Package 'npRmpi'

February 16, 2026

Version 0.60-20

Date 2026-02-11

Imports boot, cubature, methods, quadprog, quantreg, stats, parallel

Suggests crs, MASS, logspline, ks, testthat, np, Rmpi

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

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

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

Repository CRAN

NeedsCompilation yes

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

Date/Publication 2026-02-16 17:20:13 UTC

Contents

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`	data, an n x k matrix, each column being a data series.
`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)`.
`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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

data("cps71")
mpi.bcast.Robj2slave(cps71)

attach(cps71)

plot(age, logwage, xlab="Age", ylab="log(wage)")

detach(cps71)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## End(Not run)
```

`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",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.stop(force=TRUE)` then restart.
npRmpi.start(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(attach(Engel95),
              caller.execute=TRUE)

mpi.bcast.cmd(model.iv <- npregiv(y=food,z=logexp,w=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(food~logexp,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(logexp,food,
     ylab="Food Budget Share",
     xlab="log(Total Expenditure)",
     xlim=quantile(logexp,c(trim,1-trim)),
     ylim=quantile(food,c(trim,1-trim)),
     main="Nonparametric Instrumental Kernel Regression",
     type="p",
     cex=.5,
     col="lightgrey")

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

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

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

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)
## 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, ...)

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

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

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

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

Arguments

`x`	an object for which the extraction of gradients is meaningful.
`...`	other arguments.
`errors`	a logical value specifying whether or not standard errors of gradients are desired. Defaults to `FALSE`.

Details

This function provides a generic interface for extraction of gradients from objects.

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.

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

set.seed(42)

x <- runif(10)
y <- x + rnorm(10, sd = 0.1)
mydat <- data.frame(x,y)
rm(x,y)

mpi.bcast.Robj2slave(mydat)

mpi.bcast.cmd(model <- npreg(y~x, data=mydat, gradients=TRUE),
              caller.execute=TRUE)

gradients(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.stop()` 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.stop()              ## soft close (may keep slaves alive)
## npRmpi.stop(force=TRUE)  ## hard close

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

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

attach(Italy)

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

detach(Italy)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## End(Not run)
```

lamhosts *Hosts Information*

Description

lamhosts finds the host name associated with its node number. Can be used by mpi.spawn.Rslaves to spawn R slaves on selected hosts. This is a MPI implementation specific function.

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

mpi.hostinfo finds an individual host information including rank and size in a comm.

slave.hostinfo is executed only by master and find all master and slaves host information in a comm.

Usage

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

Arguments

comm	a communicator number
short	if true, a short form is printed

Value

lamhosts returns CPUs nodes numbers with their host names.

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

mpi.hostinfo sends to stdio a host name, rank, size and comm.

slave.hostname sends to stdio a list of host, rank, size, and comm information for all master and slaves. With short=TRUE and 8 slaves or more, the first 3 and last 2 slaves are shown.

Author(s)

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

See Also

mpi.spawn.Rslaves

mpi.abort *MPI_Abort API*

Description

mpi.abort makes a “best attempt" to abort all tasks in a comm.

Usage

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

Arguments

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/

See Also

mpi.finalize

mpi.any.source *MPI Constants*

Description

Find MPI constants: MPI_ANY_SOURCE, MPI_ANY_TAG, or MPI_PROC_NULL

Usage

```
mpi.any.source()
mpi.any.tag()
mpi.proc.null()
```

Arguments

None

Details

These constants are mainly used by mpi.send, mpi.recv, and mpi.probe. Different implementation of MPI may use different integers for MPI_ANY_SOURCE, MPI_ANY_TAG, and MPI_PROC_NULL. Hence one should use these functions instead of real integers for MPI communications.

Value

Each function returns an integer value.

References

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

See Also

mpi.send, mpi.recv.

`mpi.apply` *Scatter an array to slaves and then apply a FUN*

Description

An array (length <= total number of slaves) is scattered to slaves so that the first slave calls `FUN` with arguments `x[[1]]` and `...`, the second one calls with arguments `x[[2]]` and `...`, and so on. `mpi.iapply` is a nonblocking version of `mpi.apply` so that it will not consume CPU on master node.

Usage

```
mpi.apply(X, FUN, ..., comm=1)
mpi.iapply(X, FUN, ..., comm=1, sleep=0.01)
```

Arguments

`X`	an array
`FUN`	a function
`...`	optional arguments to `FUN`
`comm`	a communicator number
`sleep`	a sleep interval on master node (in sec)

Value

A list of the results is returned. Its length is the same as that of `x`. In case the call `FUN` with arguments `x[[i]]` and `...` fails on ith slave, corresponding error message will be returned in the returning list.

Author(s)

Hao Yu

Examples

```
## Not run:
# Not run in checks: requires pre-spawned slaves and a live worker communicator.
# Running this without the expected MPI session can deadlock.
#Assume that there are at least 5 slaves running
#Otherwise run mpi.spawn.Rslaves(nslaves=5)
x=c(10,20)
mpi.apply(x,runif)
meanx=1:5
mpi.apply(meanx,rnorm,n=2,sd=4)

## End(Not run)
```

---

mpi.applyLB *(Load balancing) parallel apply*

Description

(Load balancing) parallel`lapply` and related functions.

Usage

```
mpi.applyLB(X, FUN, ..., apply.seq=NULL, comm=1)
mpi.parApply(X, MARGIN, FUN, ..., job.num = mpi.comm.size(comm)-1,
                    apply.seq=NULL, comm=1)
mpi.parLapply(X, FUN, ..., job.num=mpi.comm.size(comm)-1, apply.seq=NULL,
comm=1)
mpi.parSapply(X, FUN, ..., job.num=mpi.comm.size(comm)-1, apply.seq=NULL,
simplify=TRUE, USE.NAMES = TRUE, comm=1)
mpi.parRapply(X, FUN, ..., job.num=mpi.comm.size(comm)-1, apply.seq=NULL,
comm=1)
mpi.parCapply(X, FUN, ..., job.num=mpi.comm.size(comm)-1, apply.seq=NULL,
comm=1)
mpi.parReplicate(n, expr, job.num=mpi.comm.size(comm)-1, apply.seq=NULL,
simplify = TRUE, comm=1)
mpi.parMM (A, B, job.num=mpi.comm.size(comm)-1, comm=1)
```

Arguments

`X`	an array or matrix.
`MARGIN`	vector specifying the dimensions to use.
`FUN`	a function.
`simplify`	logical; should the result be simplified to a vector or matrix if possible?

`USE.NAMES`	logical; if `TRUE` and if `X` is character, use `X` as `names` for the result unless it had names already.
`n`	number of replications.
`A`	a matrix
`B`	a matrix
`expr`	expression to evaluate repeatedly.
`job.num`	Total job numbers. If job numbers is bigger than total slave numbers (default value), a load balancing approach is used.
`apply.seq`	if reproducing the same computation (simulation) is desirable, set it to the integer vector .mpi.applyLB generated in previous computation (simulation).
`...`	optional arguments to `FUN`
`comm`	a communicator number

Details

Unless length of `X` is no more than total slave numbers (slave.num) and in this case `mpi.applyLB` is the same as `mpi.apply`, `mpi.applyLB` sends a next job to a slave who just delivered a finished job. The sequence of slaves who deliver results to master are saved into `.mpi.applyLB`. It keeps track of which slaves do which parts of the results. `.mpi.applyLB` can be used to reproduce the same simulation result if the same seed is used and the argument `apply.seq` is equal to `.mpi.applyLB`.

With the default value of argument `job.num` which is slave.num, `mpi.parApply`, `mpi.parLapply`, `mpi.parSapply`, `mpi.parRapply`, `mpi.parCapply`, `mpi.parSapply`, and `mpi.parMM` are clones of **snow**'s parApply, parLappy, parSapply, parRapply, parCapply, parSapply, and parMM, respectively. When `job.num` is bigger than slave.num, a load balancing approach is used.

Value

Returns an object with the same structure as the corresponding base apply call (typically a list or simplified vector/array when ‘simplify = TRUE‘).

Warning

When using the argument `apply.seq` with `.mpi.applyLB`, be sure all settings are the same as before, i.e., the same data, job.num, slave.num, and seed. Otherwise a deadlock could occur. Notice that `apply.seq` is useful only if `job.num` is bigger than slave.num.

See Also

`mpi.apply`

Examples

```
## Not run:
# Not run in checks: requires pre-spawned slaves and load-balancing state.
# A mismatched communicator or apply.seq can deadlock.
#Assume that there are some slaves running
```

```
#mpi.applyLB
x=1:7
mpi.applyLB(x,rnorm,mean=2,sd=4)

#get the same simulation
mpi.remote.exec(set.seed(111))
mpi.applyLB(x,rnorm,mean=2,sd=4)
mpi.remote.exec(set.seed(111))
mpi.applyLB(x,rnorm,mean=2,sd=4,apply.seq=.mpi.applyLB)

#mpi.parApply
x=1:24
dim(x)=c(2,3,4)
mpi.parApply(x, MARGIN=c(1,2), FUN=mean,job.num = 5)

#mpi.parLapply
mdat <- matrix(c(1,2,3, 7,8,9), nrow = 2, ncol=3, byrow=TRUE,
                    dimnames = list(c("R.1", "R.2"), c("C.1", "C.2", "C.3")))
mpi.parLapply(mdat, rnorm)

#mpi.parSapply
mpi.parSapply(mdat, rnorm)

#mpi.parMM
A=matrix(1:1000^2,ncol=1000)
mpi.parMM(A,A)

## End(Not run)
```

`mpi.barrier` *MPI_Barrier API*

Description

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

Usage

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

Arguments

`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

`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.
`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

cmd	a command to be sent from master.
...	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).
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

Details

mpi.bcast.cmd is a collective call. This means all members in a communicator must execute it at the same time. If slaves are spawned (created) by using slavedaemon.R (Rprofile script), then they are running mpi.bcast.cmd in infinite loop (idle state). Hence master can execute mpi.bcast.cmd alone to start computation. On the master, cmd and ... are put together as a list which is then broadcasted (after serialization) to all slaves (using for loop with mpi.send and mpi.recv pair). All slaves will return an expression which will be evaluated by either slavedaemon.R, or by whatever an R script based on slavedaemon.R.

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>)

See Also

mpi.remote.exec

---

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)
mpi.bcast.Rfun2slave(comm = 1)
mpi.bcast.data2slave(obj, comm = 1, buffunit = 100)
```

Arguments

obj	an R object to be transmitted from the sender
rank	the sender.
comm	a communicator number.
all	a logical. If TRUE, all R objects on master are transmitted to slaves.
buffunit	a buffer unit number.

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.

`mpi.bcast.data2slave` transfers data (a double vector or a matrix) natively without (un)serilization. It should be used with a huge vector or matrix. It results in less memory usage and faster transmission. Notice that data with missing values (NA) are allowed.

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. `mpi.bcast.Rfun2slave` transmits all master's functions to slaves and returns no value. `mpi.bcast.data2slave` transmits a double vector or a matrix to slaves and returns no value.

Author(s)

Hao Yu

See Also

`mpi.send.Robj`, `mpi.recv.Robj`,

`mpi.cart.coords` *MPI_Cart_coords*

Description

`mpi.cart.coords` translates a rank to its Cartesian topology coordinate.

Usage

```
mpi.cart.coords(comm=3, rank, maxdims)
```

Arguments

`comm`	Communicator with Cartesian structure
`rank`	rank of a process within group
`maxdims`	length of vector coord in the calling program

Details

This function is the rank-to-coordinates translator. It is the inverse map of `mpi.cart.rank`. maxdims is at least as big as ndims as returned by `mpi.cartdim.get`.

Value

`mpi.cart.coords` returns an integer array containing the Cartesian coordinates of a specified process.

Author(s)

Alek Hunchak and Hao Yu

References

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

See Also

`mpi.cart.rank`

Examples

```
## Not run:
# Not run in checks: requires a Cartesian communicator built from spawned slaves.
#Need at least 9 slaves
mpi.bcast.cmd(mpi.cart.create(1,c(3,3),c(F,T)))
mpi.cart.create(1,c(3,3),c(F,T))
mpi.cart.coords(3,4,2)

## End(Not run)
```

`mpi.cart.create` *MPI_Cart_create*

Description

`mpi.cart.create` creates a Cartesian structure of arbitrary dimension.

Usage

```
mpi.cart.create(commold=1, dims, periods, reorder=FALSE, commcart=3)
```

Arguments

`commold`	Input communicator
`dims`	Integery array of size ndims specifying the number of processes in each dimension
`periods`	Logical array of size ndims specifying whether the grid is periodic or not in each dimension
`reorder`	ranks may be reordered or not
`commcart`	The new communicator to which the Cartesian topology information is attached

Details

If reorder = false, then the rank of each process in the new group is the same as its rank in the old group. If the total size of the Cartesian grid is smaller than the size of the group of commold, then some processes are returned mpi.comm.null. The call is erroneous if it specifies a grid that is larger than the group size.

Value

`mpi.cart.create` returns 1 if success and 0 otherwise.

Author(s)

Alek Hunchak and Hao Yu

References

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

Examples

```
## Not run:
# Not run in checks: requires a multi-rank MPI session with spawned slaves.
#Need at least 9 slaves
mpi.bcast.cmd(mpi.cart.create(1,c(3,3),c(F,T)))
mpi.cart.create(1,c(3,3),c(F,T))

## End(Not run)
```

`mpi.cart.get` *MPI_Cart_get*

Description

`mpi.cart.get` provides the user with information on the Cartesian topology associated with a comm.

Usage

```
 mpi.cart.get(comm=3, maxdims)
```

Arguments

`comm`	Communicator with Cartesian structure
`maxdims`	length of vectors dims, periods, and coords in the calling program

Details

The coords are as given for the rank of the calling process as shown.

Value

mpi.cart.get returns a vector containing information on the Cartesian topology associated with comm. maxdims must be at least ndims as returned by mpi.cartdim.get.

Author(s)

Alek Hunchak and Hao Yu

References

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

See Also

mpi.cart.create,mpi.cartdim.get

Examples

```
## Not run:
# Not run in checks: requires a Cartesian communicator built from spawned slaves.
#Need at least 9 slaves
mpi.bcast.cmd(mpi.cart.create(1,c(3,3),c(F,T)))
mpi.cart.create(1,c(3,3),c(F,T))
mpi.remote.exec(mpi.cart.get(3,2))

## End(Not run)
```

---

mpi.cart.rank *MPI_Cart_rank*

Description

mpi.cart.rank translates a Cartesian topology coordinate to its rank.

Usage

```
mpi.cart.rank(comm=3, coords)
```

Arguments

comm	Communicator with Cartesian structure
coords	Specifies the Cartesian coordinates of a process

Details

For a process group with a Cartesian topology, this function translates the logical process coordinates to process ranks as they are used by the point-to-point routines. It is the inverse map of mpi.cart.coords.

Value

mpi.cart.rank returns the rank of the specified process.

Author(s)

Alek Hunchak and Hao Yu

References

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

See Also

mpi.cart.coords

Examples

```
## Not run:
# Not run in checks: requires a Cartesian communicator built from spawned slaves.
#Need at least 9 slaves
mpi.bcast.cmd(mpi.cart.create(1,c(3,3),c(F,T)))
mpi.cart.create(1,c(3,3),c(F,T))
mpi.cart.rank(3,c(1,0))

## End(Not run)
```

---

mpi.cart.shift *MPI_Cart_shift*

Description

mpi.cart.shift shifts the Cartesian topology in both manners, displacement and direction.

Usage

```
mpi.cart.shift(comm=3, direction, disp)
```

Arguments

comm	Communicator with Cartesian structure
direction	Coordinate dimension of the shift
disp	displacement (>0 for upwards or left shift, <0 for downwards or right shift)

Details

mpi.cart.shift provides neighbor ranks from given direction and displacement. The direction argument indicates the dimension of the shift. direction=1 means the first dim, direction=2 means the second dim, etc. disp=1 or -1 provides immediate neighbor ranks and disp=2 or -2 provides neighbor's neighbor ranks. Negative ranks mean out of boundary. They correspond to mpi.proc.null.

Value

`mpi.cart.shift` returns a vector containing information regarding the rank of the source process and rank of the destination process.

Author(s)

Alek Hunchak and Hao Yu

References

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

See Also

`mpi.cart.create`,`mpi.proc.null`

Examples

```
## Not run:
# Not run in checks: requires a Cartesian communicator built from spawned slaves.
#Need at least 9 slaves
mpi.bcast.cmd(mpi.cart.create(1,c(3,3),c(F,T)))
mpi.cart.create(1,c(3,3),c(F,T))
mpi.remote.exec(mpi.cart.shift(3,2,1))#get neighbor ranks
mpi.remote.exec(mpi.cart.shift(3,1,1))

## End(Not run)
```

`mpi.cartdim.get` *MPI_Cartdim_get*

Description

`mpi.cartdim.get` gets dim information about a Cartesian topology.

Usage

```
mpi.cartdim.get(comm=3)
```

Arguments

`comm`	Communicator with Cartesian structure

Details

Can be used to provide other functions with the correct size of arrays.

Value

`mpi.cartdim.get` returns the number of dimensions of the Cartesian structure

Author(s)

Alek Hunchak and Hao Yu

References

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

See Also

`mpi.cart.get`

Examples

```
## Not run:
# Not run in checks: requires a Cartesian communicator built from spawned slaves.
#Need at least 9 slaves
mpi.bcast.cmd(mpi.cart.create(1,c(3,3),c(F,T)))
mpi.cart.create(1,c(3,3),c(F,T))
mpi.cartdim.get(comm=3)

## End(Not run)
```

`mpi.comm.disconnect` *MPI_Comm_disconnect API*

Description

`mpi.comm.disconnect` disconnects itself from a communicator and then deallocates the communicator so it points to MPI_COMM_NULL.

Usage

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

Arguments

`comm`	a communicator number

Details

When members associated with a communicator finish jobs or exit, they have to call `mpi.comm.disconnect` to release resource if the communicator was created from an intercommunicator by `mpi.intercomm.merge`. If `mpi.comm.free` is used instead, `mpi.finalize` called by slaves may cause undefined impacts on master who wishes to stay.

Value

1 if success. Otherwise 0.

Author(s)

Hao Yu

References

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

See Also

`mpi.comm.free`

`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

`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. If the comm was created from an intercommunicator by `mpi.intercomm.merge`, use `mpi.comm.disconnect` instead.

Value

1 if success. Otherwise 0.

Author(s)

Hao Yu

References

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

See Also

`mpi.comm.disconnect`

`mpi.comm.get.parent` *MPI_Comm_get_parent, MPI_Comm_remote_size, MPI_Comm_test_inter APIs*

Description

`mpi.comm.get.parent` is mainly used by slaves to find the intercommunicator or the parent who spawns them. The intercommunicator is saved in the specified comm number.

`mpi.comm.remote.size` is mainly used by master to find the total number of slaves spawned.

`mpi.comm.test.inter` tests if a comm is an intercomm or not.

Usage

```
mpi.comm.get.parent(comm = 2)
mpi.comm.remote.size(comm = 2)
mpi.comm.test.inter(comm = 2)
```

Arguments

`comm`	an intercommunicator number.

Value

`mpi.comm.get.parent` and `mpi.comm.test.inter` return 1 if success and 0 otherwise.

`mpi.comm.remote.size` returns the total number of members in the remote group in an intercomm.

Author(s)

Hao Yu

References

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

See Also

`mpi.intercomm.merge`

mpi.comm.set.errhandler *MPI_Comm_set_errhandler API*

Description

`mpi.comm.set.errhandler` sets a communicator to MPI_ERRORS_RETURN instead of MPI_ERRORS_ARE_FATAL (default) which crashes R on any type of MPI errors. Almost all MPI API calls return errcodes which can map to specific MPI error messages. All MPI related error messages come from predefined MPI_Error_string.

Usage

```
mpi.comm.set.errhandler(comm = 1)
```

Arguments

`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.comm.size *MPI_Comm_c2f, MPI_Comm_dup, MPI_Comm_rank, and MPI_Comm_size APIs*

Description

`mpi.comm.c2f` converts the comm (a C communicator) and returns an integer that can be used as the communicator in external FORTRAN code. `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.c2f(comm=1)
  mpi.comm.dup(comm, newcomm)
  mpi.comm.rank(comm = 1)
  mpi.comm.size(comm = 1)
```

Arguments

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

Value

- `mpi.comm.c2f`: integer communicator for use in FORTRAN code.
- `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.comm.spawn` *MPI_Comm_spawn API*

Description

`mpi.comm.spawn` tries to start `nslaves` identical copies of `slaves`, establishing communication with them and returning an intercommunicator. The spawned slaves are referred to as children, and the process that spawned them is called the parent (master). The children have their own MPI_COMM_WORLD represented by comm 0. To make communication possible among master and slaves, all slaves should use `mpi.comm.get.parent` to find their parent and use `mpi.intercomm.merge` to merger an intercomm to a comm.

