Package ‘ROptEst’

August 3, 2018

Version 1.1.0
Date 2018-08-01
Title Optimally Robust Estimation
Description Optimally robust estimation in general smoothly parameterized models using S4 classes and methods.
Depends R(>= 2.14.0), methods, distr(>= 2.5.2), distrEx(>= 2.5),
distrMod(>= 2.5.2), RandVar(>= 0.9.2), RobAStBase(>= 1.0)
Imports startupmsg, MASS, stats, graphics, utils, grDevices
Suggests RobLox
ByteCompile yes
License LGPL-3
URL http://robast.r-forge.r-project.org/
Encoding latin1

LastChangedDate ${LastChangedDate: 2018-08-01 05:06:49 +0200 (Mi, 01.
Aug 2018) $}
LastChangedRevision ${LastChangedRevision: 1083 $}
VCS/SVNRevision 1081

NeedsCompilation no

Author Matthias Kohl [cre, cph],
   Mykhailo Pupashenko [ctb] (contributed wrapper functions for diagnostic
   plots),
   Gerald Kroisandt [ctb] (contributed testing routines),
   Peter Ruckdeschel [aut, cph]

Maintainer Matthias Kohl <Matthias.Kohl@stamats.de>

Repository CRAN
Date/Publication 2018-08-03 12:00:03 UTC
### R topics documented:

- ROptEst-package .............................................. 3
- asAnscombe .................................................... 4
- asAnscombe-class ................................................ 5
- asL1 ............................................................ 6
- asL1-class ...................................................... 7
- asL4 ............................................................. 8
- asL4-class ...................................................... 9
- cniperCont ...................................................... 10
- CniperPointPlot ............................................... 14
- comparePlot-methods ........................................... 15
- get.asGRisk.fct-methods ....................................... 16
- getAsRisk ...................................................... 17
- getBiasIC ...................................................... 22
- getFixClip .................................................... 23
- getFixRobIC .................................................. 24
- getIneffDiff .................................................. 26
- getInfCent .................................................... 28
- getInfClip ..................................................... 30
- getInfGamma ................................................... 32
- getInfLM ....................................................... 34
- getInfRad ..................................................... 37
- getInfRobIC ................................................... 39
- getInfStand ................................................... 44
- getInfV ........................................................ 45
- getL1normL2deriv ............................................ 47
- getL2normL2deriv ............................................ 48
- getMaxIneff .................................................. 49
- getModifyIC ................................................... 50
- getRadius ...................................................... 52
- getReq ........................................................ 53
- getRiskFctBV-methods ......................................... 55
- getRiskIC ...................................................... 55
- getStartIC-methods ............................................ 57
- inputGenerators .............................................. 59
- leastFavorableRadius .......................................... 61
- lowerCaseRadius .............................................. 62
- minmaxBias ..................................................... 64
- optIC .......................................................... 66
- optRisk ........................................................ 69
- ORobEstimate-class ........................................... 70
- plot-methods .................................................. 72
- radiusMinimaxIC ............................................... 73
- RMXEOMSEMBREOBRE .......................................... 75
- robest ......................................................... 80
- roptest ........................................................ 84
- updateNorm-methods ............................................ 91
Optimally robust estimation in general smoothly parameterized models using S4 classes and methods.
See Also

distr-package, distrEx-package, distrMod-package, RandVar-package, RobAStBase-package

Examples

## don't test to reduce check time on CRAN

library(ROptEst)

## Example: Rutherford-Geiger (1910); cf. Feller-(1968), Section VI.7 (a)
x <- c(rep(0, 57), rep(1, 203), rep(2, 383), rep(3, 525), rep(4, 532),
      rep(5, 408), rep(6, 273), rep(7, 139), rep(8, 45), rep(9, 27),
      rep(10, 10), rep(11, 4), rep(12, 0), rep(13, 1), rep(14, 1))

## ML-estimate from package distrMod
MLest <- MLEstimator(x, PoisFamily())
MLest

## confidence interval based on CLT
confint(MLest)

## compute optimally (w.r.t to MSE) robust estimator (unknown contamination)
robEst <- roptest(x, PoisFamily(), eps.upper = 0.1, steps = 3)
estimate(robEst)

## check influence curve
pIC(robEst)
checkIC(pIC(robEst))

## plot influence curve
plot(pIC(robEst))

## confidence interval based on LAN - neglecting bias
confint(robEst)

## confidence interval based on LAN - including bias
confint(robEst, method = symmetricBias())

---

asAnscombe  Generating function for asAnscombe-class

Description

Generates an object of class "asAnscombe".

Usage

asAnscombe(eff = .95, biastype = symmetricBias(), normtype = NormType())

Arguments

eff  value in (0,1]: ARE in the ideal model
biastype  a bias type of class BiasType
normtype  a norm type of class NormType
Value

Object of class asAnscombe

Author(s)

Peter Ruckdeschel <peter.ruckdeschel@fraunhofer.itwm.de>

References


See Also

asAnscombe-class

Examples

asAnscombe()

```r
## The function is currently defined as
function(ef = .95, biastype = symmetricBias(), normtype = NormType()){
  new("asAnscombe", ef = ef, biastype = biastype, normtype = normtype)
}
```

---

asAnscombe-class  Asymptotic Anscombe risk

Description

Class of asymptotic Anscombe risk which is the ARE (asymptotic relative efficiency) in the ideal model obtained by an optimal bias robust IC.

Objects from the Class

Objects can be created by calls of the form `new("asAnscombe", ...)`. More frequently they are created via the generating function `asAnscombe`.

Slots

type  Object of class "character": “optimal bias robust IC (OBRI) for given ARE (asymptotic relative efficiency)”.
eff  Object of class "numeric": given ARE (asymptotic relative efficiency) to be attained in the ideal model.
biastype Object of class "BiasType": symmetric, one-sided or asymmetric
Extends

Class "asRiskwithBias", directly.
Class "asRisk", by class "asRiskwithBias". Class "RiskType", by class "asRisk".

Methods

eff signature(object = "asAnscombe"): accessor function for slot eff.

show signature(object = "asAnscombe")

Author(s)

Peter Ruckdeschel <peter.ruckdeschel@fraunhofer.itwm.de>

References


See Also

asRisk-class, asAnscombe

Examples

new("asAnscombe")

asL1 Generating function for asMSE-class

Description

Generates an object of class "asMSE".

Usage

asL1(biastype = symmetricBias(), normtype = NormType())

Arguments

  biastype a bias type of class BiasType
  normtype a norm type of class NormType
### Value

Object of class "asMSE"

### Author(s)

Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

### References


### See Also

asL1-class, asMSE, asL4

### Examples

```r
asL1()
```

```r
## The function is currently defined as
function(biastype = symmetricBias(), normtype = NormType()){
  new("asL1", biastype = biastype, normtype = normtype )
}
```

---

<table>
<thead>
<tr>
<th>asL1-class</th>
<th>Asymptotic mean absolute error</th>
</tr>
</thead>
</table>

### Description

Class of asymptotic mean absolute error.

### Objects from the Class

Objects can be created by calls of the form `new("asL1", ...)`.

### Slots

- **type**: Object of class "character": "asymptotic mean square error"
- **biastype**: Object of class "BiasType": symmetric, one-sided or asymmetric
- **normtype**: Object of class "NormType": norm in which a multivariate parameter is considered

### Extends

- Class "asGRisk", directly.
- Class "asRiskwithBias", by class "asGRisk".
- Class "asRisk", by class "asRiskwithBias".
- Class "RiskType", by class "asGRisk".
Methods
No methods defined with class "asL1" in the signature.

Author(s)
Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

References

See Also
asGRisk-class, asMSE, asMSE-class, asL4-class, asL1

Examples
new("asMSE")

---

asL4

Generating function for asL4-class

Description
Generates an object of class "asL4".

Usage
asl4(biastype = symmetricBias(), normtype = NormType())

Arguments
biastype      a bias type of class BiasType
normtype     a norm type of class NormType

Value
Object of class "asL4"

Author(s)
Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

References
See Also
asl4-class, asMSE, asL1

Examples

```r
asl4()

## The function is currently defined as
function(biastype = symmetricBias(), normtype = NormType()){
  new("asl4", biastype = biastype, normtype = normtype) }
```

### asl4-class

Asymptotic mean power 4 error

Description

Class of asymptotic mean power 4 error.

Objects from the Class

Objects can be created by calls of the form `new("asl4", ...)`. More frequently they are created via the generating function `asl4`.

Slots

- `type` Object of class "character": “asymptotic mean square error”.
- `biastype` Object of class "BiasType": symmetric, one-sided or asymmetric
- `normtype` Object of class "NormType": norm in which a multivariate parameter is considered

Extends

Class "asGRisk", directly.
Class "asRiskwithBias", by class "asGRisk".
Class "asRisk", by class "asRiskwithBias".
Class "RiskType", by class "asGRisk".

Methods

No methods defined with class "asL4" in the signature.

Author(s)

Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

References

See Also

asGRisk-class, asMSE, asMSE-class, asL1-class, asL4

Examples

new("asMSE")

cniperCont Functions for Computation and Plot of Cniper Contamination and Cniper Points.

Description

These functions and their methods can be used to determine cniper contamination as well as cniper points. That is, under which (Dirac) contamination is the risk of one procedure larger than the risk of some other procedure.

Usage

cniperCont(IC1, IC2, data = NULL, ..., neighbor, risk, lower=setqdstrOption("DistrResolution"),
upper=1-getdistrOption("DistrResolution"), n = 101,
with.automatic.grid = TRUE, scaleX = FALSE, scaleX.fct,
setScale.inv, scaleY = FALSE, scaleY.fct = pnorm, scaleY.inv=qnorm,
setScaleN = 9, x.ticks = NULL, y.ticks = NULL, cex.pts = 1,
cex.pts.fun = NULL, col.pts = par("col"), pch.pts = 19,
cex.npts = 0.6, cex.npts.fun = NULL, col.npts = "red", pch.npts = 20,
jit.fac = 1, jit.tol = .Machine$double.eps, with.lab = FALSE,
lab.pts = NULL, lab.font = NULL, alpha.trsp = NA, which.lbs = NULL,
which.Order = NULL, which.nonlbs = NULL, attr.pre = FALSE,
return.Order = FALSE, withSubst = TRUE)

cniperPoint(L2Fam, neighbor, risk, lower, upper)

cniperPointPlot(L2Fam, data=asNULL, ..., neighbor, risk= asMSE(),
lower=setqdstrOption("DistrResolution"),
upper=1-getdistrOption("DistrResolution"), n = 101,
withMaxRisk = TRUE, with.automatic.grid = TRUE,
setScale = FALSE, scaleX.fct, scaleX.inv,
setScaleY = FALSE, scaleY.fct = pnorm, scaleY.inv=qnorm,
setScaleN = 9, x.ticks = NULL, y.ticks = NULL,
cex.pts = 1, cex.pts.fun = NULL, col.pts = par("col"),
pch.pts = 19,
cex.npts = 1, cex.npts.fun = NULL, col.npts = par("col"),
pch.npts = 19,
jit.fac = 1, jit.tol = .Machine$double.eps,
with.lab = FALSE,
cniperCont

lab.pts = NULL, lab.font = NULL, alpha.trsp = NA,
which.lbs = NULL, which.nonlbs = NULL,
which.Order = NULL, attr.pre = FALSE, return.Order = FALSE,
withSubst = TRUE, withMakeIC = FALSE)

Arguments

IC1  object of class IC
IC2  object of class IC
L2Fam object of class L2ParamFamily
neighbor object of class Neighborhood
risk  object of class RiskType
...  additional parameters (in particular to be passed on to plot).
data  data to be plotted in
lower, upper the lower and upper end points of the contamination interval (in prob-scale).
n  number of points between lower and upper
withMaxRisk logical; if TRUE, for risk comparison uses the maximal risk of the classically optimal IC \( \psi \) in all situations with contamination in Dirac points 'no larger' than the respective evaluation point and the optimally-robust IC \( \eta \) at its least favorable contamination situation ('over all real Dirac contamination points'). This is the default and was the behavior prior to package version 0.9). If FALSE it uses exactly the situation with Dirac contamination in the evaluation point for both ICs \( \psi \) and \( \eta \) which amounts to calling \text{cniperCont} with IC1=psi, IC2=eta.
with.automatic.grid logical; should a grid be plotted alongside with the ticks of the axes, automatically? If TRUE a respective call to grid in argument panel.first is ignored.
scaleX logical; shall X-axis be rescaled (by default according to the cdf of the underlying distribution)?
scaleY logical; shall Y-axis be rescaled (by default according to a probit scale)?
scaleX.fct an isotone, vectorized function mapping the domain of the IC(s) to \([0,1]\); if scaleX is TRUE and scaleX.fct is missing, the cdf of the underlying observation distribution.
scaleX.inv the inverse function to scale.fct, i.e., an isotone, vectorized function mapping \([0,1]\) to the domain of the IC(s) such that for any \(x\) in the domain, scaleX.inv(scaleX.fct(x))==x; if scaleX is TRUE and scaleX.inv is missing, the quantile function of the underlying observation distribution.
scaleY.fct an isotone, vectorized function mapping for each coordinate the range of the respective coordinate of the IC(s) to \([0,1]\); defaulting to the cdf of \(\mathcal{N}(0,1)\).
scaleY.inv an isotone, vectorized function mapping for each coordinate the range \([0,1]\) into the range of the respective coordinate of the IC(s); defaulting to the quantile function of \(\mathcal{N}(0,1)\).
scaleN integer; defaults to 9; on rescaled axes, number of x and y ticks if drawn automatically;
x.ticks numeric; defaults to NULL; (then ticks are chosen automatically); if non-NULL, user-given x-ticks (on original scale);
y.ticks numeric; defaults to NULL; (then ticks are chosen automatically); if non-NULL, user-given y-ticks (on original scale);
cex.pts size of the points of the second argument plotted (vectorized);
cex.pts.fun rescaling function for the size of the points to be plotted; either NULL (default), then \( \log(1+|x|) \) is used for the rescaling, or a function which is then used for the rescaling.
col.pts color of the points of the second argument plotted (vectorized);
pch.pts symbol of the points of the second argument plotted (vectorized);
col.npts color of the non-labelled points of the data argument plotted (vectorized);
pch.npts symbol of the non-labelled points of the data argument plotted (vectorized);
cex.npts size of the non-labelled points of the data argument plotted (vectorized);
cex.npts.fun rescaling function for the size of the non-labelled points to be plotted; either NULL (default), then \( \log(1+|x|) \) is used for each of the rescalings, or a function which is then used for each of the rescalings.
with.lab logical; shall labels be plotted to the observations?
lab.pts character or NULL; labels to be plotted to the observations; if NULL observation indices;
lab.font font to be used for labels
alpha.trsp alpha transparency to be added ex post to colors col.pch and col.lbd; if one- dim and NA all colors are left unchanged. Otherwise, with usual recycling rules alpha.trsp gets shorted/prolongated to length the data-symbols to be plotted. Coordinates of this vector alpha.trsp with NA are left unchanged, while for the remaining ones, the alpha channel in rgb space is set to the respective coordinate value of alpha.trsp. The non-NA entries must be integers in \([0,255]\) (0 invisible, 255 opaque).
jit.fac jittering factor used in case of a DiscreteDistribution for plotting points of the second argument in a jittered fashion.
jit.tol jittering tolerance used in case of a DiscreteDistribution for plotting points of the second argument in a jittered fashion.
which.lbs either an integer vector with the indices of the observations to be plotted into graph or NULL — then no observation is excluded
which.nonlbs indices of the observations which should be plotted but not labelled; either an integer vector with the indices of the observations to be plotted into graph or NULL — then all non-labelled observations are plotted.
which.Order we order the observations (descending) according to the norm given by normtype(object); then which.Order either is an integer vector with the indices of the ordered observations (remaining after a possible reduction by argument which.lbs) to be plotted into graph or NULL — then no (further) observation is excluded.
attr.pre logical; do graphical attributes for plotted data refer to indices prior (TRUE) or posterior to selection via arguments which.lbs, which.Order, which.nonlbs (FALSE)?
return.Order logical; if TRUE, an order vector is returned; more specifically, the order of
the (remaining) observations given by their original index is returned (remaining
means: after a possible reduction by argument which lbs, and ordering
is according to the norm given by normtype(object)); otherwise we return
invisible() as usual.

withSubst logical; if TRUE (default) pattern substitution for titles and lables is used; other-
wise no substitution is used.

withMakeIC logical; if TRUE the [p]IC is passed through makeIC before return.

Details

In case of cniperCont the difference between the risks of two ICs is plotted.

The function cniperPoint can be used to determine cniper points. That is, points such that the
optimally robust estimator has smaller minimax risk than the classical optimal estimator under
contamination with Dirac measures at the cniper points.

As such points might be difficult to find, we provide the function cniperPointPlot which can be
used to obtain a plot of the risk difference; in this function the usual arguments for plot can be
used. For arguments col, lwd, vectors can be used; then the first coordinate is taken for the curve,
the second one for the balancing line. For argument lty, a list can be used; its first component is
then taken for the curve, the second one for the balancing line.

If argument withSubst is TRUE, in all title and axis label arguments of cniperCont and cniperPointPlot,
the following patterns are substituted:

"%C" class of argument L2Fam (for cniperPointPlot)
"%A" deparsed argument L2Fam (for cniperPointPlot)
"%C1" class of argument IC1 (for cniperCont)
"%A1" deparsed argument IC1 (for cniperCont)
"%C2" class of argument IC2 (for cniperCont)
"%A2" deparsed argument IC2 (for cniperCont)
"%D" time/date-string when the plot was generated

For more details about cniper contamination and cniper points we refer to Section 3.5 of Kohl et

Value

The cniper point is returned by cniperPoint. In case of cniperPointPlot, we return an S3 object
of class c("plotInfo", "DiagnInfo"), i.e., a list containing the information needed to produce the
respective plot, which at a later stage could be used by different graphic engines (like, e.g. ggplot)
to produce the plot in a different framework. A more detailed description will follow in a subsequent
version.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>
References


Examples

```r
## cniper contamination
P <- PoisFamily(lambda = 4)
RobP1 <- InfRobModel(center = P, neighbor = ContNeighborhood(radius = 0.1))
IC1 <- optIC(model=RobP1, risk=asMSE())
RobP2 <- InfRobModel(center = P, neighbor = ContNeighborhood(radius = 1))
IC2 <- optIC(model=RobP2, risk=asMSE())
cniperCont(IC1 = IC1, IC2 = IC2,
neighbor = ContNeighborhood(radius = 0.5),
risk = asMSE(),
lower = 0, upper = 8, n = 101)

## cniper point plot
cniperPointPlot(P, neighbor = ContNeighborhood(radius = 0.5),
risk = asMSE(), lower = 0, upper = 10)

## Don't run to reduce check time on CRAN

## cniper point

cniperPoint(P, neighbor = ContNeighborhood(radius = 0.5),
risk = asMSE(), lower = 0, upper = 4)
cniperPoint(P, neighbor = ContNeighborhood(radius = 0.5),
risk = asMSE(), lower = 4, upper = 8)
```

CniperPointPlot

### Description

The wrapper `CniperPointPlot` (capital C!) takes most of arguments to the `cniperPointPlot` (lower case c!) function by default and gives a user possibility to run the function with low number of arguments.
Usage

\[
\text{CniperPointPlot}(\text{fam, ...}, \\
\quad \text{lower} = \text{getdistroOption("DistrResolution")}, \\
\quad \text{upper} = 1 - \text{getdistroOption("DistrResolution")}, \\
\quad \text{with.legend} = \text{TRUE}, \text{rescale} = \text{FALSE}, \text{withCall} = \text{TRUE})
\]

Arguments

- **fam**: object of class L2ParamFamily
- **...**: additional parameters (in particular to be passed on to `plot`
- **lower**: the lower end point of the contamination interval
- **upper**: the upper end point of the contamination interval
- **with.legend**: the flag for showing the legend of the plot
- **rescale**: the flag for rescaling the axes for better view of the plot
- **withCall**: the flag for the call output

Value

invisible(NULL)

Details

Calls `cniperPointPlot` with suitably chosen defaults; if `withCall == TRUE`, the call to `cniperPointPlot` is returned.

Examples

```r
L2fam <- NormLocationScaleFamily()
CniperPointPlot(fam=L2fam, main = "Normal location and scale", 
\quad lower = 0, upper = 2.5, withCall = FALSE)
```

Description

Plots 2-4 influence curves to the same model.

