

# Package ‘ROptEst’

February 14, 2012

**Version** 0.8.1

**Date** 2011-09-02

**Title** Optimally robust estimation

**Description** Optimally robust estimation in general smoothly parameterized models using S4 classes and methods.

**Depends** R(>= 2.7.0), methods, distr(>= 2.0), distrEx(>= 2.0),distrMod(>= 2.0), Rand-Var(>= 0.6.4), RobAStBase

**Suggests** MASS, RobLox

**Author** Matthias Kohl, Peter Ruckdeschel

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

**LazyLoad** yes

**License** LGPL-3

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

**Encoding** latin1

**LastChangedDate** {\$LastChangedDate: 2011-01-05 20:09:20 +0100 (Mi, 05 Jan 2011) \$}

**LastChangedRevision** {\$LastChangedRevision: 444 \$}

**SVNRevision** 439

**Repository** CRAN

**Date/Publication** 2011-10-19 13:17:33

**R topics documented:**

ROptEst-package . . . . .	3
asAnscombe . . . . .	4
asAnscombe-class . . . . .	5
asL1 . . . . .	6
asL1-class . . . . .	7
asL4 . . . . .	8
asL4-class . . . . .	9
cniperCont . . . . .	10
get.asGRisk.fct-methods . . . . .	12
getAsRisk . . . . .	13
getBiasIC . . . . .	17
getFiRisk . . . . .	18
getFixClip . . . . .	20
getFixRobIC . . . . .	21
getIneffDiff . . . . .	22
getInfCent . . . . .	24
getInfClip . . . . .	26
getInfGamma . . . . .	28
getInfLM . . . . .	30
getInfRobIC . . . . .	32
getInfStand . . . . .	36
getInfV . . . . .	38
getL1normL2deriv . . . . .	40
getL2normL2deriv . . . . .	41
getMaxIneff . . . . .	41
getModifyIC . . . . .	43
getReq . . . . .	44
getRiskIC . . . . .	46
leastFavorableRadius . . . . .	47
lowerCaseRadius . . . . .	49
minmaxBias . . . . .	50
optIC . . . . .	53
optRisk . . . . .	55
radiusMinimaxIC . . . . .	57
roptest . . . . .	59
updateNorm-methods . . . . .	64

---

ROptEst-package	<i>Optimally robust estimation</i>
-----------------	------------------------------------

---

**Description**

Optimally robust estimation in general smoothly parameterized models using S4 classes and methods.

**Details**

Package: ROptEst  
Version: 0.8  
Date: 2010-12-03  
Depends: R(>= 2.7.0), methods, distr(>= 2.0), distrEx(>= 2.0), distrMod(>= 2.0), RandVar(>= 0.6.4), RobAStBase  
LazyLoad: yes  
License: LGPL-3  
URL: <http://robast.r-forge.r-project.org/>  
SVNRevision: 439

**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](#), [RandVar-package](#), [RobAStBase-package](#)

**Examples**

```

library(ROptEst)

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

## ML-estimate from package distrMod
MLest <- MLEstimator(x, PoisFamily())
MLest
## confidence interval based on CLT
confint(MLest)

## compute optimally (w.r.t to MSE) robust estimator (unknown contamination)
robust <- roptest(x, PoisFamily(), eps.upper = 0.1, steps = 3)
estimate(robust)
## check influence curve
checkIC(pIC(robust))
## plot influence curve
plot(pIC(robust))
## confidence interval based on LAN - neglecting bias
confint(robust)
## confidence interval based on LAN - including bias
confint(robust, method = symmetricBias())

```

---

asAnscombe

*Generating function for asAnscombe-class*


---

**Description**

Generates an object of class "asAnscombe".

**Usage**

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

**Arguments**

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

**Value**

Object of class asAnscombe

**Author(s)**

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

**References**

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**

[asAnscombe-class](#)

**Examples**

```
asAnscombe()

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

---

asAnscombe-class	<i>Asymptotic Anscombe risk</i>
------------------	---------------------------------

---

**Description**

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

**Objects from the Class**

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

**Slots**

`type` Object of class "character": "optimal bias robust IC (OBRI) for given ARE (asymptotic relative efficiency)".

`eff` Object of class "numeric": given ARE (asymptotic relative efficiency) to be attained in the ideal model.

`biastype` Object of class "BiasType": symmetric, one-sided or asymmetric

**Extends**

Class "asRiskwithBias", directly.

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

**Methods**

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

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

**Author(s)**

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

**References**

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**

[asRisk-class](#), [asAnscombe](#)

**Examples**

```
new("asAnscombe")
```

---

asL1

*Generating function for asMSE-class*

---

**Description**

Generates an object of class "asMSE".

**Usage**

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

**Arguments**

biastype            a bias type of class BiasType

normtype           a norm type of class NormType

**Value**

Object of class "asMSE"

**Author(s)**

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

**References**

Ruckdeschel, P. and Rieder, H. (2004) Optimal Influence Curves for General Loss Functions. *Statistics & Decisions* 22, 201-223.

**See Also**

[asL1-class](#), [asMSE](#), [asL4](#)

**Examples**

```
asL1()

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

---

asL1-class

*Asymptotic mean absolute error*

---

**Description**

Class of asymptotic mean absolute error.

**Objects from the Class**

Objects can be created by calls of the form `new("asL1", ...)`. More frequently they are created via the generating function `asL1`.

**Slots**

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

**Extends**

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

**Methods**

No methods defined with class "asL1" in the signature.

**Author(s)**

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

**References**

Ruckdeschel, P. and Rieder, H. (2004) Optimal Influence Curves for General Loss Functions. *Statistics & Decisions* 22, 201-223.

**See Also**

[asGRisk-class](#), [asMSE](#), [asMSE-class](#), [asL4-class](#), [asL1](#)

**Examples**

```
new("asMSE")
```

---

asL4

*Generating function for asL4-class*

---

**Description**

Generates an object of class "asL4".

**Usage**

```
asL4(biastype = symmetricBias(), normtype = NormType())
```

**Arguments**

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

**Value**

Object of class "asL4"

**Author(s)**

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

**References**

Ruckdeschel, P. and Rieder, H. (2004) Optimal Influence Curves for General Loss Functions. *Statistics & Decisions* 22, 201-223.

**See Also**

[asL4-class](#), [asMSE](#), [asL1](#)

**Examples**

```
asL4()

## The function is currently defined as
function(biastype = symmetricBias(), normtype = NormType()){
  new("asL4", biastype = biastype, normtype = normtype) }
```

---

asL4-class

*Asymptotic mean power 4 error*


---

**Description**

Class of asymptotic mean power 4 error.

**Objects from the Class**

Objects can be created by calls of the form `new("asL4", ...)`. More frequently they are created via the generating function `asL4`.

**Slots**

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

**Extends**

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

**Methods**

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

**Author(s)**

Peter Ruckdeschel <[peter.ruckdeschel@itwm.fraunhofer.de](mailto:peter.ruckdeschel@itwm.fraunhofer.de)>

**References**

Ruckdeschel, P. and Rieder, H. (2004) Optimal Influence Curves for General Loss Functions. *Statistics & Decisions* 22, 201-223.

**See Also**

[asGRisk-class](#), [asMSE](#), [asMSE-class](#), [asL1-class](#), [asL4](#)

**Examples**

```
new("asMSE")
```

---

cnipecCont

*Generic Functions for Computation and Plot of Cnipec Contamination and Cnipec Points.*

---

**Description**

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

**Usage**

```
cnipecCont(IC1, IC2, L2Fam, neighbor, risk, ...)
## S4 method for signature 'IC,IC,L2ParamFamily,ContNeighborhood,asMSE'
cnipecCont(IC1,
           IC2, L2Fam, neighbor, risk, lower, upper, n = 101)

cnipecPoint(L2Fam, neighbor, risk, ...)
## S4 method for signature 'L2ParamFamily,ContNeighborhood,asMSE'
cnipecPoint(L2Fam,
            neighbor, risk, lower, upper)

cnipecPointPlot(L2Fam, neighbor, risk, ...)
## S4 method for signature 'L2ParamFamily,ContNeighborhood,asMSE'
cnipecPointPlot(L2Fam,
                neighbor, risk, lower, upper, n = 101, ...)
```

**Arguments**

IC1	object of class IC
IC2	object of class IC
L2Fam	object of class L2ParamFamily
neighbor	object of class Neighborhood
risk	object of class RiskType
...	additional parameters (in particular to be passed on to plot).
lower, upper	the lower and upper end points of the contamination interval.
n	number of points between lower and upper

## Details

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

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

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

For more details about cniPer contamination and cniPer points we refer to Section~3.5 of Kohl et al. (2008) as well as Ruckdeschel (2004) and the Introduction of Kohl (2005).

## Value

`invisible()` resp. cniPer point is returned.

## Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

## References

Kohl, M. and Ruckdeschel, H. and Rieder, H. (2008). Infinitesimally Robust Estimation in General Smoothly Parametrized Models. Unpublished Manuscript.

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

Ruckdeschel, P. (2004). Higher Order Asymptotics for the MSE of M-Estimators on Shrinking Neighborhoods. Unpublished Manuscript.