Usage

```
mpi.comm.spawn(slave, slavearg = character(0),
               nslaves = mpi.universe.size(), info = 0,
               root = 0, intercomm = 2, quiet = FALSE)
```

Arguments

`slave`	a file name to an executable program.
`slavearg`	an argument list (a char vector) to slave.
`nslaves`	number of slaves to be spawned.
`info`	an info number.
`root`	the root member who spawns slaves.
`intercomm`	an intercomm number.
`quiet`	a logical. If TRUE, do not print anything unless an error occurs.

Value

Unless `quiet = TRUE`, a message is printed to indicate how many slaves are successfully spawned and how many failed.

Author(s)

Hao Yu

References

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

See Also

`mpi.comm.get.parent`, `mpi.intercomm.merge`.

`mpi.dims.create` *MPI_Dims_create*

Description

`mpi.dims.create` Create a Cartesian dimension used by `mpi.cart.create`.

Usage

```
mpi.dims.create(nnodes, ndims, dims=integer(ndims))
```

Arguments

`nnodes`	Number of nodes in a cluster
`ndims`	Number of dimension in a Cartesian topology
`dims`	Initial dimension numbers

Details

The entries in the return value are set to describe a Cartesian grid with `ndims` dimensions and a total of `nnodes` nodes. The dimensions are set to be as close to each other as possible, using an appropriate divisibility algorithm. The return value can be constrained by specifying positive number(s) in `dims`. Only those 0 values in `dims` are modified by `mpi.dims.create`.

Value

`mpi.dims.create` returns the dimension vector used by that in `mpi.cart.create`.

Author(s)

Hao Yu

References

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

See Also

`mpi.cart.create`

Examples

```
## Not run:
## Not run in checks when toggled to dontrun: this MPI utility example is
## intended for manual interactive use.
#What is the dim numbers of 2 dim Cartersian topology under a grid of 36 nodes
mpi.dims.create(36,2) #return c(6,6)
#Constrained dim numbers
mpi.dims.create(12,2,c(0,4)) #return c(9,4)

## End(Not run)
```

`mpi.exit` *Exit MPI Environment*

Description

`mpi.exit` terminates MPI execution environment and detaches the library Rmpi. After that, you can still work on R.

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

Usage

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

Arguments

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

Details

Normally, `mpi.finalize` is used to clean all MPI states. However, it will not detach the library Rmpi. To be more safe leaving MPI, `mpi.exit` not only calls `mpi.finalize` but also detaches the library Rmpi. This will make reloading of the library Rmpi impossible.

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

Value

`mpi.exit` always returns 1

Author(s)

Hao Yu

See Also

`mpi.finalize`

`mpi.finalize` *MPI_Finalize API*

Description

Terminates MPI execution environment.

Usage

```
mpi.finalize()
```

Arguments

None

Details

This routines must be called by each slave (master) before it exits. This call cleans all MPI state. Once `mpi.finalize` has been called, no MPI routine may be called. To be more safe leaving MPI, please use `mpi.exit` which not only calls `mpi.finalize` but also detaches the library Rmpi. This will make reload the library Rmpi impossible.

Value

Always return 1

Author(s)

Hao Yu

References

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

See Also

`mpi.exit`

`mpi.gather` *MPI_Gather, MPI_Gatherv, MPI_Allgather, and MPI_Allgatherv APIs*

Description

`mpi.gather` and `mpi.gatherv` (vector variant) gather each member's message to the member specified by the argument `root`. The root member receives the messages and stores them in rank order. `mpi.allgather` and `mpi.allgatherv` are the same as `mpi.gather` and `mpi.gatherv` except that all members receive the result instead of just the root.

Usage

```
mpi.gather(x, type, rdata, root = 0, comm = 1)
mpi.gatherv(x, type, rdata, rcounts, root = 0, comm = 1)

mpi.allgather(x, type, rdata, comm = 1)
mpi.allgatherv(x, type, rdata, rcounts, comm = 1)
```

Arguments

`x`	data to be gathered. Must be the same type.
`type`	1 for integer, 2 for double, and 3 for character. Others are not supported.
`rdata`	the receive buffer. Must be the same type as the sender and big enough to include all message gathered.
`rcounts`	int vector specifying the length of each message.
`root`	rank of the receiver
`comm`	a communicator number

Details

For `mpi.gather` and `mpi.allgather`, the message to be gathered must be the same dim and the same type. The receive buffer can be prepared as either integer(size * dim) or double(size * dim), where size is the total number of members in a comm. For `mpi.gatherv` and `mpi.allgatherv`, the message to be gathered can have different dims but must be the same type. The argument `rcounts` records these different dims into an integer vector in rank order. Then the receive buffer can be prepared as either integer(sum(rcounts)) or double(sum(rcounts)).

Value

For `mpi.gather` or `mpi.gatherv`, it returns the gathered message for the root member. For other members, it returns what is in rdata, i.e., rdata (or rcounts) is ignored. For `mpi.allgather` or `mpi.allgatherv`, it returns the gathered message for all members.

Author(s)

Hao Yu

References

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

See Also

`mpi.scatter`, `mpi.scatterv`.

Examples

```
## Not run:
# Not run in checks: requires a fixed number of spawned slaves and rank-specific buffers.
# Running this with a different communicator layout can deadlock.
#Need 3 slaves to run properly
#Or use mpi.spawn.Rslaves(nslaves=3)
 mpi.bcast.cmd(id <-mpi.comm.rank(.comm), comm=1)
mpi.bcast.cmd(mpi.gather(letters[id],type=3,rdata=string(1)))
mpi.gather(letters[10],type=3,rdata=string(4))

 mpi.bcast.cmd(x<-rnorm(id))
 mpi.bcast.cmd(mpi.gatherv(x,type=2,rdata=double(1),rcounts=1))
 mpi.gatherv(double(1),type=2,rdata=double(sum(1:3)+1),rcounts=c(1,1:3))

mpi.bcast.cmd(out1<-mpi.allgatherv(x,type=2,rdata=double(sum(1:3)+1),
rcounts=c(1,1:3)))
mpi.allgatherv(double(1),type=2,rdata=double(sum(1:3)+1),rcounts=c(1,1:3))

## End(Not run)
```

`mpi.gather.Robj` *Extentions of MPI_Gather and MPI_Allgather APIs*

Description

`mpi.gather.Robj` gathers each member's object to the member specified by the argument `root`. The root member receives the objects as a list. `mpi.allgather.Robj` is the same as `mpi.gather.Robj` except that all members receive the result instead of just the root.

Usage

```
mpi.gather.Robj(obj=NULL, root = 0, comm = 1, ...)

mpi.allgather.Robj(obj=NULL, comm = 1)
```

Arguments

`obj`	data to be gathered. Could be different type.
`root`	rank of the gather
`comm`	a communicator number
`...`	optional arugments to `sapply`.

Details

Since sapply is used to gather all results, its default option "simplify=TRUE" is to simplify outputs. In some situations, this option is not desirable. Using "simplify=FALSE" as in the place of ... will tell sapply not to simplify and a list of outputs will be returned.

Value

For `mpi.gather.Robj`, it returns a list, the gathered message for the root member. For `mpi.allgatherv.Robj`, it returns a list, the gathered message for all members.

Author(s)

Hao Yu and Wei Xia

References

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

See Also

`mpi.gather`, `mpi.allgatherv`.

Examples

```
## Not run:
# Not run in checks: requires pre-spawned slaves and a live worker communicator.
#Assume that there are some slaves running
mpi.bcast.cmd(id<-mpi.comm.rank())
mpi.bcast.cmd(x<-rnorm(id))
mpi.bcast.cmd(mpi.gather.Robj(x))
x<-"test mpi.gather.Robj"
mpi.gather.Robj(x)

mpi.bcast.cmd(obj<-rnorm(id+10))
mpi.bcast.cmd(nn<-mpi.allgather.Robj(obj))
obj<-rnorm(5)
mpi.allgather.Robj(obj)
mpi.remote.exec(nn)

## End(Not run)
```

mpi.get.count *MPI_Get_count API*

Description

`mpi.get.count` finds the length of a received message.

Usage

```
mpi.get.count(type, status = 0)
```

Arguments

`type`	1 for integer, 2 for double, 3 for char.
`status`	a status number

Details

When `mpi.recv` is used to receive a message, the receiver buffer can be set to be bigger than the incoming message. To find the exact length of the received message, `mpi.get.count` is used to find its exact length. `mpi.get.count` must be called immediately after calling `mpi.recv` otherwise the status may be changed.

Value

the length of a received message.

Author(s)

Hao Yu

References

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

See Also

`mpi.send`, `mpi.recv`, `mpi.get.sourcetag`, `mpi.probe`.

`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

`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.sourcetag` *Utility for finding the source and tag of a received message*

Description

`mpi.get.sourcetag` finds the source and tag of a received message.

Usage

```
mpi.get.sourcetag(status = 0)
```

Arguments

`status`	a status number

Details

When mpi.any.source and/or mpi.any.tag are used by mpi.recv or mpi.probe, one can use mpi.get.sourcetag to find who sends the message or with what tag number. mpi.get.sourcetag must be called immediately after calling mpi.recv or mpi.probe otherwise the obtained information may not be right.

Value

2 dim int vector. The first integer is the source and the second is the tag.

Author(s)

Hao Yu

References

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

See Also

mpi.send, mpi.recv, mpi.probe, mpi.get.count

---

mpi.iapplyLB *(Load balancing) parallel apply with nonblocking features*

Description

(Load balancing) parallellapply and related functions.

Usage

```
mpi.iapplyLB(X, FUN, ..., apply.seq=NULL, comm=1, sleep=0.01)
mpi.iparApply(X, MARGIN, FUN, ..., job.num = mpi.comm.size(comm)-1,
                    apply.seq=NULL, comm=1, sleep=0.01)
mpi.iparLapply(X, FUN, ..., job.num=mpi.comm.size(comm)-1, apply.seq=NULL,
    comm=1,sleep=0.01)
mpi.iparSapply(X, FUN, ..., job.num=mpi.comm.size(comm)-1, apply.seq=NULL,
simplify=TRUE, USE.NAMES = TRUE, comm=1, sleep=0.01)
mpi.iparRapply(X, FUN, ..., job.num=mpi.comm.size(comm)-1, apply.seq=NULL,
comm=1, sleep=0.01)
mpi.iparCapply(X, FUN, ..., job.num=mpi.comm.size(comm)-1, apply.seq=NULL,
comm=1,sleep=0.01)
mpi.iparReplicate(n, expr, job.num=mpi.comm.size(comm)-1, apply.seq=NULL,
simplify = TRUE, comm=1,sleep=0.01)
mpi.iparMM(A, B, comm=1, sleep=0.01)
```

Arguments

`X`	an array or matrix.
`MARGIN`	vector specifying the dimensions to use.
`FUN`	a function.
`simplify`	logical; should the result be simplified to a vector or matrix if possible?
`USE.NAMES`	logical; if `TRUE` and if `X` is character, use `X` as `names` for the result unless it had names already.
`n`	number of replications.
`A`	a matrix
`B`	a matrix
`expr`	expression to evaluate repeatedly.
`job.num`	Total job numbers. If job numbers is bigger than total slave numbers (default value), a load balancing approach is used.
`apply.seq`	if reproducing the same computation (simulation) is desirable, set it to the integer vector .mpi.applyLB generated in previous computation (simulation).
`...`	optional arguments to `Fun`
`comm`	a communicator number
`sleep`	a sleep interval on master node (in sec)

Details

`mpi.iparApply`, `mpi.iparLapply`, `mpi.iparSapply`, `mpi.iparRapply`, `mpi.iparCapply`, `mpi.iparSapply`, `mi.iparReplicate`, and `mpi.iparMM` are nonblocking versions of `mpi.parApply`, `mpi.parLapply`, `mpi.parSapply`, `mpi.parRapply`, `mpi.parCapply`, `mpi.parSapply`, `mpi.parReplicate`, and `mpi.parMM` respectively. The main difference is that `mpi.iprobe` and `Sys.sleep` are used so that master node consumes almost no CPU cycles while waiting for slaves results. However, due to frequent wake/sleep cycles on master, those functions are not suitable for running small jobs on slave nodes. If anticipated computing time for each job is relatively long, e.g., minutes or hours, setting sleep to be 1 second or longer will further reduce load on master (only slightly).

Value

Returns an object with the same structure as the corresponding base or ‘mpi.par*‘ apply call (typically a list or simplified vector/array when ‘simplify = TRUE‘).

See Also

`mpi.iapply`

mpi.info.create *MPI_Info_create, MPI_Info_free, MPI_Info_get, MPI_Info_set APIs*

Description

Many MPI APIs take an info argument for additional information passing. An info is an object which consists of many (key,value) pairs. Rmpi uses an internal memory to store an info object.

`mpi.info.create` creates a new info object.

`mpi.info.free` frees an info object and sets it to MPI_INFO_NULL.

`mpi.info.get` retrieves the value associated with key in an info.

`mpi.info.set` adds the key and value pair to info.

Usage

```
mpi.info.create(info = 0)
mpi.info.free(info = 0)
mpi.info.get(info = 0, key, valuelen)
mpi.info.set(info = 0, key, value)
```

Arguments

`info`	an info number.
`key`	a char (length 1).
`valuelen`	the length (nchar) of a key
`value`	a char (length 1).

Value

`mpi.info.create`, `mpi.info.free`, and `mpi.info.set` return 1 if success and 0 otherwise.

`mpi.info.get` returns the value (a char) for a given info and valuelen.

Author(s)

Hao Yu

See Also

`mpi.spawn.Rslaves`

`mpi.intercomm.merge` *MPI_Intercomm_merge API*

Description

Creates an intracommunicator from an intercommunicator

Usage

```
mpi.intercomm.merge(intercomm=2, high=0, comm=1)
```

Arguments

`intercomm`	an intercommunicator number
`high`	Used to order the groups of the two intracommunicators within comm when creating the new communicator
`comm`	a (intra)communicator number

Details

When master spawns slaves, an intercommunicator is created. To make communications (point-to-point or groupwise) among master and slaves, an intracommunicator must be created. `mpi.intercomm.merge` is used for that purpose. This is a collective call so all master and slaves call together. R slaves spawned by `mpi.spawn.Rslaves` should use `mpi.comm.get.parent` to get (set) an intercomm to a number followed by merging antercomm to an intracomm. One can use `mpi.comm.test.inter` to test if a communicator is an intercommunicator or not.

Value

1 if success. Otherwise 0.

Author(s)

Hao Yu

References

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

See Also

`mpi.comm.test.inter`

`mpi.parSim` *Parallel Monte Carlo Simulation*

Description

Carry out parallel Monte Carlo simulation on R slaves spawned by using slavedaemon.R script and all executed results are returned back to master.

Usage

```
mpi.parSim(n=100, rand.gen=rnorm, rand.arg=NULL,statistic,
nsim=100, run=1, slaveinfo=FALSE, sim.seq=NULL, simplify=TRUE, comm=1, ...)
```

Arguments

`n`	sample size.
`rand.gen`	the random data generating function. See the details section
`rand.arg`	additional argument list to `rand.gen`.
`statistic`	the statistic function to be simulated. See the details section
`nsim`	the number of simulation carried on a slave which is counted as one slave job.
`run`	the number of looping. See the details section.
`slaveinfo`	if TRUE, the numbers of jobs finished by slaves will be displayed.
`sim.seq`	if reproducing the same simulation is desirable, set it to the integer vector .mpi.parSim generated in previous simulation.
`simplify`	logical; should the result be simplified to a vector or matrix if possible?
`comm`	a communicator number
`...`	optional arguments to `statistic`

Details

It is assumed that one simulation is carried out as `statistic(rand.gen(n))`, where `rand.gen(n)` can return any values as long as `statistic` can take them. Additional arguments can be passed to `rand.gen` by `rand.arg` as a list. Optional arguments can also be passed to `statistic` by the argument `...`.

Each slave job consists of `replicate(nsim,statistic(rand.gen(n)))`, i.e., each job runs `nsim` number of simulation. The returned values are transported from slaves to master.

The total number of simulation (TNS) is calculated as follows. Let slave.num be the total number of slaves in a `comm` and it is `mpi.comm.size(comm)-1`. Then TNS=slave.num*nsim*run and the total number of slave jobs is slave.num*run, where `run` is the number of looping from master perspective. If run=1, each slave will run one slave job. If run=2, each slave will run two slaves jobs on average, and so on.

The purpose of using `run` has two folds. It allows a tuneup of slave job size and total number of slave jobs to deal with two different cluster environments. On a cluster of slaves with equal CPU power, `run=1` is often enough. But if `nsim` is too big, one can set `run=2` and the slave job size to be `nsim/2` so that TNS=slave.num*(nsim/2)*(2*run). This may improve R computation efficiency slightly. On a cluster of slaves with different CPU power, one can choose a big value of `run` and a small value of `nsim` so that master can dispatch more jobs to slaves who run faster than others. This will keep all slaves busy so that load balancing is achieved.

The sequence of slaves who deliver results to master are saved into `.mpi.parSim`. It keeps track of which slaves do which parts of the results. `.mpi.parSim` can be used to reproduce the same simulation result if the same seed is used and the argument `sim.seq` is equal to `.mpi.parSim`.

See the warning section before you use `mpi.parSim`.

Value

The returned values depend on values returned by `replicate` of `statistic(rand.gen(n))` and the total number of simulation (TNS). If `statistic` returns a single value, then the result is a vector of length TNS. If `statistic` returns a vector (list) of length `nrow`, then the result is a matrix of dimension `c(nrow, TNS)`.

Warning

It is assumed that a parallel RNG is used on all slaves. Run `mpi.setup.rngstream` on the master to set up a parallel RNG. Though `mpi.parSim` works without a parallel RNG, the quality of simulation is not guarantied.

`mpi.parSim` will automatically transfer `rand.gen` and `statistic` to slaves. However, any functions that `rand.gen` and `statistic` reply on but are not on slaves must be transfered to slaves before using `mpi.parSim`. You can use `mpi.bcast.Robj2slave` for that purpose. The same is applied to required packages or C/Fortran codes. You can use either `mpi.bcast.cmd` or put `required(package)` and/or `dyn.load(so.lib)` into `rand.gen` and `statistic`.

If `simplify` is TRUE, sapply style simplication is applied. Otherwise a list of length slave.num*run is returned.