Details

S4-Method `comparePlot` for signature IC,IC has been enhanced compared to its original definition in RobAStBase so that if argument MBR is NA, it is filled automatically by a call to optIC which computes the MBR-IC on the fly. To this end, there is an additional argument `n.MBR` defaulting to 10000 to determine the number of evaluation points.
Examples

N0 <- NormLocationScaleFamily(mean=0, sd=1)
N0.Rob1 <- InfRobModel(center = N0,
neighbor = ContNeighborhood(radius = 0.5))

## Don't run to reduce check time on CRAN
## Not run:
IC1 <- optIC(model = N0, risk = asCov())
IC2 <- optIC(model = N0.Rob1, risk = asMSE())

comparePlot(IC1,IC2, withMBR=TRUE)

## End(Not run)

description

get.asGRisk.fct-methods to produce a function in r,s,b for computing a particular asGRisk

Usage

gasGRisk.fct(Risk)
## S4 method for signature 'asMSE'
gasGRisk.fct(Risk)
## S4 method for signature 'asL1'
gasGRisk.fct(Risk)
## S4 method for signature 'asL4'
gasGRisk.fct(Risk)

Arguments

Risk a risk of class "asGRisk"

Details

gasGRisk.fct is used internally in functions gasAsRisk and gasReq.

Value

gasGRisk.fct a function with arguments r (radius), s (square root of (trace of) variance), b bias to compute the respective risk of an IC with this bias and variance at the respective radius.
getAsRisk

Methods

get.asGRisk.fct signature(Risk = "asMSE"): method for asymptotic mean squared error.
get.asGRisk.fct signature(Risk = "asL1"): method for asymptotic mean absolute error.
get.asGRisk.fct signature(Risk = "asL4"): method for asymptotic mean power 4 error.

Author(s)

Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

getAsRisk

Generic Function for Computation of Asymptotic Risks

Description

Generic function for the computation of asymptotic risks. This function is rarely called directly. It is used by other functions.

Usage

getAsRisk(risk, L2deriv, neighbor, biastype, ...)

## S4 method for signature 'asMSE,UnivariateDistribution,Neighborhood,ANY'
getAsRisk(risk,
    L2deriv, neighbor, biastype, normtype = NULL, clip = NULL, cent = NULL,
    stand, trafo, ...)

## S4 method for signature 'asL1,UnivariateDistribution,Neighborhood,ANY'
getAsRisk(risk,
    L2deriv, neighbor, biastype, normtype = NULL, clip = NULL, cent = NULL,
    stand, trafo, ...)

## S4 method for signature 'asL4,UnivariateDistribution,Neighborhood,ANY'
getAsRisk(risk,
    L2deriv, neighbor, biastype, normtype = NULL, clip = NULL, cent = NULL,
    stand, trafo, ...)

getAsRisk(risk,
    L2deriv, neighbor, biastype, normtype = NULL, clip = NULL, cent = NULL,
    stand = NULL, trafo = NULL)

## S4 method for signature 'asBias,UnivariateDistribution,ContNeighborhood,ANY'
getAsRisk(risk,
    L2deriv, neighbor, biastype, normtype = NULL, clip = NULL, cent = NULL,
    stand = NULL, trafo = NULL)
## S4 method for signature
## 'asBias,UnivariateDistribution,ContNeighborhood,onesidedBias'
getAsRisk(
  risk, l2deriv, neighbor, biastype, normtype = NULL, clip = NULL, cent = NULL,
  stand = NULL, trafo, ...
)

## S4 method for signature
## 'asBias,UnivariateDistribution,ContNeighborhood,asymmetricBias'
getAsRisk(
  risk, l2deriv, neighbor, biastype, normtype = NULL, clip = NULL, cent = NULL,
  stand = NULL, trafo, ...
)

## S4 method for signature
## 'asBias,UnivariateDistribution,TotalVarNeighborhood,ANY'
getAsRisk(
  risk, l2deriv, neighbor, biastype, normtype = NULL, clip = NULL, cent = NULL,
  stand = NULL, trafo, ...
)

## S4 method for signature 'asBias,RealRandVariable,ContNeighborhood,ANY'
getAsRisk(
  risk, l2deriv, neighbor, biastype, normtype = NULL, clip = NULL, cent = NULL,
  stand = NULL, Distr, DistrSymm, L2derivSymm,
  L2derivDistrSymm, Finfo, trafo, z.start, A.start, maxiter, tol,
  warn, verbose = NULL, ...
)

## S4 method for signature 'asBias,RealRandVariable,TotalVarNeighborhood,ANY'
getAsRisk(
  risk, l2deriv, neighbor, biastype, normtype = NULL, clip = NULL, cent = NULL,
  stand = NULL, Distr, DistrSymm, L2derivSymm,
  L2derivDistrSymm, Finfo, trafo, z.start, A.start, maxiter, tol,
  warn, verbose = NULL, ...
)

## S4 method for signature 'asCov,UnivariateDistribution,ContNeighborhood,ANY'
getAsRisk(
  risk, l2deriv, neighbor, biastype, normtype = NULL, clip, cent, stand,
  trafo = NULL, ...
)

## S4 method for signature
## 'asCov,UnivariateDistribution,TotalVarNeighborhood,ANY'
getAsRisk(
  risk, l2deriv, neighbor, biastype, normtype = NULL, clip, cent, stand,
  trafo = NULL, ...
)

## S4 method for signature 'asCov,RealRandVariable,ContNeighborhood,ANY'
getAsRisk(risk,
  L2deriv, neighbor, biastype, normtype = NULL, clip = NULL, cent, stand,
  Distr, trafo = NULL, V.comp = matrix(TRUE, ncol = nrow(stand),
  nrow = nrow(stand)), w, ...)
getAsRisk

```r
## S4 method for signature
## 'trAsCov,UnivariateDistribution,UncondNeighborhood,ANY'
getAsRisk(
  risk, L2deriv, neighbor, biastype, normtype = NULL, clip, cent, stand,
  trafo = NULL, ...)

## S4 method for signature 'trAsCov,RealRandVariable,ContNeighborhood,ANY'
getAsRisk(risk,
  L2deriv, neighbor, biastype, normtype, clip, cent, stand, Distr,
  trafo = NULL, V.comp = matrix(TRUE, ncol = nrow(stand),
  nrow = nrow(stand)), w, ...)

## S4 method for signature
## 'asAnscombe,UnivariateDistribution,UncondNeighborhood,ANY'
getAsRisk(
  risk, L2deriv, neighbor, biastype, normtype = NULL, clip, cent, stand,
  trafo = NULL, FI, ...)

## S4 method for signature 'asAnscombe,RealRandVariable,ContNeighborhood,ANY'
getAsRisk(risk,
  L2deriv, neighbor, biastype, normtype, clip, cent, stand, Distr, trafo = NULL,
  V.comp = matrix(TRUE, ncol = nrow(stand), nrow = nrow(stand)),
  FI, w, ...)

## S4 method for signature
## 'asUnOvShoot,UnivariateDistribution,UncondNeighborhood,ANY'
getAsRisk(
  risk, L2deriv, neighbor, biastype, normtype = NULL, clip, cent, stand,
  trafo, ...)

## S4 method for signature
## 'asSemivar,UnivariateDistribution,Neighborhood,onesidedBias'
getAsRisk(
  risk, L2deriv, neighbor, biastype, normtype = NULL, clip, cent, stand,
  trafo, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>risk</td>
<td>object of class &quot;asRisk&quot;.</td>
</tr>
<tr>
<td>L2deriv</td>
<td>L2-derivative of some L2-differentiable family of probability distributions.</td>
</tr>
<tr>
<td>neighbor</td>
<td>object of class &quot;Neighborhood&quot;.</td>
</tr>
<tr>
<td>biastype</td>
<td>object of class &quot;ANY&quot;.</td>
</tr>
<tr>
<td>...</td>
<td>additional parameters; often used to enable flexible calls.</td>
</tr>
<tr>
<td>clip</td>
<td>optimal clipping bound.</td>
</tr>
<tr>
<td>cent</td>
<td>optimal centering constant.</td>
</tr>
<tr>
<td>stand</td>
<td>standardizing matrix.</td>
</tr>
</tbody>
</table>
```
getAsRisk

Finfo matrix: the Fisher Information of the parameter.
trafo matrix: transformation of the parameter.
Distr object of class "Distribution".
DistrSymm object of class "DistributionSymmetry".
L2derivSymm object of class "FunSymmList".
L2derivDistrSymm object of class "DistrSymmList".
z.start initial value for the centering constant.
A.start initial value for the standardizing matrix.
maxiter the maximum number of iterations
tol the desired accuracy (convergence tolerance).
warn logical: print warnings.
normtype object of class "NormType".
V.comp matrix: indication which components of the standardizing matrix have to be computed.
w object of class RobWeight; current weight
FI trace of the respective Fisher Information
verbose logical: if TRUE some diagnostics are printed out.

Details
This function is rarely called directly. It is used by other functions/methods.

Value
The asymptotic risk is computed.

Methods

risk = "asMSE", L2deriv = "UnivariateDistribution", neighbor = "Neighborhood", biastype = "ANY": computes asymptotic mean square error in methods for function getInfRobIC.

risk = "asL1", L2deriv = "UnivariateDistribution", neighbor = "Neighborhood", biastype = "ANY": computes asymptotic mean absolute error in methods for function getInfRobIC.

risk = "asL4", L2deriv = "UnivariateDistribution", neighbor = "Neighborhood", biastype = "ANY": computes asymptotic mean power 4 error in methods for function getInfRobIC.

risk = "asMSE", L2deriv = "EuclRandVariable", neighbor = "Neighborhood", biastype = "ANY": computes asymptotic mean square error in methods for function getInfRobIC.

risk = "asBias", L2deriv = "UnivariateDistribution", neighbor = "ContNeighborhood", biastype = "ANY": computes standardized asymptotic bias in methods for function getInfRobIC.

risk = "asBias", L2deriv = "UnivariateDistribution", neighbor = "ContNeighborhood", biastype = "onesidedBias": computes standardized asymptotic bias in methods for function getInfRobIC.

risk = "asBias", L2deriv = "UnivariateDistribution", neighbor = "ContNeighborhood", biastype = "asymmetricBias": computes standardized asymptotic bias in methods for function getInfRobIC.
getAsRisk

risk = "asBias", L2deriv = "UnivariateDistribution", neighbor = "TotalVarNeighborhood", biastype = "ANY": computes standardized asymptotic bias in methods for function getInfRobIC.

risk = "asBias", L2deriv = "RealRandVariable", neighbor = "ContNeighborhood", biastype = "ANY": computes standardized asymptotic bias in methods for function getInfRobIC.

risk = "asCov", L2deriv = "UnivariateDistribution", neighbor = "ContNeighborhood", biastype = "ANY": computes asymptotic covariance in methods for function getInfRobIC.

risk = "asCov", L2deriv = "RealRandVariable", neighbor = "ContNeighborhood", biastype = "ANY": computes asymptotic covariance in methods for function getInfRobIC.

risk = "asCov", L2deriv = "RealRandVariable", neighbor = "ContNeighborhood", biastype = "ANY": computes asymptotic covariance in methods for function getInfRobIC.

risk = "asAnscombe", L2deriv = "UnivariateDistribution", neighbor = "UncondNeighborhood", biastype = "ANY": computes the ARE in the ideal model in methods for function getInfRobIC.

risk = "asAnscombe", L2deriv = "RealRandVariable", neighbor = "ContNeighborhood", biastype = "ANY": computes the ARE in the ideal model in methods for function getInfRobIC.

risk = "asUnOvShoot", L2deriv = "UnivariateDistribution", neighbor = "UncondNeighborhood", biastype = "ANY": computes asymptotic under-/overshoot risk in methods for function getInfRobIC.

risk = "asSemivar", L2deriv = "UnivariateDistribution", neighbor = "Neighborhood", biastype = "onesidedBias": computes asymptotic semivariance in methods for function getInfRobIC.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

asRisk-class
getBiasIC

Generic function for the computation of the asymptotic bias for an IC

Description

Generic function for the computation of the asymptotic bias for an IC.

Usage

getBiasIC(IC, neighbor, ...)

## S4 method for signature 'HampIC,UncondNeighborhood'
getBiasIC(IC, neighbor, L2Fam, ..., withCheck = TRUE)

Arguments

IC object of class "InfluenceCurve"
neighbor object of class "Neighborhood".
L2Fam object of class "L2ParamFamily".
... additional parameters
withCheck logical: should a call to checkIC be done to check accuracy (defaults to TRUE; ignored if nothing is computed but simply a slot is read out).

Details

This function is rarely called directly. It is used by other functions/methods.

Value

The bias of the IC is computed.

Methods

IC = "HampIC", neighbor = "UncondNeighborhood" reads off the as. bias from the risks-slot of the IC.

IC = "TotalVarIC", neighbor = "UncondNeighborhood" reads off the as. bias from the risks-slot of the IC, resp. if this is NULL from the corresponding Lagrange Multipliers.

Note

This generic function is still under construction.

Author(s)

Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>
References


See Also

genericRiskIC-methods, InfRobModel-class

getFixClip

Generic Function for the Computation of the Optimal Clipping Bound

description

Generic function for the computation of the optimal clipping bound in case of robust models with fixed neighborhoods. This function is rarely called directly. It is used to compute optimally robust ICs.

Usage

genericRiskIC_nodes(clip, distr, risk, neighbor, ...)

## S4 method for signature 'genericRiskIC' object

genericRiskIC_nodes(clip, distr, risk, neighbor)

## S4 method for signature 'genericRiskIC' object

genericRiskIC_nodes(clip, distr, risk, neighbor)

Arguments

clip positive real: clipping bound
Distr object of class "Distribution".
risk object of class "RiskType".
neighbor object of class "Neighborhood".
... additional parameters.

Value

The optimal clipping bound is computed.
Methods

- clip = "numeric", Distr = "Norm", risk = "fiUnOvShoot", neighbor = "ContNeighborhood"
  optimal clipping bound for finite-sample under-/overshoot risk.
- clip = "numeric", Distr = "Norm", risk = "fiUnOvShoot", neighbor = "TotalVarNeighborhood"
  optimal clipping bound for finite-sample under-/overshoot risk.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

ContIC-class, TotalVarIC-class

getFixRobIC Generic Function for the Computation of Optimally Robust ICs

Description

Generic function for the computation of optimally robust ICs in case of robust models with fixed neighborhoods. This function is rarely called directly.

Usage

getFixRobIC(Distr, risk, neighbor, ...)

## S4 method for signature 'Norm,fiUnOvShoot,UncondNeighborhood'
getFixRobIC(Distr, risk, neighbor,
  sampleSize, upper, lower, maxiter, tol, warn, Algo, cont)

Arguments

- Distr: object of class "Distribution".
- risk: object of class "RiskType".
- neighbor: object of class "Neighborhood".
- ...: additional parameters.
- sampleSize: integer: sample size.
- upper: upper bound for the optimal clipping bound.
getFixRobIC

lower      lower bound for the optimal clipping bound.
maxiter    the maximum number of iterations.
tol        the desired accuracy (convergence tolerance).
warn       logical: print warnings.
Algo       "A" or "B".
cont       "left" or "right".

Details

Computation of the optimally robust IC in sense of Huber (1968) which is also treated in Kohl (2005). The Algorithm used to compute the exact finite sample risk is introduced and explained in Kohl (2005). It is based on FFT.

Value

The optimally robust IC is computed.

Methods

Distr = "Norm", risk = "fiUnOvShoot", neighbor = "UncondNeighborhood" computes the optimally robust influence curve for one-dimensional normal location and finite-sample under-/overshoot risk.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

FixRobModel-class
**getIneffDiff**  
*Generic Function for the Computation of Inefficiency Differences*

**Description**

Generic function for the computation of inefficiency differences. This function is rarely called directly. It is used to compute the radius minimax IC and the least favorable radius.

**Usage**

```r
getIneffDiff(radius, L2Fam, neighbor, risk, ...)
```

```r
# S4 method for signature 'numeric,L2ParamFamily,Uncond Neighborhood,asMSE'
getIneffDiff(
  radius, L2Fam, neighbor, risk, loRad, upRad, loRisk, upRisk,
  z.start = NULL, A.start = NULL, upper.b = NULL, lower.b = NULL,
  OptOrIter = "iterate", MaxIter, eps, warn, loNorm = NULL, upNorm = NULL,
  verbose = NULL, ..., withRetIneff = FALSE)
```

**Arguments**

- `radius`: neighborhood radius.
- `L2Fam`: L2-differentiable family of probability measures.
- `neighbor`: object of class "Neighborhood".
- `risk`: object of class "RiskType".
- `loRad`: the lower end point of the interval to be searched.
- `upRad`: the upper end point of the interval to be searched.
- `loRisk`: the risk at the lower end point of the interval.
- `upRisk`: the risk at the upper end point of the interval.
- `z.start`: initial value for the centering constant.
- `A.start`: initial value for the standardizing matrix.
- `upper.b`: upper bound for the optimal clipping bound.
- `lower.b`: lower bound for the optimal clipping bound.
- `OptOrIter`: character; which method to be used for determining Lagrange multipliers A and a: if (partially) matched to "optimize", getLagrangeMultByOptim is used; otherwise: by default, or if matched to "iterate" or to "doubleiterate", getLagrangeMultByIter is used. More specifically, when using getLagrangeMultByIter, and if argument risk is of class "asGRisk", by default and if matched to "iterate" we use only one (inner) iteration, if matched to "doubleiterate" we use up to MaxIter (inner) iterations.
- `MaxIter`: the maximum number of iterations
- `eps`: the desired accuracy (convergence tolerance).
getIneffDiff

warn logical: print warnings.

loNorm object of class "NormType"; used in selfstandardization to evaluate the bias of
the current IC in the norm of the lower bound

upNorm object of class "NormType"; used in selfstandardization to evaluate the bias of
the current IC in the norm of the upper bound

verbose logical: if TRUE, some messages are printed

... further arguments to be passed on to getInfRobIC

withRetIneff logical: if TRUE, getIneffDiff returns the vector of lower and upper ineffi-
ciency (components named "lo" and "up"), otherwise (default) the difference. The latter was used in radiusMinimaxIC up to version 0.8 for a call to uniroot
directly. In order to speed up things (i.e., not to call the expensive getInfRobIC
once again at the zero, up to version 0.8 we had some awkward assign-sys.frame
construction to modify the caller writing the upper inefficiency already com-
puted to the caller environment; having capsulated this into try from version
0.9 on, this became even more awkward, so from version 0.9 onwards, we in-
stead use the TRUE-alternative when calling it from radiusMinimaxIC.

Value

The inefficiency difference between the left and the right margin of a given radius interval is com-
puted.

Methods

radius = "numeric", L2Fam = "L2ParamFamily", neighbor = "UncondNeighborhood", risk = "asMSE":
computes difference of asymptotic MSE–inefficiency for the boundaries of a given radius in-
terval.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References

Methods and Applications, 17(1) 13-40.

Appeared as discussion paper Nr. 81. SFB 373 (Quantification and Simulation of Economic Pro-
cesses), Humboldt University, Berlin; also available under www.uni-bayreuth.de/departments/
math/org/matheW/RIEDER/pubs/RR.pdf

sertation.

See Also

radiusMinimaxIC, leastFavorableRadius
getInfCent

Generic Function for the Computation of the Optimal Centering Constant/Lower Clipping Bound

Description

Generic function for the computation of the optimal centering constant (contamination neighborhoods) respectively, of the optimal lower clipping bound (total variation neighborhood). This function is rarely called directly. It is used to compute optimally robust ICs.

Usage

getInfCent(L2deriv, neighbor, biastype, ...)

## S4 method for signature 'UnivariateDistribution,ContNeighborhood,BiasType'
getInfCent(L2deriv,
neighbor, biastype, clip, cent, tol.z, symm, trafo)

## S4 method for signature
## 'UnivariateDistribution,TotalVarNeighborhood,BiasType'
getInfCent(L2deriv,
neighbor, biastype, clip, cent, tol.z, symm, trafo)

## S4 method for signature 'RealRandVariable,ContNeighborhood,BiasType'
getInfCent(L2deriv,
neighbor, biastype, Distr, z.comp, w, tol.z = .Machine$double.eps*.5)

## S4 method for signature 'RealRandVariable,TotalVarNeighborhood,BiasType'
getInfCent(L2deriv,
neighbor, biastype, Distr, z.comp, w, tol.z = .Machine$double.eps*.5)

## S4 method for signature
## 'UnivariateDistribution,ContNeighborhood,onesidedBias'
getInfCent(L2deriv,
neighbor, biastype, clip, cent, tol.z, symm, trafo)

## S4 method for signature
## 'UnivariateDistribution,ContNeighborhood,asymmetricBias'
getInfCent(L2deriv,
neighbor, biastype, clip, cent, tol.z, symm, trafo)

Arguments

L2deriv L2-derivative of some L2-differentiable family of probability measures.
neighbor object of class "Neighborhood".
biastype object of class "BiasType"
additional parameters.
clip optimal clipping bound.
cent optimal centering constant.
tol.z the desired accuracy (convergence tolerance).
symm logical: indicating symmetry of L2deriv.
trafo matrix: transformation of the parameter.
Distr object of class Distribution.
z.comp logical vector: indication which components of the centering constant have to be computed.
w object of class RobWeight; current weight

Value
The optimal centering constant is computed.