## Examples

```
## cniPer contamination
P <- PoisFamily(lambda = 4)
RobP1 <- InfRobModel(center = P, neighbor = ContNeighborhood(radius = 0.1))
IC1 <- optIC(model=RobP1, risk=asMSE())
RobP2 <- InfRobModel(center = P, neighbor = ContNeighborhood(radius = 1))
IC2 <- optIC(model=RobP2, risk=asMSE())
cniPerCont(IC1 = IC1, IC2 = IC2, L2Fam = P,
           neighbor = ContNeighborhood(radius = 0.5),
           risk = asMSE(),
           lower = 0, upper = 8, n = 101)

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

## cniPer point
cniPerPoint(P, neighbor = ContNeighborhood(radius = 0.5),
```

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

---

get.asGRisk.fct-methods

*Methods for Function get.asGRisk.fct in Package 'ROptEst'*

---

## Description

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

## Usage

```
get.asGRisk.fct(Risk)
## S4 method for signature 'asMSE'
get.asGRisk.fct(Risk)
## S4 method for signature 'asL1'
get.asGRisk.fct(Risk)
## S4 method for signature 'asL4'
get.asGRisk.fct(Risk)
```

## Arguments

Risk                    a risk of class "asGRisk"

## Details

get.asGRisk.fct is used internally in functions [getAsRisk](#) and [getReq](#).

## Value

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

## Methods

**get.asGRisk.fct** signature(Risk = "asMSE"): method for asymptotic mean squared error.

**get.asGRisk.fct** signature(Risk = "asL1"): method for asymptotic mean absolute error.

**get.asGRisk.fct** signature(Risk = "asL4"): method for asymptotic mean power 4 error.

## Author(s)

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

---

getAsRisk

*Generic Function for Computation of Asymptotic Risks*


---

### Description

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

### Usage

```

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

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

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

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

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

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

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

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

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

## S4 method for signature 'asBias,RealRandVariable,ContNeighborhood,ANY'
getAsRisk(
  risk,L2deriv, neighbor, biastype, normtype = NULL, clip = NULL, cent = NULL, stand = NULL,

```

```

    Distr, DistrSymm, L2derivSymm,
    L2derivDistrSymm, Finfo, trafo, z.start, A.start, maxiter, tol,
    warn, verbose = NULL, ...)
## S4 method for signature 'asBias,RealRandVariable,TotalVarNeighborhood,ANY'
getAsRisk(
  risk, L2deriv, neighbor, biastype, normtype = NULL,
  clip = NULL, cent = NULL, stand = NULL, Distr, DistrSymm, L2derivSymm,
  L2derivDistrSymm, Finfo, trafo, z.start, A.start, maxiter, tol,
  warn, verbose = NULL, ...)

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

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

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

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

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

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

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

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

```

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

### Arguments

risk	object of class "asRisk".
L2deriv	L2-derivative of some L2-differentiable family of probability distributions.
neighbor	object of class "Neighborhood".
biastype	object of class "ANY".
...	additional parameters; often used to enable flexible calls.
clip	optimal clipping bound.
cent	optimal centering constant.
stand	standardizing matrix.
Finfo	matrix: the Fisher Information of the parameter.
trafo	matrix: transformation of the parameter.
Distr	object of class "Distribution".
DistrSymm	object of class "DistributionSymmetry".
L2derivSymm	object of class "FunSymmList".
L2derivDistrSymm	object of class "DistrSymmList".
z.start	initial value for the centering constant.
A.start	initial value for the standardizing matrix.
maxiter	the maximum number of iterations
tol	the desired accuracy (convergence tolerance).
warn	logical: print warnings.
normtype	object of class "NormType".
V.comp	matrix: indication which components of the standardizing matrix have to be computed.
w	object of class RobWeight; current weight
FI	trace of the respective Fisher Information
verbose	logical: if TRUE some diagnostics are printed out.

### Details

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

### Value

The asymptotic risk is computed.

## Methods

- risk = "asMSE", L2deriv = "UnivariateDistribution", neighbor = "Neighborhood", biastype = "ANY":**  
 computes asymptotic mean square error in methods for function getInfRobIC.
- risk = "asL1", L2deriv = "UnivariateDistribution", neighbor = "Neighborhood", biastype = "ANY":**  
 computes asymptotic mean absolute error in methods for function getInfRobIC.
- risk = "asL4", L2deriv = "UnivariateDistribution", neighbor = "Neighborhood", biastype = "ANY":**  
 computes asymptotic mean power 4 error in methods for function getInfRobIC.
- risk = "asMSE", L2deriv = "EuclRandVariable", neighbor = "Neighborhood", biastype = "ANY":**  
 computes asymptotic mean square error in methods for function getInfRobIC.
- risk = "asBias", L2deriv = "UnivariateDistribution", neighbor = "ContNeighborhood", biastype = "ANY":**  
 computes standardized asymptotic bias in methods for function getInfRobIC.
- risk = "asBias", L2deriv = "UnivariateDistribution", neighbor = "ContNeighborhood", biastype = "onesidedBias":**  
 computes standardized asymptotic bias in methods for function getInfRobIC.
- risk = "asBias", L2deriv = "UnivariateDistribution", neighbor = "ContNeighborhood", biastype = "asymmetricBias":**  
 computes standardized asymptotic bias in methods for function getInfRobIC.
- risk = "asBias", L2deriv = "UnivariateDistribution", neighbor = "TotalVarNeighborhood", biastype = "ANY":**  
 computes standardized asymptotic bias in methods for function getInfRobIC.
- risk = "asBias", L2deriv = "RealRandVariable", neighbor = "ContNeighborhood", biastype = "ANY":**  
 computes standardized asymptotic bias in methods for function getInfRobIC.
- risk = "asCov", L2deriv = "UnivariateDistribution", neighbor = "ContNeighborhood", biastype = "ANY":**  
 computes asymptotic covariance in methods for function getInfRobIC.
- risk = "asCov", L2deriv = "UnivariateDistribution", neighbor = "TotalVarNeighborhood", biastype = "ANY":**  
 computes asymptotic covariance in methods for function getInfRobIC.
- risk = "asCov", L2deriv = "RealRandVariable", neighbor = "ContNeighborhood", biastype = "ANY":**  
 computes asymptotic covariance in methods for function getInfRobIC.
- risk = "trAsCov", L2deriv = "UnivariateDistribution", neighbor = "UncondNeighborhood", biastype = "ANY":**  
 computes trace of asymptotic covariance in methods for function getInfRobIC.
- risk = "trAsCov", L2deriv = "RealRandVariable", neighbor = "ContNeighborhood", biastype = "ANY":**  
 computes trace of asymptotic covariance in methods for function getInfRobIC.
- risk = "asAnscombe", L2deriv = "UnivariateDistribution", neighbor = "UncondNeighborhood", biastype = "ANY":**  
 computes the ARE in the ideal model in methods for function getInfRobIC.
- risk = "asAnscombe", L2deriv = "RealRandVariable", neighbor = "ContNeighborhood", biastype = "ANY":**  
 computes the ARE in the ideal model in methods for function getInfRobIC.
- risk = "asUnOvShoot", L2deriv = "UnivariateDistribution", neighbor = "UncondNeighborhood", biastype = "ANY":**  
 computes asymptotic under-/overshoot risk in methods for function getInfRobIC.
- risk = "asSemivar", L2deriv = "UnivariateDistribution", neighbor = "Neighborhood", biastype = "onesidedBias":**  
 computes asymptotic semivariance in methods for function getInfRobIC.

## Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

## References

- Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.
- Ruckdeschel, P. and Rieder, H. (2004) Optimal Influence Curves for General Loss Functions. *Statistics & Decisions* 22, 201-223.
- Ruckdeschel, P. (2005) Optimally One-Sided Bounded Influence Curves. *Mathematical Methods in Statistics* 14(1), 105-131.
- Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

## See Also

[asRisk-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 'HampIC,UncondNeighborhood'  
getBiasIC(IC, neighbor, L2Fam, ...)
```

## Arguments

IC	object of class "InfluenceCurve"
neighbor	object of class "Neighborhood".
L2Fam	object of class "L2ParamFamily".
...	additional parameters

## Details

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

## Value

The bias of the IC is computed.

**Methods**

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

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

**Note**

This generic function is still under construction.

**Author(s)**

Peter Ruckdeschel <Peter.Ruckdeschel@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](#)

---

getFiRisk

*Generic Function for Computation of Finite-Sample Risks*

---

**Description**

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

**Usage**

```
getFiRisk(risk, Distr, neighbor, ...)

## S4 method for signature 'fiUnOvShoot, Norm, ContNeighborhood'
getFiRisk(risk, Distr,
          neighbor, clip, stand, sampleSize, Algo, cont)

## S4 method for signature 'fiUnOvShoot, Norm, TotalVarNeighborhood'
```

```
getFiRisk(risk, Distr,
          neighbor, clip, stand, sampleSize, Algo, cont)
```

### Arguments

risk	object of class "RiskType".
Distr	object of class "Distribution".
neighbor	object of class "Neighborhood".
...	additional parameters.
clip	positive real: clipping bound
stand	standardizing constant/matrix.
sampleSize	integer: sample size.
Algo	"A" or "B".
cont	"left" or "right".

### Details

The computation of the finite-sample under-/overshoot risk is based on FFT. For more details we refer to Section 11.3 of Kohl (2005).

### Value

The finite-sample risk is computed.

### Methods

**risk = "fiUnOvShoot", Distr = "Norm", neighbor = "ContNeighborhood"** computes finite-sample under-/overshoot risk in methods for function getFixRobIC.

**risk = "fiUnOvShoot", Distr = "Norm", neighbor = "TotalVarNeighborhood"** computes finite-sample under-/overshoot risk in methods for function getFixRobIC.

### Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

### References

Huber, P.J. (1968) Robust Confidence Limits. *Z. Wahrscheinlichkeitstheor. Verw. Geb.* **10**:269–278.

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

[fiRisk-class](#)

---

getFixClip

*Generic Function for the Computation of the Optimal Clipping Bound*


---

### Description

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

### Usage

```
getFixClip(clip, Distr, risk, neighbor, ...)
```

```
## S4 method for signature 'numeric, Norm, fiUnOvShoot, ContNeighborhood'
getFixClip(clip, Distr, risk, neighbor)
```