Author(s)

Hao Yu

See Also

`mpi.setup.rngstream` `mpi.bcast.cmd` `mpi.bcast.Robj2slave`

`mpi.probe` *MPI_Probe and MPI_Iprobe APIs*

Description

`mpi.probe` uses the source and tag of incoming message to set a status. `mpi.iprobe` does the same except it is a nonblocking call, i.e., returns immediately.

Usage

```
mpi.probe(source, tag, comm = 1, status = 0)
mpi.iprobe(source, tag, comm = 1, status = 0)
```

Arguments

`source`	the source of incoming message or mpi.any.source() for any source.
`tag`	a tag number or mpi.any.tag() for any tag.
`comm`	a communicator number
`status`	a status number

Details

When `mpi.send` or other nonblocking sends are used to send a message, the receiver may not know the exact length before receiving it. `mpi.probe` is used to probe the incoming message and put some information into a status. Then the exact length can be found by using `mpi.get.count` to such a status. If the wild card `mpi.any.source` or `mpi.any.tag` are used, then one can use `mpi.get.sourcetag` to find the exact source or tag of a sender.

Value

`mpi.probe` returns 1 only after a matching message has been found.

`mpi.iproble` returns TRUE if there is a message that can be received; FALSE otherwise.

Author(s)

Hao Yu

References

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

See Also

`mpi.send`, `mpi.recv`, `mpi.get.count`

`mpi.realloc` *Find and increase the lengths of MPI opaques comm, request, and status*

Description

`mpi.comm.maxsize`, `mpi.request.maxsize`, and `mpi.status.maxsize` find the lengths of comm, request, and status arrays respectively.

`mpi.realloc.comm`, `mpi.realloc.request` and `mpi.realloc.status` increase the lengths of comm, request and status arrays to `newmaxsize` respectively if `newmaxsize` is bigger than the original maximum size.

Usage

```
mpi.realloc.comm(newmaxsize)
mpi.realloc.request(newmaxsize)
mpi.realloc.status(newmaxsize)
mpi.comm.maxsize()
mpi.request.maxsize()
mpi.status.maxsize()
```

Arguments

`newmaxsize`	an integer.

Details

When **Rmpi** is loaded, Rmpi allocs comm array with size 10, request array with 10,000 and status array with 5,000. They should be enough in most cases. They use less than 150KB system memory. In rare case, one can use `mpi.realloc.comm`, `mpi.realloc.request` and `mpi.realloc.status` to increase them to bigger arrays.

Value

- `mpi.realloc.comm`, `mpi.realloc.request`, `mpi.realloc.status`: no return value (called for side effects).
- `mpi.comm.maxsize`, `mpi.request.maxsize`, `mpi.status.maxsize`: integer size limits.

Author(s)

Hao Yu

References

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

`mpi.reduce` *MPI_Reduce and MPI_Allreduce APIs*

Description

`mpi.reduce` and `mpi.allreduce` are global reduction operations. `mpi.reduce` combines each member's result, using the operation op, and returns the combined value(s) to the member specified by the argument `dest`. `mpi.allreduce` is the same as `mpi.reduce` except that all members receive the combined value(s).

Usage

```
mpi.reduce(x, type=2, op=c("sum","prod","max","min","maxloc","minloc"),
dest = 0, comm = 1)

mpi.allreduce(x, type=2, op=c("sum","prod","max","min","maxloc","minloc"),
comm = 1)
```

Arguments

`x`	data to be reduced. Must be the same dim and the same type for all members.
`type`	1 for integer and 2 for double. Others are not supported.
`op`	one of "sum", "prod", "max", "min", "maxloc", or "minloc".
`dest`	rank of destination
`comm`	a communicator number

Details

It is important that all members in a comm call either all `mpi.reduce` or all `mpi.allreduce` even though the master may not be in computation. They must provide exactly the same type and dim vectors to be reduced. If the operation "maxloc" or "minloc" is used, the combined vector is twice as long as the original one since the maximum or minimum ranks are included.

Value

`mpi.reduce` returns the combined value(s) to the member specified by `dest`. `mpi.allreduce` returns the combined values(s) to every member in a comm. The combined value(s) may be the summation, production, maximum, or minimum specified by the argument `op`. If the `op` is either "maxloc" or "minloc", then the maximum (minimum) value(s) along the maximum (minimum) rank(s) will be returned.

Author(s)

Hao Yu

References

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

See Also

`mpi.gather`.

`mpi.remote.exec` *Remote Executions on R slaves*

Description

Remotely execute a command on R slaves spawned by using slavedaemon.R script and return all executed results back to master.

Usage

```
mpi.remote.exec(cmd, ..., simplify = TRUE, comm = 1, ret = TRUE)
```

Arguments

`cmd`	the command to be executed on R slaves
`...`	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.remote.exec(cmd=myfun, x).
`simplify`	logical; should the result be simplified to a data.frame if possible?
`comm`	a communicator number.
`ret`	return executed results from R slaves if TRUE.

Details

Once R slaves are spawned by `mpi.spawn.Rslaves` with the slavedaemon.R script, they are waiting for instructions from master. One can use `mpi.bcast.cmd` to send a command to R slaves. However it will not return executed results. Hence `mpi.remote.exec` can be considered an extension to `mpi.bcast.cmd`.

Value

return executed results from R slaves if the argument `ret` is set to be TRUE. The value could be a data.frame if values (integer or double) from each slave have the same dimension. Otherwise a list is returned.

Warning

`mpi.remote.exec` may have difficulty guessing invisible results on R slaves. Use `ret = FALSE` instead.

Author(s)

Hao Yu

See Also

`mpi.spawn.Rslaves`, `mpi.bcast.cmd`

Examples

```
## Not run:
# Not run in checks: requires pre-spawned slaves and a live worker communicator.
mpi.remote.exec(mpi.comm.rank())
 x=5
mpi.remote.exec(rnorm,x)

## End(Not run)
```

`mpi.scatter` *MPI_Scatter and MPI_Scatterv APIs*

Description

`mpi.scatter` and `mpi.scatterv` are the inverse operations of `mpi.gather` and `mpi.gatherv` respectively.

Usage

```
mpi.scatter(x, type, rdata, root = 0,  comm = 1)
mpi.scatterv(x, scounts, type, rdata, root = 0, comm = 1)
```

Arguments

`x`	data to be scattered.
`type`	1 for integer, 2 for double, and 3 for character. Others are not supported.
`rdata`	the receive buffer. Must be the same type as the sender
`scounts`	int vector specifying the block length inside a message to be scattered to other members.
`root`	rank of the receiver
`comm`	a communicator number

Details

`mpi.scatter` scatters the message x to all members. Each member receives a portion of x with dim as length(x)/size in rank order, where size is the total number of members in a comm. So the receive buffer can be prepared as either integer(length(x)/size) or double(length(x)/size). For `mpi.scatterv`, scounts counts the portions (different dims) of x sent to each member. Each member needs to prepare the receive buffer as either integer(scounts[i]) or double(scounts[i]).

Value

For non-root members, `mpi.scatter` or `scatterv` returns the scattered message and ignores whatever is in x (or scounts). For the root member, it returns the portion belonging to itself.

Author(s)

Hao Yu

References

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

See Also

mpi.gather, mpi.gatherv.

Examples

```
## Not run:
# Not run in checks: requires a fixed number of spawned slaves and rank-specific buffers.
# Running this with a different communicator layout can deadlock.
#Need 3 slaves to run properly
#Or run  mpi.spawn.Rslaves(nslaves=3)
  num="123456789abcd"
  scounts<-c(2,3,1,7)
  mpi.bcast.cmd(strnum<-mpi.scatter(integer(1),type=1,rdata=integer(1),root=0))
  strnum<-mpi.scatter(scounts,type=1,rdata=integer(1),root=0)
  mpi.bcast.cmd(ans <- mpi.scatterv(string(1),scounts=0,type=3,rdata=string(strnum),
root=0))
  mpi.scatterv(as.character(num),scounts=scounts,type=3,rdata=string(strnum),root=0)
  mpi.remote.exec(ans)

## End(Not run)
```

---

mpi.scatter.Robj *Extensions of MPI_ SCATTER and MPI_SCATTERV*

Description

mpi.scatter.Robj and mpi.scatter.Robj2slave are used to scatter a list to all members. They are more efficient than using any parallel apply functions.

Usage

```
mpi.scatter.Robj(obj = NULL, root = 0, comm = 1)
mpi.scatter.Robj2slave(obj, comm = 1)
```

Arguments

obj	a list object to be scattered from the root or master
root	rank of the scatter.
comm	a communicator number.

Details

mpi.scatter.Robj is an extension of mpi.scatter for scattering a list object from a sender (root) to everyone. mpi.scatter.Robj2slave scatters a list to all slaves.

Value

mpi.scatter.Robj for non-root members, returns the scattered R object. For the root member, it returns the portion belonging to itself. mpi.scatter.Robj2slave returns no value for the master and all slaves get their corresponding components in the list, i.e., the first slave gets the first component in the list.

Author(s)

Hao Yu and Wei Xia

See Also

mpi.scatter, mpi.gather.Robj,

Examples

```
## Not run:
# Not run in checks: requires pre-spawned slaves and a live worker communicator.
#assume that there are three slaves running
mpi.bcast.cmd(x<-mpi.scatter.Robj())

xx <- list("master",rnorm(3),letters[2],1:10)
mpi.scatter.Robj(obj=xx)

mpi.remote.exec(x)

#scatter a matrix to slaves
dat=matrix(1:24,ncol=3)
splitmatrix = function(x, ncl) lapply(.splitIndices(nrow(x), ncl), function(i) x[i,])
dat2=splitmatrix(dat,3)
mpi.scatter.Robj2slave(dat2)
mpi.remote.exec(dat2)

## End(Not run)
```

---

mpi.send *MPI_Send, MPI_Isend, MPI_Recv, and MPI_Irecv APIs*

Description

The pair mpi.send and mpi.recv are two most used blocking calls for point-to-point communications. An int, double or char vector can be transmitted from any source to any destination.

The pair mpi.isend and mpi.irecv are the same except that they are nonblocking calls.

Blocking and nonblocking calls are interchangeable, e.g., nonblocking sends can be matched with blocking receives, and vice-versa.

Usage

```
mpi.send(x, type, dest, tag,  comm = 1)
mpi.isend(x, type, dest, tag,  comm = 1, request=0)
mpi.recv(x, type, source, tag,  comm = 1, status = 0)
mpi.irecv(x, type, source, tag,  comm = 1, request = 0)
```

Arguments

`x`	data to be sent or received. Must be the same type for source and destination. The receive buffer must be as large as the send buffer.
`type`	1 for integer, 2 for double, and 3 for character. Others are not supported.
`dest`	the destination rank. Use `mpi.proc.null` for a fake destination.
`source`	the source rank. Use `mpi.any.source` for any source. Use `mpi.proc.null` for a fake source.
`tag`	non-negative integer. Use `mpi.any.tag` for any tag flag.
`comm`	a communicator number.
`request`	a request number.
`status`	a status number.

Details

The pair `mpi.send` (or `mpi.isend`) and `mpi.recv` (or `mpi.irecv`) must be used together, i.e., if there is a sender, then there must be a receiver. Any mismatch will result a deadlock situation, i.e., programs stop responding. The receive buffer must be large enough to contain an incoming message otherwise programs will be crashed. One can use `mpi.probe` (or mpi.iprobe) and `mpi.get.count` to find the length of an incoming message before calling `mpi.recv`. If `mpi.any.source` or `mpi.any.tag` is used in `mpi.recv`, one can use `mpi.get.sourcetag` to find out the source or tag of the received message. To send/receive an R object rather than an int, double or char vector, please use the pair `mpi.send.Robj` and `mpi.recv.Robj`.

Since `mpi.irecv` is a nonblocking call, `x` with enough buffer must be created before using it. Then use nonblocking completion calls such as `mpi.wait` or `mpi.test` to test if `x` contains data from sender.

If multiple nonblocking sends or receives are used, please use request number consecutively from 0. For example, to receive two messages from two slaves, try mpi.irecv(x,1,source=1,tag=0,comm=1,request=0) mpi.irecv(y,1,source=2,tag=0,comm=1,request=1) Then `mpi.waitany`, `mpi.waitsome` or `mpi.waitall` can be used to complete the operations.

Value

`mpi.send` and `mpi.isend` return no value. `mpi.recv` returns the int, double or char vector sent from `source`. However, `mpi.irecv` returns no value. See details for explanation.

Author(s)

Hao Yu

References

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

See Also

mpi.send.Robj, mpi.recv.Robj, mpi.probe, mpi.wait, mpi.get.count, mpi.get.sourcetag.

Examples

```
## Not run:
# Not run in checks: send/recv calls must be paired across ranks.
# Running one side without a matching peer can deadlock.
#on a slave
mpi.send(1:10,1,0,0)

#on master
x <- integer(10)
mpi.irecv(x,1,1,0)
x
mpi.wait()
x

## End(Not run)
```

---

mpi.send.Robj *Extensions of MPI_Send and MPI_Recv APIs*

Description

mpi.send.Robj and mpi.recv.Robj are two extensions of mpi.send and mpi.recv. They are used to transmit a general R object from any source to any destination.

mpi.isend.Robj is a nonblocking version of mpi.send.Robj.

Usage

```
mpi.send.Robj(obj, dest, tag, comm = 1)
mpi.isend.Robj(obj, dest, tag, comm = 1, request=0)
mpi.recv.Robj(source, tag, comm = 1, status = 0)
```

Arguments

obj	an R object. Can be any R object.
dest	the destination rank.
source	the source rank or mpi.any.source() for any source.
tag	non-negative integer or mpi.any.tag() for any tag.
comm	a communicator number.
request	a request number.
status	a status number.

Details

`mpi.send.Robj` and `mpi.isend.Robj` use `serialize` to encode an R object into a binary char vector. It sends the message to the destination. The receiver decode the message back into an R object by using `unserialize`.

If `mpi.isend.Robj` is used, `mpi.wait` or `mpi.test` must be used to check the object has been sent.

Value

`mpi.send.Robj` or `mpi.isend.Robj` return no value. `mpi.recv.Robj` returns the the transmitted R object.

Author(s)

Hao Yu

References

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

See Also

`mpi.send`, `mpi.recv`, `mpi.wait`, `serialize`, `unserialize`,

`mpi.sendrecv` *MPI_Sendrecv and MPI_Sendrecv_replace APIs*

Description

`mpi.sendrecv` and `mpi.sendrecv.replace` execute blocking send and receive operations. Both of them combine the sending of one message to a destination and the receiving of another message from a source in one call. The source and destination are possibly the same. The send buffer and receive buffer are disjoint for `mpi.sendrecv`, while the buffers are not disjoint for `mpi.sendrecv.replace`.

Usage

```
mpi.sendrecv(senddata, sendtype, dest, sendtag, recvdata, recvtype,
source, recvtag, comm = 1, status = 0)

mpi.sendrecv.replace(x, type, dest, sendtag, source, recvtag,
comm = 1, status = 0)
```

Arguments

`x`	data to be sent or recieved. Must be the same type for source and destination.
`senddata`	data to be sent. May have different datatypes and lengths
`recvdata`	data to be recieved. May have different datatypes and lengths
`type`	type of the data to be sent or recieved. 1 for integer, 2 for double, and 3 for character. Others are not supported.
`sendtype`	type of the data to be sent. 1 for integer, 2 for double, and 3 for character. Others are not supported.
`recvtype`	type of the data to be recieved. 1 for integer, 2 for double, and 3 for character. Others are not supported.
`dest`	the destination rank. Use `mpi.proc.null` for a fake destination.
`source`	the source rank. Use `mpi.any.source` for any source. Use `mpi.proc.null` for a fake source.
`sendtag`	non-negative integer. Use `mpi.any.tag` for any tag flag.
`recvtag`	non-negative integer. Use `mpi.any.tag` for any tag flag.
`comm`	a communicator number.
`status`	a status number.

Details

The receive buffer must be large enough to contain an incoming message otherwise programs will be crashed. There is compatibility between send-receive and normal sends and receives. A message sent by a send-receive can be received by a regular receive and a send-receive can receive a message sent by a regular send.

Value

Returns the int, double or char vector sent from the send buffers.

Author(s)

Kris Chen

References

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

See Also

`mpi.send.Robj`, `mpi.recv.Robj`, `mpi.probe`. `mpi.get.sourcetag`.

Examples

```
## Not run:
## Not run in checks when toggled to dontrun: paired send/recv calls are
## documented for manual MPI sessions.
mpi.sendrecv(as.integer(11:20),1,0,33,integer(10),1,0,33,comm=0)
mpi.sendrecv.replace(seq(1,2,by=0.1),2,0,99,0,99,comm=0)

## End(Not run)
```

`mpi.setup.rngstream` *Setup parallel RNG on all slaves*

Description

`mpi.setup.rngstream` setups RNGstream on all slaves.

Usage

```
mpi.setup.rngstream(iseed=NULL, comm = 1)
```

Arguments

`iseed`	An integer to be supplied to `set.seed`, or NULL not to set reproducible seeds.
`comm`	A comm number.

Details

`mpi.setup.rngstream` can be run only on master node. It can be run later on with the same or different iseed.

Value

No value returned.

Author(s)

Hao Yu

mpi.spawn.Rslaves *Spawn and Close R Slaves*

Description

mpi.spawn.Rslaves spawns R slaves to those hosts automatically chosen by MPI or specific hosts assigned by the argument hosts. Those R slaves are running in R BATCH mode with a specific Rscript file. The default Rscript file "slavedaemon.R" provides interactive R slave environments.

mpi.close.Rslaves shuts down R slaves spawned by mpi.spawn.Rslaves.

tailslave.log view (from tail) R slave log files (assuming they are all in one working directory).

