

# Package ‘RobAStBase’

July 2, 2014

**Version** 0.9

**Date** 2013-09-11

**Title** Robust Asymptotic Statistics

**Description** Base S4-classes and functions for robust asymptotic statistics.

**Depends** R(>= 2.14.0), methods, rrcov, distr(>= 2.5.2), distrEx(>= 2.4), distrMod(>= 2.5.2), RandVar(>= 0.9.2)

**Suggests** ROptEst, RUnit (>= 0.4.26)

**Author** Matthias Kohl, Peter Ruckdeschel

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

**ByteCompile** yes

**LazyLoad** yes

**License** LGPL-3

**Encoding** latin1

**URL** <http://robast.r-forge.r-project.org/>

**LastChangedDate** {\$LastChangedDate: 2013-09-12 11:37:28 +0200 (Do, 12. Sep 2013) \$}

**LastChangedRevision** {\$LastChangedRevision: 701 \$}

**SVNRevision** 694

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2013-09-13 17:40:49

**R topics documented:**

RobAStBase-package . . . . .	3
ALEstimate-class . . . . .	4
BdStWeight-class . . . . .	6
BoundedWeight-class . . . . .	7
checkIC . . . . .	8
ComparePlot . . . . .	9
comparePlot-methods . . . . .	10
ContIC . . . . .	15
ContIC-class . . . . .	16
ContNeighborhood . . . . .	18
ContNeighborhood-class . . . . .	19
cutoff . . . . .	20
cutoff-class . . . . .	21
ddPlot-methods . . . . .	22
evalIC . . . . .	26
FixRobModel . . . . .	27
FixRobModel-class . . . . .	28
generateIC . . . . .	29
generateIC.fct-methods . . . . .	30
getBiasIC . . . . .	31
getBoundedIC . . . . .	32
getRiskFctBV-methods . . . . .	33
getRiskIC . . . . .	33
getweight-methods . . . . .	36
HampelWeight-class . . . . .	38
HampIC-class . . . . .	39
IC . . . . .	40
IC-class . . . . .	42
InfluenceCurve . . . . .	43
InfluenceCurve-class . . . . .	44
InfoPlot . . . . .	46
infoPlot . . . . .	47
InfRobModel . . . . .	51
InfRobModel-class . . . . .	52
interpolRisk-class . . . . .	53
kStepEstimate-class . . . . .	54
kStepEstimator . . . . .	56
kStepEstimator.start-methods . . . . .	59
locMEstimator . . . . .	60
makeIC-methods . . . . .	62
masked-methods . . . . .	64
MEstimate-class . . . . .	65
movToRef-methods . . . . .	66
Neighborhood-class . . . . .	67
oneStepEstimator . . . . .	68
optIC . . . . .	70

OptionalInfluenceCurve-class . . . . .	71
outlyingPlotIC . . . . .	72
plot-methods . . . . .	74
PlotIC . . . . .	78
qqplot . . . . .	79
RobAStBaseMASK . . . . .	81
RobAStBaseOptions . . . . .	82
RobAStControl-class . . . . .	83
RobModel-class . . . . .	84
RobWeight-class . . . . .	85
samplesize-methods . . . . .	86
TotalVarIC . . . . .	86
TotalVarIC-class . . . . .	88
TotalVarNeighborhood . . . . .	90
TotalVarNeighborhood-class . . . . .	91
UncondNeighborhood-class . . . . .	92

## Index 93

---

RobAStBase-package      *Robust Asymptotic Statistics*

---

### Description

Base S4-classes and functions for robust asymptotic statistics.

### Details

Package:            RobAStBase  
Version:            0.9  
Date:                2013-09-11  
Depends:           R(>= 2.12.0), methods, distr(>= 2.0), distrEx(>= 2.0), distrMod(>= 2.0), RandVar(>= 0.6.3)  
LazyLoad:          yes  
License:            LGPL-3  
URL:                <http://robast.r-forge.r-project.org/>  
SVNRevision:      694

### Package versions

Note: The first two numbers of package versions do not necessarily reflect package-individual development, but rather are chosen for the RobAStXXX family as a whole in order to ease updating "depends" information.

**Author(s)**

Peter Ruckdeschel <Peter.Ruckdeschel@itwm.fraunhofer.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

Maintainer: Matthias Kohl <matthias.kohl@stamats.de>

**References**

M. Kohl (2005). Numerical Contributions to the Asymptotic Theory of Robustness. Dissertation. University of Bayreuth.

**See Also**

[distr-package](#), [distrEx-package](#), [distrMod-package](#)

**Examples**

```
library(RobAStBase)

## some L2 differentiable parametric family from package distrMod, e.g.
B <- BinomFamily(size = 25, prob = 0.25)

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

---

ALEstimate-class	<i>ALEstimate-class.</i>
------------------	--------------------------

---

**Description**

Class of asymptotically linear estimates.

**Objects from the Class**

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

**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  
**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.

### Extends

Class "Estimate", directly.

### Methods

**pIC** signature(object = "ALEstimate"): accessor function for slot pIC.  
**show** signature(object = "ALEstimate")  
**confint** signature(object = "ALEstimate", method = "missing"): compute asymptotic (LAN-based) confidence interval neglecting any bias.  
**confint** signature(object = "ALEstimate", method = "symmetricBias"): compute asymptotic (LAN-based) confidence interval incorporating bias symmetrically.  
**confint** signature(object = "ALEstimate", method = "onesidedBias"): compute asymptotic (LAN-based) confidence interval incorporating bias one-sided; i.e., positive or negative, respectively.  
**confint** signature(object = "ALEstimate", method = "asymmetricBias"): compute asymptotic (LAN-based) confidence interval incorporating bias asymmetrically.

### Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

### See Also

[Estimate-class](#)

### Examples

```
## prototype
new("ALEstimate")
```

---

BdStWeight-class      *Robust Weight classes for bounded, standardized weights*

---

### Description

Classes for bounded, robust, standardized weights.

### Objects from the Class

Objects can be created by calls of the form `new("BdStWeight", ...)`; to fill slot `weight`, you will use the generating functions `getweight` and `minbiasweight`.

### Slots

`name` Object of class "character"; inherited from class `RobWeight`.

`weight` Object of class "function" — the weight function; inherited from class `RobWeight`.

`clip` Object of class "numeric" — clipping bound(s); inherited from class `BoundedWeight`.

`stand` Object of class "matrix" — standardization.

### Extends

Class "RobWeight", via class "BoundedWeight". Class "BoundedWeight", directly.

### Methods

**stand** signature(object = "BdStWeight"): accessor function for slot `stand`.

**stand<-** signature(object = "BdStWeight", value = "matrix"): replacement function for slot `stand`. This replacement method should be used with great care, as the slot `weight` is not simultaneously updated and hence, this may lead to inconsistent objects.

### Author(s)

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

### References

Hampel et al. (1986) *Robust Statistics*. The Approach Based on Influence Functions. New York: Wiley.

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.

Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

### See Also

[BoundedWeight-class](#), [RobWeight-class](#), [IC](#), [InfluenceCurve-class](#)

**Examples**

```
## prototype  
new("BdStWeight")
```

---

BoundedWeight-class     *Robust Weight classes for bounded weights*

---

**Description**

Classes for bounded, robust weights.

**Objects from the Class**

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

**Slots**

`name` Object of class "character"; inherited from class `RobWeight`.

`weight` Object of class "function" — the weight function; inherited from class `RobWeight`.

`clip` Object of class "numeric" — clipping bound(s).

**Extends**

Class "RobWeight", directly.

**Methods**

`clip` signature(`x1` = "BoundedWeight"): accessor function for slot `clip`.

`clip<-` signature(`object` = "BoundedWeight", `value` = "numeric"): replacement function for slot `clip`. This replacement method should be used with great care, as the slot `weight` is not simultaneously updated and hence, this may lead to inconsistent objects.

**Author(s)**

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

**References**

Hampel et al. (1986) *Robust Statistics*. The Approach Based on Influence Functions. New York: Wiley.

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.

Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[RobWeight-class](#), [IC](#), [InfluenceCurve-class](#)

**Examples**

```
## prototype  
new("BoundedWeight")
```

---

`checkIC`*Generic Function for Checking ICs*

---

**Description**

Generic function for checking centering and Fisher consistency of ICs.

**Usage**

```
checkIC(IC, L2Fam, ...)
```

**Arguments**

IC	object of class "IC"
L2Fam	L2-differentiable family of probability measures.
...	additional parameters

**Details**

The precisions of the centering and the Fisher consistency are computed.

**Value**

The maximum deviation from the IC properties is returned.

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.  
Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[L2ParamFamily-class](#), [IC-class](#)

**Examples**

```
IC1 <- new("IC")  
checkIC(IC1)
```



---

ComparePlot

*Wrapper function for function comparePlot*


---

### Description

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

### Usage

```
ComparePlot(IC1, IC2, y, ..., IC3 = NULL, IC4 = NULL,
            alpha.trsp = 100, with.legend = TRUE, rescale = FALSE,
            withCall = TRUE)
```

### Arguments

IC1	object of class IC
IC2	object of class IC
IC3	object of class IC
IC4	object of class IC
y	optional data argument — for plotting observations into the plot
...	additional parameters (in particular to be passed on to plot)
alpha.trsp	the transparency argument (0 to 100) for plotting the data
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 comparePlot with suitably chosen defaults; if withCall == TRUE, the call to comparePlot is returned.

### Examples

```
# Gamma
fam <- GammaFamily()
rfam <- InfRobModel(fam, ContNeighborhood(0.5))
IC1 <- optIC(model = fam, risk = asCov())
IC2 <- makeIC(list(function(x)sin(x),function(x)x^2), L2Fam = fam)
Y <- distribution(fam)
y <- r(Y)(100)
ComparePlot(IC1, IC2, y, withCall = TRUE)
```

---

comparePlot-methods    *Compare - Plots*

---

## Description

Plots 2-4 influence curves to the same model.

## Usage

```
comparePlot(obj1, obj2, ... )
## S4 method for signature 'IC,IC'
comparePlot(obj1, obj2, obj3 = NULL, obj4 = NULL,
  data = NULL, ..., withSweave = getdistrOption("withSweave"),
  forceSameModel = FALSE,
  main = FALSE, inner = TRUE, sub = FALSE,
  col = par("col"), lwd = par("lwd"), lty,
  col.inner = par("col.main"), cex.inner = 0.8,
  bmar = par("mar")[1], tmar = par("mar")[3],
  with.legend = FALSE, legend = NULL, legend.bg = "white",
  legend.location = "bottomright", legend.cex = 0.8,
  withMBR = FALSE, MBRB = NA, MBR.fac = 2, col.MBR = par("col"),
  lty.MBR = "dashed", lwd.MBR = 0.8,
  scaleX = FALSE, scaleX.fct = scaleX.inv,
  scaleY = FALSE, scaleY.fct = pnorm, scaleY.inv=qnorm,
  scaleN = 9, x.ticks = NULL, y.ticks = NULL,
  mfColRow = TRUE, to.draw.arg = NULL,
  cex.pts = 1, col.pts = par("col"),
  pch.pts = 1, jitter.fac = 1, with.lab = FALSE,
  lab.pts = NULL, lab.font = NULL, alpha.trsp = NA,
  which.lbs = NULL, which.Order = NULL, return.Order = FALSE)
```

## Arguments

obj1	object of class "InfluenceCurve"
obj2	object of class "InfluenceCurve" to be compared with obj1
obj3	optional: object of class "InfluenceCurve" to be compared with obj1
obj4	optional: object of class "InfluenceCurve" to be compared with obj1
data	optional data argument — for plotting observations into the plot;
withSweave	logical: if TRUE (for working with Sweave) no extra device is opened
forceSameModel	logical; shall we check / enforce that the model of the ICs obj1, obj2, obj3, and obj4 be the same?
main	logical: is a main title to be used? or just as argument main in <a href="#">plot.default</a> .
col	color[s] of ICs in arguments obj1 [...,obj4].

lwd	linewidth[s] of ICs in arguments obj1 [...,obj4].
lty	line-type[s] of ICs in arguments obj1 [...,obj4].
inner	logical: do panels have their own titles? or character vector of / cast to length 'number of plotted dimensions'; if argument to.draw.arg is used, this refers to a vector of length length(to.draw.arg), the actually plotted dimensions. For further information, see also description of argument main in <a href="#">plot.default</a> .
sub	logical: is a sub-title to be used? or just as argument sub in <a href="#">plot.default</a> .
tmar	top margin – useful for non-standard main title sizes
bmar	bottom margin – useful for non-standard sub title sizes
cex.inner	magnification to be used for inner titles relative to the current setting of cex; as in <a href="#">par</a>
col.inner	character or integer code; color for the inner title
with.legend	logical; shall a legend be plotted?
legend	either NULL or a list of length (number of plotted panels) of items which can be used as argument legend in command legend.
legend.location	a valid argument x for <a href="#">legend</a> — the place where to put the legend on the last issued plot
legend.bg	background color for the legend
legend.cex	magnification factor for the legend
withMBR	logical; shall horizontal lines with min and max of MBRE be plotted for comparison?
MBRB	matrix (or NA); coerced by usual recycling rules to a matrix with as many rows as plotted panels and with first column the lower bounds and the second column the upper bounds for the respective coordinates (ideally given by the MBR-IC).
MBR.fac	positive factor; scales the bounds given by argument MBRB
col.MBR	color for the MBR lines; as usual col-argument;
lty.MBR	line type for the MBR lines; as usual lty-argument;
lwd.MBR	line width for the MBR lines; as usual lwd-argument;
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 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 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 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; 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); can be a list with one (numeric or NULL) item per panel
mfColRow	shall default partition in panels be used — defaults to TRUE
to.draw.arg	Either NULL (default; everything is plotted) or a vector of either integers (the indices of the subplots to be drawn) or characters — the names of the subplots to be drawn: these names are to be chosen either among the row names of the trafo matrix <code>rownames(trafo(eval(obj1@CallL2Fam)@param))</code> or if the last expression is NULL a vector " <code>dim&lt;dimnr&gt;</code> ", <code>dimnr</code> running through the number of rows of the trafo matrix.
cex.pts	size of the points of the data argument plotted
col.pts	color of the points of the data argument plotted
pch.pts	symbol of the points of the data argument plotted
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 <code>col.pch</code> and <code>col.lbl</code> ; if one-dim and NA all colors are left unchanged. Otherwise, with usual recycling rules <code>alpha.trsp</code> gets shorted/prolongated to length the data-symbols to be plotted. Coordinates of this vector <code>alpha.trsp</code> with NA are left unchanged, while for the remaining ones, the alpha channel in rgb space is set to the respective coordinate value of <code>alpha.trsp</code> . The non-NA entries must be integers in [0,255] (0 invisible, 255 opaque).
jitter.fac	jittering factor used in case of a <code>DiscreteDistribution</code> for plotting points of the data 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.Order	for each of the given ICs, we order the observations (descending) according to the norm given by the corresponding <code>normtype(object)</code> ; then <code>which.Order</code> either is an integer vector with the indices of the <i>ordered</i> observations (remaining after a possible reduction by argument <code>which.lbs</code> ) to be plotted into graph or NULL — then no (further) observation is excluded.

return.Order logical; if TRUE, a list of length maximally four with order vectors is returned — one for the ordering w.r.t. each of the given ICs; 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.

... further arguments to be passed to plot

## Details

Any parameters of plot.default may be passed on to this particular plot method.

For main-, inner, and subtitles given as arguments main, inner, and sub, top and bottom margins are enlarged to 5 resp. 6 by default but may also be specified by tmar / bmar arguments. If main / inner / sub are logical then if the respective argument is FALSE nothing is done/plotted, but if it is TRUE, we use a default main title taking up the calling arguments in case of main, default inner titles taking up the class and (named) parameter slots of arguments in case of inner, and a "generated on <data>"-tag in case of sub. Of course, if main / inner / sub are character, this is used for the title; in case of inner it is then checked whether it has correct length. In all title arguments, the following patterns are substituted:

"%C1", "%C2", ["%C3", ["%C4"]] class of argument obj<i>, i=1,..4

"%A1", "%A2", ["%A3", ["%A4"]] deparsed argument obj<i>, i=1,..4

"%D" time/date-string when the plot was generated

If argument ... contains argument ylim, this may either be as in plot.default (i.e. a vector of length 2) or a vector of length 2\*(number of plotted dimensions); in the case of longer length, these are the values for ylim for the plotted dimensions of the IC, one pair for each dimension.

## Author(s)

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

## References

Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

## See Also

[L2ParamFamily-class](#), [IC-class](#), [plot](#)

## Examples

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

  IC1 <- optIC(model = N0, risk = asCov())
  IC2 <- optIC(model = N0.Rob1, risk = asMSE())
}
```

```

comparePlot(IC1,IC2)

set.seed(12); data <- r(N0)(20)
comparePlot(IC1, IC2, data=data, with.lab = TRUE,
            which.lbs = c(1:4,15:20),
            which.Order = 1:6,
            return.Order = TRUE)

## don't run to reduce check time on CRAN
## Not run:
## selection of subpanels for plotting
par(mfrow=c(1,1))
comparePlot(IC1, IC2 ,mfColRow = FALSE, to.draw.arg=c("mean"),
            panel.first= grid(),ylim=c(-4,4),xlim=c(-6,6))
## matrix-valued ylim
comparePlot(IC1, IC2, panel.first= grid(),ylim=c(-4,4,0,4),xlim=c(-6,6))

x <- c(data,-12,10)
comparePlot(IC1, IC2, data=x, which.Order=10,
            panel.first= grid(), ylim=c(-4,4,0,4), xlim=c(-6,6))

Y <- Chisq(df=1)* DiscreteDistribution(c(-1,1))
comparePlot(IC1, IC2, data=x, which.Order=10,
            scaleX = TRUE, scaleX.fct=pnorm, scaleX.inv=qnorm,
            scaleY = TRUE, scaleY.fct=p(Y), scaleY.inv=q(Y),
            panel.first= grid(), ylim=c(-4,4,0,4), xlim=c(-6,6))
comparePlot(IC1, IC2, data=x, which.Order=10,
            scaleX = TRUE, scaleX.fct=pnorm, scaleX.inv=qnorm,
            scaleY = TRUE, scaleY.fct=p(Y), scaleY.inv=q(Y),
            x.ticks = c(-Inf, -10, -1,0,1,10,Inf),
            y.ticks = c(-Inf, -5, -1,0,1,5,Inf),
            panel.first= grid(), ylim=c(-4,4,0,4), xlim=c(-6,6))

## with use of trafo-matrix:
G <- GammaFamily(scale = 1, shape = 2)
## explicitly transforming to
## MASS parametrization:
mtrafo <- function(x){
  nms0 <- names(c(main(param(G)),nuisance(param(G))))
  nms <- c("shape","rate")
  fval0 <- c(x[2], 1/x[1])
  names(fval0) <- nms
  mat0 <- matrix( c(0, -1/x[1]^2, 1, 0), nrow = 2, ncol = 2,
                  dimnames = list(nms,nms0))
  list(fval = fval0, mat = mat0)}
G2 <- G
trafo(G2) <- mtrafo
G2
G2.Rob1 <- InfRobModel(center = G2, neighbor = ContNeighborhood(radius = 0.5))
system.time(IC1 <- optIC(model = G2, risk = asCov()))
system.time(IC2 <- optIC(model = G2.Rob1, risk = asMSE()))
system.time(IC2.i <- optIC(model = G2.Rob1, risk = asMSE(normtype=InfoNorm())))

```

```

system.time(IC2.s <- optIC(model = G2.Rob1, risk = asMSE(normtype=SelfNorm()))
comparePlot(IC1,IC2, IC2.i, IC2.s)

## End(Not run)

}

```

---

ContIC

*Generating function for ContIC-class*


---

### Description

Generates an object of class "ContIC"; i.e., an influence curves  $\eta$  of the form

$$\eta = (A\Lambda - a) \min(1, b/|A\Lambda - a|)$$

with clipping bound  $b$ , centering constant  $a$  and standardizing matrix  $A$ .  $\Lambda$  stands for the L2 derivative of the corresponding L2 differentiable parametric family which can be created via `CallL2Fam`.