Methods
L2deriv = "UnivariateDistribution", neighbor = "ContNeighborhood", biastype = "BiasType"
computation of optimal centering constant for symmetric bias.
L2deriv = "UnivariateDistribution", neighbor = "TotalVarNeighborhood", biastype = "BiasType"
computation of optimal lower clipping bound for symmetric bias.
L2deriv = "RealRandVariable", neighbor = "TotalVarNeighborhood", biastype = "BiasType"
computation of optimal centering constant for symmetric bias.
L2deriv = "RealRandVariable", neighbor = "ContNeighborhood", biastype = "BiasType"
computation of optimal centering constant for symmetric bias.
L2deriv = "UnivariateDistribution", neighbor = "ContNeighborhood", biastype = "onesidedBias"
computation of optimal centering constant for onesided bias.
L2deriv = "UnivariateDistribution", neighbor = "ContNeighborhood", biastype = "asymmetricBias"
computation of optimal centering constant for asymmetric bias.

Author(s)
Matthias Kohl <Matthias.Kohl@stamats.de>, Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

References

See Also
ContIC-class, TotalVarIC-class
Generic Function for the Computation of the Optimal Clipping Bound

Description

Generic function for the computation of the optimal clipping bound in case of infinitesimal robust models. This function is rarely called directly. It is used to compute optimally robust ICs.

Usage

getInfClip(clip, L2deriv, risk, neighbor, ...)

## S4 method for signature
## 'numeric,UnivariateDistribution,asMSE,ContNeighborhood'
getInfClip(
  clip, L2deriv, risk, neighbor, biastype, cent, symm, trafo)

## S4 method for signature
## 'numeric,UnivariateDistribution,asMSE,TotalVarNeighborhood'
getInfClip(
  clip, L2deriv, risk, neighbor, biastype, cent, symm, trafo)

## S4 method for signature
## 'numeric,UnivariateDistribution,asL1,ContNeighborhood'
getInfClip(
  clip, L2deriv, risk, neighbor, biastype, cent, symm, trafo)

## S4 method for signature
## 'numeric,UnivariateDistribution,asL1,TotalVarNeighborhood'
getInfClip(
  clip, L2deriv, risk, neighbor, biastype, cent, symm, trafo)

## S4 method for signature
## 'numeric,UnivariateDistribution,asL4,ContNeighborhood'
getInfClip(
  clip, L2deriv, risk, neighbor, biastype, cent, symm, trafo)

## S4 method for signature
## 'numeric,UnivariateDistribution,asL4,TotalVarNeighborhood'
getInfClip(
  clip, L2deriv, risk, neighbor, biastype, cent, symm, trafo)

## S4 method for signature 'numeric, EuclRandVariable, asMSE, UncondNeighborhood'
getInfClip(
  clip, L2deriv, risk, neighbor, biastype, Distr, stand, cent, trafo)

## S4 method for signature
getInfClip

## Arguments

- **clip**
  - positive real: clipping bound

- **L2deriv**
  - L2-derivative of some L2-differentiable family of probability measures.

- **risk**
  - object of class "RiskType".

- **neighbor**
  - object of class "Neighborhood".

- **...**
  - additional parameters.

- **biastype**
  - object of class "BiasType"

- **cent**
  - optimal centering constant.

- **stand**
  - standardizing matrix.

- **Distr**
  - object of class "Distribution".

- **symm**
  - logical: indicating symmetry of L2deriv.

- **trafo**
  - matrix: transformation of the parameter.

### Value

The optimal clipping bound is computed.

### Methods

- **clip = "numeric", L2deriv = "UnivariateDistribution", risk = "asMSE", neighbor = "ContNeighborhood"**
  - optimal clipping bound for asymptotic mean square error.

- **clip = "numeric", L2deriv = "UnivariateDistribution", risk = "asMSE", neighbor = "TotalVarNeighborhood"**
  - optimal clipping bound for asymptotic mean square error.

- **clip = "numeric", L2deriv = "EuclRandVariable", risk = "asMSE", neighbor = "UncondNeighborhood"**
  - optimal clipping bound for asymptotic mean square error.

- **clip = "numeric", L2deriv = "UnivariateDistribution", risk = "asL1", neighbor = "ContNeighborhood"**
  - optimal clipping bound for asymptotic mean absolute error.

- **clip = "numeric", L2deriv = "UnivariateDistribution", risk = "asL1", neighbor = "TotalVarNeighborhood"**
  - optimal clipping bound for asymptotic mean absolute error.

- **clip = "numeric", L2deriv = "UnivariateDistribution", risk = "asL4", neighbor = "ContNeighborhood"**
  - optimal clipping bound for asymptotic mean power 4 error.

- **clip = "numeric", L2deriv = "UnivariateDistribution", risk = "asL4", neighbor = "TotalVarNeighborhood"**
  - optimal clipping bound for asymptotic mean power 4 error.
getInfGamma

`clip = "numeric", L2deriv = "UnivariateDistribution", risk = "asUnOvShoot", neighbor = "UncondNeighborhood"
optimal clipping bound for asymptotic under-/overshoot risk.

`clip = "numeric", L2deriv = "UnivariateDistribution", risk = "asSemivar", neighbor = "ContNeighborhood"
optimal clipping bound for asymptotic semivariance.

Author(s)
Matthias Kohl <Matthias.Kohl@stamats.de>, Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

References

See Also
*ContIC-class, TotalVarIC-class*

---

getInfGamma  
Generic Function for the Computation of the Optimal Clipping Bound

Description
Generic function for the computation of the optimal clipping bound. This function is rarely called directly. It is called by getInfClip to compute optimally robust ICs.

Usage
```r
getInfGamma(L2deriv, risk, neighbor, biastype, ...)
```

```
## S4 method for signature
## 'UnivariateDistribution,asGRisk,ContNeighborhood,BiasType'
getInfGamma(L2deriv, 
            risk, neighbor, biastype, cent, clip)
```

```
## S4 method for signature
## 'UnivariateDistribution,asGRisk,TotalVarNeighborhood,BiasType'
getInfGamma(L2deriv, 
            risk, neighbor, biastype, cent, clip)
```
getInfGamma

## S4 method for signature 'RealRandVariable,asMSE,ContNeighborhood,BiasType'
getInfGamma(L2deriv,  
  risk, neighbor, biastype, Distr, stand, cent, clip, power = 1L)

## S4 method for signature
## 'RealRandVariable,asMSE,TotalVarNeighborhood,BiasType'
getInfGamma(L2deriv,  
  risk, neighbor, biastype, Distr, stand, cent, clip, power = 1L)

## S4 method for signature
## 'UnivariateDistribution,asUnOvShoot,ContNeighborhood,BiasType'
getInfGamma(L2deriv,  
  risk, neighbor, biastype, cent, clip)

## S4 method for signature
## 'UnivariateDistribution,asMSE,ContNeighborhood,onesidedBias'
getInfGamma(L2deriv,  
  risk, neighbor, biastype, cent, clip)

## S4 method for signature
## 'UnivariateDistribution,asMSE,ContNeighborhood,asymmetricBias'
getInfGamma(L2deriv,  
  risk, neighbor, biastype, cent, clip)

### Arguments

- **L2deriv**: L2-derivative of some L2-differentiable family of probability measures.
- **risk**: object of class "RiskType".
- **neighbor**: object of class "Neighborhood".
- **biastype**: object of class "BiasType"
- **...**: additional parameters
- **cent**: optimal centering constant.
- **clip**: optimal clipping bound.
- **stand**: standardizing matrix.
- **Distr**: object of class "Distribution".
- **power**: exponent for the integrand; by default 1, but may also be 2, for optimization in getLagrangeMultByOptim.

### Details

The function is used in case of asymptotic G-risks; confer Ruckdeschel and Rieder (2004).

### Methods

- **L2deriv = "UnivariateDistribution", risk = "asGRisk", neighbor = "ContNeighborhood", biastype = "BiasType"**
  used by getInfClip for symmetric bias.
L2deriv = "UnivariateDistribution", risk = "asGRisk", neighbor = "TotalVarNeighborhood", biastype = "BiasType"
used by getInfClip for symmetric bias.

L2deriv = "RealRandVariable", risk = "asMSE", neighbor = "ContNeighborhood", biastype = "BiasType"
used by getInfClip for symmetric bias.

L2deriv = "RealRandVariable", risk = "asMSE", neighbor = "TotalVarNeighborhood", biastype = "BiasType"
used by getInfClip for symmetric bias.

L2deriv = "UnivariateDistribution", risk = "asUnOvShoot", neighbor = "ContNeighborhood", biastype = "BiasType"
used by getInfClip for symmetric bias.

L2deriv = "UnivariateDistribution", risk = "asMSE", neighbor = "ContNeighborhood", biastype = "onesidedBias"
used by getInfClip for onesided bias.

L2deriv = "UnivariateDistribution", risk = "asMSE", neighbor = "ContNeighborhood", biastype = "asymmetricBias"
used by getInfClip for asymmetric bias.

Author(s)
Matthias Kohl <Matthias.Kohl@stamats.de>, Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

References

See Also
asGRisk-class, asMSE-class, asUnOvShoot-class, ContIC-class, TotalVarIC-class

getInflm

Functions to determine Lagrange multipliers

description

Functions to determine Lagrange multipliers A and a in a Hampel problem or in a(n) (inner) loop in a MSE problem; can be done either by optimization or by fixed point iteration. These functions are rarely called directly.
Usage

getLagrangemultbyiter(b, L2deriv, risk, trafo,
    neighbor, biastype, normtype, Distr,
    a.start, z.start, A.start, w.start, std, z.comp,
    A.comp, maxiter, tol, verbose = NULL,
    warnit = TRUE)
getLagrangemultbyoptim(b, L2deriv, risk, FI, trafo,
    neighbor, biastype, normtype, Distr,
    a.start, z.start, A.start, w.start, std, z.comp,
    A.comp, maxiter, tol, verbose = NULL,...)

Arguments

b numeric; (> bmin; clipping bound for which the Lagrange multipliers are searched
L2deriv L2-derivative of some L2-differentiable family of probability measures.
risk object of class "RiskType".
FI matrix: Fisher information.
trafo matrix: transformation of the parameter.
neighbor object of class "Neighborhood".
biastype object of class "BiasType" — the bias type with we work.
normtype object of class "NormType" — the norm type with we work.
Distr object of class "Distribution".
a.start initial value for the centering constant (in p-space).
z.start initial value for the centering constant (in k-space).
A.start initial value for the standardizing matrix.
w.start initial value for the weight function.
std matrix of (or which may coerced to) class PosSemDefSymmMatrix for use of
different (standardizing) norm.
z.comp logical vector: indication which components of the centering constant have to
be computed.
A.comp matrix: indication which components of the standardizing matrix have to be
computed.
maxiter the maximum number of iterations.
tol the desired accuracy (convergence tolerance).
verbose logical: if TRUE, some messages are printed.
warnit logical: if TRUE warning is issued if maximal number of iterations is reached.
... additional parameters for optim.
Value

a list with items

A  
  Lagrange multiplier \( A \) (standardizing matrix)

a  
  Lagrange multiplier \( a \) (centering in \( p \)-space)

z  
  Lagrange multiplier \( z \) (centering in \( k \)-space)

w  
  weight function involving Lagrange multipliers

biastype  
  (possibly modified) bias type \( \text{biastype} \) from argument

normtype  
  (possibly modified) norm type \( \text{normtype} \) from argument

normtype.old  
  (possibly modified) norm type \( \text{normtype} \) before last (internal) update

risk  
  (possibly \([\text{norm-}]\)modified) risk \( \text{risk} \) from argument

std  
  (possibly modified) argument \( \text{std} \)

iter  
  number of iterations needed

prec  
  precision achieved

b  
  used clipping height \( b \)

call  
  call with which either \text{getLagrangeMultByIter} \text{ or getLagrangeMultByOptim} \text{ was called}

Author(s)

Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

References


See Also

InfRobModel-class
Generic Function for the Computation of the Optimal Radius for Given Clipping Bound

Description

The usual robust optimality problem for given asGRisk searches the optimal clipping height $b$ of a Hampel-type IC to given radius of the neighborhood. Instead, again for given asGRisk and for given Hampel-Type IC with given clipping height $b$ we may determine the radius of the neighborhood for which it is optimal in the sense of the first sentence. This radius is determined by getInfRad. This function is rarely called directly. It is used withing getRadius.

Usage

getInfRad(clip, L2deriv, risk, neighbor, ...)

## S4 method for signature
## 'numeric,UnivariateDistribution,asMSE,ContNeighborhood'
getInfRad(  
  clip, L2deriv, risk, neighbor, biastype, cent, symm, trafo)

## S4 method for signature
## 'numeric,UnivariateDistribution,asMSE,TotalVarNeighborhood'
getInfRad(  
  clip, L2deriv, risk, neighbor, biastype, cent, symm, trafo)

## S4 method for signature
## 'numeric,UnivariateDistribution,asL1,ContNeighborhood'
getInfRad(  
  clip, L2deriv, risk, neighbor, biastype, cent, symm, trafo)

## S4 method for signature
## 'numeric,UnivariateDistribution,asL1,TotalVarNeighborhood'
getInfRad(  
  clip, L2deriv, risk, neighbor, biastype, cent, symm, trafo)

## S4 method for signature
## 'numeric,UnivariateDistribution,asL4,ContNeighborhood'
getInfRad(  
  clip, L2deriv, risk, neighbor, biastype, cent, symm, trafo)

## S4 method for signature
## 'numeric,UnivariateDistribution,asL4,TotalVarNeighborhood'
getInfRad(  
  clip, L2deriv, risk, neighbor, biastype, cent, symm, trafo)

## S4 method for signature 'numeric,EuclRandVariable,asMSE,UncondNeighborhood'
getInfRad(
  clip, L2deriv, risk, neighbor, biastype, Distr, stand, cent, trafo)

## S4 method for signature
## 'numeric,UnivariateDistribution,asUnovShoot,UncondNeighborhood'
getInfRad(
  clip, L2deriv, risk, neighbor, biastype, cent, symm, trafo)

## S4 method for signature
## 'numeric,UnivariateDistribution,asSemivar,ContNeighborhood'
getInfRad(
  clip, L2deriv, risk, neighbor, biastype, cent, symm, trafo)

### Arguments

- **clip**: positive real: clipping bound
- **L2deriv**: $L^2$-derivative of some $L^2$-differentiable family of probability measures.
- **risk**: object of class "RiskType".
- **neighbor**: object of class "Neighborhood".
- ... additional parameters.
- **biastype**: object of class "BiasType"
- **cent**: optimal centering constant.
- **stand**: standardizing matrix.
- **Distr**: object of class "Distribution".
- **symm**: logical: indicating symmetry of L2deriv.
- **trafo**: matrix: transformation of the parameter.

### Value

The optimal clipping bound is computed.

### Methods

- **clip = "numeric", L2deriv = "UnivariateDistribution", risk = "asMSE", neighbor = "ContNeighborhood"**
optimal clipping bound for asymptotic mean square error.

- **clip = "numeric", L2deriv = "UnivariateDistribution", risk = "asMSE", neighbor = "TotalVarNeighborhood"**
optimal clipping bound for asymptotic mean square error.

- **clip = "numeric", L2deriv = "EuclRandVariable", risk = "asMSE", neighbor = "UncondNeighborhood"**
optimal clipping bound for asymptotic mean square error.

- **clip = "numeric", L2deriv = "UnivariateDistribution", risk = "asL1", neighbor = "ContNeighborhood"**
optimal clipping bound for asymptotic mean absolute error.

- **clip = "numeric", L2deriv = "UnivariateDistribution", risk = "asL1", neighbor = "TotalVarNeighborhood"**
optimal clipping bound for asymptotic mean absolute error.

- **clip = "numeric", L2deriv = "UnivariateDistribution", risk = "asL4", neighbor = "ContNeighborhood"**
optimal clipping bound for asymptotic mean power 4 error.


```r
clip = "numeric", L2deriv = "UnivariateDistribution", risk = "asL4", neighbor = "TotalVarNeighborhood"
optimal clipping bound for asymptotic mean power 4 error.

clip = "numeric", L2deriv = "UnivariateDistribution", risk = "asUnOvShoot", neighbor = "UncondNeighborhood"
optimal clipping bound for asymptotic under-/overshoot risk.

clip = "numeric", L2deriv = "UnivariateDistribution", risk = "asSemivar", neighbor = "ContNeighborhood"
optimal clipping bound for asymptotic semivariance.
```

Author(s)

Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

References


See Also

ContIC-class, TotalVarIC-class

---

## getInfRobIC

Generic Function for the Computation of Optimally Robust ICs

### Description

Generic function for the computation of optimally robust ICs in case of infinitesimal robust models. This function is rarely called directly.

### Usage

```r
getInfRobIC(L2deriv, risk, neighbor, ...)
```

```r
## S4 method for signature 'UnivariateDistribution,asCov,ContNeighborhood'
getInfRobIC(L2deriv, 
    risk, neighbor, Finfo, trafo, verbose = NULL)
```

```r
## S4 method for signature 'UnivariateDistribution,asCov,TotalVarNeighborhood'
getInfRobIC(L2deriv, 
    risk, neighbor, Finfo, trafo, verbose = NULL)
```
## S4 method for signature 'RealRandVariable,asCov,UncondNeighborhood'

getInfRobIC(L2deriv, risk, neighbor, Distr, Finfo, trafo, QuadForm = diag(nrow(trafo)),
        verbose = NULL)

## S4 method for signature 'UnivariateDistribution,asBias,UncondNeighborhood'

getInfRobIC(L2deriv, risk, neighbor, symm, trafo, maxiter, tol, warn, Finfo,
        verbose = NULL, ...)

## S4 method for signature 'RealRandVariable,asBias,UncondNeighborhood'

getInfRobIC(L2deriv, risk,
        neighbor, Distr, DistrSymm, L2derivSymm, L2derivDistrSymm, z.start, A.start, Finfo, trafo,
        maxiter, tol, warn, verbose = NULL, ...)

## S4 method for signature 'UnivariateDistribution,asHampel,UncondNeighborhood'

getInfRobIC(L2deriv, risk, neighbor, symm, Finfo, trafo, upper = NULL,
        lower=NULL, maxiter, tol, warn, noLow = FALSE,
        verbose = NULL, checkBounds = TRUE, ...)

## S4 method for signature 'RealRandVariable,asHampel,UncondNeighborhood'

getInfRobIC(L2deriv, risk,
        neighbor, Distr, DistrSymm, L2derivSymm, L2derivDistrSymm, Finfo, trafo, onesetLM = FALSE,
        z.start, A.start, upper = NULL, lower=NULL, OptOrIter = "iterate", maxiter, tol, warn,
        verbose = NULL, checkBounds = TRUE, ..., .withEvalAsVar = TRUE)

## S4 method for signature 'UnivariateDistribution,asAnscombe,UncondNeighborhood'

getInfRobIC(
        L2deriv, risk, neighbor, symm, Finfo, trafo, upper = NULL,
        lower=NULL, maxiter, tol, warn, noLow = FALSE,
        verbose = NULL, checkBounds = TRUE, ...)

## S4 method for signature 'RealRandVariable,asAnscombe,UncondNeighborhood'

getInfRobIC(L2deriv, risk, neighbor, Distr, DistrSymm, L2derivSymm, L2derivDistrSymm, Finfo, trafo, onesetLM = FALSE,
        z.start, A.start, upper = NULL, lower=NULL, OptOrIter = "iterate", maxiter, tol, warn,
        verbose = NULL, checkBounds = TRUE, ...)