```
## S4 method for signature 'numeric, Norm, fiUnOvShoot, TotalVarNeighborhood'
getFixClip(clip, Distr, risk, neighbor)
```

### Arguments

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

### Value

The optimal clipping bound is computed.

### Methods

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

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

### Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

### References

Huber, P.J. (1968) Robust Confidence Limits. *Z. Wahrscheinlichkeitstheor. Verw. Geb.* **10**:269–278.

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

**See Also**

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

---

getFixRobIC

*Generic Function for the Computation of Optimally Robust ICs*


---

**Description**

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

**Usage**

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

```
## S4 method for signature 'Norm,fiUnOvShoot,UncondNeighborhood'
```

```
getFixRobIC(Distr, risk, neighbor,
             sampleSize, upper, lower, maxiter, tol, warn, Algo, cont)
```

**Arguments**

Distr	object of class "Distribution".
risk	object of class "RiskType".
neighbor	object of class "Neighborhood".
...	additional parameters.
sampleSize	integer: sample size.
upper	upper bound for the optimal clipping bound.
lower	lower bound for the optimal clipping bound.
maxiter	the maximum number of iterations.
tol	the desired accuracy (convergence tolerance).
warn	logical: print warnings.
Algo	"A" or "B".
cont	"left" or "right".

**Details**

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

**Value**

The optimally robust IC is computed.

**Methods**

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

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

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

**See Also**

[FixRobModel-class](#)

---

getIneffDiff

*Generic Function for the Computation of Inefficiency Differences*

---

**Description**

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

**Usage**

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

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

**Arguments**

radius	neighborhood radius.
L2Fam	L2-differentiable family of probability measures.
neighbor	object of class "Neighborhood".
risk	object of class "RiskType".

loRad	the lower end point of the interval to be searched.
upRad	the upper end point of the interval to be searched.
loRisk	the risk at the lower end point of the interval.
upRisk	the risk at the upper end point of the interval.
z.start	initial value for the centering constant.
A.start	initial value for the standardizing matrix.
upper.b	upper bound for the optimal clipping bound.
lower.b	lower bound for the optimal clipping bound.
OptOrIter	character; which method to be used for determining Lagrange multipliers A and a: if (partially) matched to "optimize", getLagrangeMultByOptim is used; otherwise: by default, or if matched to "iterate" or to "doubleiterate", getLagrangeMultByIter is used. More specifically, when using getLagrangeMultByIter, and if argument risk is of class "asGRisk", by default and if matched to "iterate" we use only one (inner) iteration, if matched to "doubleiterate" we use up to MaxIter (inner) iterations.
MaxIter	the maximum number of iterations
eps	the desired accuracy (convergence tolerance).
warn	logical: print warnings.
loNorm	object of class "NormType"; used in selfstandardization to evaluate the bias of the current IC in the norm of the lower bound
upNorm	object of class "NormType"; used in selfstandardization to evaluate the bias of the current IC in the norm of the upper bound
verbose	logical: if TRUE, some messages are printed
...	further arguments to be passed on to getInfRobIC

### Value

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

### Methods

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

### Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

## References

Rieder, H., Kohl, M. and Ruckdeschel, P. (2008) The Costs of not Knowing the Radius. *Statistical Methods and Applications*, 17(1) 13-40.

Rieder, H., Kohl, M. and Ruckdeschel, P. (2001) The Costs of not Knowing the Radius. Submitted. Appeared as discussion paper Nr. 81. SFB 373 (Quantification and Simulation of Economic Processes), Humboldt University, Berlin; also available under [www.uni-bayreuth.de/departments/math/org/mathe7/RIEDER/pubs/RR.pdf](http://www.uni-bayreuth.de/departments/math/org/mathe7/RIEDER/pubs/RR.pdf)

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

## See Also

[radiusMinimaxIC](#), [leastFavorableRadius](#)

---

getInfCent	<i>Generic Function for the Computation of the Optimal Centering Constant/Lower Clipping Bound</i>
------------	--

---

## Description

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

## Usage

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

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

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

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

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

## S4 method for signature 'UnivariateDistribution,ContNeighborhood,onesidedBias'
getInfCent(L2deriv,
```

```

neighbor, biastype, clip, cent, tol.z, symm, trafo)

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

```

### Arguments

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

### Value

The optimal centering constant is computed.

### Methods

**L2deriv = "UnivariateDistribution", neighbor = "ContNeighborhood", biastype = "BiasType"**  
 computation of optimal centering constant for symmetric bias.

**L2deriv = "UnivariateDistribution", neighbor = "TotalVarNeighborhood", biastype = "BiasType"**  
 computation of optimal lower clipping bound for symmetric bias.

**L2deriv = "RealRandVariable", neighbor = "TotalVarNeighborhood", biastype = "BiasType"**  
 computation of optimal centering constant for symmetric bias.

**L2deriv = "RealRandVariable", neighbor = "ContNeighborhood", biastype = "BiasType"** computation  
 of optimal centering constant for symmetric bias.

**L2deriv = "UnivariateDistribution", neighbor = "ContNeighborhood", biastype = "onesidedBias"**  
 computation of optimal centering constant for onesided bias.

**L2deriv = "UnivariateDistribution", neighbor = "ContNeighborhood", biastype = "asymmetricBias"**  
 computation of optimal centering constant for asymmetric bias.

### Author(s)

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

**References**

- Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.
- Ruckdeschel, P. (2005) Optimally One-Sided Bounded Influence Curves. *Mathematical Methods in Statistics* 14(1), 105-131.
- Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

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

---

getInfClip

*Generic Function for the Computation of the Optimal Clipping Bound*

---

**Description**

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

**Usage**

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

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

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

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

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

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

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

```

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

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

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

```

### Arguments

clip	positive real: clipping bound
L2deriv	L2-derivative of some L2-differentiable family of probability measures.
risk	object of class "RiskType".
neighbor	object of class "Neighborhood".
...	additional parameters.
biastype	object of class "BiasType"
cent	optimal centering constant.
stand	standardizing matrix.
Distr	object of class "Distribution".
symm	logical: indicating symmetry of L2deriv.
trafo	matrix: transformation of the parameter.

### Value

The optimal clipping bound is computed.

### Methods

```

clip = "numeric", L2deriv = "UnivariateDistribution", risk = "asMSE", neighbor = "ContNeighborhood"
  optimal clipping bound for asymptotic mean square error.
clip = "numeric", L2deriv = "UnivariateDistribution", risk = "asMSE", neighbor = "TotalVarNeighborhood"
  optimal clipping bound for asymptotic mean square error.
clip = "numeric", L2deriv = "EuclRandVariable", risk = "asMSE", neighbor = "UncondNeighborhood"
  optimal clipping bound for asymptotic mean square error.
clip = "numeric", L2deriv = "UnivariateDistribution", risk = "asL1", neighbor = "ContNeighborhood"
  optimal clipping bound for asymptotic mean absolute error.
clip = "numeric", L2deriv = "UnivariateDistribution", risk = "asL1", neighbor = "TotalVarNeighborhood"
  optimal clipping bound for asymptotic mean absolute error.
clip = "numeric", L2deriv = "UnivariateDistribution", risk = "asL4", neighbor = "ContNeighborhood"
  optimal clipping bound for asymptotic mean power 4 error.

```

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

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

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

### Author(s)

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

### References

Rieder, H. (1980) Estimates derived from robust tests. *Ann. Stats.* **8**: 106–115.

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

Ruckdeschel, P. and Rieder, H. (2004) Optimal Influence Curves for General Loss Functions. *Statistics & Decisions* 22, 201-223.

Ruckdeschel, P. (2005) Optimally One-Sided Bounded Influence Curves. *Mathematical Methods in Statistics* 14(1), 105-131.

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

### See Also

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

---

getInfGamma

*Generic Function for the Computation of the Optimal Clipping Bound*

---

### Description

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

### Usage

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

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

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

```

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

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

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

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

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

```

### Arguments

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

### Details

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

### Methods

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

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

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

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

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

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

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

### Author(s)

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

### References

Rieder, H. (1980) Estimates derived from robust tests. *Ann. Stats.* **8**: 106–115.

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

Ruckdeschel, P. and Rieder, H. (2004) Optimal Influence Curves for General Loss Functions. *Statistics & Decisions* 22, 201-223.

Ruckdeschel, P. (2005) Optimally One-Sided Bounded Influence Curves. *Mathematical Methods in Statistics* 14(1), 105-131.