### Usage

```

ContIC(name, CallL2Fam = call("L2ParamFamily"),
       Curve = EuclRandVarList(RealRandVariable(Map = c(function(x){x}),
                                                Domain = Reals())),
       Risks, Infos, clip = Inf, cent = 0, stand = as.matrix(1),
       lowerCase = NULL, neighborRadius = 0, w = new("HampelWeight"),
       normtype = NormType(), biastype = symmetricBias(),
       modifyIC = NULL)

```

### Arguments

<code>name</code>	object of class "character".
<code>CallL2Fam</code>	object of class "call": creates an object of the underlying L2-differentiable parametric family.
<code>Curve</code>	object of class "EuclRandVarList"
<code>Risks</code>	object of class "list": list of risks; cf. <a href="#">RiskType-class</a> .
<code>Infos</code>	matrix of characters with two columns named <code>method</code> and <code>message</code> : additional informations.
<code>clip</code>	positive real: clipping bound.
<code>cent</code>	real: centering constant
<code>stand</code>	matrix: standardizing matrix
<code>w</code>	HampelWeight: weight object
<code>lowerCase</code>	optional constant for lower case solution.
<code>neighborRadius</code>	radius of the corresponding (unconditional) contamination neighborhood.

biastype	BiasType: type of the bias
normtype	NormType: type of the norm
modifyIC	object of class "OptionalFunction": function of two arguments, which are an L2 parametric family and an optional influence curve. Returns an object of class "IC". This function is mainly used for internal computations!

**Value**

Object of class "ContIC"

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

- Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.
- Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[IC-class](#), [ContIC](#), [HampIC-class](#)

**Examples**

```
IC1 <- ContIC()
plot(IC1)
```

---

ContIC-class

*Influence curve of contamination type*

---

**Description**

Class of (partial) influence curves of contamination type; i.e., influence curves  $\eta$  of the form

$$\eta = (A\Lambda - a) \min(1, b/|A\Lambda - a|)$$

with clipping bound  $b$ , centering constant  $a$  and standardizing matrix  $A$ .  $\Lambda$  stands for the L2 derivative of the corresponding L2 differentiable parametric family created via the call in the slot CallL2Fam.

**Objects from the Class**

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



**Slots**

**CallL2Fam**: object of class "call": creates an object of the underlying L2-differentiable parametric family.

**name**: object of class "character"

**Curve**: object of class "EuclRandVarList"

**modifyIC**: Object of class "OptionalFunction": function of two arguments, which are an L2 parametric family and an optional influence curve. Returns an object of class "IC". This slot is mainly used for internal computations!

**Risks**: object of class "list": list of risks; cf. [RiskType-class](#).

**Infos**: object of class "matrix" with two columns named method and message: additional informations.

**clip**: object of class "numeric": clipping bound.

**cent**: object of class "numeric": centering constant.

**stand**: object of class "matrix": standardizing matrix.

**weight**: object of class "HampelWeight": weight function

**biastype**: object of class "BiasType": bias type (symmetric/onsided/asymmetric)

**normtype**: object of class "NormType": norm type (Euclidean, information/self-standardized)

**lowerCase**: object of class "OptionalNumeric": optional constant for lower case solution.

**neighborRadius**: object of class "numeric": radius of the corresponding (unconditional) contamination neighborhood.

**Extends**

Class "HampIC", directly.

Class "IC", by class "HampIC".

Class "InfluenceCurve", by class "IC".

**Methods**

**CallL2Fam**<- signature(object = "ContIC"): replacement function for slot CallL2Fam.

**cent** signature(object = "ContIC"): accessor function for slot cent.

**cent**<- signature(object = "ContIC"): replacement function for slot cent.

**clip** signature(x1 = "ContIC"): accessor function for slot clip.

**clip**<- signature(object = "ContIC"): replacement function for slot clip.

**stand**<- signature(object = "ContIC"): replacement function for slot stand.

**lowerCase**<- signature(object = "ContIC"): replacement function for slot lowerCase.

**neighbor** signature(object = "ContIC"): generates an object of class "ContNeighborhood" with radius given in slot neighborRadius.

**generateIC** signature(neighbor = "ContNeighborhood", L2Fam = "L2ParamFamily"): generate an object of class "ContIC". Rarely called directly.

**show** signature(object = "ContIC")

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.

Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[IC-class](#), [ContIC](#) [HampIC-class](#)

**Examples**

```
IC1 <- new("ContIC")
plot(IC1)
```

---

ContNeighborhood

*Generating function for ContNeighborhood-class*

---

**Description**

Generates an object of class "ContNeighborhood".

**Usage**

```
ContNeighborhood(radius = 0)
```

**Arguments**

radius            non-negative real: neighborhood radius.

**Value**

Object of class "ContNeighborhood"

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.

Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[ContNeighborhood-class](#)

**Examples**

```
ContNeighborhood()  
  
## The function is currently defined as  
function(radius = 0){  
  new("ContNeighborhood", radius = radius)  
}
```

---

ContNeighborhood-class

*Contamination Neighborhood*

---

**Description**

Class of (unconditional) contamination neighborhoods.

**Objects from the Class**

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

**Slots**

type Object of class "character": "(uncond.) convex contamination neighborhood".  
radius Object of class "numeric": neighborhood radius.

**Extends**

Class "UncondNeighborhood", directly.  
Class "Neighborhood", by class "UncondNeighborhood".

**Methods**

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

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.  
Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[ContNeighborhood](#), [UncondNeighborhood-class](#)

**Examples**

```
new("ContNeighborhood")
```

---

cutoff	<i>Generating function(s) for class 'cutoff'</i>
--------	--------------------------------------------------

---

**Description**

Generating function(s) for class cutoff.

**Usage**

```
cutoff(name = "empirical", body.fct0,
        cutoff.quantile = 0.95,
        norm = NormType(), QF, nsim = 100000)
cutoff.sememp()
cutoff.chisq()
```

**Arguments**

name	argument for name slot of cutoff object
body.fct0	a call generated by code wrapped to substitute resp. quote; the body of the fct slot of the cutoff object
cutoff.quantile	numeric (in [0,1]); the corresponding slot value for the cutoff object
norm	an object of class NormType – the norm/distance by which to produce the cutoff - value.
nsim	integer: the sample size used for determining the quantiles of $(x^T Q x)^{1/2}$ for $x$ multivariate standard normal and $Q$ a corresponding quadratic form
QF	a quadratic (positive semidefinite, symmetric) matrix used as quadratic form

**Details**

cutoff generates a valid object of class "cutoff". As function slot fct may only have a formal argument data, the other arguments to determine the cutoff value, i.e. norm, QF, nsim, cutoff.quantile, nsim have to enter the scope of this function by lexical scoping; now cutoff.quantile, norm, QF are to be taken from the calling environment (not from the defining one), so we have delay evaluation of the function body, which is why we assume it to be given wrapped into substitute resp. quote. body.fct0 is by default (i.e. if argument body.fct0 is missing) set to quote(quantile(slot(norm, "fct")(data), cutoff.quantile)), internally, i.e.; to an empirical quantile of the corresponding norms.

`cutoff.sememp()` is a helper function generating the theoretical (asymptotic) quantile of (the square root of) a corresponding quadratic form, assuming multivariate normality; to determine this quantile `nsim` simulations are used.

`cutoff.chisq()` is a helper function generating the theoretical (asymptotic) quantile of (the square root of) a (self-standardized) quadratic form, assuming multivariate normality; i.e.; a corresponding quantile of a Chi-Square distribution.

### Value

Object of class "cutoff".

### Author(s)

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

### See Also

[cutoff-class](#), [ddPlot](#)

### Examples

```
cutoff()
cutoff.sememp()
cutoff.chisq()
```

---

cutoff-class

*Cutoff class for distance-distance plots*

---

### Description

Class of methods to determine cutoff point for distance-distance plots; used to derive other cutoff methods later by method dispatch.

### Objects from the Class

Objects could in principle be created by calls of the form `new("cutoff", ...)`. More frequently they are created via the generating function `cutoff`, respectively via the helper functions `cutoff.sememp` and `cutoff.chisq`.

### Slots

**name:** object of class "character"; defaults to "empirical" in prototype;

**fct:** an object of of class "function"; for this class layer, this function must only have one argument `data` (which may but need not be used to determine the cutoff point empirically); in derived classes this restriction could be dropped, if corresponding special methods for `ddPlot` are derived. Defaults to `function(data) quantile(data)`.

**cutoff.quantile:** Object of class "numeric": a probability (in [0,1]) to determine the respective quantile (empirical or theoretical) to plot the cutoff line; defaults to 0.95 in prototype;

**Methods**

**cutoff.quantile** signature(object = "cutoff"): accessor function for slot cutoff.quantile.

**cutoff.quantile<-** signature(object = "cutoff"): replacement function for slot cutoff.quantile.

**fct** signature(object = "cutoff"): accessor function for slot fct.

**name** signature(object = "cutoff"): accessor function for slot name.

**Author(s)**

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

**See Also**

[ddPlot](#), [outlyingPlotIC](#) [cutoff](#)

**Examples**

```
cutoff()
```

---

ddPlot-methods

*Methods for Function ddPlot in Package 'RobAStBase'*


---

**Description**

ddPlot-methods

**Usage**

```
ddPlot(data, dist.x, dist.y, cutoff.x, cutoff.y, ...)
## S4 method for signature 'matrix'
ddPlot(data, dist.x = NormType(), dist.y = NormType(),
       cutoff.x, cutoff.y, ...,
       cutoff.quantile.x = 0.95, cutoff.quantile.y = cutoff.quantile.x,
       transform.x, transform.y = transform.x,
       id.n, lab.pts, adj, cex.idn,
       col.idn, lty.cutoff, lwd.cutoff, col.cutoff, text.abline = TRUE,
       text.abline.x = NULL, text.abline.y = NULL,
       cex.abline = par("cex"), col.abline = col.cutoff,
       font.abline = par("font"), adj.abline = c(0,0),
       text.abline.x.x = NULL, text.abline.x.y = NULL,
       text.abline.y.x = NULL, text.abline.y.y = NULL,
       text.abline.x.fmt.cx = "%7.2f", text.abline.x.fmt.qx = "%4.2f%",
       text.abline.y.fmt.cy = "%7.2f", text.abline.y.fmt.qy = "%4.2f%",
       jitt.fac)
## S4 method for signature 'numeric'
ddPlot(data, dist.x = NormType(), dist.y = NormType(),
       cutoff.x, cutoff.y, ...,
```

```

cutoff.quantile.x = 0.95, cutoff.quantile.y = cutoff.quantile.x,
transform.x, transform.y = transform.x,
id.n, lab.pts, adj, cex.idn,
col.idn, lty.cutoff, lwd.cutoff, col.cutoff, text.abline = TRUE,
text.abline.x = NULL, text.abline.y = NULL,
cex.abline = par("cex"), col.abline = col.cutoff,
font.abline = par("font"), adj.abline = c(0,0),
text.abline.x.x = NULL, text.abline.x.y = NULL,
text.abline.y.x = NULL, text.abline.y.y = NULL,
text.abline.x.fmt.cx = "%7.2f", text.abline.x.fmt.qx = "%4.2f%%",
text.abline.y.fmt.cy = "%7.2f", text.abline.y.fmt.qy = "%4.2f%%",
jitt.fac)
## S4 method for signature 'data.frame'
ddPlot(data, dist.x = NormType(), dist.y = NormType(),
cutoff.x, cutoff.y, ...,
cutoff.quantile.x = 0.95, cutoff.quantile.y = cutoff.quantile.x,
transform.x, transform.y = transform.x,
id.n, lab.pts, adj, cex.idn,
col.idn, lty.cutoff, lwd.cutoff, col.cutoff, text.abline = TRUE,
text.abline.x = NULL, text.abline.y = NULL,
cex.abline = par("cex"), col.abline = col.cutoff,
font.abline = par("font"), adj.abline = c(0,0),
text.abline.x.x = NULL, text.abline.x.y = NULL,
text.abline.y.x = NULL, text.abline.y.y = NULL,
text.abline.x.fmt.cx = "%7.2f", text.abline.x.fmt.qx = "%4.2f%%",
text.abline.y.fmt.cy = "%7.2f", text.abline.y.fmt.qy = "%4.2f%%",
jitt.fac)

```

### Arguments

<code>data</code>	data coercable to matrix; the data at which to produce the ddPlot.
<code>...</code>	further arguments to be passed to <code>plot.default</code> , <code>text</code> , and <code>abline</code>
<code>dist.x</code>	object of class <code>NormType</code> ; the distance for the x axis.
<code>dist.y</code>	object of class <code>NormType</code> ; the distance for the y axis.
<code>cutoff.x</code>	object of class <code>cutoff</code> ; the cutoff information for the x axis (the vertical line discriminating 'good' and 'bad' points).
<code>cutoff.y</code>	object of class <code>cutoff</code> ; the cutoff information for the y axis (the horizontal line discriminating 'good' and 'bad' points).
<code>cutoff.quantile.x</code>	numeric; the cutoff quantile for the x axis.
<code>cutoff.quantile.y</code>	numeric; the cutoff quantile for the y axis.
<code>transform.x</code>	function; a transformation to be performed before determining the distances of the x axis.
<code>transform.y</code>	function; a transformation to be performed before determining the distances of the y axis.

<code>id.n</code>	a set of indices (or a corresponding logical vector); to select a subset of the data in argument data.
<code>lab.pts</code>	a vector of labels for the (unsubsetting) data.
<code>adj</code>	the corresponding argument for <code>text</code> for labelling the outliers.
<code>cex.idn</code>	the corresponding <code>cex</code> argument for <code>text</code> for labelling the outliers.
<code>col.idn</code>	the corresponding <code>col</code> argument for <code>text</code> for labelling the outliers.
<code>lty.cutoff</code>	the corresponding <code>lty</code> argument for <code>abline</code> for drawing the cutoff lines; either one <code>lty</code> -value (one value or vector) or a list of length 2 of <code>lty</code> -values.
<code>lwd.cutoff</code>	(vector cast to length 2): the corresponding <code>lwd</code> argument for <code>abline</code> for drawing the cutoff lines.
<code>col.cutoff</code>	(vector cast to length 2): the corresponding <code>col</code> argument for <code>abline</code> for drawing the cutoff lines.
<code>text.abline</code>	vector of logicals (cast to length 2): shall text be added to cutoff lines.
<code>text.abline.x</code>	text to be added to cutoff lines in x direction; if NULL (default) we use “[pp] %-cutoff = [ff]” where [pp] is the percentag up to 2 digits and [ff] is the cutoff value up to 2 digits.
<code>text.abline.y</code>	text to be added to cutoff lines in y direction; if NULL (default) we use “[pp] %-cutoff = [ff]” where [pp] is the percentag up to 2 digits and [ff] is the cutoff value up to 2 digits.
<code>cex.abline</code>	vector of numerics (cast to length 2): <code>cex</code> -value for added cutoff text.
<code>col.abline</code>	vector of length 2: color for added cutoff text.
<code>font.abline</code>	vector of length 2: font for added cutoff text.
<code>adj.abline</code>	cast to 2 x 2 matrix (by recycling rules): adjustment values for added cutoff text.
<code>text.abline.x.y</code>	y-coordinate of text to be added to cutoff lines in x direction; if NULL (default) set to mid of <code>mean(par("usr")[c(3, 4)])</code> .
<code>text.abline.y.x</code>	x-coordinate of text to be added to cutoff lines in y direction; if NULL (default) set to mid of <code>mean(par("usr")[c(1, 2)])</code> .
<code>text.abline.x.x</code>	x-coordinate of text to be added to cutoff lines in x direction; if NULL (default) set to 1.05 times the cutoff value.
<code>text.abline.y.y</code>	y-coordinate of text to be added to cutoff lines in y direction; if NULL (default) set to 1.05 times the cutoff value.
<code>text.abline.x.fmt.cx</code>	format string (see <code>gettextf</code> ) to format the cutoff value in label in x direction.
<code>text.abline.x.fmt.qx</code>	format string to format cutoff probability in label in x direction.
<code>text.abline.y.fmt.cy</code>	format string to format the cutoff value in label in y direction.
<code>text.abline.y.fmt.qy</code>	format string to format cutoff probability in label in y direction.
<code>jitt.fac</code>	factor for jittering, see <code>jitter</code> ;



## Details

The `matrix`-method calls `.ddPlot.MatNtNtCoCo`, the `numeric`- and `data.frame`-methods coerce argument `data` to `matrix` — the `numeric`-method by a call to `matrix(data, nrow=1)`, in the `data.frame`-methods by a call to `t(as.matrix(data))`.

In arguments `text.abline.x` and `text.abline.y` the following patterns are substituted:

"%qx" cutoff-quantile in x-direction

"%qy" cutoff-quantile in y-direction

"%cx" cutoff-value in x-direction

"%cy" cutoff-value in y-direction

## Value

a list with items

<code>id.x</code>	the indices of (possibly transformed) data (within subset <code>id.n</code> ) beyond the x-cutoff
<code>id.y</code>	the indices of (possibly transformed) data (within subset <code>id.n</code> ) beyond the y-cutoff
<code>id.xy</code>	the indices of (possibly transformed) data (within subset <code>id.n</code> ) beyond the x-cutoff and the y-cutoff
<code>qtx</code>	the quantiles of the distances of the (possibly transformed) data in x direction
<code>qty</code>	the quantiles of the distances of the (possibly transformed) data in y direction
<code>cutoff.x.v</code>	the cutoff value in x direction
<code>cutoff.y.v</code>	the cutoff value in y direction

## Author(s)

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

## Examples

```
MX <- matrix(rnorm(1500),nrow=6)
QM <- matrix(rnorm(36),nrow=6); QM <- QM %*% t(QM)
ddPlot(data=MX, dist.y=QFNorm(QuadF=PosSemDefSymmMatrix(QM)))
```

---

`evalIC`*Generic function for evaluating ICs*

---

**Description**

Generic function for evaluating ICs.

**Usage**

```
evalIC(IC, x)
```

**Arguments**

IC	object of class "IC"
x	numeric vector or matrix

**Details**

The list of random variables contained in the slot `Curve` is evaluated at `x`.