## S4 method for signature 'UnivariateDistribution,asGRisk,UncondNeighborhood'

getInfRobIC(L2deriv, risk, neighbor, Distr, DistrSymm, L2derivSymm, L2derivDistrSymm, Finfo, trafo, onesetLM = FALSE,
        z.start, A.start, upper = NULL, lower=NULL, OptOrIter = "iterate", maxiter, tol, warn,
        verbose = NULL, checkBounds = TRUE, ...)
getInfRobIC

    risk, neighbor, symm, Finfo, trafo, upper = NULL,
    lower = NULL, maxiter, tol, warn, noLow = FALSE,
    verbose = NULL, ...

## S4 method for signature 'RealRandVariable,asGRisk,UncondNeighborhood'
getInfRobIC(L2deriv, risk,
    neighbor, Distr, DistrSymm, L2derivSymm,
    L2derivDistrSymm, Finfo, trafo, onesetLM = FALSE, z.start,
    A.start, upper = NULL, lower = NULL, OptOrIter = "iterate",
    maxiter, tol, warn, verbose = NULL, withPICcheck = TRUE,
    ..., .withEvalAsVar = TRUE)

## S4 method for signature
## 'UnivariateDistribution,asUnOvShoot,UncondNeighborhood'
getInfRobIC(        L2deriv, risk, neighbor, symm, Finfo, trafo,
    upper, lower, maxiter, tol, warn, verbose = NULL, ...

Arguments

- **L2deriv**: L2-derivative of some L2-differentiable family of probability measures.
- **risk**: object of class "RiskType".
- **neighbor**: object of class "Neighborhood".
- **...**: additional parameters (mainly for optim).
- **Distr**: object of class "Distribution".
- **symm**: logical: indicating symmetry of L2deriv.
- **DistrSymm**: object of class "DistributionSymmetry".
- **L2derivSymm**: object of class "FunSymmList".
- **L2derivDistrSymm**: object of class "DistrSymmList".
- **Finfo**: Fisher information matrix.
- **z.start**: initial value for the centering constant.
- **A.start**: initial value for the standardizing matrix.
- **trafo**: matrix: transformation of the parameter.
- **upper**: upper bound for the optimal clipping bound.
- **lower**: lower bound for the optimal clipping bound.
- **OptOrIter**: character; which method to be used for determining Lagrange multipliers A and a: if (partially) matched to "optimize", getLagrangeMultByOptim is used; otherwise: by default, or if matched to "iterate" or to "doubleiterate", getLagrangeMultByIter is used. More specifically, when using getLagrangeMultByIter, and if argument risk is of class "asGRisk", by default and if matched to "iterate" we use only one (inner) iteration, if matched to "doubleiterate" we use up to Maxiter (inner) iterations.
- **maxiter**: the maximum number of iterations.
tol the desired accuracy (convergence tolerance).
warn logical: print warnings.
noLow logical: is lower case to be computed?
onsetLM logical: use one set of Lagrange multipliers?
QuadForm matrix of (or which may coerced to) class PosSemDefSymmMatrix for use of different (standardizing) norm
verbose logical: if TRUE, some messages are printed
checkBounds logical: if TRUE, minimal and maximal clipping bound are computed to check if a valid bound was specified.
withPICcheck logical: at the end of the algorithm, shall we check how accurately this is a pIC; this will only be done if withPICcheck && verbose.
.withEvalAsVar logical (of length 1): if TRUE, risks based on covariances are to be evaluated (default), otherwise just a call is returned.

Value
The optimally robust IC is computed.

Methods

L2deriv = "UnivariateDistribution", risk = "asCov", neighbor = "ContNeighborhood" computes the classical optimal influence curve for L2 differentiable parametric families with unknown one-dimensional parameter.

L2deriv = "UnivariateDistribution", risk = "asCov", neighbor = "TotalVarNeighborhood" computes the classical optimal influence curve for L2 differentiable parametric families with unknown one-dimensional parameter.

L2deriv = "RealRandVariable", risk = "asCov", neighbor = "UncondNeighborhood" computes the classical optimal influence curve for L2 differentiable parametric families with unknown $k$-dimensional parameter ($k > 1$) where the underlying distribution is univariate; for total variation neighborhoods only is implemented for the case where there is a $1 \times k$ transformation trafo matrix.

L2deriv = "UnivariateDistribution", risk = "asBias", neighbor = "UncondNeighborhood" computes the bias optimal influence curve for L2 differentiable parametric families with unknown one-dimensional parameter.

L2deriv = "RealRandVariable", risk = "asBias", neighbor = "UncondNeighborhood" computes the bias optimal influence curve for L2 differentiable parametric families with unknown $k$-dimensional parameter ($k > 1$) where the underlying distribution is univariate.

L2deriv = "UnivariateDistribution", risk = "asHampel", neighbor = "UncondNeighborhood" computes the optimally robust influence curve for L2 differentiable parametric families with unknown one-dimensional parameter.

L2deriv = "RealRandVariable", risk = "asHampel", neighbor = "UncondNeighborhood" computes the optimally robust influence curve for L2 differentiable parametric families with unknown $k$-dimensional parameter ($k > 1$) where the underlying distribution is univariate; for total variation neighborhoods only is implemented for the case where there is a $1 \times k$ transformation trafo matrix.
getInfRobIC

L2deriv = "UnivariateDistribution", risk = "asAnscombe", neighbor = "UncondNeighborhood"
computes the optimally bias-robust influence curve to given ARE in the ideal model for L2
differentiable parametric families with unknown one-dimensional parameter.

L2deriv = "RealRandVariable", risk = "asAnscombe", neighbor = "UncondNeighborhood"
computes the optimally bias-robust influence curve to given ARE in the ideal model for L2
differentiable parametric families with unknown k-dimensional parameter (k > 1) where the
underlying distribution is univariate; for total variation neighborhoods only is implemented
for the case where there is a 1 x k transformation trafo matrix.

L2deriv = "UnivariateDistribution", risk = "asGRisk", neighbor = "UncondNeighborhood"
computes the optimally robust influence curve for L2 differentiable parametric families with
unknown one-dimensional parameter.

L2deriv = "RealRandVariable", risk = "asGRisk", neighbor = "UncondNeighborhood" computes
the optimally robust influence curve for L2 differentiable parametric families with unknown
k-dimensional parameter (k > 1) where the underlying distribution is univariate; for total vari-
ation neighborhoods only is implemented for the case where there is a 1 x k transformation
trafo matrix.

L2deriv = "UnivariateDistribution", risk = "asUnOvShoot", neighbor = "UncondNeighborhood"
computes the optimally robust influence curve for one-dimensional L2 differentiable paramet-
ic families and asymptotic under-/overshoot risk.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>,
Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

References


See Also

InfRobModel-class
getInfStand

Generic Function for the Computation of the Standardizing Matrix

Description

Generic function for the computation of the standardizing matrix which takes care of the Fisher consistency of the corresponding IC. This function is rarely called directly. It is used to compute optimally robust ICs.

Usage

getInfStand(L2deriv, neighbor, biastype, ...)

## S4 method for signature 'UnivariateDistribution,ContNeighborhood,BiasType'
getInfStand(L2deriv,
            neighbor, biastype, clip, cent, trafo)

## S4 method for signature
## 'UnivariateDistribution,TotalVarNeighborhood,BiasType'
getInfStand(L2deriv,
            neighbor, biastype, clip, cent, trafo)

## S4 method for signature 'RealRandVariable,UncondNeighborhood,BiasType'
getInfStand(L2deriv,
            neighbor, biastype, Distr, A.comp, cent, trafo, w)

## S4 method for signature 'UnivariateDistribution,ContNeighborhood,BiasType'
getInfStand(L2deriv,
            neighbor, biastype, clip, cent, trafo)

## S4 method for signature 'UnivariateDistribution,ContNeighborhood,BiasType'
getInfStand(L2deriv,
            neighbor, biastype, clip, cent, trafo)

Arguments

L2deriv  L2-derivative of some L2-differentiable family of probability measures.
neighbor object of class "Neighborhood"
biastype object of class "BiasType"
...  additional parameters
clip  optimal clipping bound.
cent  optimal centering constant.
Distr  object of class "Distribution".
trafo  matrix: transformation of the parameter.
getInfV

A.comp

matrix: indication which components of the standardizing matrix have to be computed.

w

object of class RobWeight; current weight

Value

The standardizing matrix is computed.

Methods

L2deriv = "UnivariateDistribution", neighbor = "ContNeighborhood", biastype = "BiasType"
computes standardizing matrix for symmetric bias.

L2deriv = "UnivariateDistribution", neighbor = "TotalVarNeighborhood", biastype = "BiasType"
computes standardizing matrix for symmetric bias.

L2deriv = "RealRandVariable", neighbor = "UncondNeighborhood", biastype = "BiasType"
computes standardizing matrix for symmetric bias.

L2deriv = "UnivariateDistribution", neighbor = "ContNeighborhood", biastype = "onesidedBias"
computes standardizing matrix for onesided bias.

L2deriv = "UnivariateDistribution", neighbor = "ContNeighborhood", biastype = "asymmetricBias"
computes standardizing matrix for asymmetric bias.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>, Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

References


See Also

ContIC-class, TotalVarIC-class

getInfV

Generic Function for the Computation of the asymptotic Variance of a Hampel type IC

Description

Generic function for the computation of the optimal clipping bound in case of infinitesimal robust models. This function is rarely called directly. It is used to compute optimally robust ICs.
Usage

getInfV(L2deriv, neighbor, biastype, ...)
## S4 method for signature 'UnivariateDistribution,ContNeighborhood,BiasType'
getInfV(L2deriv,
    neighbor, biastype, clip, cent, stand)
## S4 method for signature
## 'UnivariateDistribution,TotalVarNeighborhood,BiasType'
getInfV(L2deriv,
    neighbor, biastype, clip, cent, stand)
## S4 method for signature 'RealRandVariable,ContNeighborhood,BiasType'
getInfV(L2deriv,
    neighbor, biastype, Distr, V.comp, cent, stand, w)
## S4 method for signature 'RealRandVariable,TotalVarNeighborhood,BiasType'
getInfV(L2deriv,
    neighbor, biastype, Distr, V.comp, cent, stand, w)
## S4 method for signature
## 'UnivariateDistribution,ContNeighborhood,onesidedBias'
getInfV(L2deriv,
    neighbor, biastype, clip, cent, stand)
## S4 method for signature
## 'UnivariateDistribution,ContNeighborhood,asymmetricBias'
getInfV(L2deriv,
    neighbor, biastype, clip, cent, stand)

Arguments

L2deriv L2-derivative of some L2-differentiable family of probability measures.
neighbor object of class "Neighborhood".
biastype object of class "BiasType"
... additional parameters.
clip positive real: clipping bound
cent optimal centering constant.
stand standardizing matrix.
Distr standardizing matrix.
V.comp matrix: indication which components of the standardizing matrix have to be computed.
w object of class RobWeight; current weight

Value

The asymptotic variance of an ALE to IC of Hampel type is computed.

Author(s)

Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>
References


See Also

`ContIC-class.TotalVarIC-class`

---

**getL1normL2deriv**

*Calculation of L1 norm of L2derivative*

**Description**

Methods to calculate the L1 norm of the L2derivative in a smooth parametric model.

**Usage**

```r
getL1normL2deriv(L2deriv, ...) 
## S4 method for signature 'UnivariateDistribution'
getL1normL2deriv(L2deriv,
    cent, ...)

## S4 method for signature 'RealRandVariable'
getL1normL2deriv(L2deriv,
    cent, stand, Distr, normtype, ...)
```

**Arguments**

- `L2deriv` L2derivative of the model
- `cent` centering Lagrange Multiplier
- `stand` standardizing Lagrange Multiplier
- `Distr` distribution of the L2derivative
- `normtype` object of class `NormType`; the norm under which we work
- `...` further arguments (not used at the moment)

**Value**

L1 norm of the L2derivative
**Author(s)**

Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

**Examples**

```r
##
```

---

**getL2normL2deriv**  
*Calculation of L2 norm of L2derivative*

**Description**

Function to calculate the L2 norm of the L2derivative in a smooth parametric model.

**Usage**

```r
getL2normL2deriv(aFinfo, cent, ...)
```

**Arguments**

- `aFinfo`: trace of the Fisher information
- `cent`: centering
- `...`: further arguments (not used at the moment)

**Value**

L2 norm of the L2derivative

**Author(s)**

Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

**Examples**

```r
##
```
getMaxIneff

description

computes the maximal inefficiency of an IC for the radius range [0,Inf).

usage

getMaxIneff(IC, neighbor, biastype = symmetricBias(),
            normtype = NormType(), z.start = NULL,
            A.start = NULL, maxiter = 50,
            tol = .Machine$double.eps^0.4,
            warn = TRUE, verbose = NULL)

arguments

IC some IC of class IC
neighbor object of class Neighborhood; the neighborhood at which to compute the bias.
biastype a bias type of class BiasType
normtype a norm type of class NormType
z.start initial value for the centering constant.
A.start initial value for the standardizing matrix.
maxiter the maximum number of iterations.
tol the desired accuracy (convergence tolerance).
warn logical: print warnings.
verbose logical: if TRUE, some messages are printed

value

The maximal inefficiency, i.e.; a number in [1,Inf).

author(s)

Peter Ruckdeschel <peter.ruckdeschel@fraunhofer.itwm.de>

references


Examples

```r
N0 <- NormLocationFamily(mean=2, sd=3)
## L_2 family + infinitesimal neighborhood
neighbor <- ContNeighborhood(radius = 0.5)
N0.Rob1 <- InfRobModel(center = N0, neighbor = neighbor)
## OBRE solution (ARE 95%)
N0.ICA <- optIC(model = N0.Rob1, risk = asAnscombe(.95))
## OMSE solution radius 0.5
N0.ICM <- optIC(model=N0.Rob1, risk=asMSE())
## RMX solution
N0.ICR <- radiusMinimaxIC(L2Fam=N0, neighbor=neighbor,risk=asMSE())
getMaxIneff(N0.ICA,neighbor)
getMaxIneff(N0.ICM,neighbor)
getMaxIneff(N0.ICR,neighbor)

## Don't run to reduce check time on CRAN
N0ls <- NormLocationScaleFamily()
ICsc <- makeIC(list(sin,cos),N0ls)
getMaxIneff(ICsc,neighbor)
```

`getModifyIC`  
*Generic Function for the Computation of Functions for Slot modifyIC*

**Description**

These function is used by internal computations and is rarely called directly.

**Usage**

```r
getModifyIC(L2FamIC, neighbor, risk,...)
## S4 method for signature 'L2ParamFamily,Neighborhood,asRisk'
getModifyIC(L2FamIC, neighbor, risk,...)
## S4 method for signature 'L2LocationFamily,UncondNeighborhood,asGRisk'
getModifyIC(L2FamIC, neighbor, risk,...)
## S4 method for signature 'L2LocationFamily,UncondNeighborhood,fiUnOvShoot'
getModifyIC(L2FamIC, neighbor, risk,...)
## S4 method for signature 'L2ScaleFamily,UncondNeighborhood,asGRisk'
```

---

## Examples

```r
N0 <- NormLocationFamily(mean=2, sd=3)
## L_2 family + infinitesimal neighborhood
neighbor <- ContNeighborhood(radius = 0.5)
N0.Rob1 <- InfRobModel(center = N0, neighbor = neighbor)
## OBRE solution (ARE 95%)
N0.ICA <- optIC(model = N0.Rob1, risk = asAnscombe(.95))
## OMSE solution radius 0.5
N0.ICM <- optIC(model=N0.Rob1, risk=asMSE())
## RMX solution
N0.ICR <- radiusMinimaxIC(L2Fam=N0, neighbor=neighbor,risk=asMSE())
getMaxIneff(N0.ICA,neighbor)
getMaxIneff(N0.ICM,neighbor)
getMaxIneff(N0.ICR,neighbor)

## Don't run to reduce check time on CRAN
N0ls <- NormLocationScaleFamily()
ICsc <- makeIC(list(sin,cos),N0ls)
getMaxIneff(ICsc,neighbor)
```
getModifyIC(L2FamIC,
    neighbor, risk, ..., modifyICwarn = NULL)
## S4 method for signature 'L2LocationScaleFamily,UncondNeighborhood,asGRisk'
getModifyIC(L2FamIC,
    neighbor, risk, ..., modifyICwarn = NULL)

scaleUpdateIC(neighbor,...)
## S4 method for signature 'UncondNeighborhood'
scaleUpdateIC(neighbor, sdneu, sdalt, IC)
## S4 method for signature 'ContNeighborhood'
scaleUpdateIC(neighbor, sdneu, sdalt, IC)
## S4 method for signature 'TotalVarNeighborhood'
scaleUpdateIC(neighbor, sdneu, sdalt, IC)

Arguments

L2FamIC object of class L2ParamFamily.
neighbor object of class "Neighborhood".
risk object of class "RiskType"
... further arguments to be passed over to optIC.
sdneu positive numeric of length one; the new scale.
sdalt positive numeric of length one; the new scale.
IC a Hampel-IC to be updated.
modifyICwarn logical: should a (warning) information be added if modifyIC is applied and hence some optimality information could no longer be valid? Defaults to NULL in which case this value is taken from RobAStBaseOptions.

Details

This function is used for internal computations. By setting RobAStBaseOption("all.verbose" = TRUE) somewhere globally, the generated function modifyIC will generate calls to optIC with argument verbose=TRUE.

Value

getmodifyIC Function for slot modifyIC of ICs
scaleUpdateIC a list to be digested in corresponding methods of getmodifyIC by generateIC

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References

**getRadius**

**Computation of the Optimal Radius for Given Clipping Bound**

**Description**

The usual robust optimality problem for given asGRisk searches the optimal clipping height $b$ of a Hampel-type IC to given radius of the neighborhood. Instead, again for given asGRisk and for given Hampel-Type IC with given clipping height $b$ we may determine the radius of the neighborhood for which it is optimal in the sense of the first sentence.

**Usage**

```r
getRadius(IC, risk, neighbor, L2Fam)
```

**Arguments**

- **IC**: an IC of class "HampIC".
- **risk**: object of class "RiskType".
- **neighbor**: object of class "Neighborhood".
- **L2Fam**: object of class "L2FamParameter". Can be missing; in this case it is constructed from slot CallL2Fam of IC.

**Value**

The optimal radius is computed.

**Author(s)**

Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

**References**

getReq

See Also

ContIC-class, TotalVarIC-class

Examples

N <- NormLocationFamily(mean=0, sd=1)
nb <- ContNeighborhood(); ri <- asMSE()
radIC <- radiusMinimaxIC(L2Fam=N, neighbor=nb, risk=ri, loRad=0.1, upRad=0.5)
getRadius(radIC, L2Fam=N, neighbor=nb, risk=ri)

## taken from script NormalscaleModel.R in folder scripts
N0 <- NormScaleFamily(mean=0, sd=1)
(N0.IC7 <- radiusMinimaxIC(L2Fam=N0, neighbor=nb, risk=ri, loRad=0, upRad=Inf))
getRadius(N0.IC7, risk=asMSE(), neighbor=nb, L2Fam=N0)
getRadius(N0.IC7, risk=asL4(), neighbor=nb, L2Fam=N0)

Description

(tries to) compute a radius interval where IC1 is better than IC2, respectively the number of (worst-case) outliers interval where IC1 is better than IC2.

Usage

getReq(Risk, neighbor, IC1, IC2, n=1, upper=15, radOrOutl=c("radius", "Outlier"))

Arguments

Risk an object of class "asGRisk" – the risk at which IC1 is better than IC2.
neighbor object of class "Neighborhood"; the neighborhood at which to compute the bias.
IC1 some IC of class "IC"
IC2 some IC of class "IC"
n the sample size; by default set to 1; then the radius interval refers to starting radii in the shrinking neighborhood setting of Rieder[94]. Otherwise the radius interval is scaled down accordingly.
upper the upper bound of the radius interval in which to search
radOrOutl a character string specifying whether an interval of radii or a number of outliers is returned; must be one of "radius" (default) and "Outlier".
The radius interval (given by its endpoints) where $\text{icQ}$ is better than $\text{icR}$ according to the risk. In case $\text{icR}$ is better than $\text{icQ}$ as to both variance and bias, the return value is NA.