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

### See Also

[asGRisk-class](#), [asMSE-class](#), [asUnOvShoot-class](#), [ContIC-class](#), [TotalVarIC-class](#)

---

getInfLM

*Functions to determine Lagrange multipliers*

---

### Description

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

### Usage

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

```
neighbor, biastype, normtype, Distr,
a.start, z.start, A.start, w.start, std, z.comp,
A.comp, maxiter, tol, verbose = NULL, ...)
```

### Arguments

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

### Value

a list with items	
A	Lagrange multiplier A (standardizing matrix)
a	Lagrange multiplier a (centering in p-space)
z	Lagrange multiplier z (centering in k-space)
w	weight function involving Lagrange multipliers
biastype	(possibly modified) bias type biastype from argument
normtype	(possibly modified) norm type normtype from argument

normtype.old	(possibly modified) norm type normtype before last (internal) update
risk	(possibly [norm-]modified) risk risk from argument
std	(possibly modified) argument std
iter	number of iterations needed
prec	precision achieved
b	used clipping height b
call	call with which either getLagrangeMultByIter or getLagrangeMultByOptim was called

**Author(s)**

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

**References**

- Rieder, H. (1980) Estimates derived from robust tests. *Ann. Stats.* **8**: 106-115.
- Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.
- Ruckdeschel, P. and Rieder, H. (2004) Optimal Influence Curves for General Loss Functions. *Statistics & Decisions* **22**: 201-223.
- Ruckdeschel, P. (2005) Optimally One-Sided Bounded Influence Curves. *Mathematical Methods in Statistics* *14*(1), 105-131.
- Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

**See Also**

[InfRobModel-class](#)

---

getInfRobIC

*Generic Function for the Computation of Optimally Robust ICs*

---

**Description**

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

**Usage**

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

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

```

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

## S4 method for signature 'RealRandVariable,asCov,UncondNeighborhood'
getInfRobIC(L2deriv, risk,
            neighbor, Distr, Finfo, trafo, QuadForm = diag(nrow(trafo)),
            verbose = NULL)

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

## S4 method for signature 'RealRandVariable,asBias,UncondNeighborhood'
getInfRobIC(L2deriv, risk,
            neighbor, Distr, DistrSymm, L2derivSymm,
            L2derivDistrSymm, z.start, A.start, Finfo, trafo,
            maxiter, tol, warn, verbose = NULL, ...)

## S4 method for signature 'UnivariateDistribution,asHampel,UncondNeighborhood'
getInfRobIC(L2deriv,
            risk, neighbor, symm, Finfo, trafo, upper = NULL,
            lower=NULL, maxiter, tol, warn, noLow = FALSE,
            verbose = NULL, checkBounds = TRUE)

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

## S4 method for signature 'UnivariateDistribution,asAnscombe,UncondNeighborhood'
getInfRobIC(
            L2deriv, risk, neighbor, symm, Finfo, trafo, upper = NULL,
            lower=NULL, maxiter, tol, warn, noLow = FALSE,
            verbose = NULL, checkBounds = TRUE)

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

```

```

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

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

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

```

### Arguments

L2deriv	L2-derivative of some L2-differentiable family of probability measures.
risk	object of class "RiskType".
neighbor	object of class "Neighborhood".
...	additional parameters (mainly for optim).
Distr	object of class "Distribution".
symm	logical: indicating symmetry of L2deriv.
DistrSymm	object of class "DistributionSymmetry".
L2derivSymm	object of class "FunSymmList".
L2derivDistrSymm	object of class "DistrSymmList".
Finfo	Fisher information matrix.
z.start	initial value for the centering constant.
A.start	initial value for the standardizing matrix.
trafo	matrix: transformation of the parameter.
upper	upper bound for the optimal clipping bound.
lower	lower bound for the optimal clipping bound.
OptOrIter	character; which method to be used for determining Lagrange multipliers A and a: if (partially) matched to "optimize", getLagrangeMultByOptim is used; otherwise: by default, or if matched to "iterate" or to "doubleiterate", getLagrangeMultByIter is used. More specifically, when using getLagrangeMultByIter, and if argument risk is of class "asGRisk", by default and if matched to "iterate" we use only one (inner) iteration, if matched to "doubleiterate" we use up to Maxiter (inner) iterations.
maxiter	the maximum number of iterations.

tol	the desired accuracy (convergence tolerance).
warn	logical: print warnings.
noLow	logical: is lower case to be computed?
onesetLM	logical: use one set of Lagrange multipliers?
QuadForm	matrix of (or which may coerced to) class PosSemDefSymmMatrix for use of different (standardizing) norm
verbose	logical: if TRUE, some messages are printed
checkBounds	logical: if TRUE, minimal and maximal clipping bound are computed to check if a valid bound was specified.
withPICcheck	logical: at the end of the algorithm, shall we check how accurately this is a pIC; this will only be done if withPICcheck && verbose.

### Value

The optimally robust IC is computed.

### Methods

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

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

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

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

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

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

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

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

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

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

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

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

#### Author(s)

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

#### References

- Rieder, H. (1980) Estimates derived from robust tests. *Ann. Stats.* **8**: 106-115.  
 Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.  
 Ruckdeschel, P. and Rieder, H. (2004) Optimal Influence Curves for General Loss Functions. *Statistics & Decisions* **22**: 201-223.  
 Ruckdeschel, P. (2005) Optimally One-Sided Bounded Influence Curves. *Mathematical Methods in Statistics* *14*(1), 105-131.  
 Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

#### See Also

[InfRobModel-class](#)

---

getInfStand

*Generic Function for the Computation of the Standardizing Matrix*

---

#### Description

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

**Usage**

```

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

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

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

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

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

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

```

**Arguments**

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

**Value**

The standardizing matrix is computed.

**Methods**

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

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

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

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

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

**Author(s)**

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

**References**

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

Ruckdeschel, P. (2005) Optimally One-Sided Bounded Influence Curves. *Mathematical Methods in Statistics* 14(1), 105-131.

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

**See Also**

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

---

getInfV	<i>Generic Function for the Computation of the asymptotic Variance of a Hampel type IC</i>
---------	--

---

**Description**

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

**Usage**

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

```

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

```

### Arguments

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

### Value

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

### Author(s)

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

### References

- Rieder, H. (1980) Estimates derived from robust tests. *Ann. Stats.* **8**: 106–115.
- Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.
- Ruckdeschel, P. (2005) Optimally One-Sided Bounded Influence Curves. *Mathematical Methods in Statistics* *14*(1), 105-131.
- Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

### See Also

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

---

getL1normL2deriv      *Calculation of L1 norm of L2derivative*

---

### Description

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

### Usage

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

### Arguments

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

### Value

L1 norm of the L2derivative

### Author(s)

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

### Examples

```
##
```

---

getL2normL2deriv	<i>Calculation of L2 norm of L2derivative</i>
------------------	---

---

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

L2 norm of the L2derivative

**Author(s)**

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

**Examples**

```
##
```

---

getMaxIneff	<i>getMaxIneff – computation of the maximal inefficiency of an IC</i>
-------------	---

---

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

The maximal inefficiency, i.e.; a number in  $[1, \text{Inf})$ .

**Author(s)**

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

Rieder, H., Kohl, M. and Ruckdeschel, P. (2008) The Costs of not Knowing the Radius. *Statistical Methods and Applications* 17(1) 13-40.

Rieder, H., Kohl, M. and Ruckdeschel, P. (2001) The Costs of not Knowing the Radius. Submitted. Appeared as discussion paper Nr. 81. SFB 373 (Quantification and Simulation of Economic Processes), Humboldt University, Berlin; also available under [www.uni-bayreuth.de/departments/math/org/mathe7/RIEDER/pubs/RR.pdf](http://www.uni-bayreuth.de/departments/math/org/mathe7/RIEDER/pubs/RR.pdf)

**Examples**

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

getMaxIneff(N0.ICA, neighbor)
```

```

getMaxIneff(N0.ICM,neighbor)
getMaxIneff(N0.ICR,neighbor)

N0ls <- NormLocationScaleFamily()
ICsc <- makeIC(list(sin,cos),N0ls)
getMaxIneff(ICsc,neighbor)

```

---

getModifyIC

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


---

### Description

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

### Usage

```

getModifyIC(L2FamIC, neighbor, risk,...)
## S4 method for signature 'L2ParamFamily,Neighborhood,asRisk'
getModifyIC(L2FamIC,
            neighbor, risk, ...)
## S4 method for signature 'L2LocationFamily,UncondNeighborhood,asGRisk'
getModifyIC(L2FamIC,
            neighbor, risk, ...)
## S4 method for signature 'L2LocationFamily,UncondNeighborhood,fiUnOvShoot'
getModifyIC(L2FamIC,
            neighbor, risk, ...)
## S4 method for signature 'L2ScaleFamily,UncondNeighborhood,asGRisk'
getModifyIC(L2FamIC,
            neighbor, risk, ...)
## S4 method for signature 'L2LocationScaleFamily,UncondNeighborhood,asGRisk'
getModifyIC(L2FamIC,
            neighbor, risk, ...)

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

```

### Arguments

L2FamIC	object of class L2ParamFamily.
neighbor	object of class "Neighborhood".
risk	object of class "RiskType"

... further arguments to be passed over to optIC.  
 sdneu positive numeric of length one; the new scale.  
 sdalt positive numeric of length one; the new scale.  
 IC a Hampel-IC to be updated.

### Details

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

### Value

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

### Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

### References

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

### See Also

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

---

getReq	<i>getReq – computation of the radius interval where IC1 is better than IC2</i>
--------	---

---

### Description

(tries to) compute a radius interval where IC1 is better than IC2

### Usage

```
getReq(Risk, neighbor, IC1, IC2, n=1, upper=15)
```

**Arguments**

Risk	an object of class "asGRisk" – the risk at which IC1 is better than IC2.
neighbor	object of class "Neighborhood"; the neighborhood at which to compute the bias.
IC1	some IC of class "IC"
IC2	some IC of class "IC"
n	the sample size; by default set to 1; then the radius interval refers to starting radii in the shrinking neighborhood setting of Rieder[94]. Otherwise the radius interval is scaled down accordingly.
upper	the upper bound of the radius interval in which to search

**Value**

The radius interval (given by its endpoints) where IC1 is better than IC2 according to the risk. In case IC2 is better than IC1 as to both variance and bias, the return value is NA.

**Author(s)**

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

**Examples**

```

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

getReq(asMSE(),neighbor,N0.ICA,N0.ICM,n=1)
getReq(asMSE(),neighbor,N0.ICA,N0.ICM,n=30)
getReq(asL1(),neighbor,N0.ICA,N0.ICM,n=30)
getReq(asL4(),neighbor,N0.ICA,N0.ICM,n=30)
getReq(asMSE(),neighbor,N0.ICA,N0.ICR,n=30)
getReq(asL1(),neighbor,N0.ICA,N0.ICR,n=30)
getReq(asL4(),neighbor,N0.ICA,N0.ICR,n=30)
getReq(asMSE(),neighbor,N0.ICM,N0.ICR,n=30)

### when to use MAD and when Qn

```

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

---

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 'HampIC,asCov,missing,missing'
getRiskIC(IC, risk)

## S4 method for signature 'HampIC,asCov,missing,L2ParamFamily'
getRiskIC(IC, risk, L2Fam)
## S4 method for signature 'TotalVarIC,asCov,missing,L2ParamFamily'
getRiskIC(IC, risk, L2Fam)
```

### Arguments

IC	object of class "InfluenceCurve"
risk	object of class "RiskType".
neighbor	object of class "Neighborhood"; missing in the methods described here.
...	additional parameters
L2Fam	object of class "L2ParamFamily".