**Value**

In case `x` is numeric a vector and in case `x` is matrix a matrix is returned.

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.  
Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[IC-class](#)

---

FixRobModel

*Generating function for FixRobModel-class*

---

### Description

Generates an object of class "FixRobModel".

### Usage

```
FixRobModel(center = ParamFamily(modifyParam =  
  function(theta) Norm(mean = theta)), neighbor = ContNeighborhood())
```

### Arguments

center	object of class "ProbFamily"
neighbor	object of class "UncondNeighborhood"

### Value

Object of class "FixRobModel"

### Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

### References

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.

Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

### See Also

[FixRobModel-class](#)

### Examples

```
(M1 <- FixRobModel())  
  
## The function is currently defined as  
function(center = ParamFamily(), neighbor = ContNeighborhood()){  
  new("FixRobModel", center = center, neighbor = neighbor)  
}
```

---

FixRobModel-class      *Robust model with fixed (unconditional) neighborhood*

---

### Description

Class of robust models with fixed (unconditional) neighborhoods.

### Objects from the Class

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

### Slots

`center` Object of class "ProbFamily".

`neighbor` Object of class "UncondNeighborhood".

### Extends

Class "RobModel", directly.

### Methods

`neighbor<-` signature(object = "FixRobModel"): replacement function for slot `neighbor<-`

`show` signature(object = "FixRobModel")

### Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

### References

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.

Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

### See Also

[ProbFamily-class](#), [UncondNeighborhood-class](#), [FixRobModel](#)

### Examples

```
new("FixRobModel")
```

---

generateIC	<i>Generic function for the generation of influence curves</i>
------------	----------------------------------------------------------------

---

**Description**

This function is rarely called directly. It is used by other functions to create objects of class "IC".

**Usage**

```
generateIC(neighbor, L2Fam, ...)
```

**Arguments**

neighbor	Object of class "Neighborhood".
L2Fam	L2-differentiable family of probability measures.
...	additional parameters

**Value**

Object of class "IC"

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.

Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[IC-class](#), [ContIC-class](#), [TotalVarIC-class](#)

---

generateIC.fct-methods

*Generic Function for making ICs consistent at a possibly different model*

---

### Description

Generic function for providing centering and Fisher consistency of ICs.

### Usage

```
generateIC.fct(neighbor, L2Fam, ...)
```

### Arguments

neighbor	object of class "UncondNeighborhood"
L2Fam	L2-differentiable family of probability measures; may be missing.
...	additional parameters

### Value

An IC at the model.

### Methods

**generateIC.fct** signature(IC = "UncondNeighborhood", L2Fam = "L2ParamFamily": ...)

### Author(s)

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

### References

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.  
Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

### See Also

[L2ParamFamily-class](#), [IC-class](#)

---

getBiasIC	<i>Generic function for the computation of the asymptotic bias for an IC</i>
-----------	------------------------------------------------------------------------------

---

### Description

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

### Usage

```
getBiasIC(IC, neighbor, ...)

## S4 method for signature 'IC,UncondNeighborhood'
getBiasIC(IC, neighbor, L2Fam, biastype = symmetricBias(),
          normtype = NormType(), tol = .Machine$double.eps^0.25, numbeval = 1e5)
```

### Arguments

IC	object of class "InfluenceCurve"
neighbor	object of class "Neighborhood".
...	additional parameters
L2Fam	object of class "L2ParamFamily".
biastype	object of class "BiasType"
normtype	object of class "NormType"
tol	the desired accuracy (convergence tolerance).
numbeval	number of evaluation points.

### Value

The bias of the IC is computed.

### Methods

**IC = "IC", neighbor = "UncondNeighborhood"** determines the as. bias by random evaluation of the IC; this random evaluation is done by the internal S4-method `.evalBiasIC`; this latter dispatches according to the signature `IC, neighbor, biastype`. For signature `IC="IC", neighbor = "ContNeighborhood", biastype = "BiasType"`, also an argument `normtype` is used to be able to use self- or information standardizing norms; besides this the signatures `IC="IC", neighbor = "TotalVarNeighborhood", biastype = "BiasType"`, `IC="IC", neighbor = "ContNeighborhood", biastype = "onesidedBias"`, and `IC="IC", neighbor = "ContNeighborhood", biastype = "asymmetricBias"` are implemented.

### Note

This generic function is still under construction.

**Author(s)**

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

**References**

- Huber, P.J. (1968) Robust Confidence Limits. *Z. Wahrscheinlichkeitstheor. Verw. Geb.* **10**:269–278.
- Rieder, H. (1980) Estimates derived from robust tests. *Ann. Stats.* **8**: 106–115.
- Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.
- Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.
- Ruckdeschel, P. and Kohl, M. (2005) Computation of the Finite Sample Bias of M-estimators on Neighborhoods.

**See Also**

[getRiskIC-methods](#), [InfRobModel-class](#)

---

getBoundedIC

*getBoundedIC*

---

**Description**

Generates a bounded influence curve.

**Usage**

```
getBoundedIC(L2Fam, D=trafo(L2Fam@param))
```

**Arguments**

L2Fam	object of class "L2ParamFamily"
D	matrix with as many columns as length(L2Fam@param)

**Value**

(a bounded) pIC (to matrix D) given as object of class "EuclRandVariable"

**Author(s)**

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

**References**

- Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.



---

getRiskFctBV-methods    *Methods for Function getRiskFctBV in Package ‘RobAStBase’*

---

### 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 = "asGRisk", biastype = "ANY"): returns an error that the respective method is not yet implemented.

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

**getRiskFctBV** signature(risk = "asSemivar", biastype = "onesidedBias"): returns a function with arguments bias and variance to compute the asymptotic semivariance error, i.e.  $E[(S_n - \theta)_+^2]$  resp.  $E[(S_n - \theta)_-^2]$ , for a given ALE  $S_n$  at a situation where it has one-sided bias bias (including the radius!) and variance variance.

**getRiskFctBV** signature(risk = "asSemivar", biastype = "asymmetricBias"): returns a function with arguments bias and variance to compute the asymptotic semivariance error, i.e.  $E[\nu_1(S_n - \theta)_+^2 + \nu_2(S_n - \theta)_-^2]$  for a given ALE  $S_n$  at a situation where it has one-sided bias bias (including the radius!) and variance variance.

### Examples

```
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

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

## S4 method for signature 'IC,asCov,missing,missing'
getRiskIC(IC, risk, tol = .Machine$double.eps^0.25)

## S4 method for signature 'IC,asCov,missing,L2ParamFamily'
getRiskIC(IC, risk, L2Fam, tol = .Machine$double.eps^0.25)
```

```

## S4 method for signature 'IC,trAsCov,missing,missing'
getRiskIC(IC, risk, tol = .Machine$double.eps^0.25)

## S4 method for signature 'IC,trAsCov,missing,L2ParamFamily'
getRiskIC(IC, risk, L2Fam, tol = .Machine$double.eps^0.25)

## S4 method for signature 'IC,asBias,UncondNeighborhood,missing'
getRiskIC(IC, risk, neighbor, tol = .Machine$double.eps^0.25)

## S4 method for signature 'IC,asBias,UncondNeighborhood,L2ParamFamily'
getRiskIC(IC, risk, neighbor, L2Fam, tol = .Machine$double.eps^0.25)

## S4 method for signature 'IC,asMSE,UncondNeighborhood,missing'
getRiskIC(IC, risk, neighbor, tol = .Machine$double.eps^0.25)

## S4 method for signature 'IC,asMSE,UncondNeighborhood,L2ParamFamily'
getRiskIC(IC, risk, neighbor, L2Fam, tol = .Machine$double.eps^0.25)

## S4 method for signature 'TotalVarIC,asUnOvShoot,UncondNeighborhood,missing'
getRiskIC(IC, risk, neighbor)

## S4 method for signature 'IC,fiUnOvShoot,ContNeighborhood,missing'
getRiskIC(IC, risk, neighbor, sampleSize, Algo = "A", cont = "left")

## S4 method for signature 'IC,fiUnOvShoot,TotalVarNeighborhood,missing'
getRiskIC(IC, risk, neighbor, sampleSize, Algo = "A", cont = "left")

```

### Arguments

IC	object of class "InfluenceCurve"
risk	object of class "RiskType".
neighbor	object of class "Neighborhood".
L2Fam	object of class "L2ParamFamily".
...	additional parameters
tol	the desired accuracy (convergence tolerance).
sampleSize	integer: sample size.
Algo	"A" or "B".
cont	"left" or "right".

### 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 = "IC", risk = "asCov", neighbor = "missing", L2Fam = "missing"** asymptotic covariance of IC.
- IC = "IC", risk = "asCov", neighbor = "missing", L2Fam = "L2ParamFamily"** asymptotic covariance of IC under L2Fam.
- IC = "IC", risk = "trAsCov", neighbor = "missing", L2Fam = "missing"** asymptotic covariance of IC.
- IC = "IC", risk = "trAsCov", neighbor = "missing", L2Fam = "L2ParamFamily"** asymptotic covariance of IC under L2Fam.
- IC = "IC", risk = "asBias", neighbor = "ContNeighborhood", L2Fam = "missing"** asymptotic bias of IC under convex contaminations; uses method [getBiasIC](#).
- IC = "IC", risk = "asBias", neighbor = "ContNeighborhood", L2Fam = "L2ParamFamily"** asymptotic bias of IC under convex contaminations and L2Fam; uses method [getBiasIC](#).
- IC = "IC", risk = "asBias", neighbor = "TotalVarNeighborhood", L2Fam = "missing"** asymptotic bias of IC in case of total variation neighborhoods; uses method [getBiasIC](#).
- IC = "IC", risk = "asBias", neighbor = "TotalVarNeighborhood", L2Fam = "L2ParamFamily"** asymptotic bias of IC under L2Fam in case of total variation neighborhoods; uses method [getBiasIC](#).
- IC = "IC", risk = "asMSE", neighbor = "UncondNeighborhood", L2Fam = "missing"** asymptotic mean square error of IC.
- IC = "IC", risk = "asMSE", neighbor = "UncondNeighborhood", L2Fam = "L2ParamFamily"** asymptotic mean square error of IC under L2Fam.
- IC = "TotalVarIC", risk = "asUnOvShoot", neighbor = "UncondNeighborhood", L2Fam = "missing"** asymptotic under-/overshoot risk of IC.
- IC = "IC", risk = "fiUnOvShoot", neighbor = "ContNeighborhood", L2Fam = "missing"** finite-sample under-/overshoot risk of IC.
- IC = "IC", risk = "fiUnOvShoot", neighbor = "TotalVarNeighborhood", L2Fam = "missing"** finite-sample under-/overshoot risk of IC.

## Note

This generic function is still under construction.

## Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>  
 Peter Ruckdeschel <Peter.Ruckdeschel@itwm.fraunhofer.de>

## References

- Huber, P.J. (1968) Robust Confidence Limits. *Z. Wahrscheinlichkeitstheor. Verw. Geb.* **10**:269–278.
- Rieder, H. (1980) Estimates derived from robust tests. *Ann. Stats.* **8**: 106–115.
- Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.

Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

Ruckdeschel, P. and Kohl, M. (2005) Computation of the Finite Sample Risk of M-estimators on Neighborhoods.

### See Also

[getRiskIC](#), [InfRobModel-class](#)

---

getweight-methods      *Generating weights*

---

### Description

Generates weight functions of Hampel / BdSt type for different bias and norm types.

### Usage

```
getweight(Weight, neighbor, biastype, ...)
minbiasweight(Weight, neighbor, biastype, ...)
## S4 method for signature 'HampelWeight,ContNeighborhood,BiasType'
getweight(Weight, neighbor, biastype, normW)
## S4 method for signature 'HampelWeight,ContNeighborhood,BiasType'
minbiasweight(Weight, neighbor, biastype, normW)
## S4 method for signature 'HampelWeight,ContNeighborhood,onesidedBias'
getweight(Weight, neighbor, biastype, ...)
## S4 method for signature 'HampelWeight,ContNeighborhood,onesidedBias'
minbiasweight(Weight, neighbor, biastype, ...)
## S4 method for signature 'HampelWeight,ContNeighborhood,asymmetricBias'
getweight(Weight, neighbor, biastype, ...)
## S4 method for signature 'HampelWeight,ContNeighborhood,asymmetricBias'
minbiasweight(Weight, neighbor, biastype, ...)
## S4 method for signature 'BdStWeight,TotalVarNeighborhood,BiasType'
getweight(Weight, neighbor, biastype, ...)
## S4 method for signature 'BdStWeight,TotalVarNeighborhood,BiasType'
minbiasweight(Weight, neighbor, biastype, ...)
```

### Arguments

Weight	Object of class "RobWeight".
neighbor	Object of class "Neighborhood".
biastype	Object of class "BiasType".
normW	Object of class "NormType" — only for signature HampelWeight,ContNeighborhood,BiasType.
...	possibly additional (unused) arguments — like in a call to the less specific methods.

**Details**

These functions generate the weight function in slot `weight` in a corresp. object of class `RobWeight` and descendants.

**Value**

Object of class `"HampelWeight"` resp. `"BdStWeight"`

**Methods**

**getweight** signature(`Weight = "HampelWeight"`, `neighbor = "ContNeighborhood"`,  
with additional argument `biastype` of class `"BiasType"`: produces weight slot... biastype

**minbiasweight** signature(`Weight = "HampelWeight"`, `neighbor = "ContNeighborhood"`,  
with additional argument `biastype` of class `"BiasType"`: produces weight slot... bia

**getweight** signature(`Weight = "HampelWeight"`, `neighbor = "ContNeighborhood"`,  
produces weight slot... biastype

**minbiasweight** signature(`Weight = "HampelWeight"`, `neighbor = "ContNeighborhood"`,  
produces weight slot... bia

**getweight** signature(`Weight = "HampelWeight"`, `neighbor = "ContNeighborhood"`,  
produces weight slot... biastype

**minbiasweight** signature(`Weight = "HampelWeight"`, `neighbor = "ContNeighborhood"`,  
produces weight slot... bia

**getweight** signature(`Weight = "BdStWeight"`, `neighbor = "TotalVarNeighborhood"`,  
produces weight slot... biasty

**minbiasweight** signature(`Weight = "BdStWeight"`, `neighbor = "TotalVarNeighborhood"`,  
produces weight slot... b

**Author(s)**

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

**References**

Hampel et al. (1986) *Robust Statistics. The Approach Based on Influence Functions*. New York: Wiley.

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.

Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[BdStWeight-class](#), [HampelWeight-class](#), [IC-class](#)

---

HampelWeight-class      *Robust Weight classes for weights of Hampel type*


---

### Description

Classes for weights of Hampel type.

### Objects from the Class

Objects can be created by calls of the form `new("HampelWeight", ...)`; to fill slot `weight`, you will use the generating functions `getweight` and `minbiasweight`.

### Slots

`name` Object of class "character"; inherited from class `RobWeight`.  
`weight` Object of class "function" — the weight function; inherited from class `RobWeight`.  
`clip` Object of class "numeric" — clipping bound(s); inherited from class `BoundedWeight`.  
`stand` Object of class "matrix" — standardization; inherited from class `BdStWeight`.  
`cent` Object of class "numeric" — centering.

### Extends

Class "RobWeight", via class "BoundedWeight". Class "BoundedWeight", via class "BdStWeight".  
Class "BdStWeight", directly.

### Methods

`cent` signature(object = "HampelWeight"): accessor function for slot `cent`.  
`cent<-` signature(object = "HampelWeight", value = "matrix"): replacement function for slot `cent`. This replacement method should be used with great care, as the slot `weight` is not simultaneously updated and hence, this may lead to inconsistent objects.

### Author(s)

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

### References

Hampel et al. (1986) *Robust Statistics. The Approach Based on Influence Functions*. New York: Wiley.  
Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.  
Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[BdStWeight-class](#), [BoundedWeight-class](#), [RobWeight-class](#), [IC](#), [InfluenceCurve-class](#)

**Examples**

```
## prototype
new("HampelWeight")
```

---

HampIC-class	<i>Influence curve of Hampel type</i>
--------------	---------------------------------------

---

**Description**

Class of (partial) influence curves of Hampel (= total variation or contamination) type; used as common mother class for classes ContIC and TotalVarIC.

**Objects from the Class**

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

**Slots**

`CallL2Fam` object of class "call": creates an object of the underlying L2-differentiable parametric family.

`name` object of class "character"

`Curve` object of class "EuclRandVarList"

`modifyIC` Object of class "OptionalFunction": function of two arguments, which are an L2 parametric family and an optional influence curve. Returns an object of class "IC". This slot is mainly used for internal computations!

`Risks` object of class "list": list of risks; cf. [RiskType-class](#).

`Infos` object of class "matrix" with two columns named `method` and `message`: additional informations.

`stand` object of class "matrix": standardizing matrix.

`weight` object of class "RobWeight": weight function

`biastype` object of class "BiasType": bias type (symmetric/onsided/asymmetric)

`normtype` object of class "NormType": norm type (Euclidean, information/self-standardized)

`lowerCase` object of class "OptionalNumeric": optional constant for lower case solution.

`neighborRadius` object of class "numeric": radius of the corresponding (unconditional) contamination neighborhood.

**Extends**

Class "IC", directly.

Class "InfluenceCurve", by class "IC".

**Methods**

**stand** signature(object = "HampIC"): accessor function for slot stand.  
**weight** signature(object = "HampIC"): accessor function for slot weight.  
**biastype** signature(object = "HampIC"): accessor function for slot biastype.  
**normtype** signature(object = "HampIC"): accessor function for slot normtype.  
**lowerCase** signature(object = "HampIC"): accessor function for slot lowerCase.  
**neighborRadius** signature(object = "HampIC"): accessor function for slot neighborRadius.  
**neighborRadius<-** signature(object = "HampIC"): replacement function for slot neighborRadius.

**Author(s)**

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

**References**

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.  
 Kohl, M. (2005) *Numerical Hampributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[IC-class](#)

**Examples**

```
IC1 <- new("HampIC")
plot(IC1)
```

---

 IC

---

*Generating function for IC-class*


---

**Description**

Generates an object of class "IC".

**Usage**

```
IC(name, Curve = EuclRandVarList(RealRandVariable(Map = list(function(x){x}),
                                                    Domain = Reals())),
    Risks, Infos, CallL2Fam = call("L2ParamFamily"), modifyIC = NULL)
```



**Arguments**

name	Object of class "character"; the name of the IC.
CallL2Fam	object of class "call": creates an object of the underlying L2-differentiable parametric family.
Curve	object of class "EuclRandVarList".
Risks	object of class "list": list of risks; cf. <a href="#">RiskType-class</a> .
Infos	matrix of characters with two columns named method and message: additional informations.
modifyIC	Object of class "OptionalFunction": function of two arguments, which are an L2 parametric family and an optional influence curve. Returns an object of class "IC" at the parameter value of the L2 parametric family. This function is mainly used for internal computations!

**Value**

Object of class "IC"

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

- Hampel et al. (1986) *Robust Statistics*. The Approach Based on Influence Functions. New York: Wiley.
- Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.
- Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[IC-class](#)

**Examples**

```
IC1 <- IC()
plot(IC1)
```

---

 IC-class

*Influence curve*


---

### Description

Class of (partial) influence curves.

