis, 31(2):131-145.

Chen, S. X. (2000). "Probability density function estimation using gamma kernels," Annals of the Institute of Statistical Mathematics, 52(3):471-480.

Diggle, P. (1985). "A kernel method for smoothing point process data," Journal of the Royal Statistical Society. Series C (Applied Statistics), 34(2):138-147.

Igarashi, G. and Y. Kakizawa (2014). "Re-formulation of inverse Gaussian, reciprocal inverse Gaussian, and Birnbaum-Saunders kernel estimators," Statistics & Probability Letters, 84:235-246.

Igarashi, G. and Y. Kakizawa (2015). "Bias corrections for some asymmetric kernel estimators," Journal of Statistical Planning and Inference, 159:37-63.

Igarashi, G. (2016). "Bias reductions for beta kernel estimation," Journal of Nonparametric Statistics, 28(1):1-30.

Racine, J. S. and Q. Li and Q. Wang, "Boundary-adaptive kernel density estimation: the case of (near) uniform density", Journal of Nonparametric Statistics, 2024, 36 (1), 146-164, https://doi.org/10.1080/10485252.2023.2

Scaillet, O. (2004). "Density estimation using inverse and reciprocal inverse Gaussian kernels," Journal of Nonparametric Statistics, 16(1-2):217-226.

Scott, D. W. (1992). "Multivariate density estimation: Theory, practice, and visualization," New York: Wiley.

Zhang, S. and R. J. Karunamuni (2010). "Boundary performance of the beta kernel estimators," Journal of Nonparametric Statistics, 22(1):81-104.

See Also

`np.kernels`, `np.options`, `plot` The **Ake**, **bde**, and **Conake** packages and the function `npuniden.reflect`.

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## Example 1: f(0)=0, f(1)=1, plot boundary corrected density,
## unadjusted density, and DGP
set.seed(42)
n <- 100
X <- sort(rbeta(n,5,1))
dgp <- dbeta(X,5,1)
model.g1 <- npuniden.boundary(X,kertype="gaussian1")
model.g2 <- npuniden.boundary(X,kertype="gaussian2")
model.b1 <- npuniden.boundary(X,kertype="beta1")
model.b2 <- npuniden.boundary(X,kertype="beta2")
model.fb <- npuniden.boundary(X,kertype="fb")
model.unadjusted <- npuniden.boundary(X,a=-Inf,b=Inf)
ylim <- c(0,max(c(dgp,model.g1$f,model.g2$f,model.b1$f,model.b2$f,model.fb$f)))
if (interactive()) {
  plot(X,dgp,ylab="Density",ylim=ylim,type="l")
  lines(X,model.g1$f,lty=2,col=2)
  lines(X,model.g2$f,lty=3,col=3)
  lines(X,model.b1$f,lty=4,col=4)
  lines(X,model.b2$f,lty=5,col=5)
  lines(X,model.fb$f,lty=6,col=6)
  lines(X,model.unadjusted$f,lty=7,col=7)
  rug(X)
  legend("topleft",c("DGP",
                     "Boundary Kernel (gaussian1)",
                     "Boundary Kernel (gaussian2)",
                     "Boundary Kernel (beta1)",
                     "Boundary Kernel (beta2)",
                     "Boundary Kernel (floating boundary)",
                     "Unadjusted"),col=1:7,lty=1:7,bty="n")
```

```
}

## Example 2: f(0)=0, f(1)=0, plot density, distribution, DGP, and
## asymptotic point-wise confidence intervals
set.seed(42)
X <- sort(rbeta(100,5,3))
model <- npuniden.boundary(X)
oldpar <- par(no.readonly = TRUE)
on.exit(par(oldpar), add = TRUE)
par(mfrow=c(1,2))
ylim=range(c(model$f,model$f+1.96*model$sd.f,model$f-1.96*model$sd.f,dbeta(X,5,3)))
if (interactive()) {
  plot(X,model$f,ylim=ylim,ylab="Density",type="l",)
  lines(X,model$f+1.96*model$sd.f,lty=2)
  lines(X,model$f-1.96*model$sd.f,lty=2)
  lines(X,dbeta(X,5,3),col=2)
  rug(X)
  legend("topleft",c("Density","DGP"),lty=c(1,1),col=1:2,bty="n")
}

if (interactive()) {
  plot(X,model$F,ylab="Distribution",type="l")
  lines(X,model$F+1.96*model$sd.F,lty=2)
  lines(X,model$F-1.96*model$sd.F,lty=2)
  lines(X,pbeta(X,5,3),col=2)
  rug(X)
  legend("topleft",c("Distribution","DGP"),lty=c(1,1),col=1:2,bty="n")
}

## Example 3: Age for working age males in the cps71 data set bounded
## below by 21 and above by 65
data(cps71)
with(cps71, {
model <- npuniden.boundary(age,a=21,b=65)
par(mfrow=c(1,1))
hist(age,prob=TRUE,main="")
lines(age,model$f)
lines(density(age,bw=model$h),col=2)
legend("topright",c("Boundary Kernel","Unadjusted"),lty=c(1,1),col=1:2,bty="n")
})

## End(Not run)
```

`npuniden.reflect` *Kernel Bounded Univariate Density Estimation Via Data-Reflection*

Description

`npuniden.reflect` computes kernel univariate unconditional density estimates given a vector of continuously distributed training data and, optionally, a bandwidth (otherwise likelihood cross-validation is used for its selection). Lower and upper bounds [a,b] can be supplied (default is [0,1])

and if a is set to -Inf there is only one bound on the right, while if b is set to Inf there is only one bound on the left.

Usage

```
npuniden.reflect(X = NULL,
                 Y = NULL,
                 h = NULL,
                 a = 0,
                 b = 1,
                 ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify evaluation points, observations, support bounds, and optional bandwidths.

a	an optional lower bound (defaults to 0)
b	an optional upper bound (defaults to 1)
h	an optional bandwidth (>0)
X	a required numeric vector of training data lying in $[a, b]$
Y	an optional numeric vector of evaluation data lying in $[a, b]$

Additional Arguments: Further arguments are passed to npudensbw and npudens.

...	optional arguments passed to npudensbw and npudens

Details

Documentation guide: see np.kernels for kernels, np.options for global options, plot for plotting options, and npRmpi.init for interactive/cluster MPI startup. See npRmpi.init details for performance tradeoffs (message passing/startup mode) and the inst/Rprofile manual-broadcast template.

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

```
model <- npuniden.reflect(X,a=-2,b=3)
```

npuniden.reflect implements the data-reflection method for estimating a univariate density function defined over a continuous random variable in the presence of bounds.

Note that data-reflection imposes a zero derivative at the boundary, i.e., $f'(a) = f'(b) = 0$.

Value

npuniden.reflect returns the following components:

f	estimated density at the points X
F	estimated distribution at the points X (numeric integral of f)
sd.f	asymptotic standard error of the estimated density at the points X
sd.F	asymptotic standard error of the estimated distribution at the points X
h	bandwidth used
nmulti	number of multi-starts used

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

References

Boneva, L. I., Kendall, D., and Stefanov, I. (1971). "Spline transformations: Three new diagnostic aids for the statistical data- analyst," Journal of the Royal Statistical Society. Series B (Methodological), 33(1):1-71.

Cline, D. B. H. and Hart, J. D. (1991). "Kernel estimation of densities with discontinuities or discontinuous derivatives," Statistics, 22(1):69-84.

Hall, P. and Wehrly, T. E. (1991). "A geometrical method for removing edge effects from kernel-type nonparametric regression estimators," Journal of the American Statistical Association, 86(415):665-672.

See Also

np.kernels, np.options, plot The **Ake**, **bde**, and **Conake** packages and the function npuniden.boundary.

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))
```

```
in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

## Example 1: f(0)=0, f(1)=1, plot boundary corrected density,
## unadjusted density, and DGP

set.seed(42)

n <- 100
X <- sort(rbeta(n,5,1))
dgp <- dbeta(X,5,1)

model <- npuniden.reflect(X)

model.unadjusted <- npuniden.boundary(X,a=-Inf,b=Inf)

ylim <- c(0,max(c(dgp,model$f,model.unadjusted$f)))
if (interactive()) plot(X,model$f,ylab="Density",ylim=ylim,type="l")
lines(X,model.unadjusted$f,lty=2,col=2)
lines(X,dgp,lty=3,col=3)
rug(X)
legend("topleft",c("Data-Reflection","Unadjusted","DGP"),col=1:3,lty=1:3,bty="n")

## Example 2: f(0)=0, f(1)=0, plot density, distribution, DGP, and
## asymptotic point-wise confidence intervals

set.seed(42)

X <- sort(rbeta(100,5,3))

model <- npuniden.reflect(X)

oldpar <- par(no.readonly = TRUE)
on.exit(par(oldpar), add = TRUE)
par(mfrow=c(1,2))
ylim=range(c(model$f,model$f+1.96*model$sd.f,model$f-1.96*model$sd.f,dbeta(X,5,3)))
if (interactive()) plot(X,model$f,ylim=ylim,ylab="Density",type="l",)
lines(X,model$f+1.96*model$sd.f,lty=2)
lines(X,model$f-1.96*model$sd.f,lty=2)
lines(X,dbeta(X,5,3),col=2)
rug(X)
legend("topleft",c("Density","DGP"),lty=c(1,1),col=1:2,bty="n")

if (interactive()) plot(X,model$F,ylab="Distribution",type="l")
lines(X,model$F+1.96*model$sd.F,lty=2)
lines(X,model$F-1.96*model$sd.F,lty=2)
```

```
lines(X,pbeta(X,5,3),col=2)
rug(X)
legend("topleft",c("Distribution","DGP"),lty=c(1,1),col=1:2,bty="n")


## Example 3: Age for working age males in the cps71 data set bounded
## below by 21 and above by 65

data(cps71)

model <- npuniden.reflect(cps71$age,a=21,b=65)

par(mfrow=c(1,1))
hist(cps71$age,prob=TRUE,main="",ylim=c(0,max(model$f)))
lines(cps71$age,model$f)
lines(density(cps71$age,bw=model$h),col=2)
legend("topright",c("Data-Reflection","Unadjusted"),lty=c(1,1),col=1:2,bty="n")

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

npuniden.sc *Kernel Shape Constrained Bounded Univariate Density Estimation*

Description

npuniden.sc computes shape constrained kernel univariate unconditional density estimates given a vector of continuously distributed training data and a bandwidth. Lower and upper bounds [a,b] can be supplied (default is [0,1]) and if a is set to -Inf there is only one bound on the right, while if b is set to Inf there is only one bound on the left.