### Details

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

### Value

The risk of an IC is computed.

## Methods

**IC = "HampIC", risk = "asCov", neighbor = "missing", L2Fam = "missing"** asymptotic covariance of IC read off from corresp. Risks slot.

**IC = "HampIC", risk = "asCov", neighbor = "missing", L2Fam = "L2ParamFamily"** asymptotic covariance of IC under L2Fam read off from corresp. Risks slot.

**IC = "TotalVarIC", risk = "asCov", neighbor = "missing", L2Fam = "L2ParamFamily"** asymptotic covariance of IC read off from corresp. Risks slot, resp. if this is NULL calculates it via [getInfV](#).

## Note

This generic function is still under construction.

## Author(s)

Peter Ruckdeschel <Peter.Ruckdeschel@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](#)

---

leastFavorableRadius    *Generic Function for the Computation of Least Favorable Radii*

---

## Description

Generic function for the computation of least favorable radii.

**Usage**

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

## S4 method for signature 'L2ParamFamily,UncondNeighborhood,asGRisk'
leastFavorableRadius(
  L2Fam, neighbor, risk, rho, upRad = 1,
  z.start = NULL, A.start = NULL, upper = 100,
  OptOrIter = "iterate", maxiter = 100,
  tol = .Machine$double.eps^0.4, warn = FALSE, verbose = NULL)
```

**Arguments**

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

**Value**

The least favorable radius and the corresponding inefficiency are computed.

**Methods**

**L2Fam = "L2ParamFamily", neighbor = "UncondNeighborhood", risk = "asGRisk"** computation of the least favorable radius.

**Author(s)**

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

## References

Rieder, H., Kohl, M. and Ruckdeschel, P. (2008) The Costs of not Knowing the Radius. *Statistical Methods and Applications* 17(1) 13-40.

Rieder, H., Kohl, M. and Ruckdeschel, P. (2001) The Costs of not Knowing the Radius. Submitted. Appeared as discussion paper Nr. 81. SFB 373 (Quantification and Simulation of Economic Processes), Humboldt University, Berlin; also available under [www.uni-bayreuth.de/departments/math/org/mathe7/RIEDER/pubs/RR.pdf](http://www.uni-bayreuth.de/departments/math/org/mathe7/RIEDER/pubs/RR.pdf)

Ruckdeschel, P. (2005) Optimally One-Sided Bounded Influence Curves. *Mathematical Methods in Statistics* 14(1), 105-131.

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

## See Also

[radiusMinimaxIC](#)

## Examples

```
N <- NormLocationFamily(mean=0, sd=1)
leastFavorableRadius(L2Fam=N, neighbor=ContNeighborhood(),
                    risk=asMSE(), rho=0.5)
```

---

lowerCaseRadius	<i>Computation of the lower case radius</i>
-----------------	---

---

## Description

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

## Usage

```
lowerCaseRadius(L2Fam, neighbor, risk, biastype, ...)
```

## Arguments

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

## Value

lower case radius

**Methods**

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

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

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

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

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

**Author(s)**

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

**References**

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

Ruckdeschel, P. (2005) Optimally One-Sided Bounded Influence Curves. *Mathematical Methods in Statistics* 14(1), 105-131.

**See Also**

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

**Examples**

```
lowerCaseRadius(BinomFamily(size = 10), ContNeighborhood(), asMSE())
lowerCaseRadius(BinomFamily(size = 10), TotalVarNeighborhood(), asMSE())
```

---

 minmaxBias

*Generic Function for the Computation of Bias-Optimally Robust ICs*

---

**Description**

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

**Usage**

```

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

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

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

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

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

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

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

```

**Arguments**

L2deriv	L2-derivative of some L2-differentiable family of probability measures.
neighbor	object of class "Neighborhood".
biastype	object of class "BiasType".
normtype	object of class "NormType".
...	additional parameters.
Distr	object of class "Distribution".
symm	logical: indicating symmetry of L2deriv.
z.start	initial value for the centering constant.
A.start	initial value for the standardizing matrix.
z.comp	logical indicator which indices need to be computed and which are 0 due to symmetry.
A.comp	matrix of logical indicator which indices need to be computed and which are 0 due to symmetry.
trafo	matrix: transformation of the parameter.

maxiter	the maximum number of iterations.
tol	the desired accuracy (convergence tolerance).
warn	logical: print warnings.
Finfo	Fisher information matrix.
verbose	logical: if TRUE, some messages are printed

### Value

The bias-optimally robust IC is computed.

### Methods

**L2deriv = "UnivariateDistribution", neighbor = "ContNeighborhood", biastype = "BiasType"**  
 computes the bias optimal influence curve for symmetric bias for L2 differentiable parametric families with unknown one-dimensional parameter.

**L2deriv = "UnivariateDistribution", neighbor = "ContNeighborhood", biastype = "asymmetricBias"**  
 computes the bias optimal influence curve for asymmetric bias for L2 differentiable parametric families with unknown one-dimensional parameter.

**L2deriv = "UnivariateDistribution", neighbor = "TotalVarNeighborhood", biastype = "BiasType"**  
 computes the bias optimal influence curve for symmetric bias for L2 differentiable parametric families with unknown one-dimensional parameter.

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

**L2deriv = "RealRandVariable", neighbor = "TotalNeighborhood", biastype = "BiasType"** computes the bias optimal influence curve for symmetric bias for L2 differentiable parametric families in a setting where we are interested in a  $p = 1$  dimensional aspect of an unknown  $k$ -dimensional parameter ( $k > 1$ ) where the underlying distribution is univariate.

### Author(s)

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

### References

- Rieder, H. (1980) Estimates derived from robust tests. *Ann. Stats.* **8**: 106–115.
- Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.
- Ruckdeschel, P. (2005) Optimally One-Sided Bounded Influence Curves. *Mathematical Methods in Statistics* *14*(1), 105-131.
- Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.

### See Also

[InfRobModel-class](#)

---

 optIC

*Generic function for the computation of optimally robust ICs*


---

**Description**

Generic function for the computation of optimally robust ICs.

**Usage**

```
optIC(model, risk, ...)

## S4 method for signature 'InfRobModel,asRisk'
optIC(model, risk, z.start = NULL, A.start = NULL,
       upper = 1e4, lower = 1e-4,
       OptOrIter = "iterate", maxiter = 50,
       tol = .Machine$double.eps^0.4, warn = TRUE,
       noLow = FALSE, verbose = NULL, ...)

## S4 method for signature 'InfRobModel,asUnOvShoot'
optIC(model, risk, upper = 1e4,
       lower = 1e-4, maxiter = 50,
       tol = .Machine$double.eps^0.4, warn = TRUE)

## S4 method for signature 'FixRobModel,fiUnOvShoot'
optIC(model, risk, sampleSize, upper = 1e4, lower = 1e-4,
       maxiter = 50, tol = .Machine$double.eps^0.4,
       warn = TRUE, Algo = "A", cont = "left",
       verbose = NULL)
```

**Arguments**

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

cont	"left" or "right".
noLow	logical: is lower case to be computed?
OptOrIter	character; which method to be used for determining Lagrange multipliers A and a: if (partially) matched to "optimize", getLagrangeMultByOptim is used; otherwise: by default, or if matched to "iterate" or to "doubleiterate", getLagrangeMultByIter is used. More specifically, when using getLagrangeMultByIter, and if argument risk is of class "asGRisk", by default and if matched to "iterate" we use only one (inner) iteration, if matched to "doubleiterate" we use up to Maxiter (inner) iterations.
verbose	logical: if TRUE, some messages are printed

### Details

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

### Value

Some optimally robust IC is computed.

### Methods

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

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

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

### Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

### References

- Huber, P.J. (1968) Robust Confidence Limits. *Z. Wahrscheinlichkeitstheor. Verw. Geb.* **10**:269–278.
- Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.
- Kohl, M. and Ruckdeschel, P. (2010): R package distrMod: Object-Oriented Implementation of Probability Models. *J. Statist. Softw.* **35**(10), 1–27
- Kohl, M. and Ruckdeschel, P., and Rieder, H. (2010): Infinitesimally Robust Estimation in General Smoothly Parametrized Models. *Stat. Methods Appl.*, **19**, 333–354.
- Rieder, H. (1980) Estimates derived from robust tests. *Ann. Stats.* **8**: 106–115.
- Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.

Rieder, H., Kohl, M. and Ruckdeschel, P. (2008) The Costs of not Knowing the Radius. *Statistical Methods and Applications* **17**(1) 13-40.

Rieder, H., Kohl, M. and Ruckdeschel, P. (2001) The Costs of not Knowing the Radius. Appeared as discussion paper Nr. 81. SFB 373 (Quantification and Simulation of Economic Processes), Humboldt University, Berlin; also available under [www.uni-bayreuth.de/departments/math/org/mathe7/RIEDER/pubs/RR.pdf](http://www.uni-bayreuth.de/departments/math/org/mathe7/RIEDER/pubs/RR.pdf)

## See Also

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

## Examples

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

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

---

optRisk

*Generic function for the computation of the minimal risk*

---

## Description

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

## Usage

```
optRisk(model, risk, ...)

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

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

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

**Arguments**

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

**Details**

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

**Value**

The minimal risk is computed.