### Objects from the Class

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

### Slots

`CallL2Fam` Object of class "call": creates an object of the underlying L2-differentiable parametric family.

`modifyIC` Object of class "OptionalFunction": function of two arguments, which are an L2 parametric family and an optional influence curve. Returns an object of class "IC". This slot is mainly used for internal computations!

`name` Object of class "character".

`Curve` Object of class "EuclRandVarList".

`Risks` Object of class "list": list of risks; cf. [RiskType-class](#).

`Infos` Object of class "matrix" with two columns named `method` and `message`: additional informations.

### Extends

Class "InfluenceCurve", directly.

### Methods

**CallL2Fam** signature(object = "IC"): accessor function for slot `CallL2Fam`.

**CallL2Fam<-** signature(object = "IC"): replacement function for slot `CallL2Fam`.

**modifyIC** signature(object = "IC"): accessor function for slot `modifyIC`.

**checkIC** signature(IC = "IC", L2Fam = "missing"): check centering and Fisher consistency of IC assuming the L2-differentiable parametric family which can be generated via the slot `CallL2Fam` of IC.

**checkIC** signature(IC = "IC", L2Fam = "L2ParamFamily"): check centering and Fisher consistency of IC assuming the L2-differentiable parametric family `L2Fam`.

**evalIC** signature(IC = "IC", x = "numeric"): evaluate IC at x.

**evalIC** signature(IC = "IC", x = "matrix"): evaluate IC at the rows of x.

**infoPlot** signature(object = "IC"): Plot absolute and relative information of IC.

**plot** signature(x = "IC", y = "missing")

**show** signature(object = "IC")

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

Hampel et al. (1986) *Robust Statistics. The Approach Based on Influence Functions*. New York: Wiley.

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.

Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[InfluenceCurve-class, IC](#)

**Examples**

```
IC1 <- new("IC")
plot(IC1)
```

---

InfluenceCurve      *Generating function for InfluenceCurve-class*

---

**Description**

Generates an object of class "InfluenceCurve".

**Usage**

```
InfluenceCurve(name, Curve = EuclRandVarList(EuclRandVariable(Domain = Reals())),
               Risks, Infos)
```

**Arguments**

name	character string: name of the influence curve
Curve	object of class "EuclRandVarList"
Risks	list of risks
Infos	matrix of characters with two columns named method and message: additional informations

**Value**

Object of class "InfluenceCurve"

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

- Hampel et al. (1986) *Robust Statistics. The Approach Based on Influence Functions*. New York: Wiley.
- Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.
- Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[InfluenceCurve-class](#)

**Examples**

```
InfluenceCurve()

## The function is currently defined as
InfluenceCurve <- function(name, Curve = EuclRandVarList(EuclRandVariable(Domain = Reals())),
                          Risks, Infos){
  if(missing(name))
    name <- "influence curve"
  if(missing(Risks))
    Risks <- list()
  if(missing(Infos))
    Infos <- matrix(c(character(0),character(0)), ncol=2,
                    dimnames=list(character(0), c("method", "message")))

  return(new("InfluenceCurve", name = name, Curve = Curve,
            Risks = Risks, Infos = Infos))
}
```

---

InfluenceCurve-class *Influence curve*

---

**Description**

Class of influence curves (functions).

**Objects from the Class**

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

**Slots**

`name` object of class "character"  
`Curve` object of class "EuclRandVarList"  
`Risks` object of class "list": list of risks; cf. [RiskType-class](#).  
`Infos` object of class "matrix" with two columns named `method` and `message`: additional informations.

**Methods**

**name** signature(object = "InfluenceCurve"): accessor function for slot name.  
**name<-** signature(object = "InfluenceCurve"): replacement function for slot name.  
**Curve** signature(object = "InfluenceCurve"): accessor function for slot Curve.  
**Map** signature(object = "InfluenceCurve"): accessor function for slot Map of slot Curve.  
**Domain** signature(object = "InfluenceCurve"): accessor function for slot Domain of slot Curve.  
**Range** signature(object = "InfluenceCurve"): accessor function for slot Range of slot Curve.  
**Infos** signature(object = "InfluenceCurve"): accessor function for slot Infos.  
**Infos<-** signature(object = "InfluenceCurve"): replacement function for slot Infos.  
**addInfo<-** signature(object = "InfluenceCurve"): function to add an information to slot Infos.  
**Risks** signature(object = "InfluenceCurve"): accessor function for slot Risks. By means of internal function `.evalListRec` recursively evaluates all non evaluated calls and writes back the evaluated calls to the calling environment.  
**Risks<-** signature(object = "InfluenceCurve"): replacement function for slot Risks.  
**addRisk<-** signature(object = "InfluenceCurve"): function to add a risk to slot Risks.  
**show** signature(object = "InfluenceCurve")

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

Hampel et al. (1986) *Robust Statistics. The Approach Based on Influence Functions*. New York: Wiley.  
Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.  
Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[InfluenceCurve](#), [RiskType-class](#)

**Examples**

```
new("InfluenceCurve")
```

---

InfoPlot

*Wrapper function for information plot method*


---

### Description

The wrapper InfoPlot (capital I!) takes most of arguments to the plot method infoPlot (lower case i!) by default and gives a user possibility to run the function with low number of arguments.

### Usage

```
InfoPlot(IC, data, ..., alpha.trsp = 100,
         with.legend = TRUE, rescale = FALSE, withCall = TRUE)
```

### Arguments

IC	object of class IC
data	optional data argument — for plotting observations into the plot
...	additional parameters (in particular to be passed on to plot)
alpha.trsp	the transparency argument (0 to 100) for plotting the data
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 infoPlot with suitably chosen defaults. If withCall == TRUE, the call to infoPlot is returned

### Examples

```
# Gamma
fam <- GammaFamily()
IC <- optIC(model = fam, risk = asCov())
Y <- distribution(fam)
data <- r(Y)(1000)
InfoPlot(IC, data, withCall = FALSE)
```

infoPlot

*Plot absolute and relative information***Description**

Plot absolute and relative information of influence curves.

**Usage**

```
infoPlot(object, ...)
## S4 method for signature 'IC'
infoPlot(object, data = NULL, ...,
          withSweave = getdistrOption("withSweave"),
          col = par("col"), lwd = par("lwd"), lty,
          colI = grey(0.5), lwdI = 0.7*par("lwd"), ltyI = "dotted",
          main = FALSE, inner = TRUE, sub = FALSE,
          col.inner = par("col.main"), cex.inner = 0.8,
          bmar = par("mar")[1], tmar = par("mar")[3],
          with.legend = TRUE, legend = NULL, legend.bg = "white",
          legend.location = "bottomright", legend.cex = 0.8,
          scaleX = FALSE, scaleX.fct = scaleX.inv,
          scaleY = FALSE, scaleY.fct = pnorm, scaleY.inv=qnorm,
          scaleN = 9, x.ticks = NULL, y.ticks = NULL,
          mfColRow = TRUE, to.draw.arg = NULL,
          cex.pts = 1, col.pts = par("col"),
          pch.pts = 1, jitter.fac = 1, with.lab = FALSE,
          lab.pts = NULL, lab.font = NULL, alpha.trsp = NA,
          which.lbs = NULL, which.Order = NULL, return.Order = FALSE,
          ylab.abs = "absolute information",
          ylab.rel= "relative information")
```

**Arguments**

object	object of class "InfluenceCurve"
data	optional data argument — for plotting observations into the plot;
withSweave	logical: if TRUE (for working with Sweave) no extra device is opened
main	logical: is a main title to be used? or just as argument main in <a href="#">plot.default</a> .
inner	logical: do panels have their own titles? or character vector of / cast to length 'number of compared dimensions'; if argu- ment to.draw.arg is used, this refers to a vector of length 1 (absolute informa- tion) + length(to.draw.arg), the actually plotted relative informations. For further information, see also main in <a href="#">plot.default</a> .
sub	logical: is a sub-title to be used? or just as argument sub in <a href="#">plot.default</a> .

<code>tmar</code>	top margin – useful for non-standard main title sizes; may be a vector with individual values for each of the panels to be plotted.
<code>bmar</code>	bottom margin – useful for non-standard sub title sizes; may be a vector with individual values for each of the panels to be plotted.
<code>col</code>	color of IC in argument object.
<code>lwd</code>	linewidth of IC in argument object.
<code>lty</code>	line-type of IC in argument object.
<code>colI</code>	color of the classically optimal IC.
<code>lwdI</code>	linewidth of the classically optimal IC.
<code>ltyI</code>	line-type of the classically optimal IC.
<code>cex.inner</code>	magnification to be used for inner titles relative to the current setting of <code>cex</code> ; as in <code>par</code> .
<code>col.inner</code>	character or integer code; color for the inner title
<code>with.legend</code>	logical; shall a legend be plotted?
<code>legend</code>	either NULL or a list of length (number of plotted panels) of items which can be used as argument <code>legend</code> in command <code>legend</code> .
<code>legend.location</code>	a valid argument <code>x</code> for <code>legend</code> — the place where to put the legend on the last issued plot — or a list of length (number of plotted panels) of such arguments, one for each plotted panel.
<code>legend.bg</code>	background color for the legend
<code>legend.cex</code>	magnification factor for the legend
<code>scaleX</code>	logical; shall X-axis be rescaled (by default according to the cdf of the underlying distribution)?
<code>scaleY</code>	logical; shall Y-axis be rescaled for <code>abs.info-plot</code> (by default according to a probit scale)?
<code>scaleX.fct</code>	an isotone, vectorized function mapping the domain of the IC to $[0,1]$ ; if <code>scaleX</code> is TRUE and <code>scaleX.fct</code> is missing, the cdf of the underlying observation distribution.
<code>scaleX.inv</code>	the inverse function to <code>scale.fct</code> , i.e., an isotone, vectorized function mapping $[0,1]$ to the domain of the IC such that for any $x$ in the domain, $scaleX.inv(scaleX.fct(x))=x$ ; if <code>scaleX</code> is TRUE and <code>scaleX.inv</code> is missing, the quantile function of the underlying observation distribution.
<code>scaleY.fct</code>	an isotone, vectorized function mapping the range of the norm of the IC to $[0,1]$ ; defaulting to the cdf of $\mathcal{N}(0, 1)$ .
<code>scaleY.inv</code>	an isotone, vectorized function mapping $[0,1]$ into the range of the norm of the IC; defaulting to the quantile function of $\mathcal{N}(0, 1)$ .
<code>scaleN</code>	integer; defaults to 9; on rescaled axes, number of x and y ticks if drawn automatically;
<code>x.ticks</code>	numeric; defaults to NULL; (then ticks are chosen automatically); if non-NULL, user-given x-ticks (on original scale);



<code>y.ticks</code>	numeric; defaults to NULL; (then ticks are chosen automatically); if non-NULL, user-given y-ticks (on original scale); can be a list with one (numeric or NULL) item per panel
<code>mfColRow</code>	shall default partition in panels be used — defaults to TRUE
<code>to.draw.arg</code>	Either NULL (default; everything is plotted) or a vector making a selection among the relative information plots; the absolute information being plotted in any case. This vector is either a vector of integers (the indices of the subplots to be drawn) or characters — the names of the subplots to be drawn: these names are to be chosen either among the row names of the trafo matrix <code>rownames(trafo(eval(object@CallL2Fam)@par</code> or if the last expression is NULL a vector " <code>dim&lt;dimnr&gt;</code> ", <code>dimnr</code> running through the number of rows of the trafo matrix.
<code>cex.pts</code>	size of the points of the data argument plotted
<code>col.pts</code>	color of the points of the data argument plotted
<code>pch.pts</code>	symbol of the points of the data argument plotted
<code>with.lab</code>	logical; shall labels be plotted to the observations?
<code>lab.pts</code>	character or NULL; labels to be plotted to the observations; if NULL observation indices;
<code>lab.font</code>	font to be used for labels
<code>alpha.trsp</code>	alpha transparency to be added ex post to colors <code>col.pch</code> and <code>col.lbl</code> ; if one-dim and NA all colors are left unchanged. Otherwise, with usual recycling rules <code>alpha.trsp</code> gets shorted/prolongated to length the data-symbols to be plotted. Coordinates of this vector <code>alpha.trsp</code> with NA are left unchanged, while for the remaining ones, the alpha channel in rgb space is set to the respective coordinate value of <code>alpha.trsp</code> . The non-NA entries must be integers in <code>[0,255]</code> (0 invisible, 255 opaque).
<code>jitter.fac</code>	jittering factor used in case of a <code>DiscreteDistribution</code> for plotting points of the data argument in a jittered fashion.
<code>which.lbs</code>	either an integer vector with the indices of the observations to be plotted into graph or NULL — then no observation is excluded
<code>which.Order</code>	we order the observations (descending) according to the norm given by <code>normtype(object)</code> ; then <code>which.Order</code> either is an integer vector with the indices of the <i>ordered</i> observations (remaining after a possible reduction by argument <code>which.lbs</code> ) to be plotted into graph or NULL — then no (further) observation is excluded.
<code>return.Order</code>	logical; if TRUE, a list of length two with order vectors is returned — one for ordering w.r.t. the given IC, one for ordering w.r.t. the classically optimal IC; more specifically, the order of the (remaining) observations given by their original index is returned (remaining means: after a possible reduction by argument <code>which.lbs</code> , and ordering is according to the norm given by <code>normtype(object)</code> ); otherwise we return <code>invisible()</code> as usual.
<code>ylab.abs</code>	character; label to be used for y-axis in absolute information panel
<code>ylab.rel</code>	character; label to be used for y-axis in relative information panel
<code>...</code>	further parameters for plot

## Details

Absolute information is defined as the square of the length of an IC. The relative information is defined as the absolute information of one component with respect to the absolute information of the whole IC; confer Section 8.1 of Kohl (2005).

Any parameters of `plot.default` may be passed on to this particular plot method.

For main-, inner-, and subtitles given as arguments `main`, `inner`, and `sub`, top and bottom margins are enlarged to 5 resp. 6 by default but may also be specified by `tmar` / `bmar` arguments. If `main` / `inner` / `sub` are logical then if the respective argument is `FALSE` nothing is done/plotted, but if it is `TRUE`, we use a default main title taking up the calling arguments in case of `main`, default inner titles taking up the class and (named) parameter slots of arguments in case of `inner`, and a "generated on <data>"-tag in case of `sub`. Of course, if `main` / `inner` / `sub` are character, this is used for the title; in case of `inner` it is then checked whether it has correct length. In all title arguments, the following patterns are substituted:

```
"%C" class of argument object
"%A" deparsed argument object
"%D" time/date-string when the plot was generated
```

If argument `...` contains argument `ylim`, this may either be as in `plot.default` (i.e. a vector of length 2) or a vector of length  $2 * (\text{number of plotted dimensions} + e)$ , where `e` is 1 or 0 depending on whether absolute information is plotted or not; in the case of longer length, if `e` is 1, the first two elements are the values for `ylim` in panel "Abs", while the last  $2 * (\text{number of plotted dimensions})$  are the values for `ylim` for the plotted dimensions of the IC, one pair for each dimension.

Similarly, if argument `...` contains arguments `xaxt` or `yaxt`, these may be vectorized, with one value for each of the panels to be plotted. This is useful for stacking panels over each other, using a common x-axis (see example below).

The `...` argument may also contain an argument `withbox` which if `TRUE` warrants that even if `xaxt` and `yaxt` both are `FALSE`, a box is drawn around the respective panel.

## Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

## References

Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

## See Also

[L2ParamFamily-class](#), [IC-class](#)

## Examples

```
N <- NormLocationScaleFamily(mean=0, sd=1)
IC1 <- optIC(model = N, risk = asCov())
infoPlot(IC1)
```

```

## don't run to reduce check time on CRAN
## Not run:
## selection of subpanels for plotting
par(mfrow=c(1,2))
infoPlot(IC1, mfColRow = FALSE, to.draw.arg=c("Abs","sd"))
infoPlot(IC1, mfColRow = FALSE, to.draw.arg=c("Abs","sd"), log="y")

infoPlot(IC1, mfColRow = FALSE, to.draw.arg=c("Abs","mean"),
         panel.first= grid(), ylim = c(0,4), xlim = c(-6,6))
infoPlot(IC1, mfColRow = FALSE, to.draw.arg=c("Abs","mean"),
         panel.first= grid(), ylim = c(0,4,-3,3), xlim = c(-6,6))

par(mfrow=c(1,3))
infoPlot(IC1, mfColRow = FALSE, panel.first= grid(),
         ylim = c(0,4,0,.3,0,.8), xlim=c(-6,6))
par(mfrow=c(1,1))

data <- r(N)(20)
par(mfrow=c(1,3))
infoPlot(IC1, data=data, mfColRow = FALSE, panel.first= grid(),
         with.lab = TRUE, cex.pts=2,
         which.lbs = c(1:4,15:20), which.Order = 1:6,
         return.Order = TRUE)
infoPlot(IC1, data=data[1:10], mfColRow = FALSE, panel.first= grid(),
         with.lab = TRUE, cex.pts=0.7)
par(mfrow=c(1,1))

ICr <- makeIC(list(function(x)sign(x),function(x)sign(abs(x)-qnorm(.75))),N)
data <- r(N)(600)
data.c <- c(data, 1000*data[1:30])
par(mfrow=c(3,1))
infoPlot(ICr, data=data.c, tmar=c(4.1,0,0), bmar=c(0,0,4.1),
         xaxt=c("n","n","s"), mfColRow = FALSE, panel.first= grid(),
         cex.pts=c(.9,.9), alpha.trsp=20, lwd=2, lwdI=1.5, col=3,
         col.pts=c(3,2), colI=2, pch.pts=c(20,20), inner=FALSE,
         scaleX = TRUE, scaleX.fct=pnorm, scaleX.inv=qnorm,
         scaleY=TRUE, scaleY.fct=function(x) pchisq(x,df=1),
         scaleY.inv=function(x)qchisq(x,df=1),legend.cex = 1.0)

## End(Not run)

```

## Description

Generates an object of class "InfRobModel".

**Usage**

```
InfRobModel(center = L2ParamFamily(), neighbor = ContNeighborhood())
```

**Arguments**

```
center      object of class "ProbFamily"
neighbor    object of class "UncondNeighborhood"
```

**Value**

Object of class "FixRobModel"

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.  
 Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[RobModel-class](#), [FixRobModel-class](#)

**Examples**

```
(M1 <- InfRobModel())

## The function is currently defined as
function(center = L2ParamFamily(), neighbor = ContNeighborhood()){
  new("InfRobModel", center = center, neighbor = neighbor)
}
```

---

InfRobModel-class      *Robust model with infinitesimal (unconditional) neighborhood*

---

**Description**

Class of robust models with infinitesimal (unconditional) neighborhoods; i.e., the neighborhood is shrinking at a rate of  $\sqrt{n}$ .

**Objects from the Class**

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

**Slots**

center Object of class "ProbFamily".  
neighbor Object of class "UncondNeighborhood".

**Extends**

Class "RobModel", directly.