Usage

```
npuniden.sc(X = NULL,
            Y = NULL,
            h = NULL,
            a = 0,
            b = 1,
            lb = NULL,
            ub = NULL,
            extend.range = 0,
            num.grid = 0,
            function.distance = TRUE,
            integral.equal = FALSE,
            constraint = c("density",
                           "mono.incr",
                           "mono.decr",
                           "concave",
                           "convex",
                           "log-concave",
                           "log-convex"))
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify evaluation points, observations, support bounds, and optional bandwidths.

a	an optional lower bound on the support of X or Y (defaults to 0)
b	an optional upper bound on the support of X or Y (defaults to 1)
h	a bandwidth (> 0)
X	a required numeric vector of training data lying in $[a, b]$
Y	an optional numeric vector of evaluation data lying in $[a, b]$

Density Bounds: These arguments set optional lower and upper density bounds used with constraint = \"density\".

lb	a scalar lower bound (≥ 0) to be used in conjunction with constraint="density"
ub	a scalar upper bound (≥ 0 and $\geq$ lb) to be used in conjunction with constraint="density"

Grid And Distance Controls: These arguments control grid construction, distance metric, and mass preservation.

extend.range	number specifying the fraction by which the range of the training data should be extended for the additional grid points (passed to the function extendrange)
function.distance	a logical value that, if TRUE, minimizes the squared deviation between the constrained and unconstrained estimates, otherwise, minimizes the squared deviation between the constrained and unconstrained weights
integral.equal	a logical value, that, if TRUE, adjusts the constrained estimate to have the same probability mass over the range X, Y, and the additional grid points

`num.grid` number of additional grid points (in addition to `X` and `Y`) placed on an equi-spaced grid spanning `extendrange(c(X,Y),f=extend.range)` (if `num.grid=0` no additional grid points will be used regardless of the value of `extend.range`)

Shape Constraint: This argument chooses the monotonicity, convexity, or log-shape constraint.

`constraint` a character string indicating whether the estimate is to be constrained to be monotonically increasing (`constraint="mono.incr"`), decreasing (`constraint="mono.decr"`), convex (`constraint="convex"`), concave (`constraint="concave"`), log-convex (`constraint="log-convex"`), or log-concave (`constraint="log-concave"`)

Details

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, `plot` for plotting options, and `npRmpi.init` for interactive/cluster MPI startup. See `npRmpi.init` details for performance tradeoffs (message passing/startup mode) and the `inst/Rprofile` manual-broadcast template. For interactive and cluster batch workflows, see `npRmpi.init`.

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

```
model <- npuniden.sc(X,a=-2,b=3)
```

`npuniden.sc` implements a methods for estimating a univariate density function defined over a continuous random variable in the presence of bounds subject to a variety of shape constraints. The bounded estimates use the truncated Gaussian kernel function.

Note that for the log-constrained estimates, the derivative estimate returned is that for the log-constrained estimate not the non-log value of the estimate returned by the function. See Example 5 below hat manually plots the log-density and returned derivative (no transformation is needed when plotting the density estimate itself).

If the quadratic program solver fails to find a solution, the unconstrained estimate is returned with an immediate warning. Possible causes to be investigated are undersmoothing, sparsity, and the presence of non-sample grid points. To investigate the possibility of undersmoothing try using a larger bandwidth, to investigate sparsity try decreasing `extend.range`, and to investigate non-sample grid points try setting `num.grid` to `0`.

Mean square error performance seems to improve generally when using additional grid points in the empirical support of `X` and `Y` (i.e., in the observed range of the data sample) but appears to deteriorate when imposing constraints beyond the empirical support (i.e., when `extend.range` is positive). Increasing the number of additional points beyond a hundred or so appears to have a limited impact.

The option `function.distance=TRUE` appears to perform better for imposing convexity, concavity, log-convexity and log-concavity, while `function.distance=FALSE` appears to perform better for imposing monotonicity, whether increasing or decreasing (based on simulations for the Beta(s1,s2) distribution with sample size $n = 100$).

Value

A list with the following elements:

f	unconstrained density estimate
f.sc	shape constrained density estimate
se.f	asymptotic standard error of the unconstrained density estimate
se.f.sc	asymptotic standard error of the shape constrained density estimate
f.deriv	unconstrained derivative estimate (of order 1 or 2 or log thereof)
f.sc.deriv	shape constrained derivative estimate (of order 1 or 2 or log thereof)
F	unconstrained distribution estimate
F.sc	shape constrained distribution estimate
integral.f	the integral of the unconstrained estimate over X, Y, and the additional grid points
integral.f.sc	the integral of the constrained estimate over X, Y, and the additional grid points
solve.QP	logical, if TRUE solve.QP succeeded, otherwise failed
attempts	number of attempts when solve.QP fails (max = 9)

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

References

Du, P. and C. Parmeter and J. Racine (2024), "Shape Constrained Kernel PDF and PMF Estimation", Statistica Sinica, 34 (1), 257-289, doi:10.5705/ss.202021.0112

See Also

np.kernels, np.options, plot The **logcondens**, **LogConDEAD**, and **scdensity** packages, and the function npuniden.boundary.

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
n <- 100
set.seed(42)

## Example 1: N(0,1), constrain the density to lie within lb=.1 and ub=.2

X <- sort(rnorm(n))
h <- npuniden.boundary(X,a=-Inf,b=Inf)$h
foo <- npuniden.sc(X,h=h,constraint="density",a=-Inf,b=Inf,lb=.1,ub=.2)
ylim <- range(c(foo$f.sc,foo$f))
if (interactive()) plot(X,foo$f.sc,type="l",ylim=ylim,xlab="X",ylab="Density")
lines(X,foo$f,col=2,lty=2)
rug(X)
legend("topleft",c("Constrained","Unconstrained"),lty=1:2,col=1:2,bty="n")
```

```
## Example 2: Beta(5,1), DGP is monotone increasing, impose valid
## restriction

X <- sort(rbeta(n,5,1))
h <- npuniden.boundary(X)$h

foo <- npuniden.sc(X=X,h=h,constraint=c("mono.incr"))

oldpar <- par(no.readonly = TRUE)
on.exit(par(oldpar), add = TRUE)
par(mfrow=c(1,2))
ylim <- range(c(foo$f.sc,foo$f))
if (interactive()) plot(X,foo$f.sc,type="l",ylim=ylim,xlab="X",ylab="Density")
lines(X,foo$f,col=2,lty=2)
rug(X)
legend("topleft",c("Constrained","Unconstrained"),lty=1:2,col=1:2,bty="n")

ylim <- range(c(foo$f.sc.deriv,foo$f.deriv))
if (interactive()) plot(X,foo$f.sc.deriv,type="l",ylim=ylim,xlab="X",ylab="First Derivative")
lines(X,foo$f.deriv,col=2,lty=2)
abline(h=0,lty=2)
rug(X)
legend("topleft",c("Constrained","Unconstrained"),lty=1:2,col=1:2,bty="n")

## Example 3: Beta(1,5), DGP is monotone decreasing, impose valid
## restriction

X <- sort(rbeta(n,1,5))
h <- npuniden.boundary(X)$h

foo <- npuniden.sc(X=X,h=h,constraint=c("mono.decr"))

par(mfrow=c(1,2))
ylim <- range(c(foo$f.sc,foo$f))
if (interactive()) plot(X,foo$f.sc,type="l",ylim=ylim,xlab="X",ylab="Density")
lines(X,foo$f,col=2,lty=2)
rug(X)
legend("topleft",c("Constrained","Unconstrained"),lty=1:2,col=1:2,bty="n")

ylim <- range(c(foo$f.sc.deriv,foo$f.deriv))
if (interactive()) plot(X,foo$f.sc.deriv,type="l",ylim=ylim,xlab="X",ylab="First Derivative")
lines(X,foo$f.deriv,col=2,lty=2)
abline(h=0,lty=2)
rug(X)
legend("topleft",c("Constrained","Unconstrained"),lty=1:2,col=1:2,bty="n")

## Example 4: N(0,1), DGP is log-concave, impose invalid concavity
## restriction

X <- sort(rnorm(n))
h <- npuniden.boundary(X,a=-Inf,b=Inf)$h
```

```
foo <- npuniden.sc(X=X,h=h,a=-Inf,b=Inf,constraint=c("concave"))

par(mfrow=c(1,2))
ylim <- range(c(foo$f.sc,foo$f))
if (interactive()) plot(X,foo$f.sc,type="l",ylim=ylim,xlab="X",ylab="Density")
lines(X,foo$f,col=2,lty=2)
rug(X)
legend("topleft",c("Constrained","Unconstrained"),lty=1:2,col=1:2,bty="n")
ylim <- range(c(foo$f.sc.deriv,foo$f.deriv))

if (interactive()) plot(X,foo$f.sc.deriv,type="l",ylim=ylim,xlab="X",ylab="Second Derivative")
lines(X,foo$f.deriv,col=2,lty=2)
abline(h=0,lty=2)
rug(X)
legend("topleft",c("Constrained","Unconstrained"),lty=1:2,col=1:2,bty="n")

## Example 45: Beta(3/4,3/4), DGP is convex, impose valid restriction

X <- sort(rbeta(n,3/4,3/4))
h <- npuniden.boundary(X)$h

foo <- npuniden.sc(X=X,h=h,constraint=c("convex"))

par(mfrow=c(1,2))
ylim <- range(c(foo$f.sc,foo$f))
if (interactive()) plot(X,foo$f.sc,type="l",ylim=ylim,xlab="X",ylab="Density")
lines(X,foo$f,col=2,lty=2)
rug(X)
legend("topleft",c("Constrained","Unconstrained"),lty=1:2,col=1:2,bty="n")

ylim <- range(c(foo$f.sc.deriv,foo$f.deriv))
if (interactive()) plot(X,foo$f.sc.deriv,type="l",ylim=ylim,xlab="X",ylab="Second Derivative")
lines(X,foo$f.deriv,col=2,lty=2)
abline(h=0,lty=2)
rug(X)
legend("topleft",c("Constrained","Unconstrained"),lty=1:2,col=1:2,bty="n")

## Example 6: N(0,1), DGP is log-concave, impose log-concavity
## restriction

X <- sort(rnorm(n))
h <- npuniden.boundary(X,a=-Inf,b=Inf)$h

foo <- npuniden.sc(X=X,h=h,a=-Inf,b=Inf,constraint=c("log-concave"))

par(mfrow=c(1,2))

ylim <- range(c(log(foo$f.sc),log(foo$f)))
if (interactive()) plot(X,log(foo$f.sc),type="l",ylim=ylim,xlab="X",ylab="Log-Density")
lines(X,log(foo$f),col=2,lty=2)
rug(X)
legend("topleft",c("Constrained-log","Unconstrained-log"),lty=1:2,col=1:2,bty="n")
```

```
ylim <- range(c(foo$f.sc.deriv,foo$f.deriv))
if (interactive()) plot(X,
                        foo$f.sc.deriv,
                        type="l",
                        ylim=ylim,
                        xlab="X",
                        ylab="Second Derivative of Log-Density")
lines(X,foo$f.deriv,col=2,lty=2)
abline(h=0,lty=2)
rug(X)
legend("topleft",c("Constrained-log","Unconstrained-log"),lty=1:2,col=1:2,bty="n")

## End(Not run)
```

---

npunitest *Kernel Consistent Univariate Density Equality Test with Mixed Data Types*

Description

npunitest implements the consistent metric entropy test of Maasoumi and Racine (2002) for two arbitrary, stationary univariate nonparametric densities on common support.