Author(s)

Peter Ruckdeschel <peter.ruckdeschel@fau.de>

References


Examples

```r
N0 <- NormLocationFamily(mean=2, sd=3)
## L_2 family + infinitesimal neighborhood
neighbor <- ContNeighborhood(radius = 0.5)
N0.Rob1 <- InfRobModel(center = N0, neighbor = neighbor)
## OBRE solution (ARE 95%)
N0.ICA <- optIC(model = N0.Rob1, risk = asAnscombe(.95))
## MSE solution
N0.ICM <- optIC(model = N0.Rob1, risk = asMSE())

getReq(asMSE(), neighbor, N0.ICA, N0.ICM, n=1)
getReq(asMSE(), neighbor, N0.ICA, N0.ICM, n=30)

## Don't test to reduce check time on CRAN

## RMX solution
N0.ICR <- radiusMinimaxIC(L2Fam=N0, neighbor=neighbor, risk=asMSE())

getReq(asL1(), neighbor, N0.ICA, N0.ICM, n=30)
getReq(asL4(), neighbor, N0.ICA, N0.ICM, n=30)
getReq(asMSE(), neighbor, N0.ICA, N0.ICR, n=30)
getReq(asL1(), neighbor, N0.ICA, N0.ICR, n=30)
getReq(asL4(), neighbor, N0.ICA, N0.ICR, n=30)
getReq(asMSE(), neighbor, N0.ICM, N0.ICR, n=30)

## when to use MAD and when Qn
## for Qn, see C. Croux, P. Rousseeuw (1993). Alternatives to the Median
## Absolute Deviation, JASA 88(424):1273-1283
L2M <- NormScaleFamily()
IC.mad <- makeIC(function(x) sign(abs(x)-qnorm(.75)), L2M)
d.qn <- (2^(.5*qnorm(5/8)))-1
IC.qn <- makeIC(function(x) d.qn*(1/4 - pnorm(x+1/d.qn) + pnorm(x-1/d.qn)), L2M)
getReq(asMSE(), neighbor, IC.mad, IC.qn)
getReq(asMSE(), neighbor, IC.mad, IC.qn, radOut1 = "Outlier", n = 30)
# => MAD is better once $r > 0.5144$ (i.e. for more than 2 outliers for $n = 30$)
```
**getRiskFctBV-methods**

Methods for Function `getRiskFctBV` in Package `ROptEst`

**Description**

`getRiskFctBV` for a given object of S4 class `asGRisk` returns a function in bias and variance to compute the asymptotic risk.

**Methods**

- `getRiskFctBV` signature(`risk = "asL1", biastype = "ANY"`): returns a function with arguments `bias` and `variance` to compute the asymptotic absolute (L1) error for a given ALE at a situation where it has bias `bias` (including the radius!) and variance `variance`.

- `getRiskFctBV` signature(`risk = "asL4", biastype = "ANY"`): returns a function with arguments `bias` and `variance` to compute the asymptotic L4 error for a given ALE at a situation where it has bias `bias` (including the radius!) and variance `variance`.

**Examples**

```r
myrisk <- asMSE()
getRiskFctBV(myrisk)
```

---

**getRiskIC**

Generic function for the computation of a risk for an IC

**Description**

Generic function for the computation of a risk for an IC.

**Usage**

```r
getRiskIC(IC, risk, neighbor, L2Fam, ...)
```

```r
## S4 method for signature 'HampIC,asCov,missing,missing'
getRiskIC(IC, risk, withCheck= TRUE)

## S4 method for signature 'HampIC,asCov,missing,L2ParamFamily'
getRiskIC(IC, risk, L2Fam, withCheck= TRUE)

## S4 method for signature 'TotalVarIC,asCov,missing,L2ParamFamily'
getRiskIC(IC, risk, L2Fam, withCheck = TRUE)
```
Arguments

IC object of class "InfluenceCurve"
risk object of class "RiskType".
neighbor object of class "Neighborhood"; missing in the methods described here.
... additional parameters
L2Fam object of class "L2ParamFamily".
withCheck logical: should a call to checkIC be done to check accuracy (defaults to TRUE; ignored if nothing is computed but simply a slot is read out).

Details

To make sure that the results are valid, it is recommended to include an additional check of the IC properties of IC using checkIC.

Value

The risk of an IC is computed.

Methods

IC = "HampIC", risk = "asCov", neighbor = "missing", L2Fam = "missing" asymptotic covariance of IC read off from corresp. Risks slot.
IC = "HampIC", risk = "asCov", neighbor = "missing", L2Fam = "L2ParamFamily" asymptotic covariance of IC under L2Fam read off from corresp. Risks slot.
IC = "TotalVarIC", risk = "asCov", neighbor = "missing", L2Fam = "L2ParamFamily" asymptotic covariance of IC read off from corresp. Risks slot, resp. if this is NULL calculates it via getInfV.

Note

This generic function is still under construction.

Author(s)

Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

References

Description

getStartIC computes the optimally-robust IC to be used as argument ICstart in kStepEstimator.

Usage

getStartIC(model, risk, ...)

## S4 method for signature 'ANY,ANY'
getStartIC(model, risk, ...)

## S4 method for signature 'L2ParamFamily,asGRisk'
getStartIC(model, risk, ..., 
withEvalAsVar = TRUE, withMakeIC = FALSE, ..debug=FALSE, 
modifyICwarn = NULL)

## S4 method for signature 'L2ParamFamily,asBias'
getStartIC(model, risk, ..., withMakeIC = FALSE, 
..debug=FALSE, modifyICwarn = NULL)

## S4 method for signature 'L2ParamFamily,asCov'
getStartIC(model, risk, ..., withMakeIC = FALSE, 
..debug=FALSE)

## S4 method for signature 'L2ParamFamily,transCov'
getStartIC(model, risk, ..., withMakeIC = FALSE, 
..debug=FALSE)

## S4 method for signature 'L2ParamFamily,anscombe'
getStartIC(model, risk, ..., 
withEvalAsVar = TRUE, withMakeIC = FALSE, ..debug=FALSE, 
modifyICwarn = NULL)

Arguments

model normtype of class NormType
risk normtype of class NormType
... further arguments to be passed to specific methods.
withEvalAsVar logical (of length 1): if TRUE, risks based on covariances are to be evaluated (default), otherwise just a call is returned.

withMakeIC logical; if TRUE the IC is passed through makeIC before return.

.. .debug logical; if TRUE information for debugging is issued.

modifyICwarn logical; should a (warning) information be added if modifyIC is applied and hence some optimality information could no longer be valid? Defaults to NULL in which case this value is taken from RobStBaseOptions.

Details
getStartIC is used internally in functions robest and roptest to compute the optimally robust influence function according to the arguments given to them.

Value
An IC of type HampIC.

Methods

getStartIC signature(model = "ANY", risk = "ANY"): issue that this is not yet implemented.

getStartIC signature(model = "L2ParamFamily", risk = "asGRisk"): depending on the values of argument eps (to be passed on through the ... argument) computes the optimally robust influence function on the fly via calls to optIC or radiusMinimaxIC.

getStartIC signature(model = "L2ParamFamily", risk = "asBias"): computes the most-bias-robust influence function on the fly via calls to optIC.

getStartIC signature(model = "L2ParamFamily", risk = "asCov"): computes the classically optimal influence function on the fly via calls to optIC.

getStartIC signature(model = "L2ParamFamily", risk = "trAsCov"): computes the classically optimal influence function on the fly via calls to optIC.

Author(s)
Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

See Also

robest, optIC, radiusMinimaxIC
Description

Generating functions to generate structured input for function `robest`.

Usage

```r
genKStepCtrl(uselast = getRobAStBaseOption("kStepUseLast"),
              withUpdateInKer = getRobAStBaseOption("withUpdateInKer"),
              IC.UpdateInKer = getRobAStBaseOption("IC.UpdateInKer"),
              withICList = getRobAStBaseOption("withICList"),
              withPICList = getRobAStBaseOption("withPICList"),
              scalename = "scale", withLogScale = TRUE,
              withEvalAsVar = NULL, withMakeIC = FALSE)

genStartCtrl(initial = NULL, initial.est.ArgList = NULL,
              startPar = NULL, distance = CvMDist, withMDE = NULL)

genBBCtrl(neighbor = ContNeighborhood(), eps, eps.lower, eps.upper)

genStartICCtrl(withMakeIC = FALSE, withEvalAsVar = NULL, modifyICwarn = NULL)
```

Arguments

- **uselast**
  
  which parameter estimate (initial estimate or k-step estimate) shall be used to fill the slots `pIC`, `asvar` and `asbias` of the return value.

- **withUpdateInKer**
  
  if there is a non-trivial trafo in the model with matrix \( D \), shall the parameter be updated on \( \ker(D) \)?

- **IC.UpdateInKer**
  
  if there is a non-trivial trafo in the model with matrix \( D \), the IC to be used for this; if NULL the result of `getboundedIC(L2Fam, D)` is taken; this IC will then be projected onto \( \ker(D) \).

- **withICList**
  
  logical: shall slot `ICList` of return value be filled?

- **withPICList**
  
  logical: shall slot `PICList` of return value be filled?

- **scalename**
  
  character: name of the respective scale component.

- **withLogScale**
  
  logical: shall a scale component (if existing and found with name `scalename`) be computed on log-scale and backtransformed afterwards? This avoids crossing 0.

- **withEvalAsVar**
  
  logical or NULL: if TRUE (default), tells R to evaluate the asymptotic variance or if FALSE just to produces a call to do so. If `withEvalAsVar` is NULL (default), the content of slot `.withEvalAsVar` in the L2 family is used instead to take this decision.

- **withMakeIC**
  
  logical: if TRUE the [p]IC is passed through `makeIC` before return.

- **modifyICwarn**
  
  logical: should a (warning) information be added if `modifyIC` is applied and hence some optimality information could no longer be valid? Defaults to NULL in which case this value is taken from `RobAStBaseOptions`. 
initial.est  initial estimate for unknown parameter. If missing minimum distance estimator is computed.

initial.est.ArgList  a list of arguments to be given to argument start if the latter is a function; this list by default already starts with two unnamed items, the sample x, and the model L2Fam.

startPar  initial information used by optimize resp. optim; i.e; if (total) parameter is of length 1, startPar is a search interval, else it is an initial parameter value; if NULL slot startPar of ParamFamily is used to produce it; in the multivariate case, startPar may also be of class Estimate, in which case slot untransformed.estimate is used.

distance  distance function

withMDE  logical or NULL: Shall a minimum distance estimator be used as starting estimator in roptest() / robest()—in addition to the function given in argument startPar of the current function or, if the argument is NULL, in slot startPar of the L2 family? If NULL (default) the content of slot .withMDE in the L2 family is used instead to take this decision.

neighbor  object of class "UncondNeighborhood"

eps  positive real (0 < eps <= 0.5): amount of gross errors. See details below.

eps.lower  positive real (0 <= eps.lower <= eps.upper): lower bound for the amount of gross errors. See details below.

eps.upper  positive real (eps.lower <= eps.upper <= 0.5): upper bound for the amount of gross errors. See details below.

Details

All these functions bundle their respective input to (reusable) lists which can be used as arguments in function robest. For details, see this function.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>,
Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

See Also

roblox, L2ParamFamily-class UncondNeighborhood-class, RiskType-class

Examples

  genkStepCtrl()
  genstartICCtrl()
  genstartCtrl()
  gennbCtrl()
leastFavorableRadius  

Generic Function for the Computation of Least Favorable Radii

Description

Generic function for the computation of least favorable radii.

Usage

leastFavorableRadius(L2Fam, neighbor, risk, ...)

## S4 method for signature 'L2ParamFamily,UncondNeighborhood,asGRisk'

leastFavorableRadius(
  L2Fam, neighbor, risk, rho, upRad = 1,
  z.start = NULL, A.start = NULL, upper = 100,
  OptOrIter = "iterate", maxiter = 100,
  tol = .Machine$double.eps^0.4, warn = FALSE, verbose = NULL)

Arguments

L2Fam  L2-differentiable family of probability measures.
neighbor  object of class "Neighborhood".
risk  object of class "RiskType".
...  additional parameters
upRad  the upper end point of the radius interval to be searched.
rho  The considered radius interval is: \([r \rho, r/\rho]\) with \(\rho \in (0,1)\).
z.start  initial value for the centering constant.
A.start  initial value for the standardizing matrix.
upper  upper bound for the optimal clipping bound.
OptOrIter  character; which method to be used for determining Lagrange multipliers \(A\) and \(a\): if (partially) matched to "optimize", getLagrangeMultByOptim is used; otherwise: by default, or if matched to "iterate" or to "doubleiterate", getLagrangeMultByIter is used. More specifically, when using getLagrangeMultByIter, and if argument risk is of class "asGRisk", by default and if matched to "iterate" we use only one (inner) iteration, if matched to "doubleiterate" we use up to Maxiter (inner) iterations.
maxiter  the maximum number of iterations
tol  the desired accuracy (convergence tolerance).
warn  logical: print warnings.
verbose  logical: if TRUE, some messages are printed

Value

The least favorable radius and the corresponding inefficiency are computed.
Methods

\texttt{L2Fam = "L2ParamFamily", neighbor = "UncondNeighborhood", risk = "asGRisk"} computation of the least favorable radius.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>, Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

References


See Also

\texttt{radiusMinimaxIC}

Examples

\begin{verbatim}
N <- NormLocationFamily(mean=0, sd=1)
leastFavorableRadius(L2Fam=N, neighbor=ContNeighborhood(),
risk=asMSE(), rho=0.5)
\end{verbatim}

\begin{verbatim}
lowerCaseRadius \hspace{1cm} \textit{Computation of the lower case radius}
\end{verbatim}

Description

The lower case radius is computed; confer Subsection 2.1.2 in Kohl (2005) and formula (4.5) in Ruckdeschel (2005).

Usage

\begin{verbatim}
lowerCaseRadius(L2Fam, neighbor, risk, biastype, ...)
\end{verbatim}
Arguments

L2Fam: L2 differentiable parametric family
neighbor: object of class "Neighborhood"
risk: object of class "RiskType"
biastype: object of class "BiasType"
additional parameters

Value

lower case radius

Methods

L2Fam = "L2ParamFamily", neighbor = "ContNeighborhood", risk = "asMSE", biastype = "BiasType"
lower case radius for risk "asMSE" in case of "ContNeighborhood" for symmetric bias.

L2Fam = "L2ParamFamily", neighbor = "TotalVarNeighborhood", risk = "asMSE", biastype = "BiasType"
lower case radius for risk "asMSE" in case of "TotalVarNeighborhood"; (argument biastype is just for signature reasons).

L2Fam = "L2ParamFamily", neighbor = "ContNeighborhood", risk = "asMSE", biastype = "onesidedBias"
lower case radius for risk "asMSE" in case of "ContNeighborhood" for onesided bias.

L2Fam = "L2ParamFamily", neighbor = "ContNeighborhood", risk = "asMSE", biastype = "asymmetricBias"
lower case radius for risk "asMSE" in case of "ContNeighborhood" for asymmetric bias.

L2Fam = "UnivariateDistribution", neighbor = "ContNeighborhood", risk = "asMSE", biastype = "onesidedBias"
used only internally; trick to be able to call lower case radius from within minmax bias solver

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>, Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

References


See Also

L2ParamFamily-class, Neighborhood-class

Examples

lowerCaseRadius(BinomFamily(size = 10), ContNeighborhood(), asMSE())
lowerCaseRadius(BinomFamily(size = 10), TotalVarNeighborhood(), asMSE())
Generic Function for the Computation of Bias-Optimally Robust ICs

Description

Generic function for the computation of bias-optimally robust ICs in case of infinitesimal robust models. This function is rarely called directly.

Usage

```r
minmaxBias(L2deriv, neighbor, biastype, ...)
```

```
## S4 method for signature 'UnivariateDistribution,ContNeighborhood,BiasType'
minmaxBias(L2deriv,
           neighbor, biastype, symm, trafo, maxiter, tol, warn, Finfo, verbose = NULL)
```

```
## S4 method for signature
## 'UnivariateDistribution,ContNeighborhood,asymmetricBias'
minmaxBias(
  L2deriv, neighbor, biastype, symm, trafo, maxiter, tol, warn, Finfo, verbose = NULL)
```

```
## S4 method for signature
## 'UnivariateDistribution,ContNeighborhood,onesidedBias'
minmaxBias(
  L2deriv, neighbor, biastype, symm, trafo, maxiter, tol, warn, Finfo, verbose = NULL)
```

```
## S4 method for signature
## 'UnivariateDistribution,TotalVarNeighborhood,BiasType'
minmaxBias(
  L2deriv, neighbor, biastype, symm, trafo, maxiter, tol, warn, Finfo, verbose = NULL)
```

```
## S4 method for signature 'RealRandVariable,ContNeighborhood,BiasType'
minmaxBias(L2deriv,
           neighbor, biastype, normtype, Distr, z.start, A.start, z.comp, A.comp,
           Finfo, trafo, maxiter, tol, verbose = NULL)
```

```
## S4 method for signature 'RealRandVariable,TotalVarNeighborhood,BiasType'
minmaxBias(L2deriv,
           neighbor, biastype, normtype, Distr, z.start, A.start, z.comp, A.comp,
           Finfo, trafo, maxiter, tol, verbose = NULL)
```

Arguments

- **L2deriv**: L2-derivative of some L2-differentiable family of probability measures.
- **neighbor**: object of class "Neighborhood".
- **biastype**: object of class "BiasType".
**normtype**

object of class "NormType".

... additional parameters.

**Distr**

object of class "Distribution".

**symm**

logical: indicating symmetry of L2deriv.

**z.start**

initial value for the centering constant.

**A.start**

initial value for the standardizing matrix.

**z.comp**

logical indicator which indices need to be computed and which are 0 due to symmetry.

**A.comp**

matrix of logical indicator which indices need to be computed and which are 0 due to symmetry.

**trafo**

matrix: transformation of the parameter.

**maxiter**

the maximum number of iterations.

**tol**

the desired accuracy (convergence tolerance).

**warn**

logical: print warnings.

**Finfo**

Fisher information matrix.

**verbose**

logical: if TRUE, some messages are printed.

**Value**

The bias-optimally robust IC is computed.

**Methods**

\texttt{L2deriv = "UnivariateDistribution", neighbor = "ContNeighborhood", biastype = "BiasType"}

computes the bias optimal influence curve for symmetric bias for L2 differentiable parametric families with unknown one-dimensional parameter.

\texttt{L2deriv = "UnivariateDistribution", neighbor = "ContNeighborhood", biastype = "asymmetricBias"}

computes the bias optimal influence curve for asymmetric bias for L2 differentiable parametric families with unknown one-dimensional parameter.

\texttt{L2deriv = "UnivariateDistribution", neighbor = "TotalVarNeighborhood", biastype = "BiasType"}

computes the bias optimal influence curve for symmetric bias for L2 differentiable parametric families with unknown one-dimensional parameter.

\texttt{L2deriv = "RealRandVariable", neighbor = "ContNeighborhood", biastype = "BiasType"}

computes the bias optimal influence curve for symmetric bias for L2 differentiable parametric families with unknown \(k\)-dimensional parameter \((k > 1)\) where the underlying distribution is univariate.

\texttt{L2deriv = "RealRandVariable", neighbor = "TotalNeighborhood", biastype = "BiasType"}

computes the bias optimal influence curve for symmetric bias for L2 differentiable parametric families in a setting where we are interested in a \(p = 1\) dimensional aspect of an unknown \(k\)-dimensional parameter \((k > 1)\) where the underlying distribution is univariate.

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>, Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>
References


See Also

InfRobModel-class

optIC

Generic function for the computation of optimally robust ICs

Description

Generic function for the computation of optimally robust ICs.

Usage

optIC(model, risk, ...)