Usage

```
mpi.spawn.Rslaves(Rscript=system.file("slavedaemon.R", package="npRmpi"),
                  nslaves=mpi.universe.size(),
                  root = 0,
                  intercomm = 2,
                  comm = 1,
                  hosts = NULL,
                  needlog = FALSE,
                  mapdrive=TRUE,
                  quiet = FALSE,
                  nonblock=TRUE,
                  sleep=0.1)

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

Arguments

Rscript	an R script file used to run R in BATCH mode.
nslaves	number of slaves to be spawned.
root	the rank number of the member who spawns R slaves.
intercomm	an intercommunicator number
comm	a communicator number merged from an intercomm.
hosts	NULL or LAM node numbers to specify where R slaves are to be spawned.
needlog	a logical. If TRUE, R BATCH outputs will be saved in log files. If FALSE, the outputs will send to /dev/null.
mapdrive	a logical. If TRUE and master's working dir is on a network, mapping network drive is attemped on remote nodes under windows platform.
quiet	a logical. If TRUE, do not print anything unless an error occurs.
nonblock	a logical. If TRUE, a nonblock procedure is used on all slaves 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.
`dellog`	a logical specifying if R slave's log files are deleted or not.
`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 (i.e., keeps daemons alive for reuse).
`nlines`	number of lines to view from tail in R slave's log files.

Details

The R slaves that `mpi.spawn.Rslaves` spawns are really running a shell program which can be found in `system.file("Rslaves.sh",package="npRmpi")` which takes a Rscript file as one of its arguments. Other arguments are used to see if a log file (R output) is needed and how to name it. The master process id and the comm number, along with host names where R slaves are running are used to name these log files.

Once R slaves are successfully spawned, the mergers from an intercomm (default 'intercomm = 2') to a comm (default 'comm = 1') are automatically done on master and slaves (should be done if the default Rscript is replaced). If additional sets of R slaves are needed, please use 'comm = 3', 'comm = 4', etc to spawn them. At most a comm number up to 10 can be used. Notice that the default comm number for R slaves (using slavedaemon.R) is always 1 which is saved as .comm.

On some systems (notably macOS+MPICH), repeatedly spawning and tearing down slaves in the same R session can lead to hangs/crashes. To avoid this, npRmpi may reuse an existing slave pool when `options(npRmpi.reuse.slaves=TRUE)`. In this mode, `mpi.spawn.Rslaves()` becomes idempotent and `mpi.close.Rslaves(force=FALSE)` performs a soft-close.

To spawn R slaves to specific hosts, please use the argument `hosts` with a list of those node numbers (an integer vector). Total node numbers along their host names can be found by using `mpi.hostinfo`. Notice that this is MPI implementation specific.

Value

Unless `quiet` = `TRUE`, `mpi.spawn.Rslaves` prints to stdio how many slaves are successfully spawned and where they are running.

`mpi.close.Rslaves` returns a status code. When `options(npRmpi.reuse.slaves=TRUE)` and `force=FALSE`, this may be a no-op (soft-close) so that spawned daemons can be reused within the same R session.

`tailslave.log` returns last lines of R slave's log files.

Author(s)

Hao Yu

See Also

`mpi.comm.spawn`, `mpi.hostinfo`.

Examples

```
## Not run:
# Not run in checks: spawning/tearing down MPI daemons is environment-dependent
# and can interfere with later examples in the same session.
mpi.spawn.Rslaves(nslaves=2)
tailslave.log()
mpi.remote.exec(rnorm(10))
mpi.close.Rslaves()

## End(Not run)
```

`mpi.universe.size` *MPI_Universe_size API*

Description

`mpi.universe.size` returns the total number of CPUs available in a cluster. Some MPI implements may not have this MPI call available.

Usage

```
mpi.universe.size()
```

Arguments

None.

Value

An integer giving the total number of CPUs available in the MPI universe for the current configuration.

Author(s)

Hao Yu

References

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

mpi.wait *Nonblocking completion operations*

Description

mpi.cancel cancels a nonblocking send or receive request.

mpi.test.cancelled tests if mpi.cancel cancels or not.

wait, waitall, waitany, and waitsome are used to complete nonblocking send or receive requests. They are not local.

test, testall, testany, and testsome are used to complete nonblocking send and receive requests. They are local.

Usage

```
mpi.cancel(request)
mpi.test.cancelled(status=0)
mpi.test(request, status=0)
mpi.testall(count)
mpi.testany(count, status=0)
mpi.testsome(count)
mpi.wait(request, status=0)
mpi.waitall(count)
mpi.waitany(count, status=0)
mpi.waitsome(count)
```

Arguments

count	total number of nonblocking operations.
request	a request number.
status	a status number.

Details

mpi.wait and mpi.test are used to complete a nonblocking send and receive request: use the same request number by mpi.isend or mpi.irecv. Once completed, the associated request is set to MPI_REQUEST_NULL and status contains information such as source, tag, and length of message.

If multiple nonblocking sends or receives are initiated, the following calls are more efficient. Make sure that request numbers are used consecutively as request=0, request=1, request=2, etc. In this way, the following calls can find request information in system memory.

mpi.waitany and mpi.testany are used to complete one out of several requests.

mpi.waitall and mpi.testall are used to complete all requests.

mpi.waitsome and mpi.testsome are used to complete all enabled requests.

Value

`mpi.cancel` returns no value.

`mpi.test.cancelled` returns TRUE if a nonblocking call is cancelled; FALSE otherwise.

`mpi.wait` returns no value. Instead status contains information that can be retrieved by `mpi.get.count` and `mpi.get.sourcetag`.

`mpi.test` returns TRUE if a request is complete; FALSE otherwise. If TRUE, it is the same as `mpi.wait`.

`mpi.waitany` returns which request (index) has been completed. In addition, status contains information that can be retrieved by `mpi.get.count` and `mpi.get.sourcetag`.

`mpi.testany` returns a list: index— request index; flag—TRUE if a request is complete; FALSE otherwise (index is no use in this case). If flag is TRUE, it is the same as `mpi.waitany`.

`mpi.waitall` returns no value. Instead statuses 0, 1, ..., count-1 contain corresponding information that can be retrieved by `mpi.get.count` and `mpi.get.sourcetag`.

`mpi.testall` returns TRUE if all requests are complete; FALSE otherwise. If TRUE, it is the same as `mpi.waitall`.

`mpi.waitsome` returns a list: count— number of requests that have been completed; indices—an integer vector of size `count` of those completed request numbers (in 0, 1 ,..., count-1). In addition, statuses 0, 1, ..., `count`-1 contain corresponding information that can be retrieved by `mpi.get.count` and `mpi.get.sourcetag`.

`mpi.testsome` is the same as `mpi.waitsome` except that `count` may be 0 and in this case `indices` is no use.

Author(s)

Hao Yu

References

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

See Also

`mpi.isend`, `mpi.irecv`, `mpi.get.count`, `mpi.get.sourcetag`.

`np.mpi.initialize` *Initialize Master and Slave Nodes for the np Package*

Description

`np.mpi.initialize` is used to initialize master and slave nodes.

Usage

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

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`>

`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

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

Details

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

`npudens`, `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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

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

mpi.bcast.Robj2slave(USArrests)
mpi.bcast.Robj2slave(y_vars)

mpi.bcast.cmd(pair_list <- np.pairs(y_vars = y_vars, y_dat = USArrests,
                                    ckertype = "epanechnikov",
                                    bwscaling = TRUE),
              caller.execute=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.stop()` 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.stop()               ## soft close (may keep slaves alive)
## npRmpi.stop(force=TRUE)  ## hard close

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)
## 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 'call'
npcdens(bws, ...)

## 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,
        ...)

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

Arguments

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.

`gradients`	a logical value specifying whether to return estimates of the gradients at the evaluation points. Defaults to `FALSE`.
`...`	additional arguments supplied to specify the bandwidth type, kernel types, and so on. This is necessary if you specify bws as a p+q-vector and not a `conbandwidth` object, and you do not desire the default behaviours. To do this, you may specify any of `bwmethod`, `bwscaling`, `bwtype`, `cxkertype`, `cxkerorder`, `cykertype`, `cykerorder`, `uxkertype`, `uykertype`, `oxkertype`, `oykertype`, as described in `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.
`newdata`	An optional data frame in which to look for evaluation data. If omitted, the training data are used.
`txdat`	a p-variate data frame of sample realizations of explanatory data (training data). Defaults to the training data used to compute the bandwidth object.
`tydat`	a q-variate data frame of sample realizations of dependent data (training data). Defaults to the training data used to compute the bandwidth object.
`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`.

Details

`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.

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
`congerr`	if invoked with `gradients = TRUE`, standard errors of the gradients at the evaluation points
`log_likelihood`	log likelihood of the conditional density estimate

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.

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

`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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

mpi.bcast.cmd(data("Italy"),
              caller.execute=TRUE)
mpi.bcast.cmd(attach(Italy),
              caller.execute=TRUE)

mpi.bcast.cmd(bw <- npcdensbw(formula=gdp~ordered(year)),
              caller.execute=TRUE)

mpi.bcast.cmd(fhat <- npcdens(bws=bw),
              caller.execute=TRUE)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)
## 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 'NULL'
npcdensbw(xdat = stop("data 'xdat' missing"),
          ydat = stop("data 'ydat' missing"),
          bws, ...)

## S3 method for class 'conbandwidth'
npcdensbw(xdat = stop("data 'xdat' missing"),
          ydat = stop("data 'ydat' missing"),
          bws,
          bandwidth.compute = TRUE,
          nmulti,
          remin = TRUE,
          itmax = 10000,
          ftol = 1.490116e-07,
          tol = 1.490116e-04,
          small = 1.490116e-05,
          memfac = 500,
          lbc.dir = 0.5,
          dfc.dir = 3,
          cfac.dir = 2.5*(3.0-sqrt(5)),
          initc.dir = 1.0,
          lbd.dir = 0.1,
          hbd.dir = 1,
          dfac.dir = 0.25*(3.0-sqrt(5)),
          initd.dir = 1.0,
          lbc.init = 0.1,
          hbc.init = 2.0,
          cfac.init = 0.5,
          lbd.init = 0.1,
          hbd.init = 0.9,
          dfac.init = 0.375,
          scale.init.categorical.sample = FALSE,
          transform.bounds = FALSE,
          invalid.penalty = c("baseline","dbmax"),
          penalty.multiplier = 10,
          ...)

## Default S3 method:
npcdensbw(xdat = stop("data 'xdat' missing"),
          ydat = stop("data 'ydat' missing"),
          bws,
          bandwidth.compute = TRUE,
          nmulti,
          remin,
          itmax,
          ftol,
          tol,
          small,
          memfac,
          lbc.dir,
          dfc.dir,
          cfac.dir,
          initc.dir,
          lbd.dir,
          hbd.dir,
          dfac.dir,
          initd.dir,
          lbc.init,
          hbc.init,
          cfac.init,
          lbd.init,
          hbd.init,
          dfac.init,
          scale.init.categorical.sample,
          transform.bounds,
          invalid.penalty,
          penalty.multiplier,
          bwmethod,
          bwscaling,
          bwtype,
          cxkertype,
          cxkerorder,
          cykertype,
          cykerorder,
          uxkertype,
          uykertype,
          oxkertype,
          oykertype,
          ...)
```

Arguments

`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.
`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.
`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.
`...`	additional arguments supplied to specify the bandwidth type, kernel types, selection methods, and so on, detailed below.
`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
`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`.
`cxkertype`	character string used to specify the continuous kernel type for `xdat`. Can be set as `gaussian`, `epanechnikov`, or `uniform`. Defaults to `gaussian`.
`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`.
`cykertype`	character string used to specify the continuous kernel type for `ydat`. 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`.
`uxkertype`	character string used to specify the unordered categorical kernel type. Can be set as `aitchisonaitken` or `liracine`. Defaults to `aitchisonaitken`.
`uykertype`	character string used to specify the unordered categorical kernel type. Can be set as `aitchisonaitken` or `liracine`.
`oxkertype`	character string used to specify the ordered categorical kernel type. Can be set as `wangvanryzin` or `liracine`. Defaults to `liracine`.
`oykertype`	character string used to specify the ordered categorical kernel type. Can be set as `wangvanryzin` or `liracine`.
`nmulti`	integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points
`remin`	a logical value which when set as `TRUE` the search routine restarts from located minima for a minor gain in accuracy. Defaults to `TRUE`.
`itmax`	integer number of iterations before failure in the numerical optimization routine. Defaults to `10000`.
`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)).

`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))`.

`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))`.

`lbc.dir`, `dfc.dir`, `cfac.dir`, `initc.dir`
: lower bound, chi-square degrees of freedom, stretch factor, and initial non-random values for direction set search for Powell's algorithm for `numeric` variables. See Details

`lbd.dir`, `hbd.dir`, `dfac.dir`, `initd.dir`
: lower bound, upper bound, stretch factor, and initial non-random values for direction set search for Powell's algorithm for categorical variables. See Details

`lbc.init`, `hbc.init`, `cfac.init`
: lower bound, upper bound, and non-random initial values for scale factors for `numeric` variables for Powell's algorithm. See Details

`lbd.init`, `hbd.init`, `dfac.init`
: lower bound, upper bound, and non-random initial values 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

`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`.

`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"`.

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

`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.

Details

`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.

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`.

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(5, 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.

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

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

mpi.bcast.cmd(data("Italy"),
              caller.execute=TRUE)
mpi.bcast.cmd(attach(Italy),
              caller.execute=TRUE)

mpi.bcast.cmd(bw <- npcdensbw(formula=gdp~ordered(year)),
              caller.execute=TRUE)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)
## 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 'call'
npcdist(bws, ...)

## 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,
        ...)

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

Arguments

`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.

`gradients`	a logical value specifying whether to return estimates of the gradients at the evaluation points. Defaults to `FALSE`.
`...`	additional arguments supplied to specify the bandwidth type, kernel types, and so on. This is necessary if you specify bws as a $p+q$-vector and not a `condbandwidth` object, and you do not desire the default behaviours. To do this, you may specify any of `bwmethod`, `bwscaling`, `bwtype`, `cxkertype`, `cxkerorder`, `cykertype`, `cykerorder`, `uxkertype`, `oxkertype`, `oykertype`, as described in `npcdistbw`.
`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.
`newdata`	An optional data frame in which to look for evaluation data. If omitted, the training data are used.
`txdat`	a p-variate data frame of sample realizations of explanatory data (training data). Defaults to the training data used to compute the bandwidth object.
`tydat`	a q-variate data frame of sample realizations of dependent data (training data). Defaults to the training data used to compute the bandwidth object.
`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`.

Details

`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.

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
congerr	if invoked with gradients = TRUE, standard errors of the gradients at the evaluation points
log_likelihood	log likelihood of the cumulative conditional distribution estimate

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.

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

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

data("Italy")

mpi.bcast.Robj2slave(Italy)

mpi.bcast.cmd(bw <- npcdistbw(formula=gdp~ordered(year),
                              data=Italy),
              caller.execute=TRUE)

mpi.bcast.cmd(F <- npcdist(bws=bw),
              caller.execute=TRUE)

summary(F)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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 'NULL'
npcdistbw(xdat = stop("data 'xdat' missing"),
          ydat = stop("data 'ydat' missing"),
          bws, ...)

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

## Default S3 method:
npcdistbw(xdat = stop("data 'xdat' missing"),
          ydat = stop("data 'ydat' missing"),
          gydat,
          bws,
          bandwidth.compute = TRUE,
          nmulti,
          remin,
          itmax,
          do.full.integral,
          ngrid,
          ftol,
          tol,
          small,
          memfac,
          lbc.dir,
          dfc.dir,
          cfac.dir,
          initc.dir,
          lbd.dir,
          hbd.dir,
          dfac.dir,
          initd.dir,
          lbc.init,
          hbc.init,
          cfac.init,
          lbd.init,
          hbd.init,
          dfac.init,
          scale.init.categorical.sample,
          transform.bounds,
          invalid.penalty,
          penalty.multiplier,
          bwmethod,
          bwscaling,
          bwtype,
          cxkertype,
          cxkerorder,
          cykertype,
          cykerorder,
          uxkertype,
          oxkertype,
          oykertype,
          ...)
```

Arguments

`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.
`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).
`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.
`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`).

`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.
`...`	additional arguments supplied to specify the bandwidth type, kernel types, selection methods, and so on, detailed below.
`bwmethod`	which method to use to select bandwidths. `cv.ls` specifies least-squares cross-validation (Li, Lin and Racine (2013), 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.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
`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`.
`cxkertype`	character string used to specify the continuous kernel type for `xdat`. Can be set as `gaussian`, `epanechnikov`, or `uniform`. Defaults to `gaussian`.
`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`.
`cykertype`	character string used to specify the continuous kernel type for `ydat`. 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`.
`uxkertype`	character string used to specify the unordered categorical kernel type. Can be set as `aitchisonaitken` or `liracine`. Defaults to `aitchisonaitken`.
`oxkertype`	character string used to specify the ordered categorical kernel type. Can be set as `wangvanryzin` or `liracine`. Defaults to `liracine`.
`oykertype`	character string used to specify the ordered categorical kernel type. Can be set as `wangvanryzin` or `liracine`.
`nmulti`	integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points
`remin`	a logical value which when set as `TRUE` the search routine restarts from located minima for a minor gain in accuracy. Defaults to `TRUE`.
`itmax`	integer number of iterations before failure in the numerical optimization routine. Defaults to `10000`.
`do.full.integral`	a logical value which when set as `TRUE` evaluates the moment-based integral on the entire sample.
`ngrid`	integer number of grid points to use when computing the moment-based integral. Defaults to `100`.
`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$double.eps)).
`tol`	tolerance on the position of located minima of the cross-validation function (tol should generally be no smaller than the square root of your machine's floating point precision). Defaults to `1.490116e-04 (1.0e+04*sqrt(.Machine$double.eps))`.
`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))`.
`lbc.dir`, `dfc.dir`, `cfac.dir`, `initc.dir`	lower bound, chi-square degrees of freedom, stretch factor, and initial non-random values for direction set search for Powell's algorithm for `numeric` variables. See Details
`lbd.dir`, `hbd.dir`, `dfac.dir`, `initd.dir`	lower bound, upper bound, stretch factor, and initial non-random values for direction set search for Powell's algorithm for categorical variables. See Details
`lbc.init`, `hbc.init`, `cfac.init`	lower bound, upper bound, and non-random initial values for scale factors for `numeric` variables for Powell's algorithm. See Details
`lbd.init`, `hbd.init`, `dfac.init`	lower bound, upper bound, and non-random initial values 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

`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`.

`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"`.

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

`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.

Details

`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 (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.

`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.

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`.

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(5, 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.

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

`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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

data("Italy")

mpi.bcast.Robj2slave(Italy)

mpi.bcast.cmd(bw <- npcdistbw(formula=gdp~ordered(year),
                              data=Italy),
              caller.execute=TRUE)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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

`formula`	a symbolic description of variables on which the test 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.
`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).
`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`.
`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`.
`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`.
`pivot`	a logical value specifying whether the statistic should be normalised such that it approaches $N(0, 1)$ in distribution. Defaults to `TRUE`.
`density.weighted`	a logical value specifying whether the statistic should be weighted by the density of `xdat`. 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 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.

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`.

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.

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

mpi.bcast.cmd(data(cps71),
              caller.execute=TRUE)

mpi.bcast.cmd(attach(cps71),
              caller.execute=TRUE)

mpi.bcast.cmd(model <- lm(logwage~age+I(age^2), x=TRUE, y=TRUE),
              caller.execute=TRUE)

mpi.bcast.cmd(npcmstest(model = model, xdat = age, ydat = logwage,
                        boot.num = 29),
              caller.execute=TRUE)

## 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.stop()` 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.stop()               ## soft close (may keep slaves alive)
## npRmpi.stop(force=TRUE)  ## hard close
## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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, ...)

## S3 method for class 'call'
npconmode(bws, ...)

## Default S3 method:
npconmode(bws, txdat, tydat, ...)

## S3 method for class 'conbandwidth'
npconmode(bws,
          txdat = stop("invoked without training data 'txdat'"),
          tydat = stop("invoked without training data 'tydat'"),
          exdat,
          eydat,
          ...)
```

Arguments

bws	a bandwidth specification. This can be set as a conbandwidth object returned from an invocation of npcdensbw
...	additional arguments supplied to specify the bandwidth type, kernel types, and so on, detailed below. This is necessary if you specify bws as a $p+q$-vector and not a conbandwidth object, and you do not desire the default behaviours.

`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.
`newdata`	An optional data frame in which to look for evaluation data. If omitted, the training data are used.
`txdat`	a p-variate data frame of explanatory data (conditioning data) used to calculate the regression estimators. Defaults to the training data used to compute the bandwidth object.
`tydat`	a one (1) dimensional vector of unordered or ordered factors, containing the dependent data. Defaults to the training data used to compute the bandwidth object.
`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`.

Value

`npconmode` returns a `conmode` object with the following components:

`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
`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

The functions `mode`, and `fitted` may be used to extract the conditional mode estimates, and the conditional density estimates at the conditional mode, respectively, from the resulting object. Also, `summary` supports `conmode` objects.

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.

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.