**Methods**

**neighbor<-** signature(object = "InfRobModel"): replacement function for slot neighbor<-  
**show** signature(object = "InfRobModel")

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.  
Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[ProbFamily-class](#), [UncondNeighborhood-class](#), [InfRobModel](#)

**Examples**

```
new("InfRobModel")
```

---

interpolRisk-class      *Interpolated Risks*

---

**Description**

Class of risks for which algorithms dispatch to speed-up algorithms

**Usage**

```
MBRRisk(samplesize=100)  
OMSRRisk(samplesize=100)  
RMXRRisk(samplesize=100)
```

**Arguments**

samplesize      sample size at which to look at the risk.

**Details**

The main purpose of classes `OMSRRisk`, `MBRRisk`, and `RMXRRisk` is to help to dispatch into speed-up algorithms later in function `roptest`. In all these risks, we assume convex contamination neighborhoods. `OMSRRisk` stands for optimal MSE-robust estimation (where we assume a radius  $r$  of 0.5), `RMXRRisk` stands for optimal optimally RMX-robust estimation and `MBRRisk` stands for optimal Bias-robust estimation. All these risks have an additional slot `samplesize`, defaulting to 100, and for which there is a replacement and an accessor method.

**Objects from the Class**

`interpolRisk` is a virtual class: No objects may be created from it. the other classes are generated via generating functions.

**Slots**

type Object of class "character": type of risk. (Inherited from `RiskType`).

**Author(s)**

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

**Examples**

```
new("OMSRRisk")
OMSRRisk()
RMXRRisk()
MBRRisk()
myrisk <- MBRRisk(samplesize=100)
samplesize(myrisk)
samplesize(myrisk) <- 20
```

---

`kStepEstimate-class`    *kStepEstimate-class*.

---

**Description**

Class of asymptotically linear estimates.

**Objects from the Class**

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

**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 kStepEstimator with argument withPICList==TRUE.  
**ICList** Optional object of class "OptionalpICList": the list of (intermediate) (total) influence curves used; only filled when called from kStepEstimator with argument withICList==TRUE.  
**start** The argument start — of class "StartClass" used in call to kStepEstimator.  
**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 kStepEstimator.  
**uksteps** Object of class "OptionalMatrix": the intermediate estimates (in  $k$ -space) for the parameter; only filled when called from kStepEstimator.

**Extends**

Class "ALEstimate", directly.  
 Class "Estimate", by class "ALEstimate"

**Methods**

- steps** signature(object = "kStepEstimate"): accessor function for slot steps.
- ksteps** signature(object = "kStepEstimate"): 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 = "kStepEstimate"): 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 = "kStepEstimate"): accessor function for slot start.
- startval** signature(object = "kStepEstimate"): accessor function for slot startval.
- ustartval** signature(object = "kStepEstimate"): accessor function for slot startval.
- ICList** signature(object = "kStepEstimate"): accessor function for slot ICList.
- pICList** signature(object = "kStepEstimate"): accessor function for slot pICList.
- show** signature(object = "kStepEstimate"): a show method;

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[ALEstimate-class](#)

---

kStepEstimator

*Function for the computation of k-step estimates*

---

**Description**

Function for the computation of k-step estimates.

**Usage**

```
kStepEstimator(x, IC, start = NULL, steps = 1L,
  useLast = getRobAStBaseOption("kStepUseLast"),
  withUpdateInKer = getRobAStBaseOption("withUpdateInKer"),
  IC.UpdateInKer = getRobAStBaseOption("IC.UpdateInKer"),
  withICList = getRobAStBaseOption("withICList"),
  withPICList = getRobAStBaseOption("withPICList"),
  na.rm = TRUE, startArgList = NULL, ...,
  withLogScale = TRUE, withEvalAsVar = TRUE)
```



**Arguments**

x	sample
IC	object of class "IC"
start	initial estimate (for full parameter, i.e. in dimension $k$ respective joint length of main and nuisance part of the parameter): either a numerical value, or an object of class "Estimate" or a function producing either a numerical value, or an object of class "Estimate" when evaluated at $x, \dots$ ; if missing or NULL, we use slot startPar of the L2family L2Fam from within IC
steps	integer: number of steps
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 $\ker(D)$ .
na.rm	logical: if TRUE, the estimator is evaluated at complete.cases(x).
startArgList	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 eval(CallL2Fam(IC)).
withPICList	logical: shall slot pICList of return value be filled?
withICList	logical: shall slot ICList of return value be filled?
...	additional parameters
withLogScale	logical; if TRUE, a scale component (if existing and found with name scalename) is computed on log-scale and backtransformed afterwards (default). This avoids crossing 0.
withEvalAsVar	logical; if TRUE (default), tells R to evaluate the asymptotic variance or just to produce a call to do so.

**Details**

Given an initial estimation start, a sample  $x$  and an influence curve IC the corresponding k-step estimator is computed.

The default value of argument useLast is set by the global option kStepUseLast which by default is set to FALSE. In case of general models useLast remains unchanged during the computations. However, if slot CallL2Fam of IC generates an object of class "L2GroupParamFamily" the value of useLast is changed to TRUE. Explicitly setting 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 useLast is set to TRUE and slot modifyIC of IC is filled with some function (which can be used to re-compute the IC for a different parameter), the computation of asvar, asbias and IC is based on the k-step estimate.

**Value**

Object of class "kStepEstimate".

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>,  
Peter Ruckdeschel <Peter.Ruckdeschel@itwm.fraunhofer.de>

**References**

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.  
Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[IC-class](#), [kStepEstimate-class](#)

**Examples**

```

if(require(ROptEst)){
## 1. generate a contaminated sample
ind <- rbinom(100, size=1, prob=0.05)
x <- rnorm(100, mean=0, sd=(1-ind) + ind*9)

## 2. Kolmogorov(-Smirnov) minimum distance estimator
(est0 <- MDEstimator(x=x, NormLocationScaleFamily()))

## 3. k-step estimation: radius known
N1 <- NormLocationScaleFamily(mean=estimate(est0)["mean"], sd=estimate(est0)["sd"])
N1.Rob <- InfRobModel(center = N1, neighbor = ContNeighborhood(radius = 0.5))
IC1 <- optIC(model = N1.Rob, risk = asMSE())
(est1 <- kStepEstimator(x, IC1, est0, steps = 3, withPIC = TRUE))
estimate(est1)
ksteps(est1)
pICList(est1)
start(est1)

## don't run to reduce check time on CRAN
## Not run:
## a transformed model
tfct <- function(x){
  nms0 <- c("mean", "sd")
  nms <- "comb"
  fval0 <- x[1]+2*x[2]
  names(fval0) <- nms
  mat0 <- matrix(c(1,2), nrow = 1, dimnames = list(nms,nms0))
  return(list(fval = fval0, mat = mat0))
}

N1.traf <- N1; trafo(N1.traf) <- tfct

```

```

N1R.traf <- N1.Rob; trafo(N1R.traf) <- tfct
IC1.traf <- optIC(model = N1R.traf, risk = asMSE())
(est0.traf <- MDEstimator(x, N1.traf))
(est1.traf <- kStepEstimator(x, IC1.traf, est0, steps = 3,
                           withIC = TRUE, withPIC = TRUE, withUpdateInKer = FALSE))
(est1a.traf <- kStepEstimator(x, IC1.traf, est0, steps = 3,
                             withIC = TRUE, withPIC = TRUE, withUpdateInKer = TRUE))
estimate(est1.traf)
ksteps(est1.traf)
pICList(est1.traf)
startval(est1.traf)

untransformed.estimate(est1.traf)
uksteps(est1.traf)
ICList(est1.traf)
ustartval(est1.traf)

estimate(est1a.traf)
ksteps(est1a.traf)
pICList(est1a.traf)
startval(est1a.traf)

untransformed.estimate(est1a.traf)
uksteps(est1a.traf)
ICList(est1a.traf)
ustartval(est1a.traf)

## End(Not run)
}

```

---

kStepEstimator.start-methods

*Methods for function kStepEstimator.start in Package 'RobAStBase'*

---

## Description

kStepEstimator.start-methods; these are called from within kStepEstimator to produce a numeric value of for the starting estimator in the end.

## Usage

```

kStepEstimator.start(start, ...)
## S4 method for signature 'numeric'
kStepEstimator.start(start, nrvalues, ...)
## S4 method for signature 'Estimate'
kStepEstimator.start(start, nrvalues, ...)
## S4 method for signature 'function'
kStepEstimator.start(start, x, nrvalues, na.rm, L2Fam, startList)

```

**Arguments**

start	the start slot of an object of class <code>kStepEstimator</code>
nrvalues	numeric; dimension $k$ of the original model, i.e.; length of the untransformed parameter, or joint length of main and nuisance part of the parameter.
x	the data at which the starting estimator is to be evaluated.
na.rm	logical: if TRUE, the estimator is evaluated at <code>complete.cases(x)</code> .
startList	a list of arguments to be given to the call to <code>start</code> if this is a function;
L2Fam	the parametric family;
...	further arguments for <code>kStepEstimator.start</code> .

**Value**

a numeric vector with the corresponding value of the start estimator (in  $k$  space)

**Methods**

**kStepEstimator.start** signature(`start = "numeric"`): returns the unchanged argument `start` if it has the correct length; otherwise throws an error.

**kStepEstimator.start** signature(`start = "Estimate"`): returns slot `untransformed.estimate` of `start` if it is not NULL, and else slot `estimate` if the latter has dimension `nrvalues`.

**kStepEstimator.start** signature(`start = "function"`): returns `kStepEstimator.start(do.call(start, args=c(list(x, na.rm)), startList))` where, if `na.rm == TRUE`, beforehand `x` has been modified to `x <- complete.cases(x)`.

**Author(s)**

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

**References**

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.

**See Also**

[kStepEstimator, ALEstimate-class](#)

---

locMEstimator

*Generic function for the computation of location M estimates*

---

**Description**

Generic function for the computation of location M estimates.

**Usage**

```
locMEstimator(x, IC, ...)  
  
## S4 method for signature 'numeric,InfluenceCurve'  
locMEstimator(x, IC, eps = .Machine$double.eps^0.5, na.rm = TRUE)
```

**Arguments**

x	sample
IC	object of class "InfluenceCurve"
...	additional parameters
eps	the desired accuracy (convergence tolerance).
na.rm	logical: if TRUE, the estimator is evaluated at complete.cases(x).

**Details**

Given some sample  $x$  and some influence curve  $IC$  an  $M$  estimate is computed by solving the corresponding  $M$  equation.

**Value**

Object of class "MEstimate"

**Methods**

$x = \text{"numeric"}$ ,  $IC = \text{"InfluenceCurve"}$  univariate location.

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

- Huber, P.J. (1964) Robust estimation of a location parameter. *Ann. Math. Stat.* **35**: 73–101.
- Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.
- Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[InfluenceCurve-class](#), [MEstimate-class](#)

---

makeIC-methods	<i>Generic Function for making ICs consistent at a possibly different model</i>
----------------	---------------------------------------------------------------------------------

---

### Description

Generic function for providing centering and Fisher consistency of ICs.

### Usage

```
makeIC(IC, L2Fam, ...)
## S4 method for signature 'IC,missing'
makeIC(IC)
## S4 method for signature 'IC,L2ParamFamily'
makeIC(IC, L2Fam)
## S4 method for signature 'list,L2ParamFamily'
makeIC(IC, L2Fam, forceIC = TRUE, name, Risks,
        Infos, modifyIC = NULL)
## S4 method for signature 'function,L2ParamFamily'
makeIC(IC, L2Fam, forceIC = TRUE, name,
        Risks, Infos, modifyIC = NULL)
```

### Arguments

IC	object of class "IC" for signature IC="IC", respectively a list of functions in one argument for signature IC="list", respectively a function in one argument for signature IC="function".
L2Fam	L2-differentiable family of probability measures; may be missing.
forceIC	logical; shall centeredness and Fisher consistency be enforced applying an affine linear transformation?
name	Object of class "character"; the name of the IC
Risks	object of class "list": list of risks; cf. <a href="#">RiskType-class</a> .
Infos	matrix of characters with two columns named method and message: additional informations.
modifyIC	Object of class "OptionalFunction": function of two arguments, which are an L2 parametric family and an optional influence curve. Returns an object of class "IC" at the parameter value of the L2 parametric family. This function is mainly used for internal computations!
...	additional parameters

### Value

An IC of class "IC" at the model.

## Methods

**makeIC** signature(IC = "IC", L2Fam = "missing"): creates an object of class "IC" at the parametric model of its own slot CallL2Fam; enforces IC conditions centeredness and Fisher consistency, applying an affine linear transformation.

**makeIC** signature(IC = "IC", L2Fam = "L2ParamFamily"): creates an object of class "IC" at the parametric model L2Fam; enforces IC conditions centeredness and Fisher consistency, applying an affine linear transformation.

**makeIC** signature(IC = "list", L2Fam = "L2ParamFamily"): creates an object of class "IC" out of a list of functions given by argument IC at the parametric model L2Fam; enforces IC conditions centeredness and Fisher consistency, applying an affine linear transformation.

**makeIC** signature(IC = "function", L2Fam = "L2ParamFamily"): creates an object of class "IC" out of a function given by argument IC at the parametric model L2Fam; enforces IC conditions centeredness and Fisher consistency, applying an affine linear transformation.

## Author(s)

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

## References

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.

Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

## See Also

[L2ParamFamily-class](#), [IC-class](#)

## Examples

```
## default IC
IC1 <- new("IC")

## L2-differentiable parametric family
B <- BinomFamily(13, 0.3)

## check IC properties
checkIC(IC1, B)

## make IC
IC2 <- makeIC(IC1, B)

## check IC properties
checkIC(IC2)

## slot modifyIC is filled in case of IC2
IC3 <- modifyIC(IC2)(BinomFamily(13, 0.2), IC2)
checkIC(IC3)
## identical to
```

```

checkIC(IC3, BinomFamily(13, 0.2))

IC4 <- makeIC(sin, B)
checkIC(IC4)

(IC5 <- makeIC(list(function(x)x^3), B, name="a try"))
plot(IC5)
checkIC(IC5)

## don't run to reduce check time on CRAN
## Not run:
N0 <- NormLocationScaleFamily()
IC6 <- makeIC(list(sin,cos),N0)
plot(IC6)
checkIC(IC6)

getRiskIC(IC6,risk=trAsCov())$trAsCov$value
getRiskIC(IC6,risk=asBias(),neighbor=ContNeighborhood())$asBias$value

## End(Not run)

```

---

masked-methods

*Masked Methods from Packages 'stats' and 'graphics' in Package 'RobAStBase'*


---

## Description

masked methods from packages **stats** and **graphics**

## Usage

```

clip(x1,...)
## S4 method for signature 'ANY'
clip(x1,x2,y1,y2)
start(x,...)
## S4 method for signature 'ANY'
start(x,...)

```

## Arguments

`x, ...` see [start](#).  
`x1,x2,y1,y2` see [clip](#).

## Details

In order to make accessible the otherwise masked functions [start](#), [clip](#), we generate corresponding S4-methods.



**Value**

see [start](#), [clip](#)

**Author(s)**

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

---

MEstimate-class	<i>MEstimate-class.</i>
-----------------	-------------------------

---

**Description**

Class of asymptotically linear estimates.

**Objects from the Class**

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

**Slots**

`name` Object of class "character": name of the estimator.  
`estimate` Object of class "ANY": estimate.  
`samplesize` Object of class "numeric": sample size.  
`asvar` Optional object of class "matrix": asymptotic variance.  
`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.  
`Mroot` Object of class "numeric": value of the M equation at the estimate.  
`Infos` object of class "matrix" with two columns named `method` and `message`: additional informations.

**Extends**

Class "ALEstimate", directly.  
 Class "Estimate", by class "ALEstimate".

**Methods**

**Mroot** `signature(object = "MEstimate")`: accessor function for slot `Mroot`.  
**show** `signature(object = "MEstimate")`

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[ALEstimate-class](#)

**Examples**

```
## prototype
new("MEstimate")
```

---

movToRef-methods	<i>Methods for Functions moving from and to reference parameter in Package 'ROptEst'</i>
------------------	------------------------------------------------------------------------------------------

---

**Description**

In optIC a gain in accuracy can be obtained when computing the optimally-robust ICs at a reference parameter of the model (instead of an arbitrary one). To this end, moveL2Fam2RefParam moved the model to the reference parameter and moveICBackFromRefParam moves the obtained optimal IC back to the original parameter.

**Usage**

```
moveL2Fam2RefParam(L2Fam, ...)
moveICBackFromRefParam(IC, L2Fam, ...)
```

**Arguments**

L2Fam	object of class L2ParamFamily
IC	IC of class HampIC
...	further arguments to be passed on.

**Details**

moveL2Fam2RefParam and moveICBackFromRefParam are used internally in functions robest and roptest to compute the optimally robust influence function according to the arguments given to them.

**Value**

moveL2Fam2RefParam	the L2 Family transformed to reference parameter.
moveICBackFromRefParam	the backtransformed IC.

**Methods**

- moveL2Fam2RefParam** signature(L2Fam = "L2ParamFamily"): returns L2Fam unchanged.
- moveL2Fam2RefParam** signature(L2Fam = "L2LocationFamily"): moves L2Fam to location 0.
- moveL2Fam2RefParam** signature(L2Fam = "L2ScaleFamily"): moves L2Fam to location 0 and scale 1.
- moveL2Fam2RefParam** signature(L2Fam = "L2LocationScaleFamily"): moves L2Fam to location 0 and scale 1.
- moveL2Fam2RefParam** signature(L2Fam = "L2LocationUnknownScaleFamily"): moves L2Fam to location 0 and scale 1.
- moveL2Fam2RefParam** signature(L2Fam = "L2ScaleUnknownLocationFamily"): moves L2Fam to location 0 and scale 1.
- moveICBackFromRefParam** signature(IC = "IC", L2Fam = "L2ParamFamily"): returns IC unchanged.
- moveICBackFromRefParam** signature(IC = "IC", L2Fam = "L2LocationFamily"): moves IC in IC back to original location in L2Fam.
- moveICBackFromRefParam** signature(IC = "IC", L2Fam = "L2ScaleFamily"): moves IC in IC back to original location and scale in L2Fam, rescaling risk where necessary.
- moveICBackFromRefParam** signature(IC = "IC", L2Fam = "L2LocationScaleFamily"): moves IC in IC back to original location and scale in L2Fam, rescaling risk where necessary.
- moveICBackFromRefParam** signature(IC = "IC", L2Fam = "L2LocationUnknownScaleFamily"): moves IC in IC back to original location and scale in L2Fam, rescaling risk where necessary.
- moveICBackFromRefParam** signature(IC = "IC", L2Fam = "L2ScaleUnknownLocationFamily"): moves IC in IC back to original location and scale in L2Fam, rescaling risk where necessary.
- moveICBackFromRefParam** signature(IC = "HampIC", L2Fam = "L2ParamFamily"): moves IC in IC back to original location and scale in L2Fam (and in addition changes Lagrange multipliers accordingly), rescaling risk where necessary.

**Author(s)**

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

---

Neighborhood-class      *Neighborhood*

---

**Description**

Class of neighborhoods of families of probability measures.

**Objects from the Class**

A virtual Class: No objects may be created from it.