Usage

```
npunitest(data.x = NULL,
          data.y = NULL,
          method = c("integration","summation"),
          bootstrap = TRUE,
          boot.num = 399,
          bw.x = NULL,
          bw.y = NULL,
          random.seed = 42,
          ...)
```

Arguments

Data, Bandwidth Inputs And Formula Interface: These arguments identify the two samples, statistic variant, and any supplied bandwidths.

bw.x, bw.y	numeric (scalar) bandwidths. Defaults to plug-in (see details below).
data.x, data.y	common support univariate vectors containing the variables.
method	a character string used to specify whether to compute the integral version or the summation version of the statistic. Can be set as integration or summation. Defaults to integration. See 'Details' below for important information regarding the use of summation when data.x and data.y lack common support and/or are sparse.

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

`boot.num` an integer value specifying the number of bootstrap replications to use. Defaults to `399`.

`bootstrap` a logical value which specifies whether to conduct the bootstrap test or not. If set to `FALSE`, only the statistic will be computed. Defaults to `TRUE`.

`random.seed` an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.

Additional Arguments: Further arguments are passed to the bandwidth-selection routines used by the test.

`...` additional arguments supplied to specify the bandwidth type, kernel types, and so on. This is used since we specify bw as a numeric scalar and not a `bandwidth` object, and is of interest if you do not desire the default behaviours. To change the defaults, you may specify any of `bwscaling`, `bwtype`, `ckertype`, `ckerorder`, `ukertype`, `okertype`.

Details

Documentation guide: see `np.kernels` for kernels, `np.options` for global options, `plot` for plotting options, and `npRmpi.init` for interactive/cluster MPI startup. See `npRmpi.init` details for performance tradeoffs (message passing/startup mode) and the `inst/Rprofile` manual-broadcast template.

`npunitest` computes the nonparametric metric entropy (normalized Hellinger of Granger, Maasoumi and Racine (2004)) for testing equality of two univariate density/probability functions, $D[f(x), f(y)]$. See Maasoumi and Racine (2002) for details. Default bandwidths are of the plug-in variety (`bw.SJ` for continuous variables and direct plug-in for discrete variables). The bootstrap is conducted via simple resampling with replacement from the pooled `data.x` and `data.y` (`data.x` only for `summation`).

The summation version of this statistic can be numerically unstable when `data.x` and `data.y` lack common support or when the overlap is sparse (the summation version involves division of densities while the integration version involves differences, and the statistic in such cases can be reported as exactly 0.5 or 0). Warning messages are produced when this occurs ('integration recommended') and should be heeded.

Numerical integration can occasionally fail when the `data.x` and `data.y` distributions lack common support and/or lie an extremely large distance from one another (the statistic in such cases will be reported as exactly 0.5 or 0). However, in these extreme cases, simple tests will reveal the obvious differences in the distributions and entropy-based tests for equality will be clearly unnecessary.

Value

`npunitest` returns an object of type `unitest` with the following components

`Srho` the statistic `Srho`

`Srho.bootstrap` contains the bootstrap replications of `Srho`

`P` the P-value of the statistic

boot.num	number of bootstrap replications
bw.x, bw.y	scalar bandwidths for data.x, data.y

summary supports object of type unitest.

Book And Method Pointers

npunitest uses an entropy-distance measure to compare univariate density or probability functions. The statistic is based on the normalized Hellinger-type distance described in the entropy-test literature referenced below.

For book-length background, see Racine (2019), Chapter 2 *Continuous Density and Cumulative Distribution Functions*, especially the entropy and density-test material. For related density and testing background, see Li and Racine (2007), Chapters 1, 12, and 13.

Usage Issues

See the example below for proper usage.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Granger, C.W. and E. Maasoumi and J.S. Racine (2004), "A dependence metric for possibly nonlinear processes", Journal of Time Series Analysis, 25, 649-669.

Maasoumi, E. and J.S. Racine (2002), "Entropy and predictability of stock market returns," Journal of Econometrics, 107, 2, pp 291-312.

See Also

np.kernels, np.options, plot npdeneqtest,npdeptest,npsdeptest,npsymtest

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.
```

```
force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

set.seed(42)

n <- 250

x <- rnorm(n)
y <- rnorm(n)

output <- npunitest(x,y,
                    method="summation",
                    bootstrap=TRUE,
                    boot.num=29)

summary(output)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

oecdpanel *Cross Country Growth Panel*

Description

Cross country GDP growth panel covering the period 1960-1995 used by Liu and Stengos (2000) and Maasoumi, Racine, and Stengos (2007). There are 616 observations in total. `data("oecdpanel")` makes available the dataset `"oecdpanel"` plus an additional object `"bw"`.

Usage

```
data("oecdpanel")
```

Format

A data frame with 7 columns, and 616 rows. This panel covers 7 5-year periods: 1960-1964, 1965-1969, 1970-1974, 1975-1979, 1980-1984, 1985-1989 and 1990-1994.

A separate local-linear `rbandwidth` object (`bw`) has been computed for the user's convenience which can be used to visualize this dataset using `plot(bw)`.

growth the first column, of type `numeric`: growth rate of real GDP per capita for each 5-year period

oecd the second column, of type `factor`: equal to 1 for OECD members, 0 otherwise

year the third column, of type `integer`

initgdp the fourth column, of type `numeric`: per capita real GDP at the beginning of each 5-year period

popgro the fifth column, of type `numeric`: average annual population growth rate for each 5-year period

inv the sixth column, of type `numeric`: average investment/GDP ratio for each 5-year period

humancap the seventh column, of type `numeric`: average secondary school enrolment rate for each 5-year period

Source

Thanasis Stengos

References

Liu, Z. and T. Stengos (1999), "Non-linearities in cross country growth regressions: a semiparametric approach," Journal of Applied Econometrics, 14, 527-538.

Maasoumi, E. and J.S. Racine and T. Stengos (2007), "Growth and convergence: a profile of distribution dynamics and mobility," Journal of Econometrics, 136, 483-508

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
data("oecdpanel")
with(oecdpanel, {
summary(oecdpanel)
})
```