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

mpi.bcast.cmd(library(MASS),caller.execute=TRUE)
mpi.bcast.cmd(data(birthwt),caller.execute=TRUE)

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)
```

```
mpi.bcast.Robj2slave(birthwt)

mpi.bcast.cmd(bw <- npcdensbw(low~
                              smoke+
                              race+
                              ht+
                              ui+
                              ftv+
                              age+
                              lwt,
                              data=birthwt),
              caller.execute=TRUE)

summary(bw)

mpi.bcast.cmd(model <- npconmode(bws=bw),
              caller.execute=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.stop()` 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.stop()               ## soft close (may keep slaves alive)
## npRmpi.stop(force=TRUE)  ## hard close

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## End(Not run)
```

npcopula *Kernel Copula Estimation with Mixed Data Types*

Description

`npcopula` implements the nonparametric mixed data kernel copula approach of Racine (2015) for an arbitrary number of dimensions

Usage

```
npcopula(bws,
         data,
         u = NULL,
         n.quasi.inv = 1000,
         er.quasi.inv = 1)
```

Arguments

`bws`	an unconditional joint distribution (`npudistbw`) or joint density (`npudensbw`) bandwidth object (if `bws` is delivered via `npudistbw` the copula distribution is estimated, while if `bws` is delivered via `npudensbw` the copula density is estimated)
`data`	a data frame containing variables used to construct `bws`
`u`	an optional matrix of real numbers lying in [0,1], each column of which corresponds to the vector of uth quantile values desired for each variable in the copula (otherwise the `u` values returned are those corresponding to the sample realizations)
`n.quasi.inv`	number of grid points generated when `u` is provided in order to compute the quasi-inverse of each marginal distribution (see details)
`er.quasi.inv`	number passed to `extendrange` when `u` is provided specifying the fraction by which the data range should be extended when constructing the grid used to compute the quasi-inverse (see details)

Details

`npcopula` computes the nonparametric copula or copula density using inversion (Nelsen (2006), page 51). For the inversion approach, we exploit Sklar's theorem (Corollary 2.3.7, Nelsen (2006)) to produce copulas directly from the joint distribution function using $C(u,v) = H(F^{-1}(u), G^{-1}(v))$ rather than the typical approach that instead uses $H(x,y) = C(F(x), G(y))$. Whereas the latter requires kernel density estimation on a d-dimensional unit hypercube which necessitates the use of boundary correction methods, the former does not.

Note that if `u` is provided then `expand.grid` is called on `u`. As the dimension increases this can become unwieldy and potentially consume an enormous amount of memory unless the number of grid points is kept very small. Given that computing the copula on a grid is typically done for graphical purposes, providing `u` is typically done for two-dimensional problems only. Even here, however, providing a grid of length 100 will expand into a matrix of dimension 10000 by 2 which, though not memory intensive, may be computationally burdensome.

The 'quasi-inverse' is computed via Definition 2.3.6 from Nelsen (2006). We compute an equi-quantile grid on the range of the data of length `n.quasi.inv/2`. We then extend the range of the data by the factor `er.quasi.inv` and compute an equi-spaced grid of points of length `n.quasi.inv/2` (e.g. using the default `er.quasi.inv=1` we go from the minimum data value minus $1\times$ the range to the maximum data value plus $1\times$ the range for each marginal). We then take these two grids, concatenate and sort, and these form the final grid of length n.quasi.inv for computing the quasi-inverse.

Note that if u is provided and any elements of (the columns of) u are such that they lie beyond the respective values of F for the evaluation data for the respective marginal, such values are reset to the minimum/maximum values of F for the respective marginal. It is therefore prudent to inspect the values of u returned by npcopula when u is provided.

Note that copula are only defined for data of type numeric or ordered.

Value

npcopula returns an object of type data.frame with the following components

copula	the copula (bandwidth object obtained from npudistbw) or copula density (bandwidth object obtained from npudensbw)
u	the matrix of marginal u values associated with the sample realizations (u=NULL) or those created via expand.grid when u is provided
data	the matrix of marginal quantiles constructed when u is provided (data returned has the same names as data inputted)

Usage Issues

See the example below for proper usage.

Author(s)

Jeffrey S. Racine <racinej@mcmaster.ca>

References

Nelsen, R. B. (2006), *An Introduction to Copulas,* Second Edition, Springer-Verlag.

Racine, J.S. (2015), “Mixed Data Kernel Copulas,” Empirical Economics, 48, 37-59.

See Also

npudensbw,npudens,npudist

Examples

```
## Not run:
## Not run in checks: excluded to keep MPI examples stable and check times short.
## Example 1: Bivariate Mixed Data

## 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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

mpi.bcast.cmd(library(MASS),caller.execute=TRUE)

set.seed(42)

## Simulate correlated Gaussian data (rho(x,y)=0.99)

n <- 1000
n.eval <- 100
rho <- 0.99
mu <- c(0,0)
Sigma <- matrix(c(1,rho,rho,1),2,2)
mydat <- mvrnorm(n=n, mu, Sigma)
mydat <- data.frame(x=mydat[,1],
                    y=ordered(as.integer(cut(mydat[,2],
                      quantile(mydat[,2],seq(0,1,by=.1)),
                      include.lowest=TRUE))-1))
q.min <- 0.0
q.max <- 1.0
grid.seq <- seq(q.min,q.max,length=n.eval)
grid.dat <- cbind(grid.seq,grid.seq)

mpi.bcast.Robj2slave(mydat)
mpi.bcast.Robj2slave(grid.dat)

## Estimate the copula (bw object obtained from npudistbw())

mpi.bcast.cmd(bw.cdf <- npudistbw(~x+y,data=mydat),
              caller.execute=TRUE)
mpi.bcast.cmd(copula <- npcopula(bws=bw.cdf,data=mydat,u=grid.dat),
              caller.execute=TRUE)

## Plot the copula


contour(grid.seq,grid.seq,matrix(copula$copula,n.eval,n.eval),
        xlab="u1",
        ylab="u2",
        main="Copula Contour")

persp(grid.seq,grid.seq,matrix(copula$copula,n.eval,n.eval),
      ticktype="detailed",
      xlab="u1",
      ylab="u2",
      zlab="Copula",zlim=c(0,1))
```

```
## Plot the empirical copula

mpi.bcast.cmd(copula.emp <- npcopula(bws=bw.cdf,data=mydat),
              caller.execute=TRUE)
plot(copula.emp$u1,copula.emp$u2,xlab="u1",ylab="u2",cex=.25,main="Empirical Copula")

## Estimate the copula density (bw object obtained from npudensbw())

mpi.bcast.cmd(bw.pdf <- npudensbw(~x+y,data=mydat),
              caller.execute=TRUE)
mpi.bcast.cmd(copula <- npcopula(bws=bw.pdf,data=mydat,u=grid.dat),
              caller.execute=TRUE)

## Plot the copula density

persp(grid.seq,grid.seq,matrix(copula$copula,n.eval,n.eval),
      ticktype="detailed",
      xlab="u1",
      ylab="u2",
      zlab="Copula Density")

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## Example 2: Bivariate Continuous Data

## 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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

mpi.bcast.cmd(library(MASS),caller.execute=TRUE)

set.seed(42)

## Simulate correlated Gaussian data (rho(x,y)=0.99)

n <- 1000
n.eval <- 100
rho <- 0.99
mu <- c(0,0)
Sigma <- matrix(c(1,rho,rho,1),2,2)
mydat <- mvrnorm(n=n, mu, Sigma)
mydat <- data.frame(x=mydat[,1],y=mydat[,2])

q.min <- 0.0
q.max <- 1.0
grid.seq <- seq(q.min,q.max,length=n.eval)
grid.dat <- cbind(grid.seq,grid.seq)

mpi.bcast.Robj2slave(mydat)
mpi.bcast.Robj2slave(grid.dat)

## Estimate the copula (bw object obtained from npudistbw())

mpi.bcast.cmd(bw.cdf <- npudistbw(~x+y,data=mydat),
              caller.execute=TRUE)
mpi.bcast.cmd(copula <- npcopula(bws=bw.cdf,data=mydat,u=grid.dat),
              caller.execute=TRUE)

## Plot the copula

contour(grid.seq,grid.seq,matrix(copula$copula,n.eval,n.eval),
        xlab="u1",
        ylab="u2",
        main="Copula Contour")

persp(grid.seq,grid.seq,matrix(copula$copula,n.eval,n.eval),
      ticktype="detailed",
      xlab="u1",
      ylab="u2",
      zlab="Copula",
      zlim=c(0,1))

## Plot the empirical copula

mpi.bcast.cmd(copula.emp <- npcopula(bws=bw.cdf,data=mydat),
              caller.execute=TRUE)
```

```
plot(copula.emp$u1,copula.emp$u2,xlab="u1",ylab="u2",cex=.25,main="Empirical Copula")

## Estimate the copula density (bw object obtained from npudensbw())

mpi.bcast.cmd(bw.pdf <- npudensbw(~x+y,data=mydat),
              caller.execute=TRUE)
mpi.bcast.cmd(copula <- npcopula(bws=bw.pdf,data=mydat,u=grid.dat),
              caller.execute=TRUE)

## Plot the copula density

persp(grid.seq,grid.seq,matrix(copula$copula,n.eval,n.eval),ticktype="detailed",xlab="u1",
      ylab="u2",zlab="Copula Density")

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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

`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).
`bw.x, bw.y`	optional bandwidth objects for `x,y`
`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 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

`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`.

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

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

mpi.bcast.cmd(set.seed(42),
              caller.execute=TRUE)

n <- 100

sample.A <- data.frame(x=rnorm(n))
sample.B <- data.frame(x=rnorm(n))

mpi.bcast.Robj2slave(sample.A)
mpi.bcast.Robj2slave(sample.B)

mpi.bcast.cmd(output <- npdeneqtest(sample.A,sample.B,boot.num=29),
              caller.execute=TRUE)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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.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	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.
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.

Details

`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`.

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

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

mpi.bcast.cmd(set.seed(42),
              caller.execute=TRUE)

n <- 100

x <- rnorm(n)
y <- 1 + x + rnorm(n)
model <- lm(y~x)
y.fit <- fitted(model)

mpi.bcast.Robj2slave(y)
mpi.bcast.Robj2slave(y.fit)

mpi.bcast.cmd(output <- npdeptest(y,
                                  y.fit,
                                  boot.num=29,
                                  method="summation"),
              caller.execute=TRUE)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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,
        ...)

## S3 method for class 'call'
npindex(bws,
        ...)

## Default S3 method:
```

```
npindex(bws,
        txdat,
        tydat,
        ...)

## S3 method for class 'sibandwidth'
npindex(bws,
        txdat = stop("training data 'txdat' missing"),
        tydat = stop("training data 'tydat' missing"),
        exdat,
        eydat,
        gradients = FALSE,
        residuals = FALSE,
        errors = FALSE,
        boot.num = 399,
        ...)
```

Arguments

`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).
`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`.
`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`.
`boot.num`	an integer specifying the number of bootstrap replications to use when performing standard error calculations. Defaults to `399`.
`...`	additional arguments supplied to specify the parameters to the `sibandwidth` S3 method, which is called during estimation.
`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.
`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`.
`txdat`	a p-variate data frame of explanatory data (training data) used to calculate the regression estimators. 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.
`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.

Details

A matrix of gradients along with average derivatives are computed and returned if `gradients=TRUE` is used.

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

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.

`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.

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

mpi.bcast.cmd(set.seed(42),
              caller.execute=TRUE)
```

```
n <- 500

x1 <- runif(n, min=-1, max=1)
x2 <- runif(n, min=-1, max=1)
y <- x1 - x2 + rnorm(n)
mydat <- data.frame(x1,x2,y)
rm(y,x1,x2)

mpi.bcast.Robj2slave(mydat)

## Ichimura, continuous y

mpi.bcast.cmd(bw <- npindexbw(formula=y~x1+x2,
                              data=mydat),
              caller.execute=TRUE)

summary(bw)

mpi.bcast.cmd(model <- npindex(bws=bw,
                               gradients=TRUE),
              caller.execute=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.stop()` 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.stop()               ## soft close (may keep slaves alive)
## npRmpi.stop(force=TRUE)  ## hard close

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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, ...)

## S3 method for class 'NULL'
npindexbw(xdat = stop("training data xdat missing"),
          ydat = stop("training data ydat missing"),
          bws,
          ...)

## Default S3 method:
npindexbw(xdat = stop("training data xdat missing"),
          ydat = stop("training data ydat missing"),
          bws,
          bandwidth.compute = TRUE,
          nmulti,
          random.seed,
          optim.method,
          optim.maxattempts,
          optim.reltol,
          optim.abstol,
          optim.maxit,
          only.optimize.beta,
          ...)

## S3 method for class 'sibandwidth'
npindexbw(xdat = stop("training data xdat missing"),
          ydat = stop("training data ydat missing"),
          bws,
          bandwidth.compute = TRUE,
          nmulti,
          random.seed = 42,
          optim.method = c("Nelder-Mead", "BFGS", "CG"),
          optim.maxattempts = 10,
          optim.reltol = sqrt(.Machine$double.eps),
          optim.abstol = .Machine$double.eps,
          optim.maxit = 500,
          only.optimize.beta = FALSE,
          ...)
```

Arguments

`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.
`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`.
`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.
`method`	the single index model method, one of either "ichimura" (Ichimura (1993)) or "kleinspady" (Klein and Spady (1993)). Defaults to `ichimura`.
`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(5,ncol(xdat))`.
`random.seed`	an integer used to seed R's random number generator. This ensures replicability of the numerical search. Defaults to 42.
`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`.
`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.maxattempts`
: maximum number of attempts taken trying to achieve successful convergence in `optim`. Defaults to `100`.

`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.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`.

`optim.maxit`
: maximum number of iterations used by `optim`. Defaults to `500`.

`only.optimize.beta`
: signals the routine to only minimize the objective function with respect to beta

`...`
: 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

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.

`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 directly 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(5, ncol(xdat)) times) as is done for a number of examples.

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.

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.

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(5, 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.

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.

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

mpi.bcast.cmd(set.seed(42),
              caller.execute=TRUE)

n <- 500

x1 <- runif(n, min=-1, max=1)
x2 <- runif(n, min=-1, max=1)
y <- x1 - x2 + rnorm(n)
mydat <- data.frame(x1,x2,y)
rm(y,x1,x2)

mpi.bcast.Robj2slave(mydat)

## Ichimura, continuous y

mpi.bcast.cmd(bw <- npindexbw(formula=y~x1+x2,
                              data=mydat),
              caller.execute=TRUE)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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,
       leave.one.out = FALSE,
       kernel.pow = 1.0,
       bandwidth.divide = FALSE,
       operator = names(ALL_OPERATORS),
       permutation.operator = names(PERMUTATION_OPERATORS),
       compute.score = FALSE,
       compute.ocg = FALSE,
       return.kernel.weights = FALSE,
       ...)

## S3 method for class 'numeric'
npksum(bws,
       txdat = stop("training data 'txdat' missing"),
       tydat,
       exdat,
       weights,
       leave.one.out,
       kernel.pow,
       bandwidth.divide,
       operator,
       permutation.operator,
       compute.score,
       compute.ocg,
       return.kernel.weights,
       ...)
```

Arguments

`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 supplied to specify the parameters to the `default` S3 method, which is called during estimation.
`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).
`bws`	a bandwidth specification. This can be set as any suitable bandwidth object returned from a bandwidth-generating function, or a numeric vector.

`weights`	a n by q matrix of weights which can optionally be applied to `tydat` in the sum. See details.
`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`.
`kernel.pow`	an integer specifying the power to which the kernels will be raised in the sum. Defaults to `1`.
`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`.
`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.
`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.
`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.
`return.kernel.weights`	a logical specifying whether or not to return the matrix of generalized product kernel weights. Defaults to `FALSE`. See details.

Details

`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.

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

n <- 100000
x <- rnorm(n)
x.eval <- seq(-4, 4, length=50)

mpi.bcast.Robj2slave(x)
mpi.bcast.Robj2slave(x.eval)

mpi.bcast.cmd(bw <- npudensbw(dat=x, bwmethod="normal-reference"),
              caller.execute=TRUE)

mpi.bcast.cmd(den.ksum <- npksum(txdat=x, exdat=x.eval, bws=bw$bw,
                   bandwidth.divide=TRUE)$ksum/n,
              caller.execute=TRUE)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## End(Not run)
```

npplot *General Purpose Plotting of Nonparametric Objects*

Description

npplot is invoked by plot and generates 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.