**Methods**

**model = "L2ParamFamily", risk = "asCov"** asymptotic covariance of L2 differentiable parametric family.

**model = "InfRobModel", risk = "asRisk"** asymptotic risk of a infinitesimal robust model.

**model = "FixRobModel", risk = "fiUnOvShoot"** finite-sample under-/overshoot risk of a robust model with fixed neighborhood.

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**

- 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.

**See Also**[RiskType-class](#)**Examples**

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

---

radiusMinimaxIC	<i>Generic function for the computation of the radius minimax IC</i>
-----------------	--

---

**Description**

Generic function for the computation of the radius minimax IC.

**Usage**

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

## S4 method for signature 'L2ParamFamily,UncondNeighborhood,asGRisk'
radiusMinimaxIC(
  L2Fam, neighbor, risk, loRad = 0, upRad = Inf, z.start = NULL, A.start = NULL,
  upper = NULL, lower = NULL, OptOrIter = "iterate",
  maxiter = 50, tol = .Machine$double.eps^0.4,
  warn = FALSE, verbose = NULL, loRad0 = 1e-3, ...)
```

**Arguments**

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

tol	the desired accuracy (convergence tolerance).
warn	logical: print warnings.
verbose	logical: if TRUE, some messages are printed
loRad0	for numerical reasons: the effective lower bound for the zero search; internally set to $\max(\text{loRad}, \text{loRad0})$ .
...	further arguments to be passed on to <code>getInfRobIC</code>

### Details

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

### Value

The radius minimax IC is computed.

### Methods

**L2Fam = "L2ParamFamily", neighbor = "UncondNeighborhood", risk = "asGRisk"**: computation of the radius minimax IC for an L2 differentiable parametric family.

### Author(s)

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

### References

Rieder, H., Kohl, M. and Ruckdeschel, P. (2008) The Costs of not Knowing the Radius. *Statistical Methods and Applications*, 17(1) 13-40.

Rieder, H., Kohl, M. and Ruckdeschel, P. (2001) The Costs of not Knowing the Radius. Appeared as discussion paper Nr. 81. SFB 373 (Quantification and Simulation of Economic Processes), Humboldt University, Berlin; also available under [www.uni-bayreuth.de/departments/math/org/mathe7/RIEDER/pubs/RR.pdf](http://www.uni-bayreuth.de/departments/math/org/mathe7/RIEDER/pubs/RR.pdf)

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

### See Also

[radiusMinimaxIC](#)

### Examples

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

---

roptest	<i>Optimally robust estimation</i>
---------	------------------------------------

---

### Description

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

### Usage

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

### Arguments

<code>x</code>	sample
<code>L2Fam</code>	object of class "L2ParamFamily"
<code>eps</code>	positive real ( $0 < \text{eps} \leq 0.5$ ): amount of gross errors. See details below.
<code>eps.lower</code>	positive real ( $0 \leq \text{eps.lower} \leq \text{eps.upper}$ ): lower bound for the amount of gross errors. See details below.
<code>eps.upper</code>	positive real ( $\text{eps.lower} \leq \text{eps.upper} \leq 0.5$ ): upper bound for the amount of gross errors. See details below.
<code>fsCor</code>	positive real: factor used to correct the neighborhood radius; see details.
<code>initial.est</code>	initial estimate for unknown parameter. If missing minimum distance estimator is computed.
<code>neighbor</code>	object of class "UncondNeighborhood"
<code>risk</code>	object of class "RiskType"
<code>steps</code>	positive integer: number of steps used for k-steps construction
<code>distance</code>	distance function
<code>startPar</code>	initial information used by optimize resp. optim; i.e; if (total) parameter is of length 1, startPar is a search interval, else it is an initial parameter value; if NULL slot startPar of ParamFamily is used to produce it; in the multivariate case, startPar may also be of class Estimate, in which case slot untransformed.estimate is used.
<code>verbose</code>	logical: if TRUE, some messages are printed

<code>useLast</code>	which parameter estimate (initial estimate or k-step estimate) shall be used to fill the slots <code>pIC</code> , <code>asvar</code> and <code>asbias</code> of the return value.
<code>OptOrIter</code>	character; which method to be used for determining Lagrange multipliers $A$ and $a$ : if (partially) matched to "optimize", <code>getLagrangeMultByOptim</code> is used; otherwise: by default, or if matched to "iterate" or to "doubleiterate", <code>getLagrangeMultByIter</code> is used. More specifically, when using <code>getLagrangeMultByIter</code> , and if argument <code>risk</code> is of class "asGRisk", by default and if matched to "iterate" we use only one (inner) iteration, if matched to "doubleiterate" we use up to <code>Maxiter</code> (inner) iterations.
<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>withPICList</code>	logical: shall slot <code>pICList</code> of return value be filled?
<code>withICList</code>	logical: shall slot <code>ICList</code> of return value be filled?
<code>na.rm</code>	logical: if TRUE, the estimator is evaluated at <code>complete.cases(x)</code> .
<code>initial.est.ArgList</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>L2Fam</code> .
<code>...</code>	further arguments

## Details

Computes the optimally robust estimator for a given L2 differentiable parametric family. The computation uses a k-step construction with an appropriate initial estimate; cf. also [kStepEstimator](#). Valid candidates are e.g. Kolmogorov(-Smirnov) or von Mises minimum distance estimators (default); cf. Rieder (1994) and Kohl (2005).

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

If the amount of gross errors (contamination) is unknown, try to find a rough estimate for the amount of gross errors, such that it lies between `eps.lower` and `eps.upper`.

In case `eps.lower` is specified and `eps.upper` is missing, `eps.upper` is set to 0.5. In case `eps.upper` is specified and `eps.lower` is missing, `eps.lower` is set to 0.

If neither `eps` nor `eps.lower` and/or `eps.upper` is specified, `eps.lower` and `eps.upper` are set to 0 and 0.5, respectively.

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

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

The default value of argument `useLast` is set by the global option `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` 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

- Kohl, M. (2005) *Numerical Contributions to the Asymptotic Theory of Robustness*. Bayreuth: Dissertation.
- Kohl, M. and Ruckdeschel, P. (2010): R package `distrMod`: Object-Oriented Implementation of Probability Models. *J. Statist. Softw.* **35**(10), 1–27
- Kohl, M. and Ruckdeschel, P., and Rieder, H. (2010): Infinitesimally Robust Estimation in General Smoothly Parametrized Models. *Stat. Methods Appl.*, **19**, 333–354.
- Rieder, H. (1994) *Robust Asymptotic Statistics*. New York: Springer.
- Rieder, H., Kohl, M. and Ruckdeschel, P. (2008) The Costs of not Knowing the Radius. *Statistical Methods and Applications* **17**(1) 13-40.
- Rieder, H., Kohl, M. and Ruckdeschel, P. (2001) The Costs of not Knowing the Radius. Appeared as discussion paper Nr. 81. SFB 373 (Quantification and Simulation of Economic Processes), Humboldt University, Berlin; also available under [www.uni-bayreuth.de/departments/math/org/mathe7/RIEDER/pubs/RR.pdf](http://www.uni-bayreuth.de/departments/math/org/mathe7/RIEDER/pubs/RR.pdf)

## See Also

[roblox](#), [L2ParamFamily-class](#) [UncondNeighborhood-class](#), [RiskType-class](#)

## Examples

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

## ML-estimate
MLest <- MLEstimator(x, BinomFamily(size = 25))
```

```

estimate(MLest)
confint(MLest)

## compute optimally robust estimator (known contamination)
robtest1 <- roptest(x, BinomFamily(size = 25), eps = 0.05, steps = 3)
estimate(robtest1)
confint(robtest1, method = symmetricBias())
## neglecting bias
confint(robtest1)
plot(pIC(robtest1))
qq1 <- qqplot(x, robtest1, cex.pch=1.5, exp.cex2.pch = -.25,
              exp.fadcol.pch = .55, jit.fac=.9)
str(qq1)

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

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

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

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

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

## compute optimally robust estimator (unknown contamination)
robtest <- roptest(x, PoisFamily(), eps.upper = 0.1, steps = 3)
estimate(robtest)
confint(robtest, symmetricBias())
plot(pIC(robtest))

```

```

qq2 <- qqplot(x, robest, cex.pch=1.5, exp.cex2.pch = -.25,
              exp.fadcol.pch = .55, jit.fac=.9)
str(qq2)

## total variation neighborhoods (unknown deviation)
robest1 <- roptest(x, PoisFamily(), eps.upper = 0.05,
                  neighbor = TotalVarNeighborhood(), steps = 3)
estimate(robest1)
confint(robest1, symmetricBias())
plot(pIC(robest1))

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

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

## compute optimally robust estimator (known contamination)
## takes some time -> you can use package RobLox for normal
## location and scale which is optimized for speed
robest <- roptest(chem, NormLocationScaleFamily(), eps = 0.05, steps = 3)
estimate(robest)
confint(robest, symmetricBias())
plot(pIC(robest))
## plot of relative and absolute information; cf. Kohl (2005)
infoPlot(pIC(robest))

qq3 <- qqplot(chem, robest, cex.pch=1.5, exp.cex2.pch = -.25,
              exp.fadcol.pch = .55, withLab = TRUE, which.Order=1:4,
              exp.cex2.lbl = .12, exp.fadcol.lbl = .45,
              nosym.pCI = TRUE, adj.lbl=c(1.7,.2),
              exact.pCI = FALSE, log ="xy")
str(qq3)

## finite-sample correction
if(require(RobLox)){
  n <- length(chem)
  r <- 0.05*sqrt(n)
  r.fi <- finiteSampleCorrection(n = n, r = r)
  fsCor <- r.fi/r
  robest <- roptest(chem, NormLocationScaleFamily(), eps = 0.05,
                    fsCor = fsCor, steps = 3)
  estimate(robest)
}

```

```

## compute optimally robust estimator (unknown contamination)
## takes some time -> use package RobLox!
robtest1 <- roptest(chem, NormLocationScaleFamily(), eps.lower = 0.05,
                  eps.upper = 0.1, steps = 3)
estimate(robtest1)
confint(robtest1, symmetricBias())
plot(pIC(robtest1))
## plot of relative and absolute information; cf. Kohl (2005)
infoPlot(pIC(robtest1))

```

---

updateNorm-methods      *Methods for Function updateNorm in Package 'ROptEst'*

---

## Description

updateNorm-methods to update norm in IC-Algo

## Usage

```

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

```

## Arguments

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

## Details

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

## Value

updateNorm      an updated object of class NormType.