```
## End(Not run)
```

---

plot.np *General Purpose Plotting of Nonparametric Objects*

Description

Plotting is provided via `plot` S3 methods, which generate plots of nonparametric statistical objects such as regressions, quantile regressions, partially linear regressions, single-index models, densities and distributions, given training data and a bandwidth object. `plot(...)` is the supported public interface.

Usage

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

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

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

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

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

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

Arguments

Plot Object: This argument identifies the object to plot.

`x` an object generated by the **npRmpi** package. The most convenient inputs are fitted estimator objects returned by invocations of `npudens`, `npcdens`, `npreg`, `npconmode`, `npplreg`, `npindex`, or `npscoef`. Bandwidth objects returned by `npudensbw`, `npcdensbw`, `npregbw`, `npplregbw`, `npindexbw`, or `npscoefbw` are also supported when plotting should construct the fitted object.

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

Data and evaluation controls

`data` an optional `data.frame` used to resolve stored formula calls when explicit training data arguments are not supplied.

`grid_control` optional detailed grid controls created by `np_grid_control`.

`neval` the number of evaluation points used for each plotted continuous margin.

`xdat`, `ydat`, `zdat` training data used when plotting bandwidth objects. `xdat` is the regressor/conditioning data, `ydat` is the response or conditional outcome data, and `zdat` is the auxiliary semiparametric covariate block when relevant.

`xq`, `yq`, `zq` quantiles used to hold non-plotted variables fixed when constructing slices.

`xtrim`, `ytrim`, `ztrim` trimming/expansion controls for the evaluation support.

Plot content controls

`coef`, `coef_index` coefficient-function plotting controls for semiparametric families where supported.

`common_scale` a logical value specifying whether multiple panels should share a common plotting scale when feasible.

`gradient_order`, `gradients` derivative plotting controls for families that support gradients/derivatives. For `npreg` local-polynomial objects, `gradient_order` requests the continuous-predictor derivative order to generate on the plot evaluation grid; if omitted the first derivative/effect is plotted. Location-scale quantile-regression `nplsqreg` objects use the same coordinate-wise local-polynomial contract while preserving their existing `tau`-indexed output; no mixed partials are generated. For `npconmode` objects, `gradients=TRUE` plots stored class-probability gradients/effects for the selected response level, computed by `npconmode(..., gradients=TRUE, level=...)`.

`level` response level to plot for class-probability displays when supported, notably `plot.conmode`. If omitted, the default for `npconmode` objects is the base/reference response level `levels(y)[1]`. These plots require probabilities stored by `npconmode(..., probabilities=TRUE)`. Currently `plot.conmode` displays stored object-fed probability/effect payloads only; probability/effect interval bands and surface grids are not yet implemented for this route.

`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.

`proper`, `proper_control`, `proper_method` controls for proper conditional density/distribution handling where implemented.

`tau` the quantile index or indices used for quantile-regression-related plotting routes when relevant. A vector of `tau` values overlays the corresponding conditional quantile curves and reuses the stored bandwidth object. With `errors="asymptotic"` or `errors="bootstrap"`, intervals are constructed separately for each requested `tau`; bootstrap work is shared across `tau` values within each resample. For `npqreg` objects whose conditional-distribution bandwidths were selected with `nomad=TRUE`, plotting reuses the stored LP `regtype.engine` and selected `degree.engine` metadata.

Interval and bootstrap controls

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

B the bootstrap replication count.

band one of "pmzsd", "pointwise", "bonferroni", "simultaneous", or "all".

boot_control optional detailed bootstrap controls created by np_boot_control. This controls non-fixed bootstrap behavior, wild-bootstrap multipliers, and dependent-bootstrap block length.

bootstrap the bootstrap resampling scheme. Regression-family plots support "wild", "inid", "fixed", and "geom"; density/distribution-family plots support "inid", "fixed", and "geom".

boxplot_outliers, factor_boxplot boxplot-style bootstrap summary controls for factor/categorical displays when available.

center one of "estimate" or "bias-corrected". When bootstrap errors are requested, center="bias-corrected" centers plotted uncertainty bands or bars on a bootstrap bias-corrected estimate. The package chooses the validated correction engine for the requested bootstrap route. Wild bootstrap regression routes use the standard wild-bootstrap correction. Pairwise and dependent-bootstrap routes "inid", "fixed", and "geom" use an oversmoothed pilot correction for fixed-bandwidth estimators where that route has been implemented and validated. Unsupported combinations fail clearly rather than silently using a weak correction. For one-dimensional regression curves, the ordinary estimate remains visible and the selected bias-corrected estimate is drawn as an additional center line when available. Fixed-bandwidth regression gradient plots use the same returned gradient-bias convention described below.

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

Rendering and view controls

data_overlay logical; when TRUE, overlay raw data on regression, quantile-regression, single-index, partially linear, and smooth coefficient plot surfaces and expand plot limits to include the data. For npqreg level plots the data layer is drawn first by default, so quantile curves are overlaid on top; ordered and factor conditioning variables use the natural boxplot-style display from plot(). The option is ignored for derivative/gradient plots and coefficient plots. Overlay points default to small, light markers and can be customized with standard points() arguments passed via ... (for example pch, cex, col, and bg). Default is TRUE.

legend legend control for legends drawn by the package, including vector-tau npqreg level plots and 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 custom labels, placement, or styling. For plot routes with more than one package-drawn legend, named sublists such as legend=list(tau=list(x="topright"), bands=FALSE) may be used when a route distinguishes those legend roles.

data_rug logical; when TRUE, draw a rug-style support cue for the plotted coordinates. For ordinary 1D base plots this behaves like a standard rug. For eligible 2D surface plots, the base renderer draws short floor-rug ticks and renderer="rgl" draws the analogous 3D floor-rug ticks on supported surface routes. In this rollout tranche, data_rug=TRUE is implemented across the current base plot families and across supported rgl surface routes; unsupported renderer/geometry combinations still fail clearly rather than being ignored silently. On mixed base plot layouts, rug marks are drawn for continuous panels and omitted for categorical panels. This option is distinct from data_overlay: data_rug shows where observations lie in the plotted covariate support, while data_overlay is the regression-only 3D response overlay used on supported surface plots. On supported regression surfaces, both options may be used together. Default is FALSE.

`perspective` a logical value specifying whether bivariate continuous surfaces are displayed with `persp`-style plots when relevant.

`phi`, `theta`, `view` view controls for bivariate continuous surface plots.

`render_control` optional detailed rendering controls created by `np_render_control`, including band-versus-bar uncertainty display controls.

`renderer` the surface-rendering backend for eligible bivariate continuous plots. `"base"` uses the existing base R `persp` path. `"rgl"` is currently a fail-closed backend for supported plain 2D surface routes in `npreg/npregbw`, `npplreg/npplregbw`, `npscoef/npscoefbw`, `npudens/npudensbw`, `npudist/npudistbw`, `npcdens/npcdensbw`, and `npcdist/npcdistbw`. In this tranche, `renderer="rgl"` requires `perspective=TRUE`, `view="fixed"` and `output="plot"`; unsupported combinations still error rather than falling back silently. The suggested package `rgl` must be installed when `renderer="rgl"` is requested. When drawn, the `rgl` surface is displayed via the active `rgl` backend, which on this rollout path is typically the RStudio Viewer or a browser/WebGL window rather than the base graphics plot pane. When `col` is omitted, the `rgl` surface uses a value-based `hcl.colors(..., "viridis")` gradient. When the inherited base-view defaults `theta=0` and `phi=20` are left unchanged, the `rgl` backend remaps that default camera to a more informative oblique view for this rollout tranche. For the currently supported `npreg`, `npplreg`, and plain mean-surface `npscoef` surface routes, `data_overlay=TRUE` adds observed data as an interactive point cloud. For all currently supported `rgl` surface routes, `errors != "none"` adds interval surfaces; `band="all"` draws pointwise, simultaneous, and bonferroni band families with distinct, colorblind-safer colors when those bands are available for the route. Unsupervised and conditional density/distribution surface routes do not use observed-data point overlays in this tranche. For `npscoef/npscoefbw`, `renderer="rgl"` currently applies only to the plain mean-surface route; coefficient-mode plots remain on the base renderer. Common `rgl` display controls may be supplied directly in `...` on supported routes, including surface arguments such as `alpha`, `back`, and `front`, and view arguments such as `fov` and `zoom`. Additional backend-specific arguments may be passed through using prefixed names in `...`, namely `rgl.persp3d.<arg>`, `rgl.view3d.<arg>`, `rgl.par3d.<arg>`, `rgl.grid3d.<arg>`, and `rgl.widget.<arg>`, and `rgl.legend3d.<arg>`. For the new overlay layers, `rgl.points3d.<arg>` customizes observed-point overlays and `rgl.surface3d.<arg>` customizes interval surfaces. When `band="all"` is used on supported `rgl` routes, a matching interactive legend is drawn, and `rgl.legend3d.<arg>` may be used to customize it, for example `rgl.legend3d.bty="o"` to restore a legend box. By default, these legends follow the base-graphics uncertainty legend convention and are drawn without a surrounding box.

Graphical parameters

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

Reproducibility controls

`random.seed` an optional random seed used to make bootstrap plotting reproducible.

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

`...` 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.

Details

For MPI startup/performance guidance (including message-passing tradeoffs and the manual-broadcast template), see npRmpi.init details and inst/Rprofile. Documentation guide: see np.kernels for kernels and np.options for global options.

The preferred public interface is plot on fitted or bandwidth objects (e.g., plot(fit) or plot(bw)).

plot is a general purpose plotting routine for visually exploring objects generated by the np library, such as regressions, quantile regressions, partially linear regressions, single-index models, densities and distributions. There is no need to call plot directly: plotting is handled by class-specific S3 plot methods for objects generated by the **np** package.

Visualizing one and two dimensional datasets is a straightforward process. The default behavior of plot is to generate a standard 2D plot to visualize univariate data, and a perspective plot for bivariate data. When visualizing higher dimensional data, plot resorts to plotting a series of 1D slices of the data. For a slice along dimension i, all other variables at indices $j \neq i$ are held constant at the quantiles specified in the jth element of xq. The default is the median.

The slice itself is evaluated on a uniformly spaced sequence of $neval$ points. The interval of evaluation is determined by the training data. The default behavior is to evaluate from min(txdat[,i]) to max(txdat[,i]). The xtrim variable allows for control over this behavior. When xtrim is set, data is evaluated from the xtrim[i]th quantile of txdat[,i] to the 1.0-xtrim[i]th quantile of txdat[,i].

Furthermore, xtrim can be set to a negative value in which case it will expand the limits of the evaluation interval beyond the support of the training data, by measuring the distance between min(txdat[,i]) and the xtrim[i]th quantile of txdat[,i], and extending the support by that distance on the lower limit of the interval. plot uses an analogous procedure to extend the upper limit of the interval.

Plot interval/error types are:

```
"pmzsd"        : point estimate +/- z_(1-alpha/2) * standard error
"pointwise"    : per-point two-sided interval from N(0,1) quantiles
"bonferroni"   : pointwise interval with alpha replaced by alpha/m
"simultaneous" : joint-band interval (bootstrap route)
```

where m is the number of evaluation points used in the plotted curve/surface ($m =$ neval for univariate curves, typically $m =$ neval^2 for full 2D perspective surfaces).

When center="bias-corrected" and output="data" (or output="both") is requested, returned plot objects that support bootstrap bias correction include bias and bias.corrected components where available. The sign convention is bias = estimate - bias.corrected, so the corrected estimate can be recovered as estimate - bias. For example, regression plot objects use mean - bias; density objects use dens - bias; and distribution objects use dist - bias. Regression-gradient plot objects use the analogous convention gbias = grad - gradient.bias.corrected, so the corrected gradient can be recovered as grad - gbias. Two-dimensional plots do not draw a second corrected surface in this tranche; use output="data" to recover the corrected surface values.

Kernel estimators are generally biased because smoothing trades variance reduction for bias. For this reason, pointwise and simultaneous error bounds centered on the fitted kernel estimate are often best interpreted as variability bounds rather than literal confidence intervals for the unknown target

function. Confidence-interval interpretation requires the smoothing bias to be negligible relative to sampling variability, or to be corrected with sufficient accuracy.

When bootstrap errors are requested, `center="bias-corrected"` recenters the plotted bootstrap bounds on a method-specific Efron-Tibshirani bootstrap bias-corrected estimate, where available. This produces bounds that are closer in spirit to confidence intervals than variability bounds centered on the original biased estimate. Bias correction for nonparametric estimators is difficult, however, and no universal correction is optimal across estimators, bandwidths, supports, and data types.

For wild-bootstrap regression routes, `center="bias-corrected"` uses the standard wild-bootstrap correction, without a second oversmoothed pilot bandwidth. This follows the wild-bootstrap regression literature and the package's finite-sample evidence that an additional pilot can underadjust in this route. For supported fixed-bandwidth regression-style routes with `bootstrap="inid"`, `"fixed"`, or `"geom"`, `center="bias-corrected"` estimates the bias center from a second pilot bootstrap matrix while retaining the ordinary bootstrap matrix for variability widths. These combinations can behave differently in finite samples, so the returned `bias` and `bias.corrected` components should be inspected when bias correction is scientifically important. For supported fixed regression gradient plots, inspect `gbias`, `gradient.bias.corrected`, or route-specific gradient-bias components such as `gc1bias`; different bootstrap schemes can give materially different derivative-bias corrections.

The pilot rule is family-specific. For fixed-bandwidth regression plots, the non-wild pilot uses the full oversmoothing-rate rule:

$$g_j = h_j n^{2r/((p+2r)(p+4r))},$$

where h_j is the selected bandwidth, r is the continuous kernel order, and p is the number of continuous smoothing variables in that regression route. For ordinary second-order kernels with one continuous smoothing variable this reduces to $g_j = h_j n^{4/45}$. Categorical variables do not enter this continuous-rate exponent. Supported semiparametric regression and single-index plot routes use the square-root version of this rule.

For unconditional density and distribution plots with continuous variables, including mixed-data routes supported by the estimator, the smooth-bootstrap perturbation rule uses $g_j = h_j$, following the density smooth-bootstrap literature. Continuous variables are perturbed and categorical variables are resampled row-wise without perturbation. The smooth perturbation law is drawn from the selected ordinary continuous kernel when that kernel defines a probability law: second-order Gaussian, uniform, and second-order Epanechnikov kernels are supported. Higher-order Gaussian and Epanechnikov kernels remain valid estimation kernels, but their signed equivalent-kernel forms do not define ordinary random perturbations, so `center="bias-corrected"` with pair/block/geometric bootstrap fails clearly for those higher-order smooth-bootstrap perturbation routes. For conditional density/distribution plots, the continuous dependent-variable side uses the same density-style rule $g_y = h_y$; categorical dependent variables are resampled row-wise without perturbation. The conditioning-variable side uses a full regression-style pilot: continuous conditioning bandwidths are enlarged, and categorical conditioning lambdas are moved toward their admissible upper bounds without crossing them. For conditional density this is

$$g_{xj} = h_{xj} n^{2r/((p+2r)(p+4r))},$$

and for conditional distribution this is

$$g_{xj} = h_{xj} n^{r/((p+r)(p+2r))},$$

where here p is the total number of continuous dependent and explanatory variables. The same conditional pilot rule is used for supported fixed-bandwidth conditional density/distribution gradient plots, including mixed-data routes. In mixed-data conditional gradients, continuous dependent variables are perturbed by the density-style rule, categorical dependent variables are resampled rowwise without perturbation, continuous conditioning variables use the conditional regression-style pilot, and categorical conditioning lambdas are moved toward their admissible upper bounds as in the corresponding level route. These are practical, evidence-backed rules-of-thumb guided by the smoothed-bootstrap and regression two-bandwidth literatures, not claims of universal optimality.

In practice, users who want confidence-style bounds should prefer bootstrap errors with `center="bias-corrected"`. The ordinary fitted curve or surface remains the default estimate because bias correction can increase variance and may have larger mean squared error than the uncorrected estimator.

For asymptotic intervals, let $T(x)$ denote the plotted functional (mean, gradient, density, distribution, etc.) and $\widehat{se}(x)$ its asymptotic standard error: $T(x) \pm z_{1-\alpha/2}\widehat{se}(x)$ for `"pmzsd"` and $[T(x)+z_{\alpha/2}\widehat{se}(x),\ T(x)+z_{1-\alpha/2}\widehat{se}(x)]$ for `"pointwise"`. `"bonferroni"` applies the same pointwise construction with α/m in place of α. For the kernel estimators in this package, asymptotic simultaneous bands are not generally available, so `"simultaneous"` with `errors="asymptotic"` returns `NA` bands. Asymptotic standard errors are taken from fitted-object components such as `merr`, `gerr`, `derr`, `conderr`, and `congerr` where implemented.

Bootstrap resampling is conducted pairwise on (y, X, Z) (i.e., by resampling rows of (y, X) or (y, X, Z) as appropriate). Bootstrap method support differs by estimator family:

```
 Regression-family (npreg/npindex/npscoef/npplreg): wild, inid, fixed, geom
Density/distribution-family (npudens/npudist/npcdens/npcdist/npconmode): inid, fixed, geom
```

hence `"wild"` is only available for regression-family plotting. Bias-corrected bootstrap centering is currently restricted to supported fixed-bandwidth plot routes. Non-fixed, adaptive-nearest-neighbor, and generalized-nearest-neighbor bandwidth routes fail closed for `center="bias-corrected"`. Conditional quantile `npqreg` bootstrap plots currently require `center="estimate"`; `center="bias-corrected"` fails clearly for this route. Copula `npcopula` and instrumental-variable `npregiv` bias-corrected bootstrap plot routes are not claimed in this release tranche and fail clearly when unsupported bias-corrected bootstrap controls are requested. Partially linear `npplreg` level plots support the regression-family bootstrap methods above; `gradients=TRUE` is not a partially linear plot route and fails clearly.

Implementation notes for speed:

```
wild            : fast np*hat linear-operator bootstrap path
inid/fixed/geom : fast direct helper path (no internal bandwidth search)
```

For non-fixed density/distribution bootstrap, an explicit experimental approximation is available via `boot_control = np_boot_control(nonfixed = "exact")` or `boot_control = np_boot_control(nonfixed = "frozen")`. The default `"exact"` route recomputes the non-fixed geometry for each resample. The experimental `"frozen"` route reuses the original-sample non-fixed geometry throughout the bootstrap run. This option is currently implemented only for unconditional and conditional density/distribution bootstrap routes and remains off by default. For generalized/adaptive nearest-neighbor runs, `"frozen"` is an approximation that can alter interval/band width by holding the original-sample nearest-neighbor geometry fixed across bootstrap resamples; `"exact"` remains the

recommended setting for production inference. This approximation can be more noticeable for conditional density/distribution plotting than for the regression-style plot families because the conditional bootstrap paths freeze both numerator and denominator nearest-neighbor geometry before recombining them. In practice, conditional distribution bands are often closer, while conditional density bands can differ more materially from `"exact"` under generalized/adaptive nearest-neighbor bandwidths. For smooth coefficient plots (`npscoef`) under non-fixed bandwidths, `"exact"` can also be much more expensive than `"frozen"` on large jobs, because the coefficient field must be recomputed for each bootstrap resample rather than reusing the original-sample geometry. This recomputation cannot in general be avoided without a more aggressive approximation: for `npscoef` the local weighted systems that define the coefficient vector depend on the bootstrap resample weights/counts at each evaluation point, so unlike `npplreg` there is no single global coefficient vector that can be updated once per draw. `inid` admits general heteroskedasticity of unknown form, though it does not allow for dependence. `fixed` conducts Kunsch's (1988) block bootstrap for dependent data, while `geom` conducts Politis and Romano's (1994) stationary bootstrap.

For local polynomial conditional density/distribution plotting (`npcdens`/`npcdist` with `regtype="ll"` or `regtype="lp"`) and `proper=TRUE`, the plotted estimate is rendered proper slice-by-slice on the fixed evaluation grid: each conditional density slice is projected to be nonnegative and to integrate to one using trapezoidal quadrature weights from the evaluation y-grid, while each conditional distribution slice is projected to be monotone and bounded in $[0, 1]$. When `errors="bootstrap"`, the bootstrap resample surfaces are computed first on that same fixed grid and then properized resample-by-resample using the same grid geometry before `"pointwise"`, `"bonferroni"`, `"simultaneous"`, and `"all"` bands are constructed. Thus the bootstrap distribution used to form these bands is built from properized resample surfaces. The final lower/upper band surfaces are interval envelopes and are not themselves separately re-projected to satisfy the density/distribution shape constraints.

For consistency of the block and stationary bootstrap, the (mean) block length b should grow with the sample size n at an appropriate rate. If b is not given, then a default growth rate of $const \times n^{1/3}$ is used. This rate is "optimal" under certain conditions (see Politis and Romano (1994) for more details). However, in general the growth rate depends on the specific properties of the DGP. A default value for $const$ (3.15) has been determined by a Monte Carlo simulation using a Gaussian AR(1) process (AR(1)-parameter of 0.5, 500 observations). $const$ has been chosen such that the mean square error for the bootstrap estimate of the variance of the empirical mean is minimized.

The default bootstrap replication count is `B=1999`. For pointwise tails, ensure $B \geq \lceil 2/\alpha - 1 \rceil$ so $\alpha(B+1)$ is feasible on the bootstrap rank grid. For interval types `"bonferroni"`, `"simultaneous"`, and `"all"`, the minimum recommended count is $B_{\min} = \lceil 2m/\alpha - 1 \rceil$, where m is the number of evaluation points used by the plotted curve/surface. For full 2D perspective grids this is typically $m = \texttt{neval}^2$. When `B` is below these thresholds, plotting proceeds but warning guidance is reported.

Typical plotting calls:

```
## Asymptotic pointwise/bonferroni intervals
plot(fit, errors="asymptotic", band="pointwise")
plot(fit, errors="asymptotic", band="bonferroni")

## Regression-family bootstrap (wild available)
plot(fit, errors="bootstrap", bootstrap="wild")

## Density/distribution-family bootstrap (use inid/fixed/geom)
plot(fit, errors="bootstrap", bootstrap="inid")
```

New public plot controls use snake_case. Dotted names are reserved for S3 methods and internal plot-engine plumbing. Use errors, bootstrap, B, band, alpha, center, output, data_overlay, data_rug, layout, and the control constructors documented in np_boot_control, np_grid_control, and np_render_control.

Value

Setting output will instruct plot what data to return. Option summary:
plot: instruct plot to just plot the data and return NULL
both: instruct plot to plot the data and return the data used to generate the plots. The data will be a list of objects of the appropriate type, with one object per plot. For example, invoking plot on 3D density data will have it return a list of three npdensity objects. If biases were calculated, they are stored in a component named bias
data: instruct plot to generate data only and no plots

Usage Issues

If you are using data of mixed types, then it is advisable to use the data.frame function to construct your input data and not cbind, since cbind will typically not work as intended on mixed data types and will coerce the data to the same type.

For npRmpi, plotting calls (including bootstrap error paths) are supported directly under npRmpi.autodispatch. No explicit MPI wrapping is required when autodispatch is enabled. Manual MPI control remains available for advanced workflows.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

References

Aitchison, J. and C.G.G. Aitken (1976), "Multivariate binary discrimination by the kernel method," Biometrika, 63, 413-420.

Cao-Abad, R. (1991), "Rate of convergence for the wild bootstrap in nonparametric regression," The Annals of Statistics, 19, 2226-2231.

Hall, P. and J.S. Racine and Q. Li (2004), "Cross-validation and the estimation of conditional probability densities," Journal of the American Statistical Association, 99, 1015-1026.

Hall, P. (1992a), "Effect of bias estimation on coverage accuracy of bootstrap confidence intervals for a probability density," The Annals of Statistics, 20, 675-694.

Hall, P. (1992b), "On bootstrap confidence intervals in nonparametric regression," The Annals of Statistics, 20, 695-711.

Hardle, W. and J.S. Marron (1991), "Bootstrap simultaneous error bars for nonparametric regression," The Annals of Statistics, 19, 778-796.

Kunsch, H.R. (1989), "The jackknife and the bootstrap for general stationary observations," The Annals of Statistics, 17, 1217-1241.

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

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

Politis, D.N. and J.P. Romano (1994), "The stationary bootstrap," Journal of the American Statistical Association, 89, 1303-1313.

Silverman, B.W. and G.A. Young (1987), "The bootstrap: to smooth or not to smooth?" Biometrika, 74, 469-479.

De Angelis, D. and G.A. Young (1992), "Smoothing the bootstrap," International Statistical Review, 60, 45-56.

Scott, D.W. (1992), *Multivariate Density Estimation. Theory, Practice and Visualization,* New York: Wiley.

Silverman, B.W. (1986), *Density Estimation,* London: Chapman and Hall.

Wang, M.C. and J. van Ryzin (1981), "A class of smooth estimators for discrete distributions," Biometrika, 68, 301-309.

See Also

np.kernels, np.options, npRmpi.init

Examples