## S4 method for signature 'InfRobModel,asRisk'

optIC(model, risk, z.start = NULL, A.start = NULL,
      upper = 1e4, lower = 1e-4,
      OptOrIter = "iterate", maxiter = 50,
      tol = .Machine$double.eps^0.4, warn = TRUE,
      noLow = FALSE, verbose = NULL, ..., .withEvalAsVar = TRUE, withMakeIC = FALSE,
      returnNAifProblem = FALSE, modifyICwarn = NULL)

## S4 method for signature 'InfRobModel,asUnOvShoot'

optIC(model, risk, upper = 1e4,
      lower = 1e-4, maxiter = 50,
      tol = .Machine$double.eps^0.4,
      withMakeIC = FALSE, warn = TRUE,
      verbose = NULL, modifyICwarn = NULL)

## S4 method for signature 'FixRobModel,fiUnOvShoot'

optIC(model, risk, sampleSize, upper = 1e4, lower = 1e-4,
      maxiter = 50, tol = .Machine$double.eps^0.4,
      withMakeIC = FALSE, warn = TRUE,
      Algo = "A", cont = "left",
      verbose = NULL, modifyICwarn = NULL)
Arguments

- **model**: probability model.
- **risk**: object of class "RiskType".
- **...**: additional parameters.
- **z.start**: initial value for the centering constant.
- **A.start**: initial value for the standardizing matrix.
- **upper**: upper bound for the optimal clipping bound.
- **lower**: lower bound for the optimal clipping bound.
- **maxiter**: the maximum number of iterations.
- **tol**: the desired accuracy (convergence tolerance).
- **warn**: logical: print warnings.
- **sampleSize**: integer: sample size.
- **Algo**: "A" or "B".
- **cont**: "left" or "right".
- **nolow**: logical: is lower case to be computed?
- **OptOrIter**: character; which method to be used for determining Lagrange multipliers A and a: if (partially) matched to "optimize", getLagrangemultbyOptim is used; otherwise: by default, or if matched to "iterate" or to "doubleiterate", getLagrangemultbyIter is used. More specifically, when using getLagrangemultbyIter, and if argument risk is of class "asGRisk", by default and if matched to "iterate" we use only one (inner) iteration, if matched to "doubleiterate" we use up to maxiter (inner) iterations.
- **verbose**: logical: if TRUE, some messages are printed.
- **.withEvalAsVar**: logical (of length 1): if TRUE, risks based on covariances are to be evaluated (default), otherwise just a call is returned.
- **withMakeIC**: logical; if TRUE the [p]IC is passed through makeIC before return.
- **returnNAifProblem**: logical (of length 1): if TRUE (not the default), in case of convergence problems in the algorithm, returns NA.
- **modifyICwarn**: logical: should a (warning) information be added if modifyIC is applied and hence some optimality information could no longer be valid? Defaults to NULL in which case this value is taken from RobAStBaseOptions.

Details

In case of the finite-sample risk "fiUnOvShoot" one can choose between two algorithms for the computation of this risk where the least favorable contamination is assumed to be left or right of some bound. For more details we refer to Section 11.3 of Kohl (2005).

Value

Some optimally robust IC is computed.
Methods

model = "InfRobModel", risk = "asRisk" computes optimally robust influence curve for robust models with infinitesimal neighborhoods and various asymptotic risks.

model = "InfRobModel", risk = "asUnOvShoot" computes optimally robust influence curve for robust models with infinitesimal neighborhoods and asymptotic under-/overshoot risk.

model = "FixRobModel", risk = "fiUnOvShoot" computes optimally robust influence curve for robust models with fixed neighborhoods and finite-sample under-/overshoot risk.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

InfluenceCurve-class, RiskType-class

Examples

B <- BinomFamily(size = 25, prob = 0.25)

## classical optimal IC
IC0 <- optIC(model = B, risk = asCov())
plot(IC0) # plot IC
checkIC(IC0, B)
optRisk

Generic function for the computation of the minimal risk

Description

Generic function for the computation of the optimal (i.e., minimal) risk for a probability model.

Usage

optRisk(model, risk, ...)

## S4 method for signature 'L2ParamFamily,asCov'
optRisk(model, risk)

## S4 method for signature 'InfRobModel,asRisk'
optRisk(model, risk, z.start = NULL,
        A.start = NULL, upper = 1e4, maxiter = 50,
        tol = .Machine$double.eps^0.4, warn = TRUE, noLow = FALSE)

## S4 method for signature 'FixRobModel,fiUnOvShoot'
optRisk(model, risk, sampleSize,
        upper = 1e4, maxiter = 50, tol = .Machine$double.eps^0.4,
        warn = TRUE, Algo = "A", cont = "left")

Arguments

model probability model
risk object of class RiskType
... additional parameters
z.start initial value for the centering constant.
A.start initial value for the standardizing matrix.
upper upper bound for the optimal clipping bound.
maxiter the maximum number of iterations
tol the desired accuracy (convergence tolerance).
warn logical: print warnings.
sampleSize integer: sample size.
Algo "A" or "B".
cont "left" or "right".
noLow logical: is lower case to be computed?

Details

In case of the finite-sample risk "fiUnOvShoot" one can choose between two algorithms for the computation of this risk where the least favorable contamination is assumed to be left or right of some bound. For more details we refer to Section 11.3 of Kohl (2005).
Value

The minimal risk is computed.

Methods

- **model = "L2ParamFamily", risk = "asCov"** asymptotic covariance of L2 differentiable parametric family.
- **model = "InfRobModel", risk = "asRisk"** asymptotic risk of a infinitesimal robust model.
- **model = "FixRobModel", risk = "fiUnOvShoot"** finite-sample under-/overshoot risk of a robust model with fixed neighborhood.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

- **RiskType-class**

Examples

```r
optRisk(model = NormLocationScaleFamily(), risk = asCov())
```

---

**ORobEstimate-class**

**ORobEstimate-class.**

Description

Class of optimally robust asymptotically linear estimates.

Objects from the Class

Objects can be created by calls of the form `new("ORobEstimate", ...)`. More frequently they are created as results of functions `roptest`, `MBREstimator`, `RMXEstimator`, or `OMSEstimator`. 
Slots

- **name** Object of class "character": name of the estimator. [*]
- **estimate** Object of class "ANY": estimate. [*]
- **estimate.call** Object of class "call": call by which estimate was produced. [*]
- **samplesize** Object of class "numeric" — the samplesize (only complete cases are counted) at which the estimate was evaluated. [*]
- **completecases** Object of class "logical" — complete cases at which the estimate was evaluated. [*]
- **asvar** Object of class "OptionalNumericOrMatrix" which may contain the asymptotic (co)variance of the estimator. [*]
- **asbias** Optional object of class "numeric": asymptotic bias. [*]
- **pIC** Optional object of class InfluenceCurve: influence curve. [*]
- **nuis.idx** Object of class "OptionalNumeric": indices of estimate belonging to the nuisance part. [*]
- **fixed** Object of class "OptionalNumeric": the fixed and known part of the parameter. [*]
- **steps** Object of class "integer": number of steps. [*]
- **Infos** Object of class "matrix" with two columns named method and message: additional informations. [*]
- **trafo** Object of class "list": a list with components fct and mat (see below). [*]
- **untransformed.estimate** Object of class "ANY": untransformed estimate. [*]
- **untransformed.asvar** Object of class "OptionalNumericOrMatrix" which may contain the asymptotic (co)variance of the untransformed estimator. [*]
- **pIClist** Optional object of class "OptionalpICList": the list of (intermediate) (partial) influence curves used; only filled when called from ORobEstimator with argument withPIClist==TRUE. [*]
- **ICList** Optional object of class "OptionalpICList": the list of (intermediate) (total) influence curves used; only filled when called from ORobEstimator with argument withICList==TRUE. [*]
- **start** The argument start — of class "StartClass" used in call to ORobEstimator. [*]
- **startval** Object of class "matrix": the starting value with which the k-step Estimator was initialized (in p-space / transformed). [*]
- **ustartval** Object of class "matrix": the starting value with which the k-step Estimator was initialized (in k-space / untransformed). [*]
- **ksteps** Object of class "OptionalMatrix": the intermediate estimates (in p-space) for the parameter; only filled when called from ORobEstimator. [*]
- **uksteps** Object of class "OptionalMatrix": the intermediate estimates (in k-space) for the parameter; only filled when called from ORobEstimator. [*]
- **robestcall** Object of class "OptionalCall", i.e., a call or NULL: only filled when called from roptest. [*]
- **roptestcall** Object of class "OptionalCall", i.e., a call or NULL: only filled when called from roptest, MBREstimator, RMXEstimator, or OMSEstimator.
Extends

Class "kStepEstimate", directly.
Class "ALEstimate" and class "Estimate", by class "kStepEstimate". All slots and methods marked with [*] are inherited.

Methods

steps signature(object = "ORobEstimate"): accessor function for slot steps. [*]
ksteps signature(object = "ORobEstimate"): accessor function for slot ksteps; has additional argument diff, defaulting to FALSE; if the latter is TRUE, the starting value from slot startval is prepended as first column; otherwise we return the corresponding increments in each step. [*]
uksteps signature(object = "ORobEstimate"): accessor function for slot uksteps; has additional argument diff, defaulting to FALSE; if the latter is TRUE, the starting value from slot ustartval is prepended as first column; otherwise we return the corresponding increments in each step. [*]
start signature(object = "ORobEstimate"): accessor function for slot start. [*]
startval signature(object = "ORobEstimate"): accessor function for slot startval. [*]
ustartval signature(object = "ORobEstimate"): accessor function for slot startval. [*]
ICList signature(object = "ORobEstimate"): accessor function for slot ICList. [*]
pICList signature(object = "ORobEstimate"): accessor function for slot pICList. [*]
robestCall signature(object = "ORobEstimate"): accessor function for slot robestCall. [*]
roptestCall signature(object = "ORobEstimate"): accessor function for slot roptestCall.
show signature(object = "ORobEstimate"): a show method; [*]

Author(s)

Peter Ruckdeschel <Peter.Ruckdeschel@uni-oldenburg.de>

See Also

ALEstimate-class, kStepEstimate-class

plot-methods

Methods for Function plot in Package ‘ROptEst’

plot-methods

Description

plot-methods

Details

S4-Method plot for for signature IC,missing has been enhanced compared to its original definition in RobAStBase so that if argument MBR is NA, it is filled automatically by a call to optIC which computes the MBR-IC on the fly. To this end, there is an additional argument n.MBR defaulting to 10000 to determine the number of evaluation points.
**Examples**

```r
N <- NormLocationScaleFamily(mean=0, sd=1)
IC <- optIC(model = N, risk = asCov())
## Don't run to reduce check time on CRAN

plot(IC, main = TRUE, panel.first= grid(),
     col = "blue", cex.main = 2, cex.inner = 0.6,
     withMBR=TRUE)
```

---

**radiusMinimaxIC**

*Generic function for the computation of the radius minimax IC*

---

**Description**

Generic function for the computation of the radius minimax IC.

**Usage**

```r
radiusMinimaxIC(L2Fam, neighbor, risk, ...)
```

```r
def radiusMinimaxIC(
    L2Fam, neighbor, risk, loRad = 0, upRad = Inf, z.start = NULL, A.start = NULL,
    upper = NULL, lower = NULL, OptOrIter = "iterate",
    maxiter = 50, tol = .Machine$double.eps*0.4,
    warn = FALSE, verbose = NULL, loRad0 = 1e-3, ...
)
```

**Arguments**

- **L2Fam**
  L2-differentiable family of probability measures.

- **neighbor**
  object of class "Neighborhood".

- **risk**
  object of class "RiskType".

- **loRad**
  the lower end point of the interval to be searched in the inner optimization (for the least favorable situation to the user-guessed radius).

- **upRad**
  the upper end point of the interval to be searched in the inner optimization (for the least favorable situation to the user-guessed radius).

- **z.start**
  initial value for the centering constant.

- **A.start**
  initial value for the standardizing matrix.

- **upper**
  upper bound for the optimal clipping bound.

- **lower**
  lower bound for the optimal clipping bound.
OptOrIter character; which method to be used for determining Lagrange multipliers \( a \) and \( a^\prime \): if (partially) matched to "optimize", `getLagrangemultByOptim` is used; otherwise: by default, or if matched to "iterate" or to "doubleiterate", `getLagrangemultByIter` is used. More specifically, when using `getLagrangemultByIter`, and if argument `risk` is of class "asGRisk", by default and if matched to "iterate" we use only one (inner) iteration, if matched to "doubleiterate" we use up to \( \text{maxiter} \) (inner) iterations.

maxiter the maximum number of iterations
tol the desired accuracy (convergence tolerance).
warn logical: print warnings.
verbose logical: if TRUE, some messages are printed
loRad0 for numerical reasons: the effective lower bound for the zero search; internally set to \( \max(\text{loRad}, \text{loRad0}) \).

... further arguments to be passed on to `getInfRobIC`
returnNAifProblem logical (of length 1): if TRUE (not the default), in case of convergence problems in the algorithm, returns NA.
loRad.s the lower end point of the interval to be searched in the outer optimization (for the user-guessed radius); if NULL (default) set to \( \text{loRad} \) in the algorithm.
upRad.s the upper end point of the interval to be searched in the outer optimization (for the user-guessed radius); if NULL (default) set to \( \text{upRad} \) in the algorithm.
modifyICwarn logical: should a (warning) information be added if modifyIC is applied and hence some optimality information could no longer be valid? Defaults to NULL in which case this value is taken from `RobAStBaseOptions`.

Details
In case the neighborhood radius is unknown, Rieder et al. (2001, 2008) and Kohl (2005) show that there is nevertheless a way to compute an optimally robust IC - the so-called radius-minimax IC - which is optimal for some radius interval.

Value
The radius minimax IC is computed.

Methods

\[
\text{L2Fam} = "L2ParamFamily", \text{neighbor} = "UncondNeighborhood", \text{risk} = "asGRisk": \text{ computation of the radius minimax IC for an L2 differentiable parametric family.}
\]

Author(s)
Matthias Kohl <Matthias.Kohl@stamats.de>, Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>
References


See Also

radiusMinimaxIC

Examples

```r
N <- NormLocationFamily(mean=0, sd=1)
radIC <- radiusMinimaxIC(L2Fam=N, neighbor=ContNeighborhood(),
                         risk=asMSE(), IoRad=0.1, upRad=0.5)
checkIC(radIC)
```

---

**RMXEOSEMSEMBREOBRE**

**Optimally robust estimation: RMXE, OMSE, MBRE, and OBRE**

**Description**

These are wrapper functions to 'roptest' to compute optimally robust estimates, more specifically RMXEs, OMSEs, MBREs, and OBREs, for L2-differentiable parametric families via k-step construction.

**Usage**

```r
RMXEstimator(x, L2Fam, fsCor = 1, initial.est, neighbor = ContNeighborhood(),
              steps = 1L, distance = CvMDist, startPar = NULL, verbose = NULL,
              OptOrIter = "iterate", useLast = getRobAStBaseOption("kStepUseLast"),
              withUpdateInKer = getRobAStBaseOption("withUpdateInKer"),
              IC.UpdateInKer = getRobAStBaseOption("IC.UpdateInKer"),
              withICList = getRobAStBaseOption("withICList"),
              withPICList = getRobAStBaseOption("withPICList"), na.rm = TRUE,
              initial.est.ArgList, ..., withLogScale = TRUE, ..withCheck=FALSE,
              withTimings = FALSE, withMDE = NULL, withEvalAsVar = NULL,
              withMakeIC = FALSE, modifyICwarn = NULL)
```

```r
OMSEstimator(x, L2Fam, eps=0.5, fsCor = 1, initial.est, neighbor = ContNeighborhood(),
              steps = 1L, distance = CvMDist, startPar = NULL, verbose = NULL,
              OptOrIter = "iterate", useLast = getRobAStBaseOption("kStepUseLast"),
              withUpdateInKer = getRobAStBaseOption("withUpdateInKer"),
```

Arguments

\[ x \]  
Sample

\[ \text{L2Fam} \]  
Object of class "L2ParamFamily"

\[ \text{eff} \]  
Positive real \((0 \leq \text{eff} \leq 1)\): amount of asymptotic efficiency loss in the ideal model. See details below.

\[ \text{eps} \]  
Positive real \((0 < \text{eps} \leq 0.5)\): amount of gross errors. See details below.

\[ \text{fsCor} \]  
Positive real: factor used to correct the neighborhood radius; see details.

\[ \text{initial.est} \]  
Initial estimate for unknown parameter. If missing minimum distance estimator is computed.

\[ \text{neighbor} \]  
Object of class "UncondNeighborhood"

\[ \text{steps} \]  
Positive integer: number of steps used for k-steps construction

\[ \text{distance} \]  
Initial information used by optimize resp. optim; i.e.; if (total) parameter is of length 1, startPar is a search interval, else it is an initial parameter value; if NULL slot startPar of ParamFamily is used to produce it; in the multivariate case, startPar may also be of class Estimate, in which case slot untransformed.estimate is used.
verbose logical: if TRUE, some messages are printed

useLast which parameter estimate (initial estimate or k-step estimate) shall be used to fill the slots pic, asvar and asbias of the return value.

OptOrIter character; which method to be used for determining Lagrange multipliers A and a: if (partially) matched to "optimize", getLagrangeMultByOptim is used; otherwise: by default, or if matched to "iterate" or to "doubleiterate", getLagrangeMultByIter is used. More specifically, when using getLagrangeMultByIter, and if argument risk is of class "asGRisk", by default and if matched to "iterate" we use only one (inner) iteration, if matched to "doubleiterate" we use up to Maxiter (inner) iterations.

withUpdateInKer if there is a non-trivial trafo in the model with matrix D, shall the parameter be updated on ker(D)?

IC.UpdateInKer if there is a non-trivial trafo in the model with matrix D, the IC to be used for this; if NULL the result of getboundedIC(L2Fam, D) is taken; this IC will then be projected onto ker(D).

withPICList logical: shall slot PICList of return value be filled?

withICList logical: shall slot ICList of return value be filled?

na.rm logical: if TRUE, the estimator is evaluated at complete.cases(x).

initial.est.ArgList a list of arguments to be given to argument start if the latter is a function; this list by default already starts with two unnamed items, the sample x, and the model L2Fam.

... further arguments

withLogScale logical: shall a scale component (if existing and found with name scalename) be computed on log-scale and backtransformed afterwards? This avoids crossing 0.

..withCheck logical: if TRUE, debugging info is issued.

withTimings logical: if TRUE, separate (and aggregate) timings for the three steps evaluating the starting value, finding the starting influence curve, and evaluating the k-step estimator is issued.

withMDE logical or NULL: Shall a minimum distance estimator be used as starting estimator—in addition to the function given in slot startPar of the L2 family? If NULL (default), the content of slot .withMDE in the L2 family is used instead to take this decision.

withEvalAsVar logical or NULL: if TRUE (default), tells R to evaluate the asymptotic variance or if FALSE just to produces a call to do so. If withEvalAsVar is NULL (default), the content of slot .withEvalAsVar in the L2 family is used instead to take this decision.

withMakeIC logical: if TRUE the [p]IC is passed through makeIC before return.

modifyICwarn logical: should a (warning) information be added if modifyIC is applied and hence some optimality information could no longer be valid? Defaults to NULL in which case this value is taken from RobAStBaseOptions.
Details

The functions compute optimally robust estimator for a given L2 differentiable parametric family; more specifically they are RMXEs, OMSEs, MBREs, and OBREs. The computation uses a k-step construction with an appropriate initial estimate; cf. also \texttt{kStepEstimator}. Valid candidates are e.g. Kolmogorov(-Smirnov) or von Mises minimum distance estimators (default); cf. Rieder (1994) and Kohl (2005).

For OMSE, i.e., the asymptotically linear estimator with minimax mean squared error on this neighborhood of given size, the amount of gross errors (contamination) is assumed to be known, and is specified by \( \varepsilon \). The radius of the corresponding infinitesimal contamination neighborhood is obtained by multiplying \( \varepsilon \) by the square root of the sample size.

If the amount of gross errors (contamination) is unknown, RMXE should be used, i.e., the radius-minimax estimator in the sense of Rieder et al. (2001, 2008), respectively Section 2.2 of Kohl (2005) is returned.

The OBRE, i.e., the optimal bias-robust (asymptotically linear) estimator; (terminology due to Hampel et al (1985)), expects an efficiency loss (at the ideal model) to be specified and then, according to an (asymptotic) Anscombe criterion computes the the bias bound achieving this efficiency loss.

The MBRE, i.e., the most bias-robust (asymptotically linear) estimator; (terminology due to Hampel et al (1985)), uses the influence curve with minimal possible bias bound, hence minimaxes bias on these neighborhoods (in an infinitesimal sense).

Finite-sample and higher order results suggest that the asymptotically optimal procedure is to liberal. Using \texttt{fsCor} the radius can be modified - as a rule enlarged - to obtain a more conservative estimate. In case of normal location and scale there is function \texttt{finiteSampleCorrection} which returns a finite-sample corrected (enlarged) radius based on the results of large Monte-Carlo studies.

The default value of argument useLast is set by the global option \texttt{kStepUseLast} which by default is set to \texttt{FALSE}. In case of general models \texttt{useLast} remains unchanged during the computations. However, if slot \texttt{callL2Fam} of IC generates an object of class "L2GroupParamFamily" the value of \texttt{useLast} is changed to \texttt{TRUE}. Explicitly setting \texttt{useLast} to \texttt{TRUE} should be done with care as in this situation the influence curve is re-computed using the value of the one-step estimate which may take quite a long time depending on the model.

If \texttt{useLast} is set to \texttt{TRUE} the computation of \texttt{asvar}, \texttt{asbias} and \texttt{IC} is based on the k-step estimate. All these estimators are realized as wrappers to function \texttt{roptest}.