**Slots**

**type** Object of class "character": type of the neighborhood.

**radius** Object of class "numeric": neighborhood radius.

**Methods**

**type** signature(object = "Neighborhood"): accessor function for slot type.

**radius** signature(object = "Neighborhood"): accessor function for slot radius.

**show** signature(object = "Neighborhood")

**radius<-** signature(object = "Neighborhood"): replacement function for slot radius.

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.

Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[ProbFamily-class](#)

---

oneStepEstimator

*Function for the computation of one-step estimates*

---

**Description**

Function for the computation of one-step estimates.

**Usage**

```
oneStepEstimator(x, IC, start = NULL,
  useLast = getRobASTBaseOption("kStepUseLast"),
  withUpdateInKer = getRobASTBaseOption("withUpdateInKer"),
  IC.UpdateInKer = getRobASTBaseOption("IC.UpdateInKer"),
  na.rm = TRUE, startArgList = NULL, ...)
```

**Arguments**

<code>x</code>	sample
<code>IC</code>	object of class "InfluenceCurve"
<code>start</code>	initial estimate (for full parameter, i.e. in dimension $k$ respective joint length of main and nuisance part of the parameter): either a numerical value, or an object of class "Estimate" or a function producing either a numerical value, or an object of class "Estimate" when evaluated at $x, \dots$ ; if missing or NULL, we use slot <code>startPar</code> of the L2family L2Fam from within IC.
<code>useLast</code>	which parameter estimate (initial estimate or one-step estimate) shall be used to fill the slots <code>pIC</code> , <code>asvar</code> and <code>asbias</code> of the return value.
<code>withUpdateInKer</code>	if there is a non-trivial trafo in the model with matrix $D$ , shall the parameter be updated on $\ker(D)$ ?
<code>IC.UpdateInKer</code>	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 <code>getboundedIC(L2Fam, D)</code> is taken; this IC will then be projected onto $\ker(D)$ .
<code>na.rm</code>	logical: if TRUE, the estimator is evaluated at <code>complete.cases(x)</code> .
<code>startArgList</code>	a list of arguments to be given to argument <code>start</code> if the latter is a function; this list by default already starts with two unnamed items, the sample $x$ , and the model <code>eval(CallL2Fam(IC))</code> ; in case IC is not of class IC, the model argument L2Fam will be set to NULL.
<code>...</code>	additional arguments

**Details**

Given an initial estimation `start`, a sample  $x$  and an influence curve IC the corresponding one-step estimator is computed.

In case IC is an object of class "IC" the slots `asvar` and `asbias` of the return value are filled (based on the initial estimate).

The default value of argument `useLast` is set by the global option `kStepUseLast` which by default is set to FALSE. In case of general models `useLast` remains unchanged during the computations. However, if slot `CallL2Fam` of IC generates an object of class "L2GroupParamFamily" the value of `useLast` is changed to TRUE. Explicitly setting `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 `useLast` is set to TRUE and slot `modifyIC` of IC is filled with some function (which can be used to re-compute the IC for a different parameter), the computation of `asvar`, `asbias` and IC is based on the one-step estimate.

**Value**

Object of class "kStepEstimate"

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>,  
Peter Ruckdeschel <Peter.Ruckdeschel@itwm.fraunhofer.de>

**References**

- Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.
- Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[InfluenceCurve-class](#), [kStepEstimate-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 'L2ParamFamily,asCov'
optIC(model, risk)
```

**Arguments**

model	probability model.
risk	object of class "RiskType".
...	additional parameters.

**Details**

The classical optimal IC which ist optimal in sense of the Cramer-Rao bound is computed.

**Value**

Some optimally robust IC is computed.

**Methods**

**model = "L2ParamFamily", risk = "asCov"** computes classical optimal influence curve for L2 differentiable parametric families.

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

- Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.
- Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**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)
```

---

OptionalInfluenceCurve-class

*Some helper Classes in package 'RobAStBase'*

---

**Description**

Some helper Classes in package 'RobAStBase': Classes OptionalInfluenceCurve, OptionalpICList, StartClass, pICList

**Class Unions**

OptionalInfluenceCurve is a class union of classes InfluenceCurve and NULL — it is the slot class of slot pIC in ALEstimate; OptionalpICList is a class union of classes pICList and NULL — it is the slot class of slot pICList in kStepEstimate; StartClass is a class union of classes function, numeric and Estimate — it is the slot class of slot start in kStepEstimate.

**List Classes**

pICList is a descendant of class list which requires its members —if any— to be of class pIC.

**Methods**

**show** signature(object = "OptionalpICList"): particular show-method.

**show** signature(object = "pICList"): particular show-method.

**Author(s)**

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

## References

- Hampel et al. (1986) *Robust Statistics. The Approach Based on Influence Functions*. New York: Wiley.
- Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.
- Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

## See Also

[InfluenceCurve](#), [RiskType-class](#)

---

outlyingPlotIC	<i>Function outlyingPlotIC in Package 'RobAStBase'</i>
----------------	--------------------------------------------------------

---

## Description

outlyingPlotIC produces an outlyingness plot based on distances applied to ICs

## Usage

```
outlyingPlotIC(data, IC.x, IC.y = IC.x, dist.x = NormType(),
               dist.y, cutoff.y = cutoff.chisq(), cutoff.x = cutoff.sememp(),
               ..., cutoff.quantile.x = 0.95,
               cutoff.quantile.y = cutoff.quantile.x, id.n, lab.pts, adj,
               cex.idn, col.idn, lty.cutoff, lwd.cutoff, col.cutoff,
               robCov.x = TRUE, robCov.y = TRUE, tf.x = data, tf.y = data,
               jitt.fac = 10,
               main = gettext("Outlyingness \n by means of a distance-distance plot")
               )
```

## Arguments

data	data coercable to matrix; the data at which to produce the ddPlot.
IC.x	object of class IC the influence curve to produce the distances for the x axis.
IC.y	object of class IC the influence curve to produce the distances for the y axis.
...	further arguments to be passed to plot.default, text, and abline
dist.x	object of class NormType; the distance for the x axis.
dist.y	object of class NormType; the distance for the y axis.
cutoff.x	object of class cutoff; the cutoff information for the x axis (the vertical line discriminating 'good' and 'bad' points).
cutoff.y	object of class cutoff; the cutoff information for the y axis (the horizontal line discriminating 'good' and 'bad' points).
cutoff.quantile.x	numeric; the cutoff quantile for the x axis.



<code>cutoff.quantile.y</code>	numeric; the cutoff quantile for the y axis.
<code>id.n</code>	a set of indices (or a corresponding logical vector); to select a subset of the data in argument data.
<code>lab.pts</code>	a vector of labels for the (unsubsetting) data.
<code>adj</code>	the corresponding argument for <code>text</code> for labelling the outliers.
<code>cex.idn</code>	the corresponding <code>cex</code> argument for <code>text</code> for labelling the outliers.
<code>col.idn</code>	the corresponding <code>col</code> argument for <code>text</code> for labelling the outliers.
<code>lty.cutoff</code>	the corresponding <code>lty</code> argument for <code>abline</code> for drawing the cutoff lines.
<code>lwd.cutoff</code>	the corresponding <code>lwd</code> argument for <code>abline</code> for drawing the cutoff lines.
<code>col.cutoff</code>	the corresponding <code>col</code> argument for <code>abline</code> for drawing the cutoff lines.
<code>robCov.x</code>	shall x-distances be based on MCD, i.e., robust covariances (TRUE) or on classical covariance be used?
<code>robCov.y</code>	shall y-distances be based on MCD, i.e., robust covariances (TRUE) or on classical covariance be used?
<code>tf.x</code>	transformation for x axis: a function returning the transformed x-coordinates when applied to the data; by default identity.
<code>tf.y</code>	transformation for y axis: a function returning the transformed y-coordinates when applied to the data; by default identity.
<code>jitt.fac</code>	factor for jittering, see <code>jitter</code> ;
<code>main</code>	the main title.

### Details

calls a corresponding `ddPlot` method to produce the plot.

### Value

a list with items

<code>id.x</code>	the indices of (possibly transformed) data (within subset <code>id.n</code> ) beyond the x-cutoff
<code>id.y</code>	the indices of (possibly transformed) data (within subset <code>id.n</code> ) beyond the y-cutoff
<code>id.xy</code>	the indices of (possibly transformed) data (within subset <code>id.n</code> ) beyond the x-cutoff and the y-cutoff
<code>qtx</code>	the quantiles of the distances of the (possibly transformed) data in x direction
<code>qty</code>	the quantiles of the distances of the (possibly transformed) data in y direction
<code>cutoff.x.v</code>	the cutoff value in x direction
<code>cutoff.y.v</code>	the cutoff value in y direction

### Author(s)

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

**Examples**

```

if(require(ROptEst)){
## generates normal location and scale family with mean = -2 and sd = 3
N0 <- NormLocationScaleFamily()
N0.IC0 <- optIC(model = N0, risk = asCov())
N0.Rob1 <- InfRobModel(center = N0, neighbor = ContNeighborhood(radius = 0.5))
N0.IC1 <- optIC(model = N0.Rob1, risk = asMSE())
xn <- c(rnorm(100),rcauchy(20)+20)
outlyingPlotIC(xn, IC.x=N0.IC0)
outlyingPlotIC(xn, IC.x=N0.IC1)
}

```

plot-methods

*Methods for Function plot in Package 'RobAStBase'***Description**

plot-methods

**Usage**

```

plot(x, y, ...)
## S4 method for signature 'IC,missing'
plot(x, ..., withSweave = getdistrOption("withSweave"),
      main = FALSE, inner = TRUE, sub = FALSE,
      col.inner = par("col.main"), cex.inner = 0.8,
      bmar = par("mar")[1], tmar = par("mar")[3],
      with.legend = FALSE, legend = NULL, legend.bg = "white",
      legend.location = "bottomright", legend.cex = 0.8,
      withMBR = FALSE, MBRB = NA, MBR.fac = 2, col.MBR = par("col"),
      lty.MBR = "dashed", lwd.MBR = 0.8,
      scaleX = FALSE, scaleX.fct, scaleX.inv,
      scaleY = FALSE, scaleY.fct = pnorm, scaleY.inv=qnorm,
      scaleN = 9, x.ticks = NULL, y.ticks = NULL,
      mfColRow = TRUE, to.draw.arg = NULL)
## S4 method for signature 'IC,numeric'
plot(x, y, ..., cex.pts = 1,
      col.pts = par("col"), pch.pts = 1, jitter.fac = 1, with.lab = FALSE,
      lab.pts = NULL, lab.font = NULL, alpha.trsp = NA,
      which.lbs = NULL, which.Order = NULL, return.Order = FALSE)

```

**Arguments**

x	object of class "IC": IC to be plotted
y	missing or numeric (a dataset, e.g.)
withSweave	logical: if TRUE (for working with Sweave) no extra device is opened

main	logical: is a main title to be used? or just as argument main in <code>plot.default</code> .
inner	logical: do panels have their own titles? or character vector of / cast to length 'number of plotted dimensions'; if argument <code>to.draw.arg</code> is used, this refers to a vector of length <code>length(to.draw.arg)</code> , the actually plotted dimensions. For further information, see also description of argument main in <code>plot.default</code> .
sub	logical: is a sub-title to be used? or just as argument sub in <code>plot.default</code> .
tmar	top margin – useful for non-standard main title sizes
bmar	bottom margin – useful for non-standard sub title sizes
cex.inner	magnification to be used for inner titles relative to the current setting of cex; as in <code>par</code>
col.inner	character or integer code; color for the inner title
with.legend	logical; shall a legend be plotted?
legend	either NULL or a list of length (number of plotted panels) of items which can be used as argument legend in command legend.
legend.location	a valid argument x for <code>legend</code> — the place where to put the legend on the last issued plot — or a list of length (number of plotted panels) of such arguments, one for each plotted panel.
legend.bg	background color for the legend
legend.cex	magnification factor for the legend
withMBR	logical; shall horizontal lines with min and max of MBRE be plotted for comparison?
MBRB	matrix (or NA); coerced by usual recycling rules to a matrix with as many rows as plotted panels and with first column the lower bounds and the second column the upper bounds for the respective coordinates (ideally given by the MBR-IC).
MBR.fac	positive factor; scales the bounds given by argument MBRB
col.MBR	color for the MBR lines; as usual col-argument;
lty.MBR	line type for the MBR lines; as usual lty-argument;
lwd.MBR	line width for the MBR lines; as usual lwd-argument;
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 to [0,1]; if <code>scaleX</code> is TRUE and <code>scaleX.fct</code> is missing, the cdf of the underlying observation distribution.
scaleX.inv	the inverse function to <code>scale.fct</code> , i.e., an isotone, vectorized function mapping [0,1] to the domain of the IC such that for any x in the domain, <code>scaleX.inv(scaleX.fct(x))=x</code> ; if <code>scaleX</code> is TRUE and <code>scaleX.inv</code> is missing, the quantile function of the underlying observation distribution.

<code>scaleY.fct</code>	an isotone, vectorized function mapping for each coordinate the range of the respective coordinate of the IC to [0,1]; defaulting to the cdf of $\mathcal{N}(0, 1)$ .
<code>scaleY.inv</code>	an isotone, vectorized function mapping for each coordinate the range [0,1] into the range of the respective coordinate of the IC; defaulting to the quantile function of $\mathcal{N}(0, 1)$ .
<code>scaleN</code>	integer; defaults to 9; on rescaled axes, number of x and y ticks if drawn automatically;
<code>x.ticks</code>	numeric; defaults to NULL; (then ticks are chosen automatically); if non-NULL, user-given x-ticks (on original scale);
<code>y.ticks</code>	numeric; defaults to NULL; (then ticks are chosen automatically); if non-NULL, user-given y-ticks (on original scale); can be a list with one (numeric or NULL) item per panel
<code>mfColRow</code>	shall default partition in panels be used — defaults to TRUE
<code>to.draw.arg</code>	Either NULL (default; everything is plotted) or a vector of either integers (the indices of the subplots to be drawn) or characters — the names of the subplots to be drawn: these names are to be chosen either among the row names of the trafo matrix <code>rownames(trafo(eval(x@CallL2Fam)@param))</code> or if the last expression is NULL a vector "dim<dimnr>", <code>dimnr</code> running through the number of rows of the trafo matrix.
<code>cex.pts</code>	size of the points of the second argument plotted
<code>col.pts</code>	color of the points of the second argument plotted
<code>pch.pts</code>	symbol of the points of the second argument plotted
<code>with.lab</code>	logical; shall labels be plotted to the observations?
<code>lab.pts</code>	character or NULL; labels to be plotted to the observations; if NULL observation indices;
<code>lab.font</code>	font to be used for labels
<code>alpha.trsp</code>	alpha transparency to be added ex post to colors <code>col.pch</code> and <code>col.lbl</code> ; if one-dim and NA all colors are left unchanged. Otherwise, with usual recycling rules <code>alpha.trsp</code> gets shorted/prolongated to length the data-symbols to be plotted. Coordinates of this vector <code>alpha.trsp</code> with NA are left unchanged, while for the remaining ones, the alpha channel in rgb space is set to the respective coordinate value of <code>alpha.trsp</code> . The non-NA entries must be integers in [0,255] (0 invisible, 255 opaque).
<code>jitter.fac</code>	jittering factor used in case of a <code>DiscreteDistribution</code> for plotting points of the second argument in a jittered fashion.
<code>which.lbs</code>	either an integer vector with the indices of the observations to be plotted into graph or NULL — then no observation is excluded
<code>which.Order</code>	we order the observations (descending) according to the norm given by <code>normtype(object)</code> ; then <code>which.Order</code> either is an integer vector with the indices of the <i>ordered</i> observations (remaining after a possible reduction by argument <code>which.lbs</code> ) to be plotted into graph or NULL — then no (further) observation is excluded.
<code>return.Order</code>	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 <code>which.lbs</code> , and ordering

is according to the norm given by `normtype(object)`); otherwise we return `invisible()` as usual.

... further parameters for `plot`

## Details

Any parameters of `plot.default` may be passed on to this particular plot method.

We start describing the IC,missing-method: For main-, inner, and subtitles given as arguments `main`, `inner`, and `sub`, top and bottom margins are enlarged to 5 resp. 6 by default but may also be specified by `tmar` / `bmar` arguments. If `main` / `inner` / `sub` are logical then if the respective argument is `FALSE` nothing is done/plotted, but if it is `TRUE`, we use a default main title taking up the calling arguments in case of `main`, default inner titles taking up the class and (named) parameter slots of arguments in case of `inner`, and a "generated on <data>"-tag in case of `sub`. Of course, if `main` / `inner` / `sub` are character, this is used for the title; in case of `inner` it is then checked whether it has correct length. In all title arguments, the following patterns are substituted:

```
"%C" class of argument object
"%A" deparsed argument object
"%D" time/date-string when the plot was generated
```

If argument ... contains argument `ylim`, this may either be as in `plot.default` (i.e. a vector of length 2) or a vector of length  $2 \times (\text{number of plotted dimensions} + 2)$ , where the first two elements are the values for `ylim` in panel "d", the first two are for `ylim` resp. `xlim` for panels "p" and "q", and the last  $2 \times (\text{number of plotted dimensions})$  are the values for `ylim` for the plotted dimensions of the L2derivative, one pair for each dimension.

The IC,numeric-method calls the IC,missing-method but in addition plots the values of a dataset into the IC.

## Examples

```
IC1 <- new("IC")
plot(IC1)
plot(IC1, main = TRUE, panel.first= grid(),
      col = "blue", cex.main = 2, cex.inner = 1)

### selection of subpanels for plotting
N <- NormLocationScaleFamily(mean=0, sd=1)
IC2 <- optIC(model = N, risk = asCov())
par(mfrow=c(1,1))
plot(IC2, main = TRUE, panel.first= grid(),
      col = "blue", cex.main = 2, cex.inner = 0.6,
      mfColRow = FALSE, to.draw.arg=c("sd"))

## xlim and ylim arguments
plot(IC2, main = TRUE, panel.first= grid(),
      ylim=c(-3,3), xlim=c(-2,3))
plot(IC2, main = TRUE, panel.first= grid(),
      ylim=c(-3,3,-1,3), xlim=c(-2,3),
      with.legend = TRUE)
```

```
data <- r(N)(30)
plot(IC2, data, panel.first= grid(),
      ylim = c(-3,3,-1,3), xlim=c(-2,3),
      cex.pts = 3, pch.pts = 1:2, col.pts="green",
      with.lab = TRUE, which.lbs = c(1:4,15:20),
      which.Order = 1:6, return.Order = TRUE)
```

---

PlotIC

*Wrapper function for plot method for IC*


---

### Description

The wrapper PlotIC takes most of arguments to the plot method by default and gives a user possibility to run the function with low number of arguments.

### Usage

```
PlotIC(IC, y, ..., alpha.trsp = 100, with.legend = TRUE,
       rescale = FALSE, withCall = TRUE)
```

### Arguments

IC	object of class IC
y	optional data argument — for plotting observations into the plot
...	additional parameters (in particular to be passed on to plot)
alpha.trsp	the transparency argument (0 to 100) for plotting the data
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 plot with suitably chosen defaults; if withCall == TRUE, the call to plot is returned.