```
## Not run:

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

# EXAMPLE 1: For this example, we load Giovanni Baiocchi's Italian GDP
# panel (see Italy for details), then create a data frame in which year
# is an ordered factor, GDP is continuous, compute bandwidths using
# likelihood cross-validation, then create a grid of data on which the
# density will be evaluated for plotting purposes

data("Italy")
with(Italy, {

data <- data.frame(ordered(year), gdp)

# Compute bandwidths using likelihood cross-validation (default). Note
# that this may take a minute or two depending on the speed of your
# computer...

bw <- npudensbw(dat=data)
fit <- npudens(tdat=data, bws=bw)
```

```
# You can always do things manually, as the following example demonstrates

# Create an evaluation data matrix

year.seq <- sort(unique(year))
gdp.seq <- seq(1,36,length=50)
data.eval <- expand.grid(year=year.seq,gdp=gdp.seq)

# Generate the estimated density computed for the evaluation data

fhat <- fitted(npudens(tdat = data, edat = data.eval, bws=bw))

# Coerce the data into a matrix for plotting with persp()

f <- matrix(fhat, length(unique(year)), 50)

# Next, create a 3D perspective plot of the PDF f

persp(as.integer(levels(year.seq)), gdp.seq, f, col="lightblue",
      ticktype="detailed", ylab="GDP", xlab="Year", zlab="Density",
      theta=300, phi=50)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# However, plot simply streamlines this process and aids in the
# visualization process (<ctrl>-C will interrupt on *NIX systems, <esc>
# will interrupt on MS Windows systems).

plot(fit)

# plot also streamlines construction of variability bounds (<ctrl>-C
# will interrupt on *NIX systems, <esc> will interrupt on MS Windows
# systems)

plot(fit, errors = "asymptotic")

# EXAMPLE 2: For this example, we simulate multivariate data, and plot the
# partial regression surfaces for a locally linear estimator and its
# derivatives.

set.seed(123)

n <- 100

x1 <- runif(n)
x2 <- runif(n)
x3 <- runif(n)
x4 <- rbinom(n, 2, .3)

y <- 1 + x1 + x2 + x3 + x4 + rnorm(n)
```

```
X <- data.frame(x1, x2, x3, ordered(x4))

bw <- npregbw(xdat=X, ydat=y, regtype="ll", bwmethod="cv.aic")
fit <- npreg(bws=bw)

plot(fit)

# Sleep for 5 seconds so that we can examine the output...

Sys.sleep(5)

# Now plot the gradients...

plot(fit, gradients=TRUE)

# Plot the partial regression surfaces with bias-corrected bootstrapped
# nonparametric confidence intervals... this may take a minute or two
# depending on the speed of your computer as the bootstrapping must be
# completed prior to results being displayed...

plot(fit,
     errors="bootstrap",
     center="bias-corrected",
     band="simultaneous")

# EXAMPLE 3: This example demonstrates how to retrieve plotting data from
# plot(). When plot() is called with the arguments
# `output="both"' (or "data"), it returns plotting objects
# named r1, r2, and so on (rg1, rg2, and so on when `gradients=TRUE' is
# set).  Each plotting object's index (1,2,...) corresponds to the index
# of the explanatory data data frame xdat (and zdat if appropriate).

# Take the cps71 data by way of example. In this case, there is only one
# object returned by default, `r1', since xdat is univariate.

data("cps71", package = "npRmpi")

# Compute bandwidths for local linear regression using cv.aic...

bw <- npregbw(xdat=cps71$age, ydat=cps71$logwage,
              regtype="ll", bwmethod="cv.aic")
fit <- npreg(bws=bw)

# Generate the plot and return plotting data, and store output in
# `plot.out' (NOTE: the call to `output' is necessary).

plot.out <- plot(fit,
                 perspective=FALSE,
                 errors="bootstrap",
                 B=25,
                 output="both")

# Now grab the r1 object that plot plotted on the screen, and take
```

```
# what you need. First, take the output, lower error bound and upper
# error bound...

logwage.eval <- fitted(plot.out$r1)
logwage.se <- se(plot.out$r1)
logwage.lower.ci <- logwage.eval + logwage.se[,1]
logwage.upper.ci <- logwage.eval + logwage.se[,2]

# Next grab the x data evaluation data. xdat is a data.frame(), so we
# need to coerce it into a vector (take the `first column' of data frame
# even though there is only one column)

age.eval <- plot.out$r1$eval[,1]

# Now we could plot this if we wished, or direct it to whatever end use
# we envisioned. We plot the results using R's plot() routines...

with(cps71, plot(age, logwage, cex=0.2, xlab="Age", ylab="log(Wage)"))
lines(age.eval,logwage.eval)
lines(age.eval,logwage.lower.ci,lty=3)
lines(age.eval,logwage.upper.ci,lty=3)

# Bias-corrected bootstrap output includes the estimated bias and the
# bias-corrected center when available. The sign convention is
# bias = estimate - bias.corrected, so bias.corrected = estimate - bias.

plot.out <- plot(fit,
                 perspective=FALSE,
                 errors="bootstrap",
                 center="bias-corrected",
                 B=25,
                 output="data")

logwage.eval <- fitted(plot.out$r1)
logwage.bias <- plot.out$r1$bias
logwage.bc <- plot.out$r1$bias.corrected
stopifnot(all.equal(logwage.bc, logwage.eval - logwage.bias))

# If you wanted plot() data for gradients, you would use the argument
# `gradients=TRUE' in the call to plot() as the following
# demonstrates...

plot.out <- plot(fit,
                 perspective=FALSE,
                 errors="bootstrap",
                 B=25,
                 output="both",
                 gradients=TRUE)

# Now grab object that plot() plotted on the screen. First, take the
# output, lower error bound and upper error bound... note that gradients
# are stored in objects rg1, rg2 etc.
```

```
grad.eval <- gradients(plot.out$rg1)
grad.se <- gradients(plot.out$rg1, errors = TRUE)
grad.lower.ci <- grad.eval + grad.se[,1]
grad.upper.ci <- grad.eval + grad.se[,2]

# Next grab the x evaluation data. xdat is a data.frame(), so we need to
# coerce it into a vector (take `first column' of data frame even though
# there is only one column)

age.eval <- plot.out$rg1$eval[,1]

# We plot the results using R's plot() routines...

plot(age.eval,grad.eval,cex=0.2,
     ylim=c(min(grad.lower.ci),max(grad.upper.ci)),
     xlab="Age",ylab="d log(Wage)/d Age",type="l")
lines(age.eval,grad.lower.ci,lty=3)
lines(age.eval,grad.upper.ci,lty=3)

# EXAMPLE 4: Variations on local polynomial conditional density
# estimation with proper = TRUE.

data("Italy")

Italy2 <- within(Italy, {
  year <- as.numeric(as.character(year))
})

# Plot only: make the plotted surface proper on the plot evaluation grid.

fhat <- npcdens(gdp ~ year, data = Italy2,
                regtype = "lp", degree = 3, nmulti = 1)

plot(fhat, proper = TRUE)

# Fit an object whose fitted values are themselves proper.

ctrl_fit <- list(
  mode = "slice",
  apply = "fitted",
  slice.grid.size = 101L,
  slice.extend.factor = 0.1
)

fhat_fit <- npcdens(
  gdp ~ year,
  data = Italy2,
  regtype = "lp",
  degree = 3,
  nmulti = 1,
  proper = TRUE,
  proper.control = ctrl_fit
)
```

```
fit_proper <- fitted(fhat_fit)
fit_raw <- fhat_fit$condens.raw

# Display the repaired and raw fitted values for cases where the raw
# fitted density is negative.

head(cbind(fit_proper, fit_raw)[which(fit_raw < 0), ])

# Predict on a common explicit y-grid for several years, and render
# those predictions proper.

g.grid <- seq(min(Italy2$gdp), max(Italy2$gdp), length.out = 200)

nd_grid <- expand.grid(
  gdp = g.grid,
  year = c(1955, 1975, 1995)
)

pred_grid <- predict(fhat, newdata = nd_grid, proper = TRUE)

# Predict on paired rows with different gdp grids by year, and still
# make the predictions proper via slice mode.

g1 <- seq(quantile(Italy2$gdp, 0.10),
          quantile(Italy2$gdp, 0.60), length.out = 60)
g2 <- seq(quantile(Italy2$gdp, 0.30),
          quantile(Italy2$gdp, 0.90), length.out = 35)

nd_slice <- rbind(
  data.frame(gdp = g1, year = rep(1960, length(g1))),
  data.frame(gdp = g2, year = rep(1985, length(g2)))
)

pred_slice <- predict(
  fhat,
  newdata = nd_slice,
  proper = TRUE,
  proper.control = list(mode = "slice")
)

# One object that carries properization for fitted values and for later
# predict() calls.

ctrl_both <- list(
  mode = "slice",
  apply = "both",
  slice.grid.size = 101L,
  slice.extend.factor = 0.1
)

fhat_both <- npcdens(
  gdp ~ year,
```

```
  data = Italy2,
  regtype = "lp",
  degree = 3,
  nmulti = 1,
  proper = TRUE,
  proper.control = ctrl_both
)

fit_both <- fitted(fhat_both)
pred_both <- predict(
  fhat_both,
  newdata = nd_slice,
  proper.control = ctrl_both
)
plot(fhat_both)
})
npRmpi.quit()
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

se *Extract Standard Errors*

Description

se is a generic function which extracts standard errors from objects.