Usage

```
npplot(bws = stop("'bws' has not been set"), ..., random.seed = 42)

## S3 method for class 'bandwidth'
npplot(bws,
       xdat,
       data = NULL,
       xq = 0.5,
       xtrim = 0.0,
       neval = 50,
       common.scale = TRUE,
       perspective = TRUE,
       main = NULL,
       type = NULL,
       border = NULL,
       cex.axis = NULL,
       cex.lab = NULL,
       cex.main = NULL,
       cex.sub = NULL,
       col = NULL,
       ylab = NULL,
       xlab = NULL,
       zlab = NULL,
       sub = NULL,
       ylim = NULL,
       xlim = NULL,
       zlim = NULL,
       lty = NULL,
       lwd = NULL,
       theta = 0.0,
       phi = 10.0,
       view = c("rotate","fixed"),
       plot.behavior = c("plot","plot-data","data"),
       plot.errors.method = c("none","bootstrap","asymptotic"),
       plot.errors.boot.method = c("inid", "fixed", "geom"),
       plot.errors.boot.blocklen = NULL,
       plot.errors.boot.num = 399,
       plot.errors.center = c("estimate","bias-corrected"),
       plot.errors.type = c("standard","quantiles"),
       plot.errors.quantiles = c(0.025,0.975),
       plot.errors.style = c("band","bar"),
       plot.errors.bar = c("|","I"),
       plot.errors.bar.num = min(neval,25),
       plot.bxp = FALSE,
       plot.bxp.out = TRUE,
       plot.par.mfrow = TRUE,
       ...,
       random.seed)

## S3 method for class 'conbandwidth'
npplot(bws,
       xdat,
       ydat,
       data = NULL,
       xq = 0.5,
       yq = 0.5,
       xtrim = 0.0,
       ytrim = 0.0,
       neval = 50,
       gradients = FALSE,
       common.scale = TRUE,
       perspective = TRUE,
       main = NULL,
       type = NULL,
       border = NULL,
       cex.axis = NULL,
       cex.lab = NULL,
       cex.main = NULL,
       cex.sub = NULL,
       col = NULL,
       ylab = NULL,
       xlab = NULL,
       zlab = NULL,
       sub = NULL,
       ylim = NULL,
       xlim = NULL,
       zlim = NULL,
       lty = NULL,
       lwd = NULL,
       theta = 0.0,
       phi = 10.0,
       tau = 0.5,
       view = c("rotate","fixed"),
       plot.behavior = c("plot","plot-data","data"),
       plot.errors.method = c("none","bootstrap","asymptotic"),
       plot.errors.boot.method = c("inid", "fixed", "geom"),
       plot.errors.boot.blocklen = NULL,
       plot.errors.boot.num = 399,
       plot.errors.center = c("estimate","bias-corrected"),
       plot.errors.type = c("standard","quantiles"),
       plot.errors.quantiles = c(0.025,0.975),
       plot.errors.style = c("band","bar"),
       plot.errors.bar = c("|","I"),
       plot.errors.bar.num = min(neval,25),
       plot.bxp = FALSE,
       plot.bxp.out = TRUE,
       plot.par.mfrow = TRUE,
       ...,
       random.seed)

## S3 method for class 'plbandwidth'
npplot(bws,
       xdat,
       ydat,
       zdat,
       data = NULL,
       xq = 0.5,
       zq = 0.5,
       xtrim = 0.0,
       ztrim = 0.0,
       neval = 50,
       common.scale = TRUE,
       perspective = TRUE,
       gradients = FALSE,
       main = NULL,
       type = NULL,
       border = NULL,
       cex.axis = NULL,
       cex.lab = NULL,
       cex.main = NULL,
       cex.sub = NULL,
       col = NULL,
       ylab = NULL,
       xlab = NULL,
       zlab = NULL,
       sub = NULL,
       ylim = NULL,
       xlim = NULL,
       zlim = NULL,
       lty = NULL,
       lwd = NULL,
       theta = 0.0,
       phi = 10.0,
       view = c("rotate","fixed"),
       plot.behavior = c("plot","plot-data","data"),
       plot.errors.method = c("none","bootstrap","asymptotic"),
       plot.errors.boot.method = c("inid", "fixed", "geom"),
       plot.errors.boot.blocklen = NULL,
       plot.errors.boot.num = 399,
       plot.errors.center = c("estimate","bias-corrected"),
       plot.errors.type = c("standard","quantiles"),
       plot.errors.quantiles = c(0.025,0.975),
       plot.errors.style = c("band","bar"),
       plot.errors.bar = c("|","I"),
       plot.errors.bar.num = min(neval,25),
       plot.bxp = FALSE,
       plot.bxp.out = TRUE,
       plot.par.mfrow = TRUE,
       ...,
       random.seed)

## S3 method for class 'rbandwidth'
npplot(bws,
       xdat,
       ydat,
       data = NULL,
       xq = 0.5,
       xtrim = 0.0,
       neval = 50,
       common.scale = TRUE,
       perspective = TRUE,
       gradients = FALSE,
       main = NULL,
       type = NULL,
       border = NULL,
       cex.axis = NULL,
       cex.lab = NULL,
       cex.main = NULL,
       cex.sub = NULL,
       col = NULL,
       ylab = NULL,
       xlab = NULL,
       zlab = NULL,
       sub = NULL,
       ylim = NULL,
       xlim = NULL,
       zlim = NULL,
       lty = NULL,
       lwd = NULL,
       theta = 0.0,
       phi = 10.0,
       view = c("rotate","fixed"),
       plot.behavior = c("plot","plot-data","data"),
       plot.errors.method = c("none","bootstrap","asymptotic"),
       plot.errors.boot.num = 399,
       plot.errors.boot.method = c("inid", "fixed", "geom"),
       plot.errors.boot.blocklen = NULL,
       plot.errors.center = c("estimate","bias-corrected"),
       plot.errors.type = c("standard","quantiles"),
       plot.errors.quantiles = c(0.025,0.975),
       plot.errors.style = c("band","bar"),
       plot.errors.bar = c("|","I"),
       plot.errors.bar.num = min(neval,25),
       plot.bxp = FALSE,
       plot.bxp.out = TRUE,
       plot.par.mfrow = TRUE,
       ...,
       random.seed)

## S3 method for class 'scbandwidth'
npplot(bws,
       xdat,
       ydat,
       zdat = NULL,
       data = NULL,
       xq = 0.5,
       zq = 0.5,
       xtrim = 0.0,
       ztrim = 0.0,
       neval = 50,
       common.scale = TRUE,
       perspective = TRUE,
       gradients = FALSE,
       main = NULL,
       type = NULL,
       border = NULL,
       cex.axis = NULL,
       cex.lab = NULL,
       cex.main = NULL,
       cex.sub = NULL,
       col = NULL,
       ylab = NULL,
       xlab = NULL,
       zlab = NULL,
       sub = NULL,
       ylim = NULL,
       xlim = NULL,
       zlim = NULL,
       lty = NULL,
       lwd = NULL,
       theta = 0.0,
       phi = 10.0,
       view = c("rotate","fixed"),
       plot.behavior = c("plot","plot-data","data"),
       plot.errors.method = c("none","bootstrap","asymptotic"),
       plot.errors.boot.num = 399,
       plot.errors.boot.method = c("inid", "fixed", "geom"),
       plot.errors.boot.blocklen = NULL,
       plot.errors.center = c("estimate","bias-corrected"),
       plot.errors.type = c("standard","quantiles"),
       plot.errors.quantiles = c(0.025,0.975),
       plot.errors.style = c("band","bar"),
       plot.errors.bar = c("|","I"),
       plot.errors.bar.num = min(neval,25),
       plot.bxp = FALSE,
       plot.bxp.out = TRUE,
       plot.par.mfrow = TRUE,
       ...,
       random.seed)

## S3 method for class 'sibandwidth'
npplot(bws,
       xdat,
       ydat,
       data = NULL,
       common.scale = TRUE,
       gradients = FALSE,
       main = NULL,
       type = NULL,
       cex.axis = NULL,
       cex.lab = NULL,
       cex.main = NULL,
       cex.sub = NULL,
       col = NULL,
       ylab = NULL,
       xlab = NULL,
       sub = NULL,
       ylim = NULL,
       xlim = NULL,
       lty = NULL,
       lwd = NULL,
       plot.behavior = c("plot","plot-data","data"),
       plot.errors.method = c("none","bootstrap","asymptotic"),
       plot.errors.boot.num = 399,
       plot.errors.boot.method = c("inid", "fixed", "geom"),
       plot.errors.boot.blocklen = NULL,
       plot.errors.center = c("estimate","bias-corrected"),
       plot.errors.type = c("standard","quantiles"),
       plot.errors.quantiles = c(0.025,0.975),
       plot.errors.style = c("band","bar"),
       plot.errors.bar = c("|","I"),
       plot.errors.bar.num = NULL,
       plot.par.mfrow = TRUE,
       ...,
       random.seed)
```

Arguments

`bws`	a bandwidth specification. This should be a bandwidth object returned from an invocation of `npudensbw`, `npcdensbw`, `npregbw`, `npplregbw`, `npindexbw`, or `npscoefbw`.
`...`	additional arguments supplied to control various aspects of plotting, depending on the type of object to be plotted, detailed 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(bws)`, typically the environment where the bandwidth object was generated.
`xdat`	a p-variate data frame of sample realizations (training data).
`ydat`	a q-variate data frame of sample realizations (training data). In a regression or conditional density context, this is the dependent data.
`zdat`	a p-variate data frame of sample realizations (training data).
`xq`	a numeric p-vector of quantiles. Each element i of `xq` corresponds to the ith column of `txdat`. Defaults to the median (0.5). See details.
`yq`	a numeric q-vector of quantiles. Each element i of `yq` corresponds to the ith column of `tydat`. Only to be specified in a conditional density context. Defaults to the median (0.5). See details.
`zq`	a numeric q-vector of quantiles. Each element i of `zq` corresponds to the ith column of `tzdat`. Only to be specified in a semiparametric model context. Defaults to the median (0.5). See details.
`xtrim`	a numeric p-vector of quantiles. Each element i of `xtrim` corresponds to the ith column of `txdat`. Defaults to `0.0`. See details.
`ytrim`	a numeric q-vector of quantiles. Each element i of `ytrim` corresponds to the ith column of `tydat`. Defaults to `0.0`. See details.
`ztrim`	a numeric q-vector of quantiles. Each element i of `ztrim` corresponds to the ith column of `tzdat`. Defaults to `0.0`. See details.
`neval`	an integer specifying the number of evaluation points. Only applies to continuous variables however, as discrete variables will be evaluated once at each category. Defaults to `50`.
`common.scale`	a logical value specifying whether or not all graphs are to be plotted on a common scale. Defaults to `TRUE`.
`perspective`	a logical value specifying whether a perspective plot should be displayed (if possible). Defaults to `TRUE`.
`gradients`	a logical value specifying whether gradients should be plotted (if possible). Defaults to `FALSE`.
`main`	optional title, see `title`. Defaults to `NULL`.

`sub`	optional subtitle, see `sub`. Defaults to `NULL`.
`type`	optional character indicating the type of plotting; actually any of the types as in `plot.default`. Defaults to `NULL`.
`border`	optional character indicating the border of plotting; actually any of the borders as in `plot.default`. Defaults to `NULL`.
`cex.axis`	The magnification to be used for axis annotation relative to the current setting of cex.
`cex.lab`	The magnification to be used for x and y labels relative to the current setting of cex.
`cex.main`	The magnification to be used for main titles relative to the current setting of cex.
`cex.sub`	The magnification to be used for sub-titles relative to the current setting of cex.
`col`	optional character indicating the color of plotting; actually any of the colours as in `plot.default`. Defaults to `NULL`.
`ylab`	optional character indicating the y axis label of plotting; actually any of the ylabs as in `plot.default`. Defaults to `NULL`.
`xlab`	optional character indicating the x axis label of plotting; actually any of the xlabs as in `plot.default`. Defaults to `NULL`.
`zlab`	optional character indicating the z axis label of plotting; actually any of the zlabs as in `plot.default`. Defaults to `NULL`.
`ylim`	optional a two-element numeric vector of the minimum and maximum y plotting limits. Defaults to `NULL`.
`xlim`	a two-element numeric vector of the minimum and maximum x plotting limits. Defaults to `NULL`.
`zlim`	a two-element numeric vector of the minimum and maximum z plotting limits. Defaults to `NULL`.
`lty`	a numeric value indicating the line type of plotting; actually any of the ltys as in `plot.default`. Defaults to `1`.
`lwd`	a numeric value indicating the width of the line of plotting; actually any of the lwds as in `plot.default`. Defaults to `1`.
`theta`	a numeric value specifying the starting azimuthal angle of the perspective plot. Defaults to `0.0`.
`phi`	a numeric value specifying the starting zenith angle of the perspective plot. Defaults to `10.0`.
`tau`	a numeric value specifying the τth quantile is desired when plotting quantile regressions. Defaults to `0.5`.
`view`	a character string used to specify the viewing mode of the perspective plot. Can be set as `rotate` or `fixed`. Defaults to `rotate`.
`plot.behavior`	a character string used to specify the net behavior of `npplot`. Can be set as `plot`, `plot-data` or `data`. Defaults to `plot`. See value.
`plot.errors.method`	a character string used to specify the method to calculate errors. Can be set as `none`, `bootstrap`, or `asymptotic`. Defaults to `none`.

`plot.errors.boot.method`	a character string used to specify the bootstrap method. Can be set as `inid`, `fixed`, or `geom` (see below for details). Defaults to `inid`.
`plot.errors.boot.blocklen`	an integer used to specify the block length b for the `fixed` or `geom` bootstrap (see below for details).
`plot.errors.boot.num`	an integer used to specify the number of bootstrap samples to use for the calculation of errors. Defaults to `399`.
`plot.errors.center`	a character string used to specify where to center the errors on the plot(s). Can be set as `estimate` or `bias-corrected`. Defaults to `estimate`.
`plot.errors.type`	a character string used to specify the type of error to calculate. Can be set as `standard` or `quantiles`. Defaults to `standard`.
`plot.errors.quantiles`	a numeric vector specifying the quantiles of the statistic to calculate for the purpose of error plotting. Defaults to `c(0.025,0.975)`.
`plot.errors.style`	a character string used to specify the style of error plotting. Can be set as `band` or `bar`. Defaults to `band`. Bands are not drawn for discrete variables.
`plot.errors.bar`	a character string used to specify the error bar shape. Can be set as `|` (vertical bar character) for a dashed vertical bar, or as `I` for an 'I' shaped error bar with horizontal bounding bars. Defaults to `|`.
`plot.errors.bar.num`	an integer specifying the number of error bars to plot. Defaults to `min(neval,25)`.
`plot.bxp`	a logical value specifying whether boxplots should be produced when appropriate. Defaults to `FALSE`.
`plot.bxp.out`	a logical value specifying whether outliers should be plotted on boxplots. Defaults to `TRUE`.
`plot.par.mfrow`	a logical value specifying whether `par(mfrow=c(,))` should be called before plotting. Defaults to `TRUE`.
`random.seed`	an integer used to seed R's random number generator. This ensures replicability of the bootstrapped errors. Defaults to 42.

Details

`npplot` is a general purpose plotting routine for visually exploring objects generated by the `npRmpi` library, such as regressions, quantile regressions, partially linear regressions, single-index models, densities and distributions. There is no need to call `npplot` directly as it is automatically invoked when `plot` is used with an object generated by the **npRmpi** package.

Visualizing one and two dimensional datasets is a straightforward process. The default behavior of `npplot` is to generate a standard 2D plot to visualize univariate data, and a perspective plot for bivariate data. When visualizing higher dimensional data, `npplot` 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. `npplot` uses an analogous procedure to extend the upper limit of the interval.

Bootstrap resampling is conducted pairwise on (y, X, Z) (i.e., by resampling from rows of the (y, X) data or (y, X, Z) data where appropriate). `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 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.

Value

Setting `plot.behavior` will instruct `npplot` what data to return. Option summary:
`plot`: instruct `npplot` to just plot the data and return `NULL`
`plot-data`: instruct `npplot` 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 `npplot` 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 `npplot` 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.

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.

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.

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.

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

x <- rnorm(100)
mpi.bcast.Robj2slave(x)
mpi.bcast.cmd(fhat <- npudens(~x),
              caller.execute=TRUE)

mpi.bcast.cmd(plot(fhat),
              caller.execute=TRUE)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## End(Not run)
```

---

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, ...)

## S3 method for class 'call'
npplreg(bws, ...)

## 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

`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.
`...`	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`, as described in `npregbw`.
`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.
`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`.
`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 training data used to compute the bandwidth object.
`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.
`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`.
`residuals`	a logical value indicating that you want residuals computed and returned in the resulting `plregression` object. Defaults to `FALSE`.

Details

`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.

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:

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
bw	the bandwidths, stored as a plbandwidth object
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

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.

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

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",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.stop(force=TRUE)` then restart.
npRmpi.start(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)

mpi.bcast.Robj2slave(x1)
mpi.bcast.Robj2slave(x2)
mpi.bcast.Robj2slave(z1)
mpi.bcast.Robj2slave(z2)
mpi.bcast.Robj2slave(y)

mpi.bcast.cmd(bw <- npplregbw(formula=y~x1+x2|z1+z2),
              caller.execute=TRUE)

mpi.bcast.cmd(pl <- npplreg(bws=bw),
              caller.execute=TRUE)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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, ...)

## S3 method for class 'NULL'
npplregbw(xdat = stop("invoked without data `xdat'"),
          ydat = stop("invoked without data `ydat'"),
          zdat = stop("invoked without data `zdat'"),
          bws,
          ...)

## Default S3 method:
npplregbw(xdat = stop("invoked without data `xdat'"),
          ydat = stop("invoked without data `ydat'"),
          zdat = stop("invoked without data `zdat'"),
          bws,
          ...,
          bandwidth.compute = TRUE,
          nmulti,
          remin,
          itmax,
          ftol,
          tol,
          small)

## 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

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.
`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)
`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.
`...`	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`, as described in `npregbw`.
`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`.
`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(5,ncol(zdat))`.
`remin`	a logical value which when set as `TRUE` the search routine restarts from located minima for a minor gain in accuracy. Defaults to `TRUE`
`itmax`	integer number of iterations before failure in the numerical optimization routine. Defaults to `10000`

`ftol`	tolerance on the value of the cross-validation function evaluated at located minima. Defaults to `1.19e-07` `(FLT_EPSILON)`
`tol`	tolerance on the position of located minima of the cross-validation function. Defaults to `1.49e-08` `(sqrt(DBL_EPSILON))`
`small`	a small number, at about the precision of the data type used. Defaults to `2.22e-16` `(DBL_EPSILON)`

Details

`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.

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.

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(5, 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.

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

`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",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.stop(force=TRUE)` then restart.
npRmpi.start(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)

mpi.bcast.Robj2slave(x1)
mpi.bcast.Robj2slave(x2)
mpi.bcast.Robj2slave(z1)
mpi.bcast.Robj2slave(z2)
mpi.bcast.Robj2slave(y)

mpi.bcast.cmd(bw <- npplregbw(formula=y~x1+x2|z1+z2),
              caller.execute=TRUE)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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

`formula`	a symbolic description of variables on which the test 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.
`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.
`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`.
`tau`	a numeric value specifying the τth quantile is desired
`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`.
`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"`.
`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`.
`pivot`	a logical value specifying whether the statistic should be normalised such that it approaches $N(0, 1)$ in distribution. Defaults to `TRUE`.
`density.weighted`	a logical value specifying whether the statistic should be weighted by the density of `xdat`. 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 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.

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`.

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.

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",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.stop(force=TRUE)` then restart.
```

```
npRmpi.start(nslaves=1)

mpi.bcast.cmd(library("quantreg"),
              caller.execute=TRUE)

mpi.bcast.cmd(data("cps71"),
              caller.execute=TRUE)
mpi.bcast.cmd(attach(cps71),
              caller.execute=TRUE)

mpi.bcast.cmd(model <- rq(logwage~age+I(age^2), tau=0.5, model=TRUE),
              caller.execute=TRUE)

mpi.bcast.cmd(X <- data.frame(age),
              caller.execute=TRUE)

# Note - this may take a few minutes depending on the speed of your
# computer...

mpi.bcast.cmd(output <- npqcmstest(model=model, xdat=X,
                                   ydat=logwage, tau=0.5,
                                   boot.num=29),
              caller.execute=TRUE)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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 'call'
npqreg(bws, ...)

## 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,
       ftol = 1.490116e-07,
       tol = 1.490116e-04,
       small = 1.490116e-05,
       itmax = 10000,
       lbc.dir = 0.5,
       dfc.dir = 3,
       cfac.dir = 2.5*(3.0-sqrt(5)),
       initc.dir = 1.0,
       lbd.dir = 0.1,
       hbd.dir = 1,
       dfac.dir = 0.25*(3.0-sqrt(5)),
       initd.dir = 1.0,
       ...)

## Default S3 method:
npqreg(bws, txdat, tydat, ...)
```

Arguments

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.
`tau`	a numeric value specifying the τth quantile is desired. Defaults to `0.5`.
`...`	additional arguments supplied to specify the regression type, bandwidth type, kernel types, training data, and so on. To do this, you may specify any of `bwmethod`, `bwscaling`, `bwtype`, `cxkertype`, `cxkerorder`, `cykertype`, `cykerorder`, `uxkertype`, `uykertype`, `oxkertype`, `oykertype`, as described in `npcdistbw`.
`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.
`newdata`	An optional data frame in which to look for evaluation data. If omitted, the training data are used.
`txdat`	a p-variate data frame of explanatory data (training data) used to calculate the regression estimators. 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.
`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`	[currently not supported] a logical value indicating that you want gradients computed and returned in the resulting `npregression` object. Defaults to `FALSE`.
`itmax`	integer number of iterations before failure in the numerical optimization routine. Defaults to `10000`.
`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$double.eps)).
`tol`	tolerance on the position of located minima of the cross-validation function (tol should generally be no smaller than the square root of your machine's floating point precision). Defaults to `1.490116e-04 (1.0e+04*sqrt(.Machine$double.eps))`.
`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))`.
`lbc.dir`, `dfc.dir`, `cfac.dir`, `initc.dir`	lower bound, chi-square degrees of freedom, stretch factor, and initial non-random values for direction set search for Powell's algorithm for `numeric` variables. See Details
`lbd.dir`, `hbd.dir`, `dfac.dir`, `initd.dir`	lower bound, upper bound, stretch factor, and initial non-random values for direction set search for Powell's algorithm for categorical variables. See Details

Details

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

`npqreg` returns a `npqregression` object. The generic functions `fitted` (or `quantile`), `se`, `predict` (when using `predict` you must add the argument `tau=` to generate predictions other than the median), and `gradients`, extract (or generate) estimated values, 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
`quantile`	estimation of the quantile regression function (conditional quantile) at the evaluation points
`quanterr`	standard errors of the quantile regression estimates
`quantgrad`	gradients at each evaluation point
`tau`	the τth quantile computed

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.