Value

Object of class "kStepEstimate". In addition, it has an attribute "timings" where computation time is stored.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>,
Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

References


See Also
roptest, robest, roblox, L2ParamFamily-class, UncondNeighborhood-class, RiskType-class

Examples

# 1. Binomial data
set.seed(123)
ind <- rbinom(100, size=1, prob=0.05)
x <- rbinom(100, size=25, prob=(1-ind)*0.25 + ind*0.9)

## ML-estimate
MLE.bin <- MLEstimator(x, BinomFamily(size = 25))

## compute optimally robust estimators
OMSE.bin <- OMSEstimator(x, BinomFamily(size = 25), steps = 3)
MBRE.bin <- MBREstimator(x, BinomFamily(size = 25), steps = 3)
estimate(MLE.bin)
estimate(MBRE.bin)
estimate(OMSE.bin)

## to reduce time load at CRAN tests
RMXE.bin <- RMXEstimator(x, BinomFamily(size = 25), steps = 3)
OBRE.bin <- OBREstimator(x, BinomFamily(size = 25), steps = 3)
estimate(RMXE.bin)
estimate(OBRE.bin)

## to reduce time load at CRAN tests

# 2. Poisson data

## Example: Rutherford-Geiger (1910); cf. Feller-(1968), Section VI.7 (a)
x <- c(rep(0, 57), rep(1, 293), rep(2, 383), rep(3, 525), rep(4, 532),
rep(5, 468), rep(6, 273), rep(7, 139), rep(8, 45), rep(9, 27),
rep(10, 10), rep(11, 4), rep(12, 0), rep(13, 1), rep(14, 1))
## Description

Function to compute optimally robust estimates for L2-differentiable parametric families via k-step construction.

## Usage

`robest(x, L2Fam, fsCor = 1, risk = asMSE(), steps = 1L, verbose = NULL, OptOrIter = "iterate", nbCtrl = gennbCtrl(), startCtrl = genstartCtrl(), startICCtrl = genstartICCtrl(), kStepCtrl = genkStepCtrl(), na.rm = TRUE, ..., debug = FALSE, withTimings = FALSE)`
**Arguments**

- **x** sample
- **L2Fam** object of class "L2ParamFamily"
- **fsCor** positive real: factor used to correct the neighborhood radius; see details.
- **risk** object of class "RiskType"
- **steps** positive integer: number of steps used for k-steps construction
- **verbose** logical: if TRUE, some messages are printed
- **OptOrIter** character; which method to be used for determining Lagrange multipliers \( A \) and \( a \): if (partially) matched to "optimize", \texttt{getLagrangeMultByOptim} is used; otherwise: by default, or if matched to "iterate" or to "doubleiterate", \texttt{getLagrangeMultByIter} is used. More specifically, when using \texttt{getLagrangeMultByIter}, and if argument risk is of class "asGRisk", by default and if matched to "iterate" we use only one (inner) iteration, if matched to "doubleiterate" we use up to \texttt{maxiter} (inner) iterations.
- **nbCtrl** a list specifying input concerning the used neighborhood; to be generated by a respective call to \texttt{gennbCtrl}.
- **startCtrl** a list specifying input concerning the used starting estimator; to be generated by a respective call to \texttt{genstartCtrl}.
- **startICCtrol** a list specifying input concerning the call to \texttt{getStartIC} which returns the starting influence curve; to be generated by a respective call to \texttt{genstartICCtrol}.
- **kStepCtrl** a list specifying input concerning the used variant of a kstepEstimator; to be generated by a respective call to \texttt{genkStepCtrl}.
- **na.rm** logical: if TRUE, the estimator is evaluated at \texttt{completecases(x)}.
- **...** further arguments
- **debug** logical: if TRUE, only the respective calls within the function are generated for debugging purposes.
- **withTimings** logical: if TRUE, separate (and aggregate) timings for the three steps evaluating the starting value, finding the starting influence curve, and evaluating the k-step estimator is issued.

**Details**

A new, more structured interface to the former function \texttt{roptest}. For details, see this function.

**Value**

Object of class "kStepEstimate". In addition, it has an attribute "timings" where computation time is stored.

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>,
Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>
See Also

roblox, L2ParamFamily-class UncondNeighborhood-class, RiskType-class

Examples

```r
## Don't test to reduce check time on CRAN

# Binomial data
# generate a sample of contaminated data
set.seed(123)
ind <- rbinom(100, size=1, prob=0.05)
x <- rbinom(100, size=25, prob=(1-ind)*0.25 + ind*0.9)

## Family
BF <- BinomFamily(size = 25)
## ML-estimate
MLEst <- MLEstimator(x, BF)
estimate(MLEst)
confint(MLEst)

## compute optimally robust estimator (known contamination)
nb <- gennbCtrl(eps=0.05)
robest1 <- robest(x, BF, nbCtrl = nb, steps = 3)
estimate(robest1)
confint(robest1, method = symmetricBias())
## neglecting bias
confint(robest1)
plot(pIC(robest1))
tmp <- qqplot(x, robest1, cex.pch=1.5, exp.cex2.pch = -.25,
exp.fadcol.pch = .55, jit.fac=.9)

## compute optimally robust estimator (unknown contamination)
nb2 <- gennbCtrl(eps.lower = 0, eps.upper = 0.2)
robest2 <- robest(x, BF, nbCtrl = nb2, steps = 3)
estimate(robest2)
confint(robest2, method = symmetricBias())
plot(pIC(robest2))

## total variation neighborhoods (known deviation)
nb3 <- gennbCtrl(eps = 0.025, neighbor = TotalVarNeighborhood())
robest3 <- robest(x, BF, nbCtrl = nb3, steps = 3)
estimate(robest3)
confint(robest3, method = symmetricBias())
plot(pIC(robest3))

## total variation neighborhoods (unknown deviation)
nb4 <- gennbCtrl(eps.lower = 0, eps.upper = 0.1,
neighbor = TotalVarNeighborhood())
robest3 <- robest(x, BF, nbCtrl = nb4, steps = 3)
```
robust4 <- robust(x, BinomFamily(size = 25), nbCtrl = nb4, steps = 3)
estimate(robust4)
confint(robust4, method = symmetricBias())
plot(pIC(robust4))

###############################
## 2. Poisson data
###############################
## Example: Rutherford-Geiger (1910); cf. Feller (1968), Section VI.7 (a)
x <- c(rep(0, 57), rep(1, 283), rep(2, 383), rep(3, 525), rep(4, 532),
      rep(5, 408), rep(6, 273), rep(7, 139), rep(8, 45), rep(9, 27),
      rep(10, 10), rep(11, 4), rep(12, 0), rep(13, 1), rep(14, 1))

## Family
PF <- PoisFamily()

## ML-estimate
MLEst <- MLEstimator(x, PF)
estimate(MLEst)
confint(MLEst)

## compute optimally robust estimator (unknown contamination)
nb1 <- gennbCtrl(eps.upper = 0.1)
robust <- robust(x, PF, nbCtrl = nb1, steps = 3)
estimate(robust)

confint(robust, symmetricBias())
plot(pIC(robust))
tmp <- qqplot(x, robust, cex.pch=1.5, exp.cex2.pch = -.25,
              exp.fadcol.pch = .55, jit.fac=.9)

## total variation neighborhoods (unknown deviation)
nb2 <- gennbCtrl(eps.upper = 0.05, neighbor = TotalVarNeighborhood())
robust1 <- robust(x, PF, nbCtrl = nb2, steps = 3)
estimate(robust1)
confint(robust1, symmetricBias())
plot(pIC(robust1))

###############################
## 3. Normal (Gaussian) location and scale
###############################
## 24 determinations of copper in wholemeal flour
library(MASS)
data(chem)
plot(chem, main = "copper in wholemeal flour", pch = 20)

## Family
NF <- NormLocationScaleFamily()
## ML-estimate
MLEst <- MLEstimator(chem, NF)
estimate(MLEst)
confint(MLest)

## Don't run to reduce check time on CRAN
## Not run:
## compute optimally robust estimator (known contamination)
## takes some time -> you can use package RobLox for normal
## location and scale which is optimized for speed
nb1 <- gennbCtrl(eps = 0.05)
robEst <- robest(chem, NF, nbCtrl = nb1, steps = 3)
estimate.call(robEst)
attr(robEst, "timings")
estimate(robest)

confint(robest, symmetricBias())
plot(pIC(robest))
## plot of relative and absolute information; cf. Kohl (2005)
infoPlot(pIC(robest))

tmp <- qqplot(chem, robest, cex.pch=1.5, exp.cex2.pch = -.25,
  exp.fadcol.pch = .55, withLab = TRUE, which.Order=1:4,
  exp.cex2.labl = .12,exp.fadcol.labl = .45,
  nosym.pCI = TRUE, adj.labl=c(1.7,..2),
  exact.pCI = FALSE, log ="xy")

## finite-sample correction
if(require(RobLox)){
  n <- length(chem)
  r <- 0.05*sqrt(n)
  r.f <- finiteSampleCorrection(n = n, r = r)
  fsCor0 <- r.f/r
  nb1 <- gennbCtrl(eps = 0.05)
  robest <- robest(chem, NF, nbCtrl = nb1, fsCor = fsCor0, steps = 3)
estimate(robest)
}

## compute optimally robust estimator (unknown contamination)
## takes some time -> use package RobLox!
nb2 <- gennbCtrl(eps.lower = 0.05, eps.upper = 0.1)
robest1 <- robest(chem, NF, nbCtrl = nb2, steps = 3)
estimate(robest1)
confint(robest1, symmetricBias())
plot(pIC(robest1))
## plot of relative and absolute information; cf. Kohl (2005)
infoPlot(pIC(robest1))

## End(Not run)
Description

Function to compute optimally robust estimates for L2-differentiable parametric families via k-step construction.

Usage

roptest(x, L2Fam, eps, eps.lower, eps.upper, fsCor = 1, initial.est, neighbor = ContNeighborhood(), risk = asMSE(), steps = 1L, distance = CvMDist, startPar = NULL, verbose = NULL, OptOrIter = "iterate", useLast = getRobAStBaseOption("kStepUseLast"), withUpdateInKer = getRobAStBaseOption("withUpdateInKer"), IC.UpdateInKer = getRobAStBaseOption("IC.UpdateInKer"), withICList = getRobAStBaseOption("withICList"), withPICList = getRobAStBaseOption("withPICList"), na.rm = TRUE, initial.est.ArgList, ..., withLogScale = TRUE, .withCheck = FALSE, withTimings = FALSE, withMDE = NULL, withEvalAsVar = NULL, withMakeIC = FALSE, modifyICwarn = NULL)

Arguments

x
sample

L2Fam
object of class "L2ParamFamily"

eps
positive real (0 < eps <= 0.5): amount of gross errors. See details below.

eps.lower
positive real (0 <= eps.lower <= eps.upper): lower bound for the amount of gross errors. See details below.

eps.upper
positive real (eps.lower <= eps.upper <= 0.5): upper bound for the amount of gross errors. See details below.

fsCor
positive real: factor used to correct the neighborhood radius; see details.

initial.est
initial estimate for unknown parameter. If missing minimum distance estimator is computed.

neighbor
object of class "UncondNeighborhood"

risk
object of class "RiskType"

steps
positive integer: number of steps used for k-steps construction.
distance function used in MDEstimator, which in turn is used as (default) starting estimator.

startPar initial information used by optimize resp. optim; i.e; if (total) parameter is of length 1, startPar is a search interval, else it is an initial parameter value; if NULL slot startPar of ParamFamily is used to produce it; in the multivariate case, startPar may also be of class Estimate, in which case slot untransformed.estimate is used.

verbose logical: if TRUE, some messages are printed

useLast which parameter estimate (initial estimate or k-step estimate) shall be used to fill the slots pIC, asvar and asbias of the return value.

OptOrIter character; which method to be used for determining Lagrange multipliers \( a \) and \( a' \): if (partially) matched to "optimize", getLagrangeMultByOptim is used; otherwise: by default, or if matched to "iterate" or to "doubleiterate", getLagrangeMultByIter is used. More specifically, when using getLagrangeMultByIter, and if argument risk is of class "asGRisk", by default and if matched to "iterate" we use only one (inner) iteration, if matched to "doubleiterate" we use up to Maxiter (inner) iterations.

withUpdateInKer
if there is a non-trivial trafo in the model with matrix \( D \), shall the parameter be updated on \( \ker(D) \)?

IC.UpdateInKer
if there is a non-trivial trafo in the model with matrix \( D \), the IC to be used for this; if NULL the result of getboundedIC(L2Fam, D) is taken; this IC will then be projected onto \( \ker(D) \).

withPICList logical: shall slot pICList of return value be filled?

withICList logical: shall slot ICList of return value be filled?

na.rm logical: if TRUE, the estimator is evaluated at complete.cases(x).

initial.est.ArgList
a list of arguments to be given to argument start if the latter is a function; this list by default already starts with two unnamed items, the sample \( x \), and the model L2Fam.

... further arguments

withLogScale logical: shall a scale component (if existing and found with name scalename) be computed on log-scale and backtransformed afterwards? This avoids crossing 0.

..withCheck logical: if TRUE, debugging info is issued.

withTimings logical: if TRUE, separate (and aggregate) timings for the three steps evaluating the starting value, finding the starting influence curve, and evaluating the k-step estimator is issued.

withMDE logical or NULL: Shall a minimum distance estimator be used as starting estimator—in addition to the function given in slot startPar of the L2 family? If NULL (default), the content of slot .withMDE in the L2 family is used instead to take this decision.

withEvalAsVar logical or NULL: if TRUE (default), tells R to evaluate the asymptotic variance or if FALSE just to produces a call to do so. If withEvalAsVar is NULL (default),
Computes the optimally robust estimator for a given L2 differentiable parametric family. The computation uses a k-step construction with an appropriate initial estimate; cf. also \texttt{kStepEstimator}. Valid candidates are e.g. Kolmogorov(-Smirnov) or von Mises minimum distance estimators (default); cf. Rieder (1994) and Kohl (2005).

Before package version 0.9, this computation was done with the code of function \texttt{roptest\_old} (with the same formals). From package version 0.9 on, this function uses the modularized function \texttt{robest} internally.

If the amount of gross errors (contamination) is known, it can be specified by \texttt{eps}. The radius of the corresponding infinitesimal contamination neighborhood is obtained by multiplying \texttt{eps} by the square root of the sample size.

If the amount of gross errors (contamination) is unknown, try to find a rough estimate for the amount of gross errors, such that it lies between \texttt{eps\_lower} and \texttt{eps\_upper}.

In case \texttt{eps\_lower} is specified and \texttt{eps\_upper} is missing, \texttt{eps\_upper} is set to 0.5. In case \texttt{eps\_upper} is specified and \texttt{eps\_lower} is missing, \texttt{eps\_lower} is set to 0.

If neither \texttt{eps} nor \texttt{eps\_lower} and/or \texttt{eps\_upper} is specified, \texttt{eps\_lower} and \texttt{eps\_upper} are set to 0 and 0.5, respectively.

If \texttt{eps} is missing, the radius-minimax estimator in sense of Rieder et al. (2001, 2008), respectively Section 2.2 of Kohl (2005) is returned.

Finite-sample and higher order results suggest that the asymptotically optimal procedure is liberal. Using \texttt{fsCor} the radius can be modified - as a rule enlarged - to obtain a more conservative estimate. In case of normal location and scale there is function \texttt{finiteSampleCorrection} which returns a finite-sample corrected (enlarged) radius based on the results of large Monte-Carlo studies.

The default value of argument \texttt{useLast} is set by the global option \texttt{kStepUseLast} which by default is set to FALSE. In case of general models \texttt{useLast} remains unchanged during the computations. However, if slot \texttt{callL2Fam} of \texttt{IC} generates an object of class "L2GroupParamFamily" the value of \texttt{useLast} is changed to TRUE. Explicitly setting \texttt{useLast} to TRUE should be done with care as in this situation the influence curve is re-computed using the value of the one-step estimate which may take quite a long time depending on the model.

If \texttt{useLast} is set to TRUE the computation of \texttt{asvar}, \texttt{asbias} and \texttt{IC} is based on the k-step estimate.

Value

Object of class "\texttt{kStepEstimate}". In addition, it has an attribute "\texttt{timings}" where computation time is stored.
Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>,
Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

References


See Also

`roblox.L2ParamFamily-class UncondNeighborhood-class RiskType-class`

Examples

```r
## Don't run to reduce check time on CRAN
## Not run:
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
## 1. Binomial data
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
## generate a sample of contaminated data
set.seed(123)
ind <- rbinom(100, size=1, prob=0.05)
x <- rbinom(100, size=25, prob=(1-ind)*0.25 + ind*0.9)

## ML-estimate
MLEst <- MLEstimator(x, BinomFamily(size = 25))
estimate(MLEst)
confint(MLEst)

## compute optimally robust estimator (known contamination)
robest1 <- roptest(x, BinomFamily(size = 25), eps = 0.05, steps = 3)
robest1.0 <- roptest.old(x, BinomFamily(size = 25), eps = 0.05, steps = 3)
identical(robest1,robest1.0)
estimate(robest1)
confint(robest1, method = symmetricBias())
## neglecting bias
confint(robest1)
```
plot(pIC(robtest1))
tmp <- qqplot(x, robtest1, cex.pch=1.5, exp.cex2.pch = -.25,
              exp.fadcol.pch = .55, jit.fac=.9)

## compute optimally robust estimator (unknown contamination)
robtest2 <- roptest(x, BinomFamily(size = 25), eps.lower = 0, eps.upper = 0.2, steps = 3)
estimate(robtest2)
confint(robtest2, method = symmetricBias())
plot(pIC(robtest2))

## total variation neighborhoods (known deviation)
robtest3 <- roptest(x, BinomFamily(size = 25), eps = 0.025,
                    neighbor = TotalVarNeighborhood(), steps = 3)
estimate(robtest3)
confint(robtest3, method = symmetricBias())
plot(pIC(robtest3))

## total variation neighborhoods (unknown deviation)
robtest4 <- roptest(x, BinomFamily(size = 25), eps.lower = 0, eps.upper = 0.1,
                    neighbor = TotalVarNeighborhood(), steps = 3)
estimate(robtest4)
confint(robtest4, method = symmetricBias())
plot(pIC(robtest4))

####################################
## 2. Poisson data
####################################
## Example: Rutherford-Geiger (1910); cf. Feller-(1968), Section VI.7 (a)
x <- c(rep(0, 57), rep(1, 203), rep(2, 383), rep(3, 525), rep(4, 532),
      rep(5, 408), rep(6, 273), rep(7, 139), rep(8, 45), rep(9, 27),
      rep(10, 10), rep(11, 4), rep(12, 0), rep(13, 1), rep(14, 1))

## ML-estimate
MLEst <- MLEstimator(x, PoisFamily())
estimate(MLEst)
confint(MLEst)

## compute optimally robust estimator (unknown contamination)
robtest <- roptest(x, PoisFamily(), eps.upper = 0.1, steps = 3)
estimate(robtest)
confint(robtest, symmetricBias())
plot(pIC(robtest))
tmp <- qqplot(x, robtest, cex.pch=1.5, exp.cex2.pch = -.25,
               exp.fadcol.pch = .55, jit.fac=.9)

## total variation neighborhoods (unknown deviation)
robtest1 <- roptest(x, PoisFamily(), eps.upper = 0.05,
                    neighbor = TotalVarNeighborhood(), steps = 3)
estimate(robtest1)
confint(robtest1, symmetricBias())
plot(pIC(robtest1))
## End(Not run)

### 3. Normal (Gaussian) location and scale

### 24 determinations of copper in wholemeal flour

```r
library(MASS)
data(chem)
plot(chem, main = "copper in wholemeal flour", pch = 20)
```

**ML-estimate**

```r
MLEst <- MaxLikEstimator(chem, NormLocationScaleFamily())
estimate(MLEst)
confint(MLEst)
```

**Don't run to reduce check time on CRAN**

**compute optimally robust estimator (known contamination)**

```r
robest <- roptest(chem, NormLocationScaleFamily(), eps = 0.05, steps = 3)
estimate(robest)
confint(robest, symmetricBias())
plot(pIC(robest))
```

**plot of relative and absolute information; cf. Kohl (2005)**

```r
infoPlot(pIC(robest))
```

**finite-sample correction**

```r
tmp <- qqplot(chem, robtest, cex.pch=1.5, exp.cex2.pch = -.25,
                exp.fadcol.pch = .55, withLab = TRUE, which.Order=1:4,
                exp.cex2.lbl = .12, exp.fadcol.lbl = .45,
                nosym.pCI = TRUE, adj.lbl=c(1.7,.2),
                exact.pCI = FALSE, log="xy")
```

**compute optimally robust estimator (unknown contamination)**

```r
robest <- roptest(chem, NormLocationScaleFamily(), eps = 0.05,
                   eps.lower = 0.05, eps.upper = 0.1, steps = 3)
estimate(robest)
confint(robest, symmetricBias())
plot(pIC(robest))
```

**plot of relative and absolute information; cf. Kohl (2005)**

```r
infoPlot(pIC(robest))
```
Methods for Function updateNorm in Package 'ROptEst'

Description
updateNorm-methods to update norm in IC-Algo

Usage
updateNorm(normtype, ...)  
## S4 method for signature 'SelfNorm'  
updateNorm(normtype, L2, neighbor, biastype, distr, V.comp,  
            cent, stand, w)

Arguments
- **normtype**
  - normtype of class NormType
- **...**
  - further arguments to be passed to specific methods.
- **L2**
  - L2derivative
- **neighbor**
  - object of class "Neighborhood".
- **biastype**
  - object of class "BiasType"
- **cent**
  - optimal centering constant.
- **stand**
  - standardizing matrix.
- **distr**
  - standardizing matrix.
- **V.comp**
  - matrix: indication which components of the standardizing matrix have to be computed.
- **w**
  - object of class RobWeight; current weight

Details
updateNorm is used internally in the opt-IC-algorithm to be able to work with a norm that depends on the current covariance (SelfNorm)

Value
- **updateNorm**
  - an updated object of class NormType.