Usage

```
se(x)
```

Arguments

Object Input: Object to interrogate for standard errors.

x	an object for which the extraction of standard errors is meaningful.

Details

This function provides a generic interface for extraction of standard errors from objects.

Value

Standard errors extracted from the model object x.

Note

This method currently only supports objects from the npRmpi library.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

See Also

fitted, residuals, coef, and gradients, for related methods; npRmpi for supported objects; npRmpi.init for MPI session startup.

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## The following example is adapted for interactive parallel execution
## in R. Here we spawn 1 slave so that there will be two compute nodes
## (master and slave).  Kindly see the batch examples in the demos
## directory (npRmpi/demos) and study them carefully. Also kindly see
## the more extensive examples in the np package itself. See the npRmpi
## vignette for further details on running parallel np programs via
## vignette("npRmpi_getting_started", package = "npRmpi").

## Start npRmpi for interactive execution. If slaves are already running and
## `options(npRmpi.reuse.slaves=TRUE)` (default on some systems), this will
## reuse the existing pool instead of respawning. To change the number of
## slaves, call `npRmpi.quit(force=TRUE)` then restart.

force.run <- nzchar(Sys.getenv("NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK"))

in.check <- !force.run && (
            nzchar(Sys.getenv("_R_CHECK_PACKAGE_NAME_")) ||
            nzchar(Sys.getenv("_R_CHECK_INTERNALS2_")) ||
            nzchar(Sys.getenv("_R_CHECK_CRAN_INCOMING_")) ||
            nzchar(Sys.getenv("_R_CHECK_EXAMPLE_TIMING_THRESHOLD_"))
          )

if (!in.check) {
npRmpi.init(nslaves=1)

set.seed(42)

x <- rnorm(10)
bw <- npudensbw(~x)
fhat <- npudens(bw)

se(fhat)

## For the interactive run only we close the slaves perhaps to proceed
## with other examples and so forth. This is redundant in batch mode.

## Note: on some systems (notably macOS+MPICH), repeatedly spawning and
## tearing down slaves in the same R session can lead to hangs/crashes.
## npRmpi may therefore keep slave daemons alive by default and
## `npRmpi.quit()` performs a "soft close". Use `force=TRUE` to
```

```
## actually shut down the slaves.
##
## You can disable reuse via `options(npRmpi.reuse.slaves=FALSE)` or by
## setting the environment variable `NP_RMPI_NO_REUSE_SLAVES=1` before
## loading the package.

npRmpi.quit()               ## soft close (may keep slaves alive)
## npRmpi.quit(force=TRUE)  ## hard close
} else {
 message("Skipping MPI spawn in check context (set NP_RMPI_RUN_MPI_EXAMPLES_IN_CHECK=1).")
}

## End(Not run)
```

---

uocquantile *Compute Quantiles*

Description

uocquantile is a function which computes quantiles of an unordered, ordered or continuous variable x.

Usage

```
uocquantile(x, prob)
```

Arguments

Variable And Probability Inputs: Variable and requested probability for mixed-type quantile extraction.

x	an ordered, unordered or continuous variable.
prob	quantile to compute.

Details

uocquantile is a function which computes quantiles of an unordered, ordered or continuous variable x. If x is unordered, the mode is returned. If x is ordered, the level for which the cumulative distribution is >= prob is returned. If x is continuous, quantile is invoked and the result returned.

Value

A quantile computed from x.

Author(s)

Tristen Hayfield <tristen.hayfield@gmail.com>, Jeffrey S. Racine <racinej@mcmaster.ca>

See Also

quantile

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
x <- rbinom(n = 100, size = 10, prob = 0.5)
uocquantile(x, 0.5)

## End(Not run)
```

---

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. `data("wage1")` makes available the dataset `"wage"` plus additional objects `"bw.all"` and `"bw.subset"`.

Usage

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

Format

A data frame with 24 columns, and 526 rows.

Two local-linear `rbandwidth` objects (`bw.all` and `bw.subset`) have been computed for the user's convenience which can be used to visualize this dataset using `plot(bw.all)`

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 dependants

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 construction industry

ndurman column 14, of type `numeric`, =1 if in non-durable manufacturing industry

trcommpu column 15, of type `numeric`, =1 if in transportation, communications, public utility

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

services column 17, of type `numeric`, =1 if in services industry

profserv column 18, of type `numeric`, =1 if in professional services industry

profocc column 19, of type `numeric`, =1 if in professional 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:
## Not run in checks: excluded to keep MPI examples stable and check times short.
data("wage1")
with(wage1, {
summary(wage1)
})