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.

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

See Also

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

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

## A quantile regression example

mpi.bcast.cmd(bw <- npcdistbw(gdp~ordered(year),data=Italy),
              caller.execute=TRUE)

summary(bw)

mpi.bcast.cmd(model <- npqreg(bws=bw, tau=0.50),
              caller.execute=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.stop()` 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.stop()              ## soft close (may keep slaves alive)
## npRmpi.stop(force=TRUE)  ## hard close

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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

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).
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).
num.eval	an optional integer specifying the length of the grid on which the quasi-inverse is computed. Defaults to 10000.

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 ....
f	an optional argument fed to extendrange. Defaults to 1. See ?extendrange for details.
...	additional arguments supplied to specify the bandwidth type, kernel types, bandwidth selection methods, and so on. See ?npudistbw for details.

Details

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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)
set.seed(42)
x <- rchisq(100,df=10)
mpi.bcast.Robj2slave(x)
mpi.bcast.cmd(npquantile(x),caller.execute=TRUE)
npRmpi.stop()
```

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, alternatively, use kd-trees (options(np.tree=TRUE) along with a bounded kernel (ckertype="epanechnikov")), both of which will reduce the computational burden appreciably.

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

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",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.stop(force=TRUE)` then restart.
```

```
npRmpi.start(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)

mpi.bcast.Robj2slave(x)
mpi.bcast.Robj2slave(tau)

## Compute kernel smoothed sample quantiles
mpi.bcast.cmd(q <- npquantile(x,tau),
              caller.execute=TRUE)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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, ...)

## S3 method for class 'call'
npreg(bws, ...)

## Default S3 method:
npreg(bws, txdat, tydat, ...)

## S3 method for class 'rbandwidth'
npreg(bws,
      txdat = stop("training data 'txdat' missing"),
      tydat = stop("training data 'tydat' missing"),
      exdat,
      eydat,
      gradients = FALSE,
      residuals = FALSE,
      ...)
```

Arguments

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.
...	additional arguments supplied to specify the regression type, bandwidth type, kernel types, training data, and so on, detailed 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(bws), typically the environment from which npregbw was called.
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.

`txdat`	a p-variate data frame of explanatory data (training data) used to calculate the regression estimators. 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.
`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.
`gradients`	a logical value indicating that you want gradients computed and returned in the resulting `npregression` object. Defaults to `FALSE`.
`residuals`	a logical value indicating that you want residuals computed and returned in the resulting `npregression` object. Defaults to `FALSE`.

Details

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:

   ## 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.stop(force=TRUE)` then restart.
   npRmpi.start(nslaves=1)
   mpi.bcast.Robj2slave(x)
   mpi.bcast.Robj2slave(y)
   mpi.bcast.cmd(bw <- npregbw(y~x),caller.execute=TRUE)
   mpi.bcast.cmd(ghat <- npreg(bw),caller.execute=TRUE)
   npRmpi.stop()

    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.stop(force=TRUE)` then restart.
   npRmpi.start(nslaves=1)
   mpi.bcast.Robj2slave(x)
   mpi.bcast.Robj2slave(y)
   mpi.bcast.cmd(ghat <- npreg(y~x),caller.execute=TRUE)
   npRmpi.stop()

    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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)
mpi.bcast.Robj2slave(x)
mpi.bcast.Robj2slave(y)
mpi.bcast.cmd(ghat <- npreg(y~x, ckertype="epanechnikov", ckerorder=4),
              caller.execute=TRUE)
npRmpi.stop()

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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)
mpi.bcast.Robj2slave(x)
mpi.bcast.Robj2slave(y)
mpi.bcast.cmd(ghat <- npreg(tydat=y, txdat=x, ckertype="epanechnikov", ckerorder=4),
              caller.execute=TRUE)
npRmpi.stop()
```

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.

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

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.

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.

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.

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

loess

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",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.stop(force=TRUE)` then restart.
npRmpi.start(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)
mydat <- data.frame(y,x,z1,z2)
rm(x,y,z1,z2)

mpi.bcast.Robj2slave(mydat)

mpi.bcast.cmd(bw <- npregbw(y~x+z1+z2,
                            regtype="lc",
                            bwmethod="cv.ls",
                            data=mydat),
              caller.execute=TRUE)

summary(bw)

mpi.bcast.cmd(model <- npreg(bws=bw,
                             data=mydat),
              caller.execute=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.stop()` performs a "soft close". Use `force=TRUE` to
## actually shut down the slaves.

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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, ...)

## S3 method for class 'NULL'
npregbw(xdat = stop("invoked without data 'xdat'"),
        ydat = stop("invoked without data 'ydat'"),
        bws,
        ...)

## Default S3 method:
npregbw(xdat = stop("invoked without data 'xdat'"),
        ydat = stop("invoked without data 'ydat'"),
        bws,
        bandwidth.compute = TRUE,
        nmulti,
        remin,
        itmax,
        ftol,
        tol,
        small,
        lbc.dir,
        dfc.dir,
        cfac.dir,
        initc.dir,
        lbd.dir,
        hbd.dir,
        dfac.dir,
        initd.dir,
        lbc.init,
        hbc.init,
        cfac.init,
        lbd.init,
        hbd.init,
        dfac.init,
        scale.init.categorical.sample,
        transform.bounds = FALSE,
        invalid.penalty = c("baseline","dbmax"),
        penalty.multiplier = 10,
        regtype,
        bwmethod,
        bwscaling,
        bwtype,
        ckertype,
        ckerorder,
        ukertype,
        okertype,
        ...)

## S3 method for class 'rbandwidth'
npregbw(xdat = stop("invoked without data 'xdat'"),
        ydat = stop("invoked without data 'ydat'"),
        bws,
        bandwidth.compute = TRUE,
        nmulti,
        remin = TRUE,
        itmax = 10000,
        ftol = 1.490116e-07,
        tol = 1.490116e-04,
        small = 1.490116e-05,
        lbc.dir = 0.5,
        dfc.dir = 3,
        cfac.dir = 2.5*(3.0-sqrt(5)),
        initc.dir = 1.0,
        lbd.dir = 0.1,
        hbd.dir = 1,
        dfac.dir = 0.25*(3.0-sqrt(5)),
        initd.dir = 1.0,
        lbc.init = 0.1,
        hbc.init = 2.0,
        cfac.init = 0.5,
        lbd.init = 0.1,
        hbd.init = 0.9,
        dfac.init = 0.375,
        scale.init.categorical.sample = FALSE,
        transform.bounds = FALSE,
        invalid.penalty = c("baseline","dbmax"),
        penalty.multiplier = 10,
        ...)
```

Arguments

`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.
`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.
`...`	additional arguments supplied to specify the bandwidth type, kernel types, selection methods, and so on, detailed below.
`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. Defaults to `lc`.
`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
`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`.
`ckertype`	character string used to specify the continuous kernel type. Can be set as `gaussian`, `epanechnikov`, or `uniform`. Defaults to `gaussian`.
`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`.
`ukertype`	character string used to specify the unordered categorical kernel type. Can be set as `aitchisonaitken` or `liracine`. Defaults to `aitchisonaitken`.
`okertype`	character string used to specify the ordered categorical kernel type. Can be set as `wangvanryzin` or `liracine`. Defaults to `liracine`.
`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(5,ncol(xdat))`.

`remin`	a logical value which when set as `TRUE` the search routine restarts from located minima for a minor gain in accuracy. Defaults to `TRUE`.
`itmax`	integer number of iterations before failure in the numerical optimization routine. Defaults to `10000`.
`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$double.eps)).
`tol`	tolerance on the position of located minima of the cross-validation function (tol should generally be no smaller than the square root of your machine's floating point precision). Defaults to `1.490116e-04 (1.0e+04*sqrt(.Machine$double.eps))`.
`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))`.
`lbc.dir`, `dfc.dir`, `cfac.dir`, `initc.dir`	lower bound, chi-square degrees of freedom, stretch factor, and initial non-random values for direction set search for Powell's algorithm for `numeric` variables. See Details
`lbd.dir`, `hbd.dir`, `dfac.dir`, `initd.dir`	lower bound, upper bound, stretch factor, and initial non-random values for direction set search for Powell's algorithm for categorical variables. See Details
`lbc.init`, `hbc.init`, `cfac.init`	lower bound, upper bound, and non-random initial values for scale factors for `numeric` variables for Powell's algorithm. See Details
`lbd.init`, `hbd.init`, `dfac.init`	lower bound, upper bound, and non-random initial values 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
`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`.
`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"`.
`penalty.multiplier`	a numeric multiplier applied to the baseline penalty when `invalid.penalty="baseline"`. Defaults to `10`.

Details

`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 (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.

`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.

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.

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.

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(5, 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.

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

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",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.stop(force=TRUE)` then restart.
npRmpi.start(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)
mydat <- data.frame(y,x,z1,z2)
rm(x,y,z1,z2)

mpi.bcast.Robj2slave(mydat)

mpi.bcast.cmd(bw <- npregbw(y~x+z1+z2,
                            regtype="lc",
                            bwmethod="cv.ls",
                            data=mydat),
               caller.execute=TRUE)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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

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.
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.
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.
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.

`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)
`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.
`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).
`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)
`...`	additional arguments supplied to `npksum`.

Details

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=5` for the first iteration.

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 5 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

`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",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.stop(force=TRUE)` then restart.
npRmpi.start(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$z),]
# rm(y,z,w)
#
# mpi.bcast.Robj2slave(ivdata)
# mpi.bcast.cmd(attach(ivdata),
#               caller.execute=TRUE)
#
# mpi.bcast.cmd(model.iv <- npregiv(y=y,z=z,w=w),
#               caller.execute=TRUE)
# phi.iv <- model.iv$phi
#
# ## Now the non-iv local linear estimator of E(y|z)
#
# mpi.bcast.cmd(ll.mean <- fitted(npreg(y~z,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
#
# 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.stop()` 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.stop()               ## soft close (may keep slaves alive)
## npRmpi.stop(force=TRUE)  ## hard close

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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

`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.
`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.
`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`.
`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`.
`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`.
`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.
`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 supplied to `npreg` and `npksum`.

Details

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=5` for the first iteration.

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 5 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

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",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.stop(force=TRUE)` then restart.
npRmpi.start(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$z),]
# rm(y,z,w,u,v)
#
# mpi.bcast.Robj2slave(ivdata)
# mpi.bcast.cmd(attach(ivdata),
#               caller.execute=TRUE)
#
# mpi.bcast.cmd(model.ivderiv <- npregivderiv(y=y,z=z,w=w),
#               caller.execute=TRUE)
#
# ylim <-c(quantile(model.ivderiv$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.stop()` 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.stop()               ## soft close (may keep slaves alive)
## npRmpi.stop(force=TRUE)  ## hard close

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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 vignette containing many of the examples found in the help files accompanying the **npRmpi** package that is intended to serve as a gentle introduction to this package can be accessed via `vignette("npRmpi", 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` (which automatically calls `npplot`) can work with the bandwidth object directly without having to do the subsequent companion function evaluation.

As of `npRmpi` version 0.20-0, we allow the user to combine these steps. When using `npRmpi` versions 0.20-0 and higher, 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 could then 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, `npplot` (which is automatically invoked by `plot`) , facilitates graphing of multivariate objects. An example for creating postscript graphs using the `npplot` utility 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. 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.

To cite the `npRmpi` package, type `citation("npRmpi")` from within `R` for details.

Details

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) $k(z) = (\exp(-z^2/2) - \exp(-b^2/2))/(\text{erf}(b/\sqrt{2})\sqrt{2\pi} - 2b\exp(-b^2/2))$ where $z = (x_i - x)/h$, $b > 0$, $|z| \le b$ 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 $x_i \ne x$, 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.

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 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.

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.

`npRmpi.start` *Start/Stop Helpers for Interactive npRmpi Sessions*

Description

Convenience helpers for interactive use of **npRmpi**. These functions provide a recommended, robust workflow: initialize a slave pool once and reuse it across multiple examples within the same R session.

Usage

```
npRmpi.start(..., nslaves = 1, comm = 1)

npRmpi.stop(force = FALSE, dellog = TRUE, comm = 1)

npRmpi.session.info(comm = 1)
```

Arguments

`...`	Additional arguments passed to `mpi.spawn.Rslaves()`.
`nslaves`	Number of slaves to spawn for interactive execution.
`comm`	Communicator used for the master+slaves pool (defaults to `1`).
`force`	Logical; when `TRUE`, force a hard shutdown of slave daemons.
`dellog`	Logical; when `TRUE`, remove slave log files (if applicable).

Details

`npRmpi.start()` ensures that a slave pool exists (spawning if needed) and runs `np.mpi.initialize()` on all ranks via `mpi.bcast.cmd()`.

`npRmpi.stop()` 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.

`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.start(nslaves=1)

## ... run np* calls here ...

## Soft-stop (may keep daemons alive for reuse)
npRmpi.stop()

## Hard-stop (actually shuts down slaves)
## npRmpi.stop(force=TRUE)

## 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, ...)

## S3 method for class 'call'
npscoef(bws, ...)
```

```
## Default S3 method:
npscoef(bws, txdat, tydat, tzdat, ...)

## S3 method for class 'scbandwidth'
npscoef(bws,
        txdat = stop("training data 'txdat' missing"),
        tydat = stop("training data 'tydat' missing"),
        tzdat = NULL,
        exdat,
        eydat,
        ezdat,
        residuals = FALSE,
        errors = TRUE,
        iterate = TRUE,
        maxiter = 100,
        tol = .Machine$double.eps,
        leave.one.out = FALSE,
        betas = FALSE,
        ...)
```

Arguments

`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.
`...`	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`.
`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.
`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`.
`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.
`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`.
`errors`	a logical value indicating whether or not asymptotic standard errors should be computed and returned in the resulting `smoothcoefficient` object. Defaults to `TRUE`.
`residuals`	a logical value indicating that you want residuals computed and returned in the resulting `smoothcoefficient` object. Defaults to `FALSE`.
`iterate`	a logical value indicating whether or not backfitted estimates should be iterated for self-consistency. Defaults to `TRUE`.
`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`.
`tol`	desired tolerance on the relative convergence of backfit estimates. Defaults to `.Machine$double.eps`.
`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`.
`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`.

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
`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

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.

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

`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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

mpi.bcast.cmd(set.seed(42),
              caller.execute=TRUE)

n <- 500

x <- runif(n)
z <- runif(n, min=-2, max=2)
y <- x*exp(z)*(1.0+rnorm(n,sd = 0.2))
mydat <- data.frame(x,y,z)
rm(x,y,z)

mpi.bcast.Robj2slave(mydat)

## A smooth coefficient model example

mpi.bcast.cmd(bw <- npscoefbw(y~x|z,data=mydat),
              caller.execute=TRUE)

summary(bw)

mpi.bcast.cmd(model <- npscoef(bws=bw, gradients=TRUE),
              caller.execute=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.stop()` 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.stop()               ## soft close (may keep slaves alive)
## npRmpi.stop(force=TRUE)  ## hard close

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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, ...)

## S3 method for class 'NULL'
npscoefbw(xdat = stop("invoked without data 'xdat'"),
          ydat = stop("invoked without data 'ydat'"),
          zdat = NULL,
          bws,
          ...)

## Default S3 method:
npscoefbw(xdat = stop("invoked without data 'xdat'"),
          ydat = stop("invoked without data 'ydat'"),
          zdat = NULL,
          bws,
          nmulti,
          random.seed,
          cv.iterate,
          cv.num.iterations,
          backfit.iterate,
          backfit.maxiter,
          backfit.tol,
          bandwidth.compute = TRUE,
          bwmethod,
          bwscaling,
          bwtype,
          ckertype,
          ckerorder,
          ukertype,
          okertype,
          optim.method,
          optim.maxattempts,
          optim.reltol,
          optim.abstol,
          optim.maxit,
          ...)

## S3 method for class 'scbandwidth'
npscoefbw(xdat = stop("invoked without data 'xdat'"),
          ydat = stop("invoked without data 'ydat'"),
          zdat = NULL,
          bws,
          nmulti,
          random.seed = 42,
          cv.iterate = FALSE,
          cv.num.iterations = 1,
          backfit.iterate = FALSE,
          backfit.maxiter = 100,
          backfit.tol = .Machine$double.eps,
          bandwidth.compute = TRUE,
          optim.method = c("Nelder-Mead", "BFGS", "CG"),
          optim.maxattempts = 10,
          optim.reltol = sqrt(.Machine$double.eps),
          optim.abstol = .Machine$double.eps,
          optim.maxit = 500,
          ...)
```

Arguments

`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.
`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`.
`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.
`...`	additional arguments supplied to specify the regression type, bandwidth type, kernel types, selection methods, and so on, detailed below.
`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`.
`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
`ckertype`	character string used to specify the continuous kernel type. Can be set as `gaussian`, `epanechnikov`, or `uniform`. Defaults to `gaussian`.

`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`.
`ukertype`	character string used to specify the unordered categorical kernel type. Can be set as `aitchisonaitken` or `liracine`. Defaults to `aitchisonaitken`.
`okertype`	character string used to specify the ordered categorical kernel type. Can be set as `wangvanryzin` or `liracine`. Defaults to `liracine`.
`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(5,ncol(xdat))`.
`random.seed`	an integer used to seed R's random number generator. This ensures replicability of the numerical search. Defaults to 42.
`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.maxattempts`	maximum number of attempts taken trying to achieve successful convergence in `optim`. Defaults to `100`.
`optim.abstol`	the absolute convergence tolerance used by `optim`. Only useful for non-negative functions, as a tolerance for reaching zero. Defaults to `.Machine$double.eps`.
`optim.reltol`	relative convergence tolerance used by `optim`. The algorithm stops if it is unable to reduce the value by a factor of 'reltol * (abs(val) + reltol)' at a step. Defaults to `sqrt(.Machine$double.eps)`, typically about `1e-8`.
`optim.maxit`	maximum number of iterations used by `optim`. Defaults to `500`.
`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`.
`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.

Details

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.

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.

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(5,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.

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

`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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

mpi.bcast.cmd(set.seed(42),
              caller.execute=TRUE)

n <- 500

x <- runif(n)
z <- runif(n, min=-2, max=2)
y <- x*exp(z)*(1.0+rnorm(n,sd = 0.2))
mydat <- data.frame(x,y,z)
rm(x,y,z)

mpi.bcast.Robj2slave(mydat)

## A smooth coefficient model example

mpi.bcast.cmd(bw <- npscoefbw(y~x|z,data=mydat),
              caller.execute=TRUE)

summary(bw)

mpi.bcast.cmd(model <- npscoef(bws=bw, gradients=TRUE),
              caller.execute=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.stop()` 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.stop()               ## soft close (may keep slaves alive)
## npRmpi.stop(force=TRUE)  ## hard close

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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`	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`	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`.
`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.

Details

`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`.

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.

See Also

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

mpi.bcast.cmd(set.seed(42),
              caller.execute=TRUE)

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))

mpi.bcast.Robj2slave(yt)

mpi.bcast.cmd(output <- npsdeptest(yt,
                                   lag.num=2,
                                   boot.num=29,
                                   method="summation"),
              caller.execute=TRUE)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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	an integer seed for the random number generator.