Methods
- **updateNorm** signature(normtype = "SelfNorm"): updates the norm in the self-standardized case; just used internally in the opt-IC-Algorithm.
Author(s)

Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

See Also

NormType-class
Index

*Topic classes
   asAnscombe-class, 5
   asL1-class, 7
   asL4-class, 9
   get.asGRisk.fct-methods, 16
   getRiskFctBV-methods, 55
   getStartIC-methods, 57
   ORobEstimate-class, 70
   updateNorm-methods, 91

*Topic distribution
   plot-methods, 72

*Topic methods
   plot-methods, 72

*Topic package
   ROptEst-package, 3

*Topic robust
   asAnscombe, 4, 6
   asL1, 6, 8, 9
   asL4, 7, 8, 10
   cniperCont, 10
   comparePlot-methods, 15
   getAsRisk, 17
   getBiasIC, 22
   getFixClip, 23
   getFixRobIC, 24
   getIneffDiff, 26
   getInfCent, 28
   getInfClip, 30
   getInfGamma, 32
   getInfLM, 34
   getInfRad, 37
   getInfRobIC, 39
   getInfStand, 44
   getInfV, 45
   getL1normL2deriv, 47
   getL2normL2deriv, 48
   getMaxIneff, 49
   getModifyIC, 50
   getRadius, 52
   getReq, 53
   getRiskIC, 55
   inputGenerators, 59
   leastFavorableRadius, 61
   lowerCaseRadius, 62
   minmaxBias, 64
   optIC, 66
   optRisk, 69
   radiusMinimaxIC, 73
   RMXEOmseBore, 75
   robest, 80
   roptest, 84

asAnscombe, 4, 6
asAnscombe-class, 5
asl1, 6, 8, 9
asl4, 7, 8, 10
asmse, 7–10
cniperCont, 10
cniperPoint (cniperCont), 10
CniperPointPlot, 14
cniperPointPlot (cniperCont), 10
comparePlot (comparePlot-methods), 15
comparePlot, IC, IC-method
   (comparePlot-methods), 15
   comparePlot-methods, 15

eff (asAnscombe-class), 5
eff, asAnscombe-method
   (asAnscombe-class), 5
finiteSampleCorrection, 78, 87
genkStepCtrl, 81
genkStepCtrl (inputGenerators), 59
genbCtrl, 81
genbCtrl (inputGenerators), 59
genstartCtrl, 81
getasRisk (inputGenerators), 59

genstartCtrl, 81

genstartICCtrl (inputGenerators), 59

getaSGrisk.fct (getasRisk. fct-methods), 16

getaSGrisk.fct, asL1-method (getasRisk.fct-methods), 16

getaSGrisk.fct, asL4-method (getasRisk.fct-methods), 16

getaSGrisk.fct, asMSE-method (getasRisk.fct-methods), 16

getaSGrisk.fct-methods, 16

getaSGrisk-methods (getasRisk), 17

getaSGrisk, aScombe, RealRandVariable, ContNeighborhood, BiasType-method (getasRisk), 17

getaSGrisk, aScombe, UnivariateDistribution, UncondNeighborhood-method (getasRisk), 17

getaSGrisk, aSbias, RealRandVariable, ContNeighborhood-method (getasRisk), 17

getaSGrisk, aSbias, RealRandVariable, TotalVarNeighborhood-method (getasRisk), 17

getaSGrisk, aSbias, UnivariateDistribution, ContNeighborhood-method (getasRisk), 17

getaSGrisk, aSbias, UnivariateDistribution, TotalVarNeighborhood-method (getasRisk), 17

getaSGrisk, aScov, RealRandVariable, ContNeighborhood-method (getasRisk), 17

getaSGrisk, aScov, UnivariateDistribution, ContNeighborhood-method (getasRisk), 17

getaSGrisk, aScov, UnivariateDistribution, TotalVarNeighborhood-method (getasRisk), 17

getaSGrisk, aSl, UnivariateDistribution, Neighborhood, asymmetric-method (getasRisk), 17

getaSGrisk, aSl1, UnivariateDistribution, Neighborhood, onesided-method (getasRisk), 17

getaSGrisk, aSl4, UnivariateDistribution, Neighborhood, ANY-methods (getasRisk), 17

getaSGrisk, aSmse, EuclRandVariable, Neighborhood, numeric-method (getasRisk), 17

getaSGrisk, aSmse, UnivariateDistribution, Neighborhood, asMSE-method (getasRisk), 17

getaSGrisk, aSemivar, UnivariateDistribution, Neighborhood, ANY-method (getasRisk), 17

getaSGrisk, aUnivOvShoot, UnivariateDistribution, Neighborhood, ANY-method (getasRisk), 17

getaSGrisk, trAsCov, RealRandVariable, ContNeighborhood-method (getasRisk), 17

getBiasIC, 22

getaBiasIC, HampIC, UncondNeighborhood-method (getBiasIC), 22

getaBiasIC, TotalVarIC, UncondNeighborhood-method (getBiasIC), 22

getaBiasIC-methods (getBiasIC), 22

getaBiasIC, numeric, Norm, fiUnOvShoot, ContNeighborhood-method (getBiasIC), 23

getaBiasIC, numeric, Norm, fiUnOvShoot, TotalVarNeighborhood-method (getBiasIC), 23

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24

getaBiasIC, fiUnOvShoot, UncondNeighborhood-method (getBiasIC), 24
getInfClip, 30
(getInfClip), 30
getInfClip, numeric, UnivariateDistribution, asMSE, ContNeighborhood, BiasType, method
(getInfClip), 30
getInfClip, numeric, UnivariateDistribution, asMSE, TotalVarNeighborhood, BiasType, method
(getInfClip), 30
getInfClip, numeric, UnivariateDistribution, asSemivar, ContNeighborhood, BiasType, method
(getInfClip), 30
getInfClip, numeric, UnivariateDistribution, asUnOvShoot, ContNeighborhood, BiasType, method
(getInfClip), 30
getInfClip-methods (getInfClip), 30
getInfGamma, 32
(getInfGamma), 32
getInfGamma, RealRandVariable, asMSE, ContNeighborhood, BiasType, method
(getInfGamma), 32
getInfGamma, RealRandVariable, asMSE, TotalVarNeighborhood, BiasType, method
(getInfGamma), 32
getInfGamma, UnivariateDistribution, asBias, ContNeighborhood, BiasType, method
(getInfGamma), 32
getInfGamma, UnivariateDistribution, asGrisk, TotalVarNeighborhood, BiasType, method
(getInfGamma), 32
getInfGamma, UnivariateDistribution, asMSE, ContNeighborhood, BiasType, method
(getInfGamma), 32
getInfGamma, UnivariateDistribution, asMSE, TotalVarNeighborhood, BiasType, method
(getInfGamma), 32
getInfGamma-methods (getInfGamma), 32
getInfFM, 34
(getInfFM), 34
getInfRad, 37
(getInfRad), 37
getInfRad, numeric, EuclRandVariable, asMSE, UncondNeighborhood, BiasType, method
(getInfRad), 37
getInfRad, numeric, UnivariateDistribution, asCov, ContNeighborhood, BiasType, method
(getInfRad), 37
getInfRad, numeric, UnivariateDistribution, asGrisk, TotalVarNeighborhood, BiasType, method
(getInfRad), 37
getInfRad, numeric, UnivariateDistribution, asUnVShoot, ContNeighborhood, BiasType, method
(getInfRad), 37
getInfRad-methods (getInfRad), 37
getInfRobIC, 39
(getInfRobIC), 39
getInfRobIC, RealRandVariable, asAnscombe, UncondNeighborhood, BiasType, method
(getInfRobIC), 39
getInfRobIC, RealRandVariable, asBias, UncondNeighborhood, BiasType, method
(getInfRobIC), 39
getInfRobIC, RealRandVariable, asCov, UncondNeighborhood, BiasType, method
(getInfRobIC), 39
getInfRobIC, RealRandVariable, asGrisk, UncondNeighborhood, BiasType, method
(getInfRobIC), 39
getInfRobIC, RealRandVariable, asHampel, UncondNeighborhood, BiasType, method
(getInfRobIC), 39
getInfRobIC, UnivariateDistribution, asAnscombe, UncondNeighborhood, BiasType, method
(getInfRobIC), 39
getInfRobIC, UnivariateDistribution, asBias, UncondNeighborhood, BiasType, method
(getInfRobIC), 39
getInfRobIC, UnivariateDistribution, asCov, UncondNeighborhood, BiasType, method
(getInfRobIC), 39
getInfRobIC, UnivariateDistribution, asGrisk, UncondNeighborhood, BiasType, method
(getInfRobIC), 39
getInfRobIC, UnivariateDistribution, asHampel, UncondNeighborhood, BiasType, method
(getInfRobIC), 39
getInfRobIC, UnivariateDistribution, asUnOvShoot, UncondNeighborhood, BiasType, method
(getInfRobIC), 39
getInfStand, 44
(getInfStand), 44
getInfStand, RealRandVariable, UncondNeighborhood, BiasType, method
(getInfStand), 44
getInfStand, RealRandVariable, UncondNeighborhood, BiasType, method
(getInfStand), 44
getInfStand, UnivariateDistribution, ContNeighborhood, asymmetric
(getInfStand), 44
getInfStand, UnivariateDistribution, ContNeighborhood, BiasType, method
(getInfStand), 44
getInfStand, UnivariateDistribution, ContNeighborhood, onesidedBias
(getInfStand), 44
getInfStand, UnivariateDistribution, TotalVarNeighborhood, BiasType, method
(getInfStand), 44
getInfV, 45, 56
(getInfV), 45
getInfV, numeric, UnivariateDistribution, asUnOvShoot, UncondNeighborhood, BiasType, method
(getInfV), 45
getInfV, UnivariateDistribution, asUnVShoot, UncondNeighborhood, BiasType, method
(getInfV), 45
getInfV, UnivariateDistribution, asUnVShoot, UncondNeighborhood, BiasType, method
(getInfV), 45
getInfV, UnivariateDistribution, ContNeighborhood, onesidedBias
(getInfV), 45
getInfV, UnivariateDistribution, ContNeighborhood, onesidedBias
(getInfV), 45

INDEX

getInfV, UnivariateDistribution, TotalVarNeighborhood, Biastype-method (getInfV), 45
getInfV-methods (getInfV), 45
getLnormL2deriv, 47
getLnormL2deriv, RealRandVariable-method (getLnormL2deriv), 47
getLnormL2deriv, UnivariateDistribution-method (getLnormL2deriv), 47
getLnormL2deriv-methods (getLnormL2deriv), 47
get2normL2deriv, 48
getLagrangeMultByIter (getInfLM), 34
getLagrangeMultByOptim (getInfLM), 34
getMaxIneff, 49
getModifyIC, 50
getModifyIC, L2LocationFamily, UncondNeighborhood-method (getModifyIC), 50
getModifyIC, L2LocationFamily, UncondNeighborhood, fiUnOvShoot-method (getModifyIC), 50
getModifyIC, L2LocationScaleFamily, UncondNeighborhood-method (getModifyIC), 50
getModifyIC, L2ParamFamily, Neighborhood-method (getModifyIC), 50
getModifyIC, L2ScaleFamily, UncondNeighborhood-method (getModifyIC), 50
getModifyIC-methods (getModifyIC), 50
getRadius, 37, 52
getReq, 16, 53
getRiskFctBV (getRiskFctBV-methods), 55
getRiskFctBV, asL1, ANY-method (getRiskFctBV-methods), 55
getRiskFctBV, asL4, ANY-method (getRiskFctBV-methods), 55
getRiskFctBV-methods, 55
getRiskIC, 55, 57
getRiskIC, HampIC, asCov, missing, L2ParamFamily-method (getRiskIC), 55
getRiskIC, HampIC, asCov, missing, missing-method (getRiskIC), 55
getRiskIC, TotalVarIC, asCov, missing, L2ParamFamily-method (getRiskIC), 55
getRiskIC-methods (getRiskIC), 55
getStartIC (getStartIC-methods), 57
getStartIC, ANY, ANY-method (getStartIC-methods), 57
getStartIC, L2ParamFamily, asAnscombe-method (getStartIC-methods), 57
getStartIC, L2ParamFamily, asBias-method (getStartIC-methods), 57
getStartIC-methods (getStartIC), 57
ICList (ORobEstimate-class), 70
ICList, ORobEstimate-method (ORobEstimate-class), 70
inputGenerators, 59
kStepEstimator, 78, 87
ksteps (ORobEstimate-class), 70
ksteps, ORobMethod (ORobEstimate-class), 70
leastFavorableRadius, 27, 61
leastFavorableRadius, L2ParamFamily, UncondNeighborhood, asGRisk-method (leastFavorableRadius), 61
leastFavorableRadius-methods (leastFavorableRadius), 61
lowerCaseRadius, 62
lowerCaseRadius, L2ParamFamily, ContNeighborhood, asMSE, ANY-method (lowerCaseRadius), 62
lowerCaseRadius, L2ParamFamily, ContNeighborhood, asMSE, asymmetric-method (lowerCaseRadius), 62
lowerCaseRadius, L2ParamFamily, ContNeighborhood, asMSE, onesided-method (lowerCaseRadius), 62
lowerCaseRadius, L2ParamFamily, TotalVarNeighborhood, asMSE, ANY-method (lowerCaseRadius), 62
lowerCaseRadius, UnivariateDistribution, ContNeighborhood-method (lowerCaseRadius), 62
lowerCaseRadius-methods (lowerCaseRadius), 62
MBREstimat or (RMEOSENMBROBRE), 75
minmaxBias, 64
minmaxBias, RealRandVariable, ContNeighborhood, BiasType-method (minmaxBias), 64
minmaxBias, RealRandVariable, TotalVarNeighborhood, BiasType-method (minmaxBias), 64
minmaxBias, UnivariateDistribution, ContNeighborhood, asymm etric-method (minmaxBias), 64
minmaxBias, UnivariateDistribution, ContNeighborhood, onesided-method (minmaxBias), 64
minmaxBias, UnivariateDistribution, ContNeighborhood, onesided-method (minmaxBias), 64
INDEX

minmaxBias, UnivariateDistribution, TotalVarNeighborhood (minmaxBias), 64
minmaxBias-methods (minmaxBias), 64

OBREstimato (RMXEMSEMBREOBRE), 75
OMSEstimator (RMXEMSEMBREOBRE), 75
optIC, 52, 58, 66
optIC, FixRobModel, fiUnOvShoot-method (optIC), 66
optIC, InfRobModel, asRisk-method (optIC), 66
optIC, InfRobModel, asUnOvShoot-method (optIC), 66
optIC-methods (optIC), 66
optRisk, 69
optRisk, FixRobModel, fiUnOvShoot-method (optRisk), 69
optRisk, InfRobModel, asRisk-method (optRisk), 69
optRisk, L2ParamFamily, asCov-method (optRisk), 69
optRisk-methods (optRisk), 69
ORobEstimate-class, 70

pIClist (ORobEstimate-class), 70
pIClist, ORobEstimate-method (ORobEstimate-class), 70
plot (plot-methods), 72
plot, IC, missing-method (plot-methods), 72
plot-methods, 72

radiusMinimaxIC, 27, 58, 62, 73, 75
radiusMinimaxIC, L2ParamFamily, UncondNeighborhood, asGRisk-method (radiusMinimaxIC), 73
radiusMinimaxIC-methods (radiusMinimaxIC), 73
RMXEMSEMBREOBRE, 75
RMXEstimator (RMXEMSEMBREOBRE), 75
robest, 58, 60, 79, 80, 87
robestCall (ORobEstimate-class), 70
robtestCall, ORobEstimate-method (ORobEstimate-class), 70
roblox, 60, 79, 82, 88
ROptEst (ROptEst-package), 3
roptest, 79, 81, 84
ROptEst-package, 3
roptestCall (ORobEstimate-class), 70

scaleUpdateIC (getModifyIC), 50
scaleUpdateIC, ContNeighborhood-method (getModifyIC), 50
scaleUpdateIC, TotalVarNeighborhood-method (getModifyIC), 50
scaleUpdateIC, UncondNeighborhood-method (getModifyIC), 50
scaleUpdateIC-methods (getModifyIC), 50
show, asAnscombe-method (asAnscombe-class), 5
show, ORobEstimate-method (ORobEstimate-class), 70
start, ORobEstimate-method (ORobEstimate-class), 70
startval (ORobEstimate-class), 70
startval, ORobEstimate-method (ORobEstimate-class), 70
steps (ORobEstimate-class), 70
steps, ORobEstimate-method (ORobEstimate-class), 70
uksteps (ORobEstimate-class), 70
uksteps, ORobEstimate-method (ORobEstimate-class), 70
updateNorm (updateNorm-methods), 91
updateNorm, SelfNorm-method (updateNorm-methods), 91
updateNorm-methods, 91
ustartval (ORobEstimate-class), 70
ustartval, ORobEstimate-method (ORobEstimate-class), 70
updatenorm (updatenorm-methods), 91
updatenorm, selfnorm-method (updatenorm-methods), 91
updatenorm-methods, 91

uksteps (ORobEstimate-class), 70
uksteps, ORobEstimate-method (ORobEstimate-class), 70
updateNorm (updateNorm-methods), 91
updateNorm, SelfNorm-method (updateNorm-methods), 91
updateNorm-methods, 91
ustartval (ORobEstimate-class), 70
ustartval, ORobEstimate-method (ORobEstimate-class), 70
updatenorm (updatenorm-methods), 91
updatenorm, selfnorm-method (updatenorm-methods), 91
updatenorm-methods, 91

uksteps (ORobEstimate-class), 70
uksteps, ORobEstimate-method (ORobEstimate-class), 70
updateNorm (updateNorm-methods), 91
updateNorm, SelfNorm-method (updateNorm-methods), 91
updateNorm-methods, 91
ustartval (ORobEstimate-class), 70
ustartval, ORobEstimate-method (ORobEstimate-class), 70
updatenorm (updatenorm-methods), 91
updatenorm, selfnorm-method (updatenorm-methods), 91
updatenorm-methods, 91

uksteps (ORobEstimate-class), 70
uksteps, ORobEstimate-method (ORobEstimate-class), 70
updateNorm (updateNorm-methods), 91
updateNorm, SelfNorm-method (updateNorm-methods), 91
updateNorm-methods, 91
ustartval (ORobEstimate-class), 70
ustartval, ORobEstimate-method (ORobEstimate-class), 70
updatenorm (updatenorm-methods), 91
updatenorm, selfnorm-method (updatenorm-methods), 91
updatenorm-methods, 91

uksteps (ORobEstimate-class), 70
uksteps, ORobEstimate-method (ORobEstimate-class), 70
updateNorm (updateNorm-methods), 91
updateNorm, SelfNorm-method (updateNorm-methods), 91
updateNorm-methods, 91
ustartval (ORobEstimate-class), 70
ustartval, ORobEstimate-method (ORobEstimate-class), 70
updatenorm (updatenorm-methods), 91
updatenorm, selfnorm-method (updatenorm-methods), 91
updatenorm-methods, 91