**Methods**

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

**Author(s)**

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

**See Also**

[NormType-class](#)

# Index

## \*Topic **classes**

- asAnscombe-class, 5
- asL1-class, 7
- asL4-class, 9
- get.asGRisk.fct-methods, 12
- updateNorm-methods, 64

## \*Topic **package**

- ROptEst-package, 3

## \*Topic **robust**

- asAnscombe, 4
- asL1, 6
- asL4, 8
- cniperCont, 10
- getAsRisk, 13
- getBiasIC, 17
- getFiRisk, 18
- getFixClip, 20
- getFixRobIC, 21
- getIneffDiff, 22
- getInfCent, 24
- getInfClip, 26
- getInfGamma, 28
- getInfLM, 30
- getInfRobIC, 32
- getInfStand, 36
- getInfV, 38
- getL1normL2deriv, 40
- getL2normL2deriv, 41
- getMaxIneff, 41
- getModifyIC, 43
- getReq, 44
- getRiskIC, 46
- leastFavorableRadius, 47
- lowerCaseRadius, 49
- minmaxBias, 50
- optIC, 53
- optRisk, 55
- radiusMinimaxIC, 57
- roptest, 59

- asAnscombe, 4, 6
- asAnscombe-class, 5
- asAnscombe-class, 5
- asGRisk-class, 8, 10, 30
- asL1, 6, 8, 9
- asL1-class, 7, 10
- asL1-class, 7
- asL4, 7, 8, 10
- asL4-class, 8, 9
- asL4-class, 9
- asMSE, 7–10
- asMSE-class, 8, 10, 30
- asRisk-class, 6, 17
- asUnOvShoot-class, 30

- cniperCont, 10
- cniperCont, IC, IC, L2ParamFamily, ContNeighborhood, asMSE-method (cniperCont), 10
- cniperCont-methods (cniperCont), 10
- cniperPoint (cniperCont), 10
- cniperPoint, L2ParamFamily, ContNeighborhood, asMSE-method (cniperCont), 10
- cniperPoint-methods (cniperCont), 10
- cniperPointPlot (cniperCont), 10
- cniperPointPlot, L2ParamFamily, ContNeighborhood, asMSE-method (cniperCont), 10
- cniperPointPlot-methods (cniperCont), 10
- ContIC-class, 21, 26, 28, 30, 38, 39

- distr-package, 3
- distrEx-package, 3
- distrMod-package, 3

- eff (asAnscombe-class), 5
- eff, asAnscombe-method (asAnscombe-class), 5

- finiteSampleCorrection, 60
- fiRisk-class, 19
- FixRobModel-class, 22

- get.asGRisk.fct  
     (get.asGRisk.fct-methods), 12  
 get.asGRisk.fct,asL1-method  
     (get.asGRisk.fct-methods), 12  
 get.asGRisk.fct,asL4-method  
     (get.asGRisk.fct-methods), 12  
 get.asGRisk.fct,asMSE-method  
     (get.asGRisk.fct-methods), 12  
 get.asGRisk.fct-methods, 12  
 getAsRisk, 12, 13  
 getAsRisk,asAnscombe,RealRandVariable,ContNeighborhood,ANY-method  
     (getAsRisk), 13  
 getAsRisk,asAnscombe,UnivariateDistribution,UncondNeighborhood,ANY-method  
     (getAsRisk), 13  
 getAsRisk,asBias,RealRandVariable,ContNeighborhood,ANY-method  
     (getAsRisk), 13  
 getAsRisk,asBias,RealRandVariable,TotalVarNeighborhood,ANY-method  
     (getAsRisk), 13  
 getAsRisk,asBias,UnivariateDistribution,ContNeighborhood,ANY-method  
     (getAsRisk), 13  
 getAsRisk,asBias,UnivariateDistribution,ContNeighborhood,AsymTriObs,ContNeighborhood-method  
     (getAsRisk), 13  
 getAsRisk,asBias,UnivariateDistribution,ContNeighborhood,onesideBias,ContNeighborhood-method  
     (getAsRisk), 13  
 getAsRisk,asBias,UnivariateDistribution,TotalVarNeighborhood,ANY-method  
     (getAsRisk), 13  
 getAsRisk,asBias,UnivariateDistribution,TotalVarNeighborhood,ContNeighborhood,asMSPadmaFamily,UncondNeighborhood,asMSPadmaFamily,ANY-method  
     (getAsRisk), 13  
 getAsRisk,asCov,RealRandVariable,ContNeighborhood,ANY-method  
     (getAsRisk), 13  
 getAsRisk,asCov,UnivariateDistribution,ContNeighborhood,RealRandVariable,ContNeighborhood,BiasType-method  
     (getAsRisk), 13  
 getAsRisk,asCov,UnivariateDistribution,TotalVarNeighborhood,RealRandVariable,TotalVarNeighborhood,BiasType-method  
     (getAsRisk), 13  
 getAsRisk,asL1,UnivariateDistribution,Neighborhood,ANY-method  
     (getAsRisk), 13  
 getAsRisk,asL4,UnivariateDistribution,Neighborhood,ANY-method  
     (getAsRisk), 13  
 getAsRisk,asMSE,EuclRandVariable,Neighborhood,ANY-method  
     (getAsRisk), 13  
 getAsRisk,asMSE,UnivariateDistribution,Neighborhood,ANY-method  
     (getAsRisk), 13  
 getAsRisk,asSemivar,UnivariateDistribution,Neighborhood,onesideBias,ContNeighborhood-method  
     (getAsRisk), 13  
 getAsRisk,asUnOvShoot,UnivariateDistribution,Neighborhood,ANY-method  
     (getAsRisk), 13  
 getAsRisk,trAsCov,RealRandVariable,ContNeighborhood,ANY-method  
     (getAsRisk), 13  
 getAsRisk,trAsCov,UnivariateDistribution,UncondNeighborhood,ANY-method  
     (getAsRisk), 13  
 getAsRisk-methods (getAsRisk), 13  
 getBiasIC, 17  
 getBiasIC,HampIC,UncondNeighborhood-method  
     (getBiasIC), 17  
 getBiasIC,TotalVarIC,UncondNeighborhood-method  
     (getBiasIC), 17  
 getBiasIC-methods (getBiasIC), 17  
 getFiRisk, 18  
 getFiRisk,fiUnOvShoot,Norm,ContNeighborhood-method  
     (getFiRisk), 18  
 getFiRisk,fiUnOvShoot,Norm,TotalVarNeighborhood-method  
     (getFiRisk), 18  
 getFiRisk-methods (getFiRisk), 18  
 getFixClip, 20  
 getFixClip,ANY-method  
     (getFixClip), 20  
 getFixClip,ANY-method,fiUnOvShoot,ContNeighborhood-method  
     (getFixClip), 20  
 getFixClip,ANY-method,fiUnOvShoot,TotalVarNeighborhood-method  
     (getFixClip), 20  
 getFixClip,ANY-method,fiUnOvShoot,TotalVarNeighborhood-method  
     (getFixClip), 20  
 getFixRobIC, 21  
 getFixRobIC,AsymTriObs,ContNeighborhood-method  
     (getFixRobIC), 21  
 getFixRobIC,onesideBias,ContNeighborhood-method  
     (getFixRobIC), 21  
 getIneffDiff, 22  
 getIneffDiff,ANY-method  
     (getIneffDiff), 22  
 getInfCent, 24  
 getInfCent,RealRandVariable,ContNeighborhood,BiasType-method  
     (getInfCent), 24  
 getInfCent,RealRandVariable,TotalVarNeighborhood,BiasType-method  
     (getInfCent), 24  
 getInfCent,UnivariateDistribution,ContNeighborhood,asymmetric,ANY-method  
     (getInfCent), 24  
 getInfCent,UnivariateDistribution,ContNeighborhood,BiasType-method  
     (getInfCent), 24  
 getInfCent,UnivariateDistribution,ContNeighborhood,onesideBias,ContNeighborhood-method  
     (getInfCent), 24  
 getInfClip, 26  
 getInfClip,ANY-method  
     (getInfClip), 26  
 getInfClip,UnivariateDistribution,asL1,ContNeighborhood,ANY-method  
     (getInfClip), 26  
 getInfClip,UnivariateDistribution,asL1,TotalVarNeighborhood,ANY-method  
     (getInfClip), 26