Details

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

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

mpi.bcast.cmd(npseed(712),
              caller.execute=TRUE)

x <- runif(10)
y <- x + rnorm(10, sd = 0.1)
mydat <- data.frame(x,y)
rm(x,y)

mpi.bcast.Robj2slave(mydat)

mpi.bcast.cmd(bw <- npregbw(y~x, data=mydat),
              caller.execute=TRUE)
```

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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 'call'
npsigtest(bws, ...)

## 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(1,ncol(xdat)),
          random.seed = 42,
          ...)
```

Arguments

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.
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".
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`.
`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`.
`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`.
`random.seed`	an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.
`...`	additional arguments supplied to specify the bandwidth type, kernel types, selection methods, and so on, detailed below.

Details

`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.

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`).

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: 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.

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

mpi.bcast.cmd(set.seed(42),
              caller.execute=TRUE)

## 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)
mydat <- data.frame(z,x1,x2,y)
rm(z,x1,x2,y)

mpi.bcast.Robj2slave(mydat)

mpi.bcast.cmd(model <- npreg(y~z+x1+x2,
                             regtype="ll",
                             bwmethod="cv.aic",
                             data=mydat),
              caller.execute=TRUE)

mpi.bcast.cmd(output <- npsigtest(model,boot.num=29),
              caller.execute=TRUE)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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	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.
boot.num	an integer value specifying the number of bootstrap replications to use. Defaults to 399.
bw	a numeric (scalar) bandwidth. Defaults to plug-in (see details below).
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.
random.seed	an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.
...	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

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.

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

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

mpi.bcast.cmd(set.seed(42),
              caller.execute=TRUE)

## 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))

mpi.bcast.Robj2slave(yt)

## A simple example of the test for symmetry

mpi.bcast.cmd(output <- npsymtest(yt,
                                  boot.num=29,
                                  boot.method="geom",
                                  method="summation"),
              caller.execute=TRUE)

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.stop()` 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.stop()               ## soft close (may keep slaves alive)
```

```
## npRmpi.stop(force=TRUE)  ## hard close

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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

`b`	Truncation radius of the kernel.

Details

`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:

$$H_{-} = a_0 \operatorname{erf}(z/2 + b) e^{-z^2/4} + a_1 z + a_2 \operatorname{erf}((z+b)/\sqrt{2}) - c_0$$

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`>

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,
        ...)

## S3 method for class 'call'
npudens(bws, ...)

## Default S3 method:
npudens(bws, tdat, ...)
```

Arguments

`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.
`...`	additional arguments supplied to specify, the training data, the bandwidth type, kernel types, and so on. This is necessary if you specify bws as a p-vector and not a `bandwidth` object, and you do not desire the default behaviours. To do this, you may specify any of `bwscaling`, `bwtype`, `ckertype`, `ckerorder`, `ukertype`, `okertype`, as described in `npudensbw`.
`tdat`	a p-variate data frame of sample realizations (training data) used to estimate the density. Defaults to the training data used to compute the bandwidth object.
`edat`	a p-variate data frame of density evaluation points. By default, evaluation takes place on the data provided by `tdat`.
`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.
`newdata`	An optional data frame in which to look for evaluation data. If omitted, the training data are used.

Details

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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)
mpi.bcast.Robj2slave(y)
mpi.bcast.cmd(bw <- npudensbw(~y),caller.execute=TRUE)
mpi.bcast.cmd(fhat <- npudens(bw),caller.execute=TRUE)
npRmpi.stop()
```

```
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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)
mpi.bcast.Robj2slave(y)
mpi.bcast.cmd(fhat <- npudens(~y),caller.execute=TRUE)
npRmpi.stop()

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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)
mpi.bcast.Robj2slave(y)
mpi.bcast.cmd(fhat <- npudens(~y, ckertype="epanechnikov", ckerorder=4),
              caller.execute=TRUE)
npRmpi.stop()

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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)
mpi.bcast.Robj2slave(y)
mpi.bcast.cmd(fhat <- npudens(tdat = y, ckertype="epanechnikov", ckerorder=4),
              caller.execute=TRUE)
npRmpi.stop()
```

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

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.

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

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

mpi.bcast.cmd(data("Italy"),
              caller.execute=TRUE)
mpi.bcast.cmd(attach(Italy),
              caller.execute=TRUE)

mpi.bcast.cmd(bw <- npudensbw(formula=~year+gdp),
              caller.execute=TRUE)

mpi.bcast.cmd(fhat <- npudens(bws=bw),
              caller.execute=TRUE)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.
## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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 'NULL'
npudensbw(dat = stop("invoked without input data 'dat'"),
          bws,
          ...)

## S3 method for class 'bandwidth'
npudensbw(dat = stop("invoked without input data 'dat'"),
          bws,
          bandwidth.compute = TRUE,
          nmulti,
          remin = TRUE,
          itmax = 10000,
          ftol = 1.490116e-07,
          tol = 1.490116e-04,
          small = 1.490116e-05,
          lbc.dir = 0.5,
          dfc.dir = 3,
          cfac.dir = 2.5*(3.0-sqrt(5)),
          initc.dir = 1.0,
          lbd.dir = 0.1,
          hbd.dir = 1,
          dfac.dir = 0.25*(3.0-sqrt(5)),
          initd.dir = 1.0,
          lbc.init = 0.1,
          hbc.init = 2.0,
          cfac.init = 0.5,
          lbd.init = 0.1,
          hbd.init = 0.9,
          dfac.init = 0.375,
          scale.init.categorical.sample = FALSE,
          transform.bounds = FALSE,
          invalid.penalty = c("baseline","dbmax"),
          penalty.multiplier = 10,
          ...)

## Default S3 method:
npudensbw(dat = stop("invoked without input data 'dat'"),
          bws,
          bandwidth.compute = TRUE,
          nmulti,
          remin,
          itmax,
          ftol,
          tol,
          small,
          lbc.dir,
          dfc.dir,
          cfac.dir,
          initc.dir,
          lbd.dir,
          hbd.dir,
          dfac.dir,
          initd.dir,
          lbc.init,
          hbc.init,
          cfac.init,
          lbd.init,
          hbd.init,
          dfac.init,
          scale.init.categorical.sample,
          transform.bounds,
          invalid.penalty,
          penalty.multiplier,
          bwmethod,
          bwscaling,
          bwtype,
          ckertype,
          ckerorder,
          ukertype,
          okertype,
          ...)
```

Arguments

`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.
`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.
`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.
`...`	additional arguments supplied to specify the bandwidth type, kernel types, selection methods, and so on, detailed below.
`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. 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
`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`.
`ckertype`	character string used to specify the continuous kernel type. Can be set as `gaussian`, `epanechnikov`, or `uniform`. Defaults to `gaussian`.
`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`.
`ukertype`	character string used to specify the unordered categorical kernel type. Can be set as `aitchisonaitken` or `liracine`.
`okertype`	character string used to specify the ordered categorical kernel type. Can be set as `wangvanryzin` or `liracine`.
`nmulti`	integer number of times to restart the process of finding extrema of the cross-validation function from different (random) initial points.
`remin`	a logical value which when set as `TRUE` the search routine restarts from located minima for a minor gain in accuracy. Defaults to `TRUE`.
`itmax`	integer number of iterations before failure in the numerical optimization routine. Defaults to `10000`.
`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$double.eps)).
`tol`	tolerance on the position of located minima of the cross-validation function (tol should generally be no smaller than the square root of your machine's floating point precision). Defaults to `1.490116e-04 (1.0e+04*sqrt(.Machine$double.eps))`.
`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))`.
`lbc.dir`, `dfc.dir`, `cfac.dir`, `initc.dir`	lower bound, chi-square degrees of freedom, stretch factor, and initial non-random values for direction set search for Powell's algorithm for `numeric` variables. See Details
`lbd.dir`, `hbd.dir`, `dfac.dir`, `initd.dir`	lower bound, upper bound, stretch factor, and initial non-random values for direction set search for Powell's algorithm for categorical variables. See Details
`lbc.init`, `hbc.init`, `cfac.init`	lower bound, upper bound, and non-random initial values for scale factors for `numeric` variables for Powell's algorithm. See Details

`lbd.init`, `hbd.init`, `dfac.init`
: lower bound, upper bound, and non-random initial values 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

`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`.

`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"`.

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

Details

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.stop(force=TRUE)` then restart.
    npRmpi.start(nslaves=1)
    mpi.bcast.Robj2slave(y)
    mpi.bcast.cmd(bw <- npudensbw(~y),caller.execute=TRUE)
    npRmpi.stop()

    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.stop(force=TRUE)` then restart.
    npRmpi.start(nslaves=1)
    mpi.bcast.Robj2slave(y)
  mpi.bcast.cmd(fhat <- npudensbw(tdat = y, ckertype="epanechnikov", ckerorder=4),
                caller.execute=TRUE)
    npRmpi.stop()
```

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.

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(5, 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

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

mpi.bcast.cmd(data("Italy"),
              caller.execute=TRUE)
mpi.bcast.cmd(attach(Italy),
              caller.execute=TRUE)

mpi.bcast.cmd(bw <- npudensbw(formula=~year+gdp),
              caller.execute=TRUE)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)
## 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,
        ...)

## S3 method for class 'call'
npudist(bws, ...)

## Default S3 method:
npudist(bws, tdat, ...)
```

Arguments

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.
...	additional arguments supplied to specify the training data, the bandwidth type, kernel types, and so on. This is necessary if you specify bws as a p-vector and not a dbandwidth object, and you do not desire the default behaviours. To do this, you may specify any of bwscaling, bwtype, ckertype, ckerorder, okertype, as described in npudistbw.
tdat	a p-variate data frame of sample realizations (training data) used to estimate the cumulative distribution. Defaults to the training data used to compute the bandwidth object.

`edat`	a p-variate data frame of cumulative distribution evaluation points. By default, evaluation takes place on the data provided by `tdat`.
`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.
`newdata`	An optional data frame in which to look for evaluation data. If omitted, the training data are used.

Details

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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)
mpi.bcast.Robj2slave(y)
mpi.bcast.cmd(bw <- npudistbw(~y),caller.execute=TRUE)
mpi.bcast.cmd(Fhat <- npudist(bw),caller.execute=TRUE)
npRmpi.stop()

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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)
mpi.bcast.Robj2slave(y)
mpi.bcast.cmd(Fhat <- npudist(~y),caller.execute=TRUE)
npRmpi.stop()

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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)
mpi.bcast.Robj2slave(y)
mpi.bcast.cmd(Fhat <- npudist(~y, ckertype="epanechnikov", ckerorder=4),
              caller.execute=TRUE)
```

```
    npRmpi.stop()

    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.stop(force=TRUE)` then restart.
    npRmpi.start(nslaves=1)
    mpi.bcast.Robj2slave(y)
  mpi.bcast.cmd(Fhat <- npudist(tdat = y, ckertype="epanechnikov", ckerorder=4),
                 caller.execute=TRUE)
    npRmpi.stop()
```

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

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.

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

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

data("Italy")

mpi.bcast.Robj2slave(Italy)
mpi.bcast.cmd(bw <- npudistbw(formula=~ordered(year)+gdp,
                              data=Italy),
              caller.execute=TRUE)

mpi.bcast.cmd(F <- npudist(bws=bw),
              caller.execute=TRUE)

summary(F)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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 'NULL'
```

```
npudistbw(dat = stop("invoked without input data 'dat'"),
          bws,
          ...)

## S3 method for class 'dbandwidth'
npudistbw(dat = stop("invoked without input data 'dat'"),
          bws,
          gdat = NULL,
          bandwidth.compute = TRUE,
          nmulti,
          remin = TRUE,
          itmax = 10000,
          do.full.integral = FALSE,
          ngrid = 100,
          ftol = 1.490116e-07,
          tol = 1.490116e-04,
          small = 1.490116e-05,
          lbc.dir = 0.5,
          dfc.dir = 3,
          cfac.dir = 2.5*(3.0-sqrt(5)),
          initc.dir = 1.0,
          lbd.dir = 0.1,
          hbd.dir = 1,
          dfac.dir = 0.25*(3.0-sqrt(5)),
          initd.dir = 1.0,
          lbc.init = 0.1,
          hbc.init = 2.0,
          cfac.init = 0.5,
          lbd.init = 0.1,
          hbd.init = 0.9,
          dfac.init = 0.375,
          scale.init.categorical.sample = FALSE,
          memfac = 500.0,
          transform.bounds = FALSE,
          invalid.penalty = c("baseline","dbmax"),
          penalty.multiplier = 10,
          ...)

## Default S3 method:
npudistbw(dat = stop("invoked without input data 'dat'"),
          bws,
          gdat,
          bandwidth.compute = TRUE,
          nmulti,
          remin,
          itmax,
          do.full.integral,
          ngrid,
          ftol,
          tol,
          small,
          lbc.dir,
          dfc.dir,
          cfac.dir,
          initc.dir,
          lbd.dir,
          hbd.dir,
          dfac.dir,
          initd.dir,
          lbc.init,
          hbc.init,
          cfac.init,
          lbd.init,
          hbd.init,
          dfac.init,
          scale.init.categorical.sample,
          memfac,
          transform.bounds,
          invalid.penalty,
          penalty.multiplier,
          bwmethod,
          bwscaling,
          bwtype,
          ckertype,
          ckerorder,
          okertype,
          ...)
```

Arguments

`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.

`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).
`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.
`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.
`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).
`...`	additional arguments supplied to specify the bandwidth type, kernel types, selection methods, and so on, detailed below.
`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
`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`.
`ckertype`	character string used to specify the continuous kernel type. Can be set as `gaussian`, `epanechnikov`, or `uniform`. Defaults to `gaussian`.

`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`.

`okertype`
: character string used to specify the ordered categorical kernel type. Can be set as `wangvanryzin`.

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

`remin`
: a logical value which when set as `TRUE` the search routine restarts from located minima for a minor gain in accuracy. Defaults to `TRUE`.

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

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

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

`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)).

`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))`.

`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))`.

`lbc.dir`, `dfc.dir`, `cfac.dir`, `initc.dir`
: lower bound, chi-square degrees of freedom, stretch factor, and initial non-random values for direction set search for Powell's algorithm for `numeric` variables. See Details

`lbd.dir`, `hbd.dir`, `dfac.dir`, `initd.dir`
: lower bound, upper bound, stretch factor, and initial non-random values for direction set search for Powell's algorithm for categorical variables. See Details

`lbc.init`, `hbc.init`, `cfac.init`
: lower bound, upper bound, and non-random initial values for scale factors for `numeric` variables for Powell's algorithm. See Details

`lbd.init`, `hbd.init`, `dfac.init`
: lower bound, upper bound, and non-random initial values 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

`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`.

`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"`.

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

`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.

Details

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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)
mpi.bcast.Robj2slave(y)
mpi.bcast.cmd(bw <- npudistbw(~y),caller.execute=TRUE)
npRmpi.stop()

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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)
mpi.bcast.Robj2slave(y)
mpi.bcast.cmd(Fhat <- npudistbw(tdat = y, ckertype="epanechnikov", ckerorder=4),
              caller.execute=TRUE)
npRmpi.stop()
```

`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`.

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(5, 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

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

data("Italy")

mpi.bcast.Robj2slave(Italy)

mpi.bcast.cmd(bw <- npudistbw(formula=~ordered(year)+gdp,
                              data=Italy),
              caller.execute=TRUE)

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.stop()` 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.stop()               ## soft close (may keep slaves alive)
## npRmpi.stop(force=TRUE)  ## hard close
## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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 = 5,
                  proper = FALSE)
```

Arguments

`X`	a required numeric vector of training data lying in $[a, b]$
`Y`	an optional numeric vector of evaluation data lying in $[a, b]$
`h`	an optional bandwidth (>0)
`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 $\max(X)$)

`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 5)
`proper`	an optional logical value indicating whether to enforce proper density and distribution function estimates over the range $[a, b]$

Details

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

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$f)))
plot(X,dgp,ylab="Density",ylim=ylim,type="l")
lines(X,model.g1$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)
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)))
```

```
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")

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)
attach(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")
detach(cps71)
par(oldpar)

## 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

`X`	a required numeric vector of training data lying in $[a, b]$
`Y`	an optional numeric vector of evaluation data lying in $[a, b]$
`h`	an optional bandwidth (>0)
`a`	an optional lower bound (defaults to 0)
`b`	an optional upper bound (defaults to 1)
`...`	optional arguments passed to `npudensbw` and `npudens`

Details

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

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

## Example 1: f(0)=0, f(1)=1, plot boundary corrected density,
## unadjusted density, and DGP

mpi.bcast.cmd(set.seed(42),
              caller.execute=TRUE)

n <- 100
X <- sort(rbeta(n,5,1))
dgp <- dbeta(X,5,1)

mpi.bcast.Robj2slave(X)

mpi.bcast.cmd(model <- npuniden.reflect(X),
              caller.execute=TRUE)

mpi.bcast.cmd(model.unadjusted <- npuniden.boundary(X,a=-Inf,b=Inf),
              caller.execute=TRUE)

ylim <- c(0,max(c(dgp,model$f,model.unadjusted$f)))
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

mpi.bcast.cmd(set.seed(42),
              caller.execute=TRUE)
```

```
X <- sort(rbeta(100,5,3))

mpi.bcast.Robj2slave(X)

mpi.bcast.cmd(model <- npuniden.reflect(X),
              caller.execute=TRUE)

oldpar <- par(no.readonly = TRUE)
par(mfrow=c(1,2))
ylim=range(c(model$f,model$f+1.96*model$sd.f,model$f-1.96*model$sd.f,dbeta(X,5,3)))
plot(X,model$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")

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

mpi.bcast.cmd(data(cps71),
              caller.execute=TRUE)

mpi.bcast.cmd(model <- npuniden.reflect(cps71$age,a=21,b=65),
              caller.execute=TRUE)

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")

par(oldpar)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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

`X`	a required numeric vector of training data lying in $[a, b]$
`Y`	an optional numeric vector of evaluation data lying in $[a, b]$

`h`	a bandwidth (> 0)
`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)
`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"`
`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`)
`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`)
`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
`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

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

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))
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)
par(mfrow=c(1,2))
ylim <- range(c(foo$f.sc,foo$f))
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))
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))
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))
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))
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))

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))
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))
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)))
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))
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")
par(oldpar)

## 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.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	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.
boot.num	an integer value specifying the number of bootstrap replications to use. Defaults to 399.

`bw.x`, `bw.y`	numeric (scalar) bandwidths. Defaults to plug-in (see details below).
`random.seed`	an integer used to seed R's random number generator. This is to ensure replicability. Defaults to 42.
`...`	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

`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`.

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

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

mpi.bcast.cmd(set.seed(42),
              caller.execute=TRUE)

n <- 250

x <- rnorm(n)
y <- rnorm(n)

mpi.bcast.Robj2slave(x)
mpi.bcast.Robj2slave(y)

mpi.bcast.cmd(output <- npunitest(x,y,
                                  method="summation",
                                  bootstrap=TRUE,
                                  boot.num=29),
              caller.execute=TRUE)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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")
attach(oecdpanel)
summary(oecdpanel)
detach(oecdpanel)

## End(Not run)
```

`se` *Extract Standard Errors*

Description

`se` is a generic function which extracts standard errors from objects.

Usage

```
se(x)
```

Arguments

`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.

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",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.stop(force=TRUE)` then restart.
npRmpi.start(nslaves=1)

set.seed(42)

x <- rnorm(10)
mpi.bcast.Robj2slave(x)

mpi.bcast.cmd(bw <- npudensbw(~x),
              caller.execute=TRUE)

mpi.bcast.cmd(fhat <- npudens(bw),
              caller.execute=TRUE)

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

## Note that in order to exit npRmpi properly avoid quit(), and instead
## use mpi.quit() as follows.

## mpi.bcast.cmd(mpi.quit(),
##               caller.execute=TRUE)

## 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

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")
attach(wage1)
summary(wage1)
detach(wage1)

## End(Not run)
```

Index