## End(Not run)
```

Index

∗ **datasets**
- cps71, 5
- Engel95, 8
- Italy, 13
- oecdpanel, 314
- wage1, 334

∗ **density**
- npcdenshat, 59
- npudenshat, 277

∗ **distribution**
- npcdisthat, 84
- npudisthat, 295

∗ **hplot**
- np.pairs, 33
- np_plot_controls, 35

∗ **instrument**
- npregiv, 198
- npregivderiv, 208

∗ **interface**
- mpi.barrier, 16
- mpi.bcast, 16
- mpi.comm.free, 21
- mpi.comm.size, 22
- mpi.get.processor.name, 24
- mpi.get.version, 24

∗ **misc**
- np.options, 30

∗ **nonparametric**
- b.star, 4
- gradients, 11
- np.kernels, 26
- npcdens, 36
- npcdensbw, 44
- npcdenshat, 59
- npcdist, 61
- npcdistbw, 69
- npcdisthat, 84
- npcmstest, 86
- npconmode, 90
- npcopula, 99
- npdeneqtest, 107
- npdeptest, 110
- npindex, 114
- npindexbw, 120
- npksum, 131
- nplsqreg, 137
- nplsqregbw, 141
- npplreg, 144
- npplregbw, 150
- npqcmstest, 160
- npqreg, 164
- npquantile, 171
- npreg, 175
- npregbw, 182
- npreghat, 196
- npRmpi.init, 218
- npscoef, 222
- npscoefbw, 228
- npsdeptest, 241
- npseed, 244
- npsemihat, 246
- npsigtest, 250
- npsymtest, 255
- npudens, 260
- npudensbw, 266
- npudenshat, 277
- npudist, 278
- npudistbw, 284
- npudisthat, 295
- npuniden.boundary, 297
- npuniden.reflect, 301
- npuniden.sc, 305
- npunitest, 311
- plot.np, 316
- se, 331
- uocquantile, 333

∗ **package**
- npRmpi, 213

∗ **regression**
nplsqreg, 137
nplsqregbw, 141
npreghat, 196
npsemihat, 246
∗ **semiparametric**
npsemihat, 246
∗ **smooth**
np.kernels, 26
npuniden.boundary, 297
npuniden.reflect, 301
npuniden.sc, 305
∗ **univar**
b.star, 4
npdeptest, 110
npsdeptest, 241
npsymtest, 255
npunitest, 311
uocquantile, 333
∗ **utilities**
lamhosts, 15
mpi.bcast.cmd, 17
mpi.bcast.Robj, 19
mpi.close.Rslaves, 20
mpi.exit, 23
mpi.hostinfo, 25

as.data.frame, *37*, *47*, *62*, *72*, *86*, *91*, *104*, *115*, *122*, *133*, *145*, *152*, *161*, *165*, *176*, *190*, *223*, *230*, *251*, *261*, *268*, *279*, *286*
as.data.frame.npcopula *(*npcopula*)*, 99

b.star, 4, *256*
boxplot.stats, *174*
bw *(*oecdpanel*)*, 314
bw.all *(*wage1*)*, 334
bw.nrd, *58*, *83*, *226*, *276*, *294*
bw.SJ, *58*, *83*, *226*, *256*, *276*, *294*, *312*
bw.subset *(*wage1*)*, 334

cbind, *40*, *57*, *66*, *82*, *88*, *96*, *105*, *109*, *118*, *129*, *135*, *140*, *148*, *158*, *162*, *169*, *180*, *194*, *214*, *226*, *238*, *253*, *264*, *275*, *282*, *293*, *324*
coef, *12*, *117*, *129*, *147*, *225*, *332*
colMeans, *135*
contour, *103, 104*
cps71, 5

data.frame, *40*, *57*, *66*, *82*, *88*, *96*, *105*, *109*, *118*, *129*, *135*, *140*, *148*, *158*, *162*, *169*, *180*, *194*, *214*, *226*, *238*, *253*, *264*, *275*, *282*, *293*, *317*, *324*
density, *264*, *282*
dimBS, 7

ecdf, *174*
Engel95, 8
expand.grid, *103*
extendrange, *101*, *172, 173*

factor, *39*, *55*, *64*, *80*, *134*, *147*, *156*, *178*, *191*, *214*, *257*, *263*, *274*
fitted, *12*, *39*, *65*, *95*, *104*, *117*, *140*, *147*, *168*, *179*, *225*, *263*, *281*, *332*
fitted.lsqregression *(*nplsqreg*)*, 137
fitted.npcopula *(*npcopula*)*, 99
fivenum, *174*

glm, *86*, *88*
gradients, 11, *12*, *39*, *65*, *95*, *117*, *140*, *168*, *179*, *332*

hcl.colors, *101*, *319*
hist, *58*, *83*, *226*, *276*, *294*

image, *103, 104*
Italy, 13

lamhosts, 15
legend, *318*
lm, *86*, *88*, *214*
loess, *181*
lsqregressionbandwidth *(*nplsqregbw*)*, 141

mpi.barrier, 16
mpi.bcast, 16, *17*, *19, 20*
mpi.bcast.cmd, 17, *17*
mpi.bcast.Robj, *17*, 19
mpi.bcast.Robj2slave, *17*
mpi.bcast.Robj2slave *(*mpi.bcast.Robj*)*, 19
mpi.close.Rslaves, 20
mpi.comm.dup *(*mpi.comm.size*)*, 22
mpi.comm.free, 21
mpi.comm.rank, *25*
mpi.comm.rank *(*mpi.comm.size*)*, 22
mpi.comm.size, *21*, 22, *25*
mpi.exit, 23

mpi.get.processor.name, 24, *25*
mpi.get.version, 24
mpi.hostinfo, *15*, 25
mpi.is.master *(*lamhosts*)*, 15
mpi.quit *(*mpi.exit*)*, 23

na.action, *47*, *72*, *122*, *134*, *152*, *190*, *231*, *268*, *287*
na.fail, *47*, *72*, *122*, *134*, *152*, *190*, *231*, *268*, *287*
na.omit, *47*, *72*, *122*, *134*, *152*, *190*, *231*, *268*, *287*
nlm, *134*
np.kernels, 26, *29, 30*, *32, 33*, *38*, *41*, *49, 50*, *55*, *58*, *64*, *66*, *75*, *79*, *83*, *87*, *89*, *93*, *97*, *103*, *105*, *108, 109*, *111*, *113*, *116*, *119*, *127*, *134*, *140*, *144*, *147*, *149*, *156*, *158*, *162, 163*, *167*, *170*, *172*, *174*, *178*, *181*, *188*, *191*, *195*, *202*, *205*, *210, 211*, *215*, *218, 219*, *226*, *232, 233*, *236*, *239*, *241*, *243*, *245*, *252*, *256, 257*, *259*, *262*, *264*, *270*, *273*, *276*, *279, 280*, *282*, *288*, *291*, *294*, *298*, *300*, *302, 303*, *307, 308*, *312, 313*, *320*, *325*
np.mpi.initialize, 29, *219*
np.options, *26*, *29*, 30, *33*, *38*, *41*, *55*, *58*, *64*, *66*, *79*, *83*, *87*, *89*, *93*, *97*, *103*, *105*, *108, 109*, *111*, *113*, *116*, *119*, *127*, *134*, *140*, *144*, *147*, *149*, *156*, *158*, *162, 163*, *167*, *170*, *172*, *174*, *178*, *181*, *191*, *195*, *202*, *205*, *210, 211*, *215*, *218, 219*, *226*, *236*, *239*, *241*, *243*, *245*, *252*, *256, 257*, *259*, *262*, *264*, *273*, *276*, *279*, *282*, *291*, *294*, *298*, *300*, *302, 303*, *307, 308*, *312, 313*, *320*, *325*
np.pairs, 33
np_boot_control, *102*, *318*, *324*
np_boot_control *(*np_plot_controls*)*, 35
np_grid_control, *317*, *324*
np_grid_control *(*np_plot_controls*)*, 35
np_plot_controls, 35
np_render_control, *319*, *324*
np_render_control *(*np_plot_controls*)*, 35
npcdens, 36, *57*, *59*, *92*, *95*, *316*
npcdensbw, *29*, *37, 38*, 44, *91–93*, *95*, *97*, *245*, *316*
npcdenshat, 59
npcdist, 61, *82*, *84*
npcdistbw, *29*, *62–64*, 69, *143*, *165–169*
npcdisthat, 84
npcmstest, 86, *245*
npconmode, 90, *316*
npcopula, 99
npdeneqtest, 107, *113*, *243*, *257*, *313*
npdeptest, *109*, 110, *243*, *257*, *313*
npindex, 114, *316*
npindexbw, *115*, *117*, *119*, 120, *316*
npindexhat *(*npsemihat*)*, 246
npksum, *29*, *128*, 131, *202*, *209*, *215*
nplsqreg, 137, *141*, *143, 144*
nplsqregbw, *138–140*, 141
npplreg, 144, *157*, *316*
npplregbw, *145–147*, 150, *245*, *316*
npplreghat *(*npsemihat*)*, 246
npqcmstest, 160, *245*
npqreg, *65*, *82*, *137*, *140*, *143*, 164, *245*
npquantile, 171, *173*
npreg, *12*, *33*, *139, 140*, *143, 144*, *149*, *158*, 175, *191*, *193*, *195*, *205*, *209*, *211*, *214*, *239*, *316*
npregbw, *29*, *87*, *89*, *134*, *142–144*, *146, 147*, *149*, *152*, *155, 156*, *158*, *161*, *163*, *176–179*, 182, *239*, *245*, *251*, *316*
npreghat, *60*, *85*, 196
npregiv, 198, *211*
npregivderiv, *205*, 208
npRmpi, *12*, *32*, *39*, *47*, *55*, *64*, *72*, *80*, *116*, *122*, *134, 135*, *146, 147*, *152*, *156*, *177*, *179*, *190, 191*, 213, *224*, *230*, *237*, *244, 245*, *256*, *263*, *268*, *274*, *281*, *286*, *292*, *331, 332*
npRmpi-package *(*npRmpi*)*, 213
npRmpi.init, *12*, *15*, *21*, *26*, *29, 30*, *32, 33*, *38*, *55*, *64*, *79*, *87*, *89*, *93*, *97*, *108*, *111*, *116*, *119*, *127*, *130*, *134*, *136*, *147*, *156*, *162, 163*, *167*, *172*, *178*, *191*, *202*, *210*, *215*, 218, *219*, *226*, *236*, *241*, *245*, *252, 253*, *256*, *259, 260*, *262*, *273*, *279*, *291*, *298*, *302*, *307*, *312*, *320*, *325*, *332*
npRmpi.quit, *21*, *29*
npRmpi.quit *(*npRmpi.init*)*, 218
npRmpi.session.info *(*npRmpi.init*)*, 218
npscoef, 222, *316*
npscoefbw, *223–226*, 228, *316*

npscoefhat (npsemihat), 246
npsdeptest, *109*, *113*, 241, *257*, *313*
npseed, 244
npsemihat, 246
npsigtest, *245*, 250
npsymtest, *109*, *113*, *243*, 255, *313*
nptgauss, *216*, 259
npudens, *33*, *41*, *58*, *66*, *83*, *105*, *226*, 260, *273*, *275–277*, *302*, *316*
npudensbw, *29*, *100, 101*, *103*, *105*, *226*, *245*, *261, 262*, *264*, 266, *302*, *316*
npudenshat, 277
npudist, *58*, *83*, *105*, *173*, *226*, *276*, 278, *281*, *291*, *293–295*
npudistbw, *29*, *100, 101*, *103*, *105*, *172, 173*, *279, 280*, *282*, 284
npudisthat, 295
npuniden.boundary, 297, *303*, *308*
npuniden.reflect, *300*, 301
npuniden.sc, 305
npunitest, *109*, *113*, *243*, *257*, 311
numeric, *111*, *135*, *172*, *214*, *241*, *257*

oecdpanel, 314
on.exit, *32*
optim, *31*, *125, 126*, *128*, *200, 201*, *235, 236*
options, *32*
ordered, *39*, *55*, *64*, *80*, *134*, *147*, *156*, *179*, *191*, *214*, *263*, *274*, *281*, *292*

persp, *102–104*, *319*
plot, *26*, *29, 30*, *32, 33*, *38, 39*, *41*, *55*, *57, 58*, *64–66*, *79*, *82, 83*, *87*, *89*, *93*, *95*, *97*, *103–105*, *108, 109*, *111*, *113*, *116, 117*, *119*, *127*, *129*, *134*, *140*, *147*, *149*, *156*, *158*, *162, 163*, *167, 168*, *170*, *172*, *174*, *178, 179*, *181*, *191*, *193*, *195*, *202*, *204, 205*, *210, 211*, *214, 215*, *218, 219*, *225, 226*, *236*, *238, 239*, *241*, *243*, *245*, *252*, *256, 257*, *259*, *262–264*, *273, 274*, *276*, *279*, *281, 282*, *291*, *293, 294*, *298*, *300*, *302, 303*, *307, 308*, *312, 313*, *315, 316*, *320*, *334*
plot.bandwidth (plot.np), 316
plot.conbandwidth (plot.np), 316
plot.conmode, *92*
plot.conmode (npconmode), 90
plot.lsqregression (nplsqreg), 137
plot.np, *38, 39*, *41*, *55*, *58*, *64*, *66*, *79, 80*, *83*, *93*, *116*, *119*, *127*, *140*, *147*, *149*, *156*, *158*, *167*, *170*, *178*, *181*, *191*, *195*, *226*, *236*, *239*, *262*, *264*, *273*, *276*, *279, 280*, *282*, *291*, *294*, 316
plot.npcopula (npcopula), 99
plot.plbandwidth (plot.np), 316
plot.qregression (npqreg), 164
plot.rbandwidth (plot.np), 316
plot.scbandwidth (plot.np), 316
plot.sibandwidth (plot.np), 316
predict, *39*, *57*, *65*, *82*, *94, 95*, *104*, *117*, *129*, *140*, *147*, *167, 168*, *179*, *193*, *225*, *238*, *263*, *274*, *281*, *293*
predict.conmode (npconmode), 90
predict.lsqregression (nplsqreg), 137
predict.npcopula (npcopula), 99
predict.npreghat (npreghat), 196
predict.qregression (npqreg), 164
print, *140*, *204*, *210*
print.lsqregression (nplsqreg), 137
print.lsqregressionbandwidth (nplsqregbw), 141
print.npcopula (npcopula), 99
print.npreghat (npreghat), 196

qkde, *174*
qlogspline, *174*
quantile, *140*, *168*, *173, 174*, *333, 334*
quantile.lsqregression (nplsqreg), 137

residuals, *12*, *117*, *140*, *147*, *179*, *225*, *332*
residuals.lsqregression (nplsqreg), 137
rq, *161*

se, *12*, *39*, *65*, *104*, *117*, *140*, *168*, *179*, *225*, *263*, *281*, 331
se.lsqregression (nplsqreg), 137
se.npcopula (npcopula), 99
set.seed, *245*
slave.hostinfo, *15*
slave.hostinfo (mpi.hostinfo), 25
summary, *39*, *57*, *65*, *82*, *88*, *95*, *104*, *109*, *112*, *117*, *129*, *140*, *162*, *168*, *179*, *193*, *204*, *210*, *225*, *238*, *242*, *252*, *256*, *263*, *274*, *281*, *293*, *313*
summary.lsqregression (nplsqreg), 137
summary.lsqregressionbandwidth (nplsqregbw), 141

summary.npcopula *(*npcopula*)*, 99

tailslave.log *(*mpi.close.Rslaves*)*, 20
ts, *241*

uocquantile, 333

vcov, *117, 118*, *147*

wage1, 334