### Examples

```
# Gamma
fam <- GammaFamily()
rfam <- InfRobModel(fam, ContNeighborhood(0.5))
IC <- optIC(model = fam, risk = asCov())
Y <- distribution(fam)
y <- r(Y)(1000)
PlotIC(IC, y, withCall = FALSE)
```

**Description**

We generalize function `qqplot` from package `stats` to be applicable to distribution and probability model objects. In this context, `qqplot` produces a QQ plot of data (argument `x`) against a (model) distribution. For arguments `y` of class `RobModel`, points at a high “distance” to the model are plotted smaller. For arguments `y` of class `kStepEstimate`, points at with low weight in the [p]IC are plotted bigger and their color gets faded out slowly. Graphical parameters may be given as arguments to `qqplot`.

**Usage**

```
qqplot(x, y, ...)
## S4 method for signature 'ANY,RobModel'
qqplot(x, y,
       n = length(x), withIdLine = TRUE, withConf = TRUE,
       withConf.pw = withConf, withConf.sim = withConf,
       plot.it = TRUE, xlab = deparse(substitute(x)),
       ylab = deparse(substitute(y)), ..., distance = NormType(),
       n.adj = TRUE)
## S4 method for signature 'ANY,InfRobModel'
qqplot(x, y,
       n = length(x), withIdLine = TRUE, withConf = TRUE,
       withConf.pw = withConf, withConf.sim = withConf,
       plot.it = TRUE, xlab = deparse(substitute(x)),
       ylab = deparse(substitute(y)), ..., n.adj = TRUE)
## S4 method for signature 'ANY,kStepEstimate'
qqplot(x, y,
       n = length(x), withIdLine = TRUE, withConf = TRUE,
       withConf.pw = withConf, withConf.sim = withConf,
       plot.it = TRUE, xlab = deparse(substitute(x)),
       ylab = deparse(substitute(y)), ...,
       exp.cex2.lbl = -.15,
       exp.cex2.pch = -.35,
       exp.fadcol.lbl = 1.85,
       exp.fadcol.pch = 1.85,
       bg = "white")
```

**Arguments**

<code>x</code>	data to be checked for compatibility with distribution/model <code>y</code> .
<code>y</code>	object of class "RobModel", of class "InfRobModel" or of class "kStepEstimate".
<code>n</code>	numeric; number of quantiles at which to do the comparison.

<code>withIdLine</code>	logical; shall line $y = x$ be plotted in?
<code>withConf</code>	logical; shall confidence lines be plotted?
<code>withConf.pw</code>	logical; shall pointwise confidence lines be plotted?
<code>withConf.sim</code>	logical; shall simultaneous confidence lines be plotted?
<code>plot.it</code>	logical; shall be plotted at all (inherited from <code>qqplot</code> )?
<code>xlab</code>	x-label
<code>ylab</code>	y-label
<code>...</code>	further parameters for method <code>qqplot</code> with signature <code>ANY, ProbFamily</code> (see <code>qqplot</code> ) or with function <code>plot</code>
<code>n.adj</code>	logical; shall sample size be adjusted for possible outliers according to radius of the corresponding neighborhood?
<code>distance</code>	a function mapping observations $x$ to the positive reals; used to determine the size of the plotted points (the larger <code>distance(x)</code> , the smaller the points are plotted).
<code>exp.cex2.lbl</code>	for objects <code>kStepEstimate</code> based on a [p]IC of class <code>HampIC</code> : exponent for the weights of this [p]IC used to magnify the labels.
<code>exp.cex2.pch</code>	for objects <code>kStepEstimate</code> based on a [p]IC of class <code>HampIC</code> : exponent for the weights of this [p]IC used to magnify the symbols.
<code>exp.fadcol.lbl</code>	for objects <code>kStepEstimate</code> based on a [p]IC of class <code>HampIC</code> : exponent for the weights of this [p]IC used to find out-fading colors.
<code>exp.fadcol.pch</code>	for objects <code>kStepEstimate</code> based on a [p]IC of class <code>HampIC</code> : exponent for the weights of this [p]IC used to find out-fading colors.
<code>bg</code>	background color to fade against

## Details

**qqplot** signature(`x = "ANY"`, `y = "RobModel"`): produces a QQ plot of a dataset  $x$  against the theoretical quantiles of distribution of robust model  $y$ .

**qqplot** signature(`x = "ANY"`, `y = "InfRobModel"`): produces a QQ plot of a dataset  $x$  against the theoretical quantiles of distribution of infinitesimally robust model  $y$ .

**qqplot** signature(`x = "ANY"`, `y = "kStepEstimate"`): produces a QQ plot of a dataset  $x$  against the theoretical quantiles of the model distribution of model at which the corresponding `kStepEstimate`  $y$  had been calibrated at. By default, if the [p]IC of the `kStepEstimate` is of class `HampIC`, i.e.; has a corresponding weight function, points (and, if `withLab==TRUE`, labels) are scaled and faded according to this weight function. Corresponding arguments `exp.cex2.pch` and `exp.fadcol.pch` control this scaling and fading, respectively (and analogously `exp.cex2.lbl` and `exp.fadcol.lbl` for the labels). The choice of these arguments has to be done on a case-by-case basis. Positive exponents induce fading, magnification with increasing weight, for negative exponents the same is true for decreasing weight; higher (absolute) values increase the speed of fading / magnification.



**Value**

As for function `qqplot` from package **stats**: a list with components

`x`                    The x coordinates of the points that were/would be plotted  
`y`                    The corresponding quantiles of the second distribution, *including NAs*.

**Author(s)**

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

**References**

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

**See Also**

`qqplot` from package **stats** – the standard QQ plot function, `qqplot` from package **distr** for comparisons of distributions, and `qqplot` from package **distrMod** (which is called intermediately by this method), as well as `qqbounds`, used by `qqplot` to produce confidence intervals.

**Examples**

```
qqplot(r(Norm(15,sqrt(30)))(40), Chisq(df=15))
RobM <- InfRobModel(center = NormLocationFamily(mean=13,sd=sqrt(28)),
                    neighbor = ContNeighborhood(radius = 0.4))
x <- r(Norm(15,sqrt(30)))(20)
qqplot(x, RobM)
qqplot(x, RobM, alpha.CI=0.9)
## further examples for ANY,kStepEstimator-method
## in example to roptest() in package R0ptEst
```

---

RobAStBaseMASK

*Masking off/by other functions in package "RobAStBase"*

---

**Description**

Provides information on the (intended) masking of and (non-intended) masking by other other functions in package **RobAStBase**

**Usage**

```
RobAStBaseMASK(library = NULL)
```

**Arguments**

`library`            a character vector with path names of R libraries, or NULL. The default value of NULL corresponds to all libraries currently known. If the default is used, the loaded packages are searched before the libraries

**Value**

no value is returned

**Author(s)**

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

**Examples**

```
RobASTBaseMASK()
```

---

RobASTBaseOptions	<i>Function to change the global variables of the package 'RobASTBase'</i>
-------------------	----------------------------------------------------------------------------

---

**Description**

With RobASTBaseOptions you can inspect and change the global variables of the package **RobAST-Base**.

**Usage**

```
RobASTBaseOptions(...)
getRobASTBaseOption(x)
```

**Arguments**

...	any options can be defined, using name = value or by passing a list of such tagged values.
x	a character string holding an option name.

**Value**

RobASTBaseOptions() returns a list of the global variables.  
 RobASTBaseOptions(x) returns the global variable x.  
 getRobASTBaseOption(x) returns the global variable x.  
 RobASTBaseOptions(x=y) sets the value of the global variable x to y.

**Global Options**

**kStepUseLast:** The default value of argument kStepUseLast is FALSE. Explicitly setting kStepUseLast to TRUE should be done with care as in this situation the influence curve in case of oneStepEstimator and kStepEstimator is re-computed using the value of the one- resp. k-step estimate which may take quite a long time depending on the model.

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

**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  $\ker(D)$ ; defaults to NULL.

**all.verbose:** argument `verbose` passed on by default to many calls of `optIC`, `radiusminimaxIC`, `getinfRobIC` etc.; well suited for testing purposes.

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

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

### Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

### See Also

[options](#), [getOption](#)

### Examples

```
RobASTBaseOptions()
RobASTBaseOptions("kStepUseLast")
RobASTBaseOptions("kStepUseLast" = TRUE)
# or
RobASTBaseOptions(kStepUseLast = 1e-6)
getRobASTBaseOption("kStepUseLast")
```

---

RobASTControl-class     *Control classes in package RobASTBase*

---

### Description

Control classes in package **RobASTBase**.

### Objects from the Class

This class is virtual; that is no objects may be created.

### Slots

name Object of class "character": name of the control object.

### Methods

**name** signature(object = "RobASTControl"): accessor function for slot name.

**name<-** signature(object = "RobASTControl", value = "character"): replacement function for slot name.

**Author(s)**

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

**References**

Hampel et al. (1986) *Robust Statistics. The Approach Based on Influence Functions*. New York: Wiley.

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.

Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

---

RobModel-class

*Robust model*

---

**Description**

Class of robust models. A robust model consists of family of probability measures center and a neighborhood neighbor about this family.

**Objects from the Class**

A virtual Class: No objects may be created from it.

**Slots**

center Object of class "ProbFamily"

neighbor Object of class "Neighborhood"

**Methods**

**center** signature(object = "RobModel"): accessor function for slot center.

**center<-** signature(object = "RobModel"): replacement function for slot center.

**neighbor** signature(object = "RobModel"): accessor function for slot neighbor.

**neighbor<-** signature(object = "RobModel"): replacement function for slot neighbor.

**trafo** signature(object = "RobModel", param = "missing"): accessor function for slot trafo of slot center.

**trafo<-** signature(object = "RobModel"): replacement function for slot trafo of slot center.

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

- Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.
- Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[ProbFamily-class](#), [Neighborhood-class](#)

---

RobWeight-class	<i>Robust Weight classes</i>
-----------------	------------------------------

---

**Description**

Classes for robust weights.

**Objects from the Class**

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

**Slots**

`name` Object of class "character".

`weight` Object of class "function" — the weight function.

**Methods**

**name** signature(object = "RobWeight"): accessor function for slot name.

**name<-** signature(object = "RobWeight"): replacement function for slot name.

**weight** signature(object = "RobWeight"): accessor function for slot weight.

**weight<-** signature(object = "RobWeight"): replacement function for slot weight.

**Author(s)**

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

**References**

- Hampel et al. (1986) *Robust Statistics. The Approach Based on Influence Functions*. New York: Wiley.
- Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.
- Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[InfluenceCurve-class, IC](#)

**Examples**

```
## prototype
new("RobWeight")
```

---

samplesize-methods      *Methods for Function samplesize in Package 'RobStBase'*

---

**Description**

samplesize-methods

**Methods**

**samplesize** signature(object = "interpolrisk"): returns the slot samplesize of an object of class "interpolrisk".

**samplesize<-** signature(object = "interpolrisk", value = "ANY"): modifies the slot samplesize of an object of class "interpolrisk".

**Examples**

```
myrisk <- MBRRisk(samplesize=100)
samplesize(myrisk)
samplesize(myrisk) <- 20
```

---

TotalVarIC      *Generating function for TotalVarIC-class*

---

**Description**

Generates an object of class "TotalVarIC"; i.e., an influence curves  $\eta$  of the form

$$\eta = c \vee A\Lambda \wedge d$$

with lower clipping bound  $c$ , upper clipping bound  $d$  and standardizing matrix  $A$ .  $\Lambda$  stands for the L2 derivative of the corresponding L2 differentiable parametric family which can be created via CallL2Fam.

**Usage**

```
TotalVarIC(name, CallL2Fam = call("L2ParamFamily"),
            Curve = EuclRandVarList(RealRandVariable(Map = c(function(x) {x}),
                                                    Domain = Reals()))),
            Risks, Infos, clipLo = -Inf, clipUp = Inf, stand = as.matrix(1),
            lowerCase = NULL, neighborRadius = 0, w = new("BdStWeight"),
            normtype = NormType(), biastype = symmetricBias(),
            modifyIC = NULL)
```

**Arguments**

name	object of class "character".
CallL2Fam	object of class "call": creates an object of the underlying L2-differentiable parametric family.
Curve	object of class "EuclRandVarList".
Risks	object of class "list": list of risks; cf. <a href="#">RiskType-class</a> .
Infos	matrix of characters with two columns named method and message: additional informations.
clipLo	negative real: lower clipping bound.
clipUp	positive real: lower clipping bound.
stand	matrix: standardizing matrix
w	BdStWeight: weight object
lowerCase	optional constant for lower case solution.
neighborRadius	radius of the corresponding (unconditional) contamination neighborhood.
biastype	BiasType: type of the bias
normtype	NormType: type of the norm
modifyIC	object of class "OptionalFunction": function of two arguments, which are an L2 parametric family and an optional influence curve. Returns an object of class "IC". This function is mainly used for internal computations!

**Value**

Object of class "TotalVarIC"

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.  
 Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[IC-class](#), [ContIC](#)

**Examples**

```
IC1 <- TotalVarIC()
plot(IC1)
```

---

TotalVarIC-class

*Influence curve of total variation type*

---

**Description**

Class of (partial) influence curves of total variation type. i.e., an influence curves  $\eta$  of the form

$$\eta = c \vee A\Lambda \wedge d$$

with lower clipping bound  $c$ , upper clipping bound  $d$  and standardizing matrix  $A$ .  $\Lambda$  stands for the L2 derivative of the corresponding L2 differentiable parametric family which can be created via `CallL2Fam`.

**Objects from the Class**

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

**Slots**

`CallL2Fam` object of class "call": creates an object of the underlying L2-differentiable parametric family.

`name` object of class "character".

`Curve` object of class "EuclRandVarList".

`modifyIC` Object of class "OptionalFunction": function of two arguments, which are an L2 parametric family and an optional influence curve. Returns an object of class "IC". This slot is mainly used for internal computations!

`Risks` object of class "list": list of risks; cf. [RiskType-class](#).

`Infos` object of class "matrix" with two columns named `method` and `message`: additional informations.

`clipLo` object of class "numeric": lower clipping bound.

`clipUp` object of class "numeric": upper clipping bound.

`stand` object of class "matrix": standardizing matrix.

`weight` object of class "BdStWeight": weight function

`biastype` object of class "BiasType": bias type (symmetric/onsided/asymmetric)

`normtype` object of class "NormType": norm type (Euclidean, information/self-standardized)

`neighborRadius` object of class "numeric": radius of the corresponding (unconditional) contamination neighborhood.



**Extends**

Class "HampIC", directly.  
 Class "IC", by class "HampIC".  
 Class "InfluenceCurve", by class "IC".

**Methods**

**CallL2Fam**<- signature(object = "TotalVarIC"): replacement function for slot CallL2Fam.  
**clipLo** signature(object = "TotalVarIC"): accessor function for slot clipLo.  
**clipLo**<- signature(object = "TotalVarIC"): replacement function for slot clipLo.  
**clipUp** signature(object = "TotalVarIC"): accessor function for slot clipUp.  
**clipUp**<- signature(object = "TotalVarIC"): replacement function for slot clipUp.  
**clip** signature(x1 = "TotalVarIC"): returns clipUp-clipLo.  
**stand**<- signature(object = "TotalVarIC"): replacement function for slot stand.  
**lowerCase**<- signature(object = "TotalVarIC"): replacement function for slot lowerCase.  
**neighbor** signature(object = "TotalVarIC"): generates an object of class "TotalVarNeighborhood" with radius given in slot neighborRadius.  
**generateIC** signature(neighbor = "TotalVarNeighborhood", L2Fam = "L2ParamFamily"): generate an object of class "TotalVarIC". Rarely called directly.  
**show** signature(object = "TotalVarIC")

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.  
 Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[IC-class](#), [ContIC](#), [HampIC-class](#)

**Examples**

```
IC1 <- new("TotalVarIC")
plot(IC1)
```

---

TotalVarNeighborhood    *Generating function for TotalVarNeighborhood-class*

---

**Description**

Generates an object of class "TotalVarNeighborhood".

**Usage**

```
TotalVarNeighborhood(radius = 0)
```

**Arguments**

radius                    non-negative real: neighborhood radius.

**Value**

Object of class "ContNeighborhood"

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.

Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[TotalVarNeighborhood-class](#)

**Examples**

```
TotalVarNeighborhood()

## The function is currently defined as
function(radius = 0){
  new("TotalVarNeighborhood", radius = radius)
}
```

---

TotalVarNeighborhood-class  
*Total variation neighborhood*

---

**Description**

Class of (unconditional) total variation neighborhoods.

**Objects from the Class**

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

**Slots**

type Object of class "character": "(uncond.) total variation neighborhood".

radius Object of class "numeric": neighborhood radius.

**Extends**

Class "UncondNeighborhood", directly.

Class "Neighborhood", by class "UncondNeighborhood".

**Methods**

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

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.

Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[TotalVarNeighborhood](#), [UncondNeighborhood-class](#)

**Examples**

```
new("TotalVarNeighborhood")
```

---

UncondNeighborhood-class

*Unconditional neighborhood*

---

**Description**

Class of unconditional (errors-in-variables) neighborhoods.

**Objects from the Class**

A virtual Class: No objects may be created from it.

**Slots**

type Object of class "character": type of the neighborhood.

radius Object of class "numeric": neighborhood radius.

**Extends**

Class "Neighborhood", directly.

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.

Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[Neighborhood-class](#)

# Index

## \*Topic **classes**

- ALEstimate-class, 4
- BdStWeight-class, 6
- BoundedWeight-class, 7
- ContIC-class, 16
- ContNeighborhood-class, 19
- cutoff-class, 21
- FixRobModel-class, 28
- getRiskFctBV-methods, 33
- HampelWeight-class, 38
- HampIC-class, 39
- IC-class, 42
- InfluenceCurve-class, 44
- InfRobModel-class, 52
- interpolRisk-class, 53
- kStepEstimate-class, 54
- kStepEstimator.start-methods, 59
- MEstimate-class, 65
- movToRef-methods, 66
- Neighborhood-class, 67
- OptionalInfluenceCurve-class, 71
- RobASTControl-class, 83
- RobModel-class, 84
- RobWeight-class, 85
- samplesize-methods, 86
- TotalVarIC-class, 88
- TotalVarNeighborhood-class, 91
- UncondNeighborhood-class, 92

## \*Topic **distribution**

- plot-methods, 74
- qqplot, 79
- RobASTBaseMASK, 81

## \*Topic **documentation**

- RobASTBaseMASK, 81

## \*Topic **hplot**

- cutoff, 20
- ddPlot-methods, 22
- outlyingPlotIC, 72
- qqplot, 79

## \*Topic **methods**

- ddPlot-methods, 22
- masked-methods, 64
- plot-methods, 74

## \*Topic **misc**

- RobASTBaseOptions, 82

## \*Topic **models**

- ContNeighborhood, 18
- ContNeighborhood-class, 19
- FixRobModel, 27
- FixRobModel-class, 28
- getBoundedIC, 32
- InfRobModel, 51
- InfRobModel-class, 52
- Neighborhood-class, 67
- RobModel-class, 84
- TotalVarNeighborhood, 90
- TotalVarNeighborhood-class, 91
- UncondNeighborhood-class, 92

## \*Topic **package**

- RobASTBase-package, 3

## \*Topic **programming**

- RobASTBaseMASK, 81

## \*Topic **robust**

- checkIC, 8
- comparePlot-methods, 10
- ContIC, 15
- evalIC, 26
- generateIC, 29
- generateIC.fct-methods, 30
- getBiasIC, 31
- getRiskIC, 33
- getweight-methods, 36
- IC, 40
- IC-class, 42
- InfluenceCurve, 43
- InfluenceCurve-class, 44
- infoPlot, 47
- kStepEstimator, 56

- locMEstimator, 60
- makeIC-methods, 62
- oneStepEstimator, 68
- optIC, 70
- OptionalInfluenceCurve-class, 71
- RobASTBaseOptions, 82
- TotalVarIC, 86
- TotalVarIC-class, 88
- \*Topic **univar**
  - kStepEstimator, 56
  - locMEstimator, 60
  - oneStepEstimator, 68
  
- abline, 24, 73
- addInfo<- (InfluenceCurve-class), 44
- addInfo<- ,InfluenceCurve-method (InfluenceCurve-class), 44
- addRisk<- (InfluenceCurve-class), 44
- addRisk<- ,InfluenceCurve-method (InfluenceCurve-class), 44
- ALEstimate-class, 4
- asbias (ALEstimate-class), 4
- asbias, ALEstimate-method (ALEstimate-class), 4
  
- BdStWeight-class, 6
- biastype, HampIC-method (HampIC-class), 39
- BoundedWeight-class, 7
  
- CallL2Fam (IC-class), 42
- CallL2Fam, IC-method (IC-class), 42
- CallL2Fam<- (IC-class), 42
- CallL2Fam<- ,ContIC-method (ContIC-class), 16
- CallL2Fam<- ,IC-method (IC-class), 42
- CallL2Fam<- ,TotalVarIC-method (TotalVarIC-class), 88
- cent (ContIC-class), 16
- cent, ContIC-method (ContIC-class), 16
- cent, HampelWeight-method (HampelWeight-class), 38
- cent<- (ContIC-class), 16
- cent<- ,ContIC-method (ContIC-class), 16
- cent<- ,HampelWeight-method (HampelWeight-class), 38
- center (RobModel-class), 84
- center, RobModel-method (RobModel-class), 84
  
- center<- (RobModel-class), 84
- center<- ,RobModel-method (RobModel-class), 84
- checkIC, 8
- checkIC, IC, L2ParamFamily-method (IC-class), 42
- checkIC, IC, missing-method (IC-class), 42
- clip, 64, 65
- clip (masked-methods), 64
- clip, ANY-method (masked-methods), 64
- clip, BoundedWeight-method (BoundedWeight-class), 7
- clip, ContIC-method (ContIC-class), 16
- clip, TotalVarIC-method (TotalVarIC-class), 88
- clip-methods (masked-methods), 64
- clip<- (ContIC-class), 16
- clip<- ,BoundedWeight-method (BoundedWeight-class), 7
- clip<- ,ContIC-method (ContIC-class), 16
- clipLo (TotalVarIC-class), 88
- clipLo, TotalVarIC-method (TotalVarIC-class), 88
- clipLo<- (TotalVarIC-class), 88
- clipLo<- ,TotalVarIC-method (TotalVarIC-class), 88
- clipUp (TotalVarIC-class), 88
- clipUp, TotalVarIC-method (TotalVarIC-class), 88
- clipUp<- (TotalVarIC-class), 88
- clipUp<- ,TotalVarIC-method (TotalVarIC-class), 88
- ComparePlot, 9
- comparePlot (comparePlot-methods), 10
- comparePlot, IC, IC-method (comparePlot-methods), 10
- comparePlot-methods, 10
- confint, ALEstimate, asymmetricBias-method (ALEstimate-class), 4
- confint, ALEstimate, missing-method (ALEstimate-class), 4
- confint, ALEstimate, onesidedBias-method (ALEstimate-class), 4
- confint, ALEstimate, symmetricBias-method (ALEstimate-class), 4
- ContIC, 15, 16, 18, 88, 89
- ContIC-class, 16
- ContNeighborhood, 18, 20

- ContNeighborhood-class, 19
- Curve (InfluenceCurve-class), 44
- Curve, InfluenceCurve-method (InfluenceCurve-class), 44
- cutoff, 20, 21, 22
- cutoff-class, 21
- cutoff.quantile (cutoff-class), 21
- cutoff.quantile, cutoff-method (cutoff-class), 21
- cutoff.quantile<- (cutoff-class), 21
- cutoff.quantile<-, cutoff-method (cutoff-class), 21
  
- ddPlot, 21, 22, 73
- ddPlot (ddPlot-methods), 22
- ddPlot, data.frame-method (ddPlot-methods), 22
- ddPlot, matrix-method (ddPlot-methods), 22
- ddPlot, numeric-method (ddPlot-methods), 22
- ddPlot-methods, 22
- Domain, InfluenceCurve-method (InfluenceCurve-class), 44
  
- evalIC, 26
- evalIC, IC, matrix-method (IC-class), 42
- evalIC, IC, numeric-method (IC-class), 42
  
- fct, cutoff-method (cutoff-class), 21
- FixRobModel, 27, 28
- FixRobModel-class, 28
  
- generateIC, 29
- generateIC, ContNeighborhood, L2ParamFamily-method (ContIC-class), 16
- generateIC, TotalVarNeighborhood, L2ParamFamily-method (TotalVarIC-class), 88
- generateIC.fct (generateIC.fct-methods), 30
- generateIC.fct, UncondNeighborhood, L2ParamFamily-method (generateIC.fct-methods), 30
- generateIC.fct-methods, 30
- getBiasIC, 31, 35
- getBiasIC, IC, UncondNeighborhood-method (getBiasIC), 31
- getBiasIC-methods (getBiasIC), 31
- getBoundedIC, 32
- getOption, 83
- getRiskFctBV (getRiskFctBV-methods), 33
- getRiskFctBV, asGRisk, ANY-method (getRiskFctBV-methods), 33
- getRiskFctBV, asMSE, ANY-method (getRiskFctBV-methods), 33
- getRiskFctBV, asSemivar, asymmetricBias-method (getRiskFctBV-methods), 33
- getRiskFctBV, asSemivar, onesidedBias-method (getRiskFctBV-methods), 33
- getRiskFctBV-methods, 33
- getRiskIC, 33, 36
- getRiskIC, IC, asBias, UncondNeighborhood, L2ParamFamily-method (getRiskIC), 33
- getRiskIC, IC, asBias, UncondNeighborhood, missing-method (getRiskIC), 33
- getRiskIC, IC, asCov, missing, L2ParamFamily-method (getRiskIC), 33
- getRiskIC, IC, asCov, missing, missing-method (getRiskIC), 33
- getRiskIC, IC, asMSE, UncondNeighborhood, L2ParamFamily-method (getRiskIC), 33
- getRiskIC, IC, asMSE, UncondNeighborhood, missing-method (getRiskIC), 33
- getRiskIC, IC, fiUnOvShoot, ContNeighborhood, missing-method (getRiskIC), 33
- getRiskIC, IC, fiUnOvShoot, TotalVarNeighborhood, missing-method (getRiskIC), 33
- getRiskIC, IC, trAsCov, missing, L2ParamFamily-method (getRiskIC), 33
- getRiskIC, IC, trAsCov, missing, missing-method (getRiskIC), 33
- getRiskIC, TotalVarIC, asUnOvShoot, UncondNeighborhood, missing-method (getRiskIC), 33
- getRiskIC-methods (getRiskIC), 33
- getRobASTBaseOption
- gettextf, 24
- getweight, 6, 38
- getweight (getweight-methods), 36
- getweight, BdStWeight, TotalVarNeighborhood, BiasType-method (getweight-methods), 36
- getweight, HampelWeight, ContNeighborhood, asymmetricBias-method (getweight-methods), 36
- getweight, HampelWeight, ContNeighborhood, BiasType-method (getweight-methods), 36
- getweight, HampelWeight, ContNeighborhood, onesidedBias-method (getweight-methods), 36
- getweight-methods, 36

- HampelWeight-class, 38
- HampIC-class, 39
- IC, 6, 7, 39, 40, 43, 86
- IC-class, 42
- ICList (kStepEstimate-class), 54
- ICList, kStepEstimate-method (kStepEstimate-class), 54
- InfluenceCurve, 43, 45, 72
- InfluenceCurve-class, 44
- InfoPlot, 46
- infoPlot, 47
- infoPlot, IC-method (infoPlot), 47
- infoPlot-methods (infoPlot), 47
- Infos (InfluenceCurve-class), 44
- Infos, InfluenceCurve-method (InfluenceCurve-class), 44
- Infos<- (InfluenceCurve-class), 44
- Infos<-, InfluenceCurve-method (InfluenceCurve-class), 44
- InfRobModel, 51, 53
- InfRobModel-class, 52
- interpolRisk-class, 53
- jitter, 73
- kStepEstimate-class, 54
- kStepEstimator, 56, 60
- kStepEstimator.start (kStepEstimator.start-methods), 59
- kStepEstimator.start, Estimate-method (kStepEstimator.start-methods), 59
- kStepEstimator.start, function-method (kStepEstimator.start-methods), 59
- kStepEstimator.start, numeric-method (kStepEstimator.start-methods), 59
- kStepEstimator.start-methods, 59
- ksteps (kStepEstimate-class), 54
- ksteps, kStepEstimate-method (kStepEstimate-class), 54
- kStepUseLast (RobAStBaseOptions), 82
- legend, 11, 48, 75
- locMEstimator, 60
- locMEstimator, numeric, InfluenceCurve-method (locMEstimator), 60
- locMEstimator-methods (locMEstimator), 60
- lowerCase (HampIC-class), 39
- lowerCase, HampIC-method (HampIC-class), 39
- lowerCase<- (ContIC-class), 16
- lowerCase<-, ContIC-method (ContIC-class), 16
- lowerCase<-, TotalVarIC-method (TotalVarIC-class), 88
- makeIC (makeIC-methods), 62
- makeIC, function, L2ParamFamily-method (makeIC-methods), 62
- makeIC, IC, L2ParamFamily-method (makeIC-methods), 62
- makeIC, IC, missing-method (makeIC-methods), 62
- makeIC, list, L2ParamFamily-method (makeIC-methods), 62
- makeIC-methods, 62
- Map, InfluenceCurve-method (InfluenceCurve-class), 44
- masked-methods, 64
- maskedMethods (masked-methods), 64
- MASKING (RobAStBaseMASK), 81
- MBRRisk (interpolRisk-class), 53
- MBRRisk-class (interpolRisk-class), 53
- MEstimate-class, 65
- minbiasweight, 6, 38
- minbiasweight (getweight-methods), 36
- minbiasweight, BdStWeight, TotalVarNeighborhood, BiasType-method (getweight-methods), 36
- minbiasweight, HampelWeight, ContNeighborhood, asymmetricBias (getweight-methods), 36
- minbiasweight, HampelWeight, ContNeighborhood, BiasType-method (getweight-methods), 36
- minbiasweight, HampelWeight, ContNeighborhood, onesidedBias-method (getweight-methods), 36
- minbiasweight-methods (getweight-methods), 36
- modifyIC (IC-class), 42
- modifyIC, IC-method (IC-class), 42
- moveICBackFromRefParam (movToRef-methods), 66
- moveICBackFromRefParam, HampIC, L2ParamFamily-method (movToRef-methods), 66
- moveICBackFromRefParam, IC, L2LocationFamily-method (movToRef-methods), 66



- moveICBackFromRefParam, IC, L2LocationScaleFamily-method (movToRef-methods), 66
- moveICBackFromRefParam, IC, L2ParamFamily-method (movToRef-methods), 66
- moveICBackFromRefParam, IC, L2ScaleFamily-method (movToRef-methods), 66
- moveICBackFromRefParam-methods (movToRef-methods), 66
- moveL2Fam2RefParam (movToRef-methods), 66
- moveL2Fam2RefParam, L2LocationFamily-method (movToRef-methods), 66
- moveL2Fam2RefParam, L2LocationScaleFamily-method (movToRef-methods), 66
- moveL2Fam2RefParam, L2ParamFamily-method (movToRef-methods), 66
- moveL2Fam2RefParam, L2ScaleFamily-method (movToRef-methods), 66
- moveL2Fam2RefParam-methods (movToRef-methods), 66
- movToRef-methods, 66
- Mroot (MEstimate-class), 65
- Mroot, MEstimate-method (MEstimate-class), 65
  
- NA, 81
- name, cutoff-method (cutoff-class), 21
- name, InfluenceCurve-method (InfluenceCurve-class), 44
- name, RobAStControl-method (RobAStControl-class), 83
- name, RobModel-method (RobModel-class), 84
- name, RobWeight-method (RobWeight-class), 85
- name<-, InfluenceCurve-method (InfluenceCurve-class), 44
- name<-, RobAStControl-method (RobAStControl-class), 83
- name<-, RobWeight-method (RobWeight-class), 85
- neighbor (RobModel-class), 84
- neighbor, ContIC-method (ContIC-class), 16
- neighbor, RobModel-method (RobModel-class), 84
- neighbor, TotalVarIC-method (TotalVarIC-class), 88
- neighbor<- (RobModel-class), 84
- neighborhood, FixRobModel-method (FixRobModel-class), 28
- neighbor<-, InfRobModel-method (InfRobModel-class), 52
- neighbor<-, RobModel-method (RobModel-class), 84
- Neighborhood-class, 67
- neighborRadius (HampIC-class), 39
- neighborRadius, HampIC-method (HampIC-class), 39
- neighborRadius<- (HampIC-class), 39
- neighborRadius<-, HampIC-method (HampIC-class), 39
- normtype, HampIC-method (HampIC-class), 39
  
- OMSRRisk (interpolRisk-class), 53
- OMSRRisk-class (interpolRisk-class), 53
- oneStepEstimator, 68
- optIC, 70
- optIC, L2ParamFamily, asCov-method (optIC), 70
- optIC-methods (optIC), 70
- OptionalInfluenceCurve-class, 71
- OptionalpICList-class (OptionalInfluenceCurve-class), 71
- options, 83
- outlyingPlotIC, 22, 72
  
- par, 11, 48, 75
- pIC (ALEstimate-class), 4
- pIC, ALEstimate-method (ALEstimate-class), 4
- pICList (kStepEstimate-class), 54
- pICList, kStepEstimate-method (kStepEstimate-class), 54
- pICList-class (OptionalInfluenceCurve-class), 71
- plot, 13
- plot (plot-methods), 74
- plot, IC, missing-method (plot-methods), 74
- plot, IC, numeric-method (plot-methods), 74
- plot-methods, 74
- plot.default, 10, 11, 47, 75
- PlotIC, 78

- qqbounds, [81](#)
- qqplot, [79](#), [79](#), [80](#), [81](#)
- qqplot, ANY, InfRobModel-method (qqplot), [79](#)
- qqplot, ANY, kStepEstimate-method (qqplot), [79](#)
- qqplot, ANY, RobModel-method (qqplot), [79](#)
- qqplot-methods (qqplot), [79](#)
  
- radius (Neighborhood-class), [67](#)
- radius, Neighborhood-method (Neighborhood-class), [67](#)
- radius<- (Neighborhood-class), [67](#)
- radius<- , Neighborhood-method (Neighborhood-class), [67](#)
- Range, InfluenceCurve-method (InfluenceCurve-class), [44](#)
- Risks (InfluenceCurve-class), [44](#)
- Risks, InfluenceCurve-method (InfluenceCurve-class), [44](#)
- Risks<- (InfluenceCurve-class), [44](#)
- Risks<- , InfluenceCurve-method (InfluenceCurve-class), [44](#)
- RMXRRisk (interpolRisk-class), [53](#)
- RMXRRisk-class (interpolRisk-class), [53](#)
- RobAStBase (RobAStBase-package), [3](#)
- RobAStBase-package, [3](#)
- RobAStBaseMASK, [81](#)
- RobAStBaseOptions, [82](#)
- RobAStControl-class, [83](#)
- RobModel-class, [84](#)
- RobWeight-class, [85](#)
  
- samplesize (samplesize-methods), [86](#)
- samplesize, interpolRisk-method (samplesize-methods), [86](#)
- samplesize-methods, [86](#)
- samplesize<- (samplesize-methods), [86](#)
- samplesize<- , interpolRisk-method (samplesize-methods), [86](#)
- show, ALEstimate-method (ALEstimate-class), [4](#)
- show, ContIC-method (ContIC-class), [16](#)
- show, FixRobModel-method (FixRobModel-class), [28](#)
- show, IC-method (IC-class), [42](#)
- show, InfluenceCurve-method (InfluenceCurve-class), [44](#)
- show, InfRobModel-method (InfRobModel-class), [52](#)
- show, kStepEstimate-method (kStepEstimate-class), [54](#)
- show, MEstimate-method (MEstimate-class), [65](#)
- show, Neighborhood-method (Neighborhood-class), [67](#)
- show, OptionalICList-method (OptionalInfluenceCurve-class), [71](#)
- show, pICList-method (OptionalInfluenceCurve-class), [71](#)
- show, TotalVarIC-method (TotalVarIC-class), [88](#)
- stand (HampIC-class), [39](#)
- stand, BdStWeight-method (BdStWeight-class), [6](#)
- stand, HampIC-method (HampIC-class), [39](#)
- stand<- (ContIC-class), [16](#)
- stand<- , BdStWeight-method (BdStWeight-class), [6](#)
- stand<- , ContIC-method (ContIC-class), [16](#)
- stand<- , TotalVarIC-method (TotalVarIC-class), [88](#)
- start, [64](#), [65](#)
- start (masked-methods), [64](#)
- start, ANY-method (masked-methods), [64](#)
- start, kStepEstimate-method (kStepEstimate-class), [54](#)
- start-methods (masked-methods), [64](#)
- StartClass-class (OptionalInfluenceCurve-class), [71](#)
- startval (kStepEstimate-class), [54](#)
- startval, kStepEstimate-method (kStepEstimate-class), [54](#)
- steps (kStepEstimate-class), [54](#)
- steps, kStepEstimate-method (kStepEstimate-class), [54](#)
  
- text, [24](#), [73](#)
- TotalVarIC, [86](#)
- TotalVarIC-class, [88](#)
- TotalVarNeighborhood, [90](#), [91](#)
- TotalVarNeighborhood-class, [91](#)
- trafo, RobModel, missing-method (RobModel-class), [84](#)

trafo<- ,RobModel-method  
    (RobModel-class), 84

type, Neighborhood-method  
    (Neighborhood-class), 67

uksteps (kStepEstimate-class), 54

uksteps, kStepEstimate-method  
    (kStepEstimate-class), 54

UncondNeighborhood-class, 92

ustartval (kStepEstimate-class), 54

ustartval, kStepEstimate-method  
    (kStepEstimate-class), 54

weight (RobWeight-class), 85

weight, HampIC-method (HampIC-class), 39

weight, RobWeight-method  
    (RobWeight-class), 85

weight<- (RobWeight-class), 85

weight<- ,RobWeight-method  
    (RobWeight-class), 85

weight<--methods (RobWeight-class), 85