- getInfClip, numeric, UnivariateDistribution, asL4, ContNeighborhood, BiasType-method  
 (getInfClip), 26
- getInfClip, numeric, UnivariateDistribution, asL4, TotalVarNeighborhood, BiasType-method  
 (getInfClip), 26
- getInfClip, numeric, UnivariateDistribution, asMSE, ContNeighborhood, BiasType-method  
 (getInfClip), 26
- getInfClip, numeric, UnivariateDistribution, asMSE, TotalVarNeighborhood, BiasType-method  
 (getInfClip), 26
- getInfClip, numeric, UnivariateDistribution, asSym, ContNeighborhood, BiasType-method  
 (getInfClip), 26
- getInfClip, numeric, UnivariateDistribution, asUnOvShoot, ContNeighborhood, BiasType-method  
 (getInfClip), 26
- getInfClip-methods (getInfClip), 26
- getInfGamma, 28
- getInfGamma, RealRandVariable, asMSE, ContNeighborhood, BiasType-method  
 (getInfGamma), 28
- getInfGamma, RealRandVariable, asMSE, TotalVarNeighborhood, BiasType-method  
 (getInfGamma), 28
- getInfGamma, UnivariateDistribution, asGRisk, ContNeighborhood, BiasType-method  
 (getInfGamma), 28
- getInfGamma, UnivariateDistribution, asGRisk, TotalVarNeighborhood, BiasType-method  
 (getInfGamma), 28
- getInfGamma, UnivariateDistribution, asMSE, ContNeighborhood, BiasType-method  
 (getInfGamma), 28
- getInfGamma, UnivariateDistribution, asMSE, ContNeighborhood, BiasType-method  
 (getInfGamma), 28
- getInfGamma, UnivariateDistribution, asMSE, ContNeighborhood, BiasType-method  
 (getInfGamma), 28
- getInfGamma, UnivariateDistribution, asUnOvShoot, ContNeighborhood, BiasType-method  
 (getInfGamma), 28
- getInfGamma-methods (getInfGamma), 28
- getInfLM, 30
- getInfRobIC, 32
- getInfRobIC, RealRandVariable, asAnscombe, UncondNeighborhood, BiasType-method  
 (getInfRobIC), 32
- getInfRobIC, RealRandVariable, asBias, UncondNeighborhood, BiasType-method  
 (getInfRobIC), 32
- getInfRobIC, RealRandVariable, asCov, UncondNeighborhood, BiasType-method  
 (getInfRobIC), 32
- getInfRobIC, RealRandVariable, asGRisk, UncondNeighborhood, BiasType-method  
 (getInfRobIC), 32
- getInfRobIC, RealRandVariable, asHampel, UncondNeighborhood, BiasType-method  
 (getInfRobIC), 32
- getInfRobIC, UnivariateDistribution, asAnscombe, UncondNeighborhood, BiasType-method  
 (getInfRobIC), 32
- getInfRobIC, UnivariateDistribution, asBias, UncondNeighborhood, BiasType-method  
 (getInfRobIC), 32
- getInfRobIC, UnivariateDistribution, asCov, ContNeighborhood, BiasType-method  
 (getInfRobIC), 32
- getInfRobIC, UnivariateDistribution, asCov, TotalVarNeighborhood, BiasType-method  
 (getInfRobIC), 32
- getInfRobIC, UnivariateDistribution, asGRisk, UncondNeighborhood, BiasType-method  
 (getInfRobIC), 32
- getInfRobIC, UnivariateDistribution, asHampel, UncondNeighborhood, BiasType-method  
 (getInfRobIC), 32
- getInfRobIC, UnivariateDistribution, asUnOvShoot, UncondNeighborhood, BiasType-method  
 (getInfRobIC), 32
- getInfRobIC-methods (getInfRobIC), 32
- getInfStand, RealRandVariable, UncondNeighborhood, BiasType-method  
 (getInfStand), 36
- getInfStand, UnivariateDistribution, ContNeighborhood, asymmetricBias-method  
 (getInfStand), 36
- getInfStand, UnivariateDistribution, ContNeighborhood, BiasType-method  
 (getInfStand), 36
- getInfStand, UnivariateDistribution, ContNeighborhood, onesidedBias-method  
 (getInfStand), 36
- getInfStand, UnivariateDistribution, TotalVarNeighborhood, BiasType-method  
 (getInfStand), 36
- getInfStand-methods (getInfStand), 36
- getInfV, RealRandVariable, ContNeighborhood, BiasType-method  
 (getInfV), 38
- getInfV, RealRandVariable, TotalVarNeighborhood, BiasType-method  
 (getInfV), 38
- getInfV, UnivariateDistribution, ContNeighborhood, asymmetricBias-method  
 (getInfV), 38
- getInfV, UnivariateDistribution, ContNeighborhood, BiasType-method  
 (getInfV), 38
- getInfV, UnivariateDistribution, ContNeighborhood, onesidedBias-method  
 (getInfV), 38
- getInfV, UnivariateDistribution, TotalVarNeighborhood, BiasType-method  
 (getInfV), 38
- getL1normL2deriv, 40
- getL1normL2deriv, RealRandVariable-method  
 (getL1normL2deriv), 40
- getL1normL2deriv, UnivariateDistribution-method  
 (getL1normL2deriv), 40
- getL1normL2deriv-methods  
 (getL1normL2deriv), 40
- getL2normL2deriv, 40
- getL2normL2deriv, UnivariateDistribution-method  
 (getL2normL2deriv), 40
- getLagrangeMultByIter (getInfLM), 30
- getLagrangeMultByIter (getInfLM), 30
- getMaxIneff, 41
- getModifyIC, 43
- getModifyIC, L2LocationFamily, UncondNeighborhood, asGRisk-method  
 (getModifyIC), 43

- getModifyIC,L2LocationFamily,UncondNeighborhood,asRisk-method  
 (getModifyIC), 43
- getModifyIC,L2LocationFamily,UncondNeighborhood,asRisk-method  
 (getModifyIC), 43
- getModifyIC,L2ParamFamily,Neighborhood,asRisk-method  
 (getModifyIC), 43
- getModifyIC,L2ScaleFamily,UncondNeighborhood,asRisk-method  
 (getModifyIC), 43
- getModifyIC-methods (getModifyIC), 43
- getReq, 12, 44
- getRiskIC, 46, 47
- getRiskIC,HampIC,asCov,missing,L2ParamFamily-method  
 (getRiskIC), 46
- getRiskIC,HampIC,asCov,missing,missing-method  
 (getRiskIC), 46
- getRiskIC>TotalVarIC,asCov,missing,L2ParamFamily-method  
 (getRiskIC), 46
- getRiskIC-methods, 18
- getRiskIC-methods (getRiskIC), 46
- IC-class, 44
- InfluenceCurve-class, 55
- InfRobModel-class, 18, 32, 36, 47, 52
- kStepEstimator, 60
- L2ParamFamily-class, 50, 61
- leastFavorableRadius, 24, 47
- leastFavorableRadius,L2ParamFamily,UncondNeighborhood,asRisk-method  
 (leastFavorableRadius), 47
- leastFavorableRadius-methods  
 (leastFavorableRadius), 47
- lowerCaseRadius, 49
- lowerCaseRadius,L2ParamFamily,ContNeighborhood,asMSE,ANY-method  
 (lowerCaseRadius), 49
- lowerCaseRadius,L2ParamFamily,ContNeighborhood,asMSE,asymmetricBias-method  
 (lowerCaseRadius), 49
- lowerCaseRadius,L2ParamFamily,ContNeighborhood,asMSE,onesidedBias-method  
 (lowerCaseRadius), 49
- lowerCaseRadius,L2ParamFamily>TotalVarNeighborhood,asMSE,ANY-method  
 (lowerCaseRadius), 49
- lowerCaseRadius,UnivariateDistribution,ContNeighborhood,asMSE,onesidedBias-method  
 (lowerCaseRadius), 49
- lowerCaseRadius-methods  
 (lowerCaseRadius), 49
- minmaxBias, 50
- minmaxBias,RealRandVariable,ContNeighborhood,asRisk-method  
 (minmaxBias), 50
- minmaxBias,RealRandVariable,ContNeighborhood,asRisk-method  
 (minmaxBias), 50
- minmaxBias,UnivariateDistribution,ContNeighborhood,BiasType-  
 (minmaxBias), 50
- minmaxBias,UnivariateDistribution,ContNeighborhood,oneside  
 (minmaxBias), 50
- minmaxBias,UnivariateDistribution,TotalVarNeighborhood,Bia  
 (minmaxBias), 50
- minmaxBias-methods (minmaxBias), 50
- Neighborhood-class, 50
- NormType-class, 65
- optIC, 44, 53
- optIC,FixRobModel,fiUnOvShoot-method  
 (optIC), 53
- optIC,InfRobModel,asRisk-method  
 (optIC), 53
- optIC,InfRobModel,asUnOvShoot-method  
 (optIC), 53
- optIC-methods (optIC), 53
- optRisk, 55
- optRisk,FixRobModel,fiUnOvShoot-method  
 (optRisk), 55
- optRisk,InfRobModel,asRisk-method  
 (optRisk), 55
- optRisk,L2ParamFamily,asCov-method  
 (optRisk), 55
- optRisk-methods (optRisk), 55
- radiusMinimaxIC, 24, 49, 57, 58
- radiusMinimaxIC,L2ParamFamily,UncondNeighborhood,asGRisk-m  
 (radiusMinimaxIC), 57
- radiusMinimaxIC-methods  
 (radiusMinimaxIC), 57
- RandVar-package, 3
- RiskType-class, 55, 57, 61
- RobASTBase-package, 3
- ROptEst (ROptEst-package), 3
- ROptEst, 3
- ROptEst-package, 3
- scaleUpdateIC (getModifyIC), 43
- scaleUpdateIC,ContNeighborhood-method  
 (getModifyIC), 43
- scaleUpdateIC>TotalVarNeighborhood-method  
 (getModifyIC), 43

scaleUpdateIC, UncondNeighborhood-method  
    (getModifyIC), [43](#)  
scaleUpdateIC-methods (getModifyIC), [43](#)  
show, asAnscombe-method  
    (asAnscombe-class), [5](#)

TotalVarIC-class, [21](#), [26](#), [28](#), [30](#), [38](#), [39](#)

UncondNeighborhood-class, [61](#)  
updateNorm (updateNorm-methods), [64](#)  
updateNorm, SelfNorm-method  
    (updateNorm-methods), [64](#)  
updateNorm-methods, [64](#)