Package ‘RobLox’

April 11, 2019

Version 1.2.0
Date 2019-04-02
Title Optimally Robust Influence Curves and Estimators for Location and Scale
Description Functions for the determination of optimally robust influence curves and estimators in case of normal location and/or scale.
Depends R(>= 3.4), stats, distrMod(>= 2.8.0), RobAStBase(>= 1.2.0)
Imports methods, lattice, RColorBrewer, Biobase, RandVar(>= 1.2.0), distr(>= 2.8.0)
Suggests MASS
ByteCompile yes
License LGPL-3
Encoding latin1
URL http://robast.r-forge.r-project.org/
LastChangedDate {$LastChangedDate: 2019-04-02 21:10:23 +0200 (Di, 02. Apr 2019) $}
LastChangedRevision {$LastChangedRevision: 1215 $}
VCS/SVNRevision 1214
NeedsCompilation no
Author Matthias Kohl [cre, cph],
Peter Ruckdeschel [aut, cph]
Maintainer Matthias Kohl <Matthias.Kohl@stamats.de>
Repository CRAN
Date/Publication 2019-04-11 06:45:19 UTC

R topics documented:

RobLox-package .......................................................... 2
finiteSampleCorrection .............................................. 4
RobLox-package

Optimally robust influence curves and estimators for location and scale

Description

Functions for the determination of optimally robust influence curves and estimators in case of normal location and/or scale.

Details

Package: RobLox
Version: 1.2.0
Date: 2019-04-02
Depends: R(>= 3.4), stats, distrMod(>= 2.8.0), RobAStBase(>= 1.2.0)
Imports: methods, lattice, RColorBrewer, Biobase, RandVar(>= 1.2.0), distr(>= 2.8.0)
Suggests: MASS
ByteCompile: yes
License: LGPL-3
URL: http://robast.r-forge.r-project.org/
VCS/SVNRevision: 1214
Package versions

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

Author(s)

Matthias Kohl <matthias.kohl@stamats.de>

References


See Also

RobASlBase-package

Examples

```r
library(RobLox)
ind <- rbinom(100, size=1, prob=0.05)
x <- rnorm(100, mean=ind*3, sd=(1-ind) + ind*9)
roblox(x)
res <- roblox(x, eps.lower = 0.01, eps.upper = 0.1, returnIC = TRUE)
estimate(res)
confint(res)
confint(res, method = symmetricBias())
pIC(res)
  ## don't run to reduce check time on CRAN
  ## Not run:
  checkIC(pIC(res))
  Risks(pIC(res))
  Infos(pIC(res))
  plot(pIC(res))
  infoPlot(pIC(res))

  ## End(Not run)
  ## row-wise application
ind <- rbinom(200, size=1, prob=0.05)
X <- matrix(rnorm(200, mean=ind*3, sd=(1-ind) + ind*9), nrow = 2)
rowRoblox(X)
```
finiteSampleCorrection

*Function to compute finite-sample corrected radii*

**Description**

Given some radius and some sample size the function computes the corresponding finite-sample corrected radius.

**Usage**

```
finiteSampleCorrection(r, n, model = "locsc")
```

**Arguments**

- `r`: asymptotic radius (non-negative numeric)
- `n`: sample size
- `model`: has to be "locsc" (for location and scale), "loc" (for location) or "sc" (for scale), respectively.

**Details**

The finite-sample correction is based on empirical results obtained via simulation studies.

Given some radius of a shrinking contamination neighborhood which leads to an asymptotically optimal robust estimator, the finite-sample empirical MSE based on contaminated samples was minimized for this class of asymptotically optimal estimators and the corresponding finite-sample radius determined and saved.

The computation is based on the saved results of these Monte-Carlo simulations.

**Value**

Finite-sample corrected radius.

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**


**rloptic**

**Description**

The function `rloptic` computes the optimally robust IC for AL estimators in case of normal location and (convex) contamination neighborhoods. The definition of these estimators can be found in Rieder (1994) or Kohl (2005), respectively.

**Usage**

```r
rloptic(r, mean = 0, sd = 1, bUp = 1000, computeIC = TRUE)
```

**Arguments**

- `r`: non-negative real: neighborhood radius.
- `mean`: specified mean.
- `sd`: specified standard deviation.
- `bUp`: positive real: the upper end point of the interval to be searched for the clipping bound b.
- `computeIC`: logical: should IC be computed. See details below.

**Details**

If 'computeIC' is 'FALSE' only the Lagrange multipliers 'A', 'a', and 'b' contained in the optimally robust IC are computed.

**Value**

If 'computeIC' is 'TRUE' an object of class "ContIC" is returned, otherwise a list of Lagrange multipliers

- `A`: standardizing constant
- `a`: centering constant; always '=' 0' is this symmetric setup
- `b`: optimal clipping bound

---

**See Also**

`roblox, rowRoblox, colRoblox`

**Examples**

```r
finiteSampleCorrection(n = 3, r = 0.001, model = "locsc")
finteSampleCorrection(n = 10, r = 0.02, model = "loc")
finteSampleCorrection(n = 250, r = 0.15, model = "sc")
```
Author(s)
Matthias Kohl <Matthias.Kohl@stamats.de>

References

See Also
ContIC-class, roblox

Examples
IC1 <- rlsopticAL(r = 0.1)
distrExOptions("ERelativeTolerance" = 1e-12)
checkIC(IC1)
distrExOptions("ERelativeTolerance" = .Machine$double.eps^0.25) # default
Risks(IC1)
cent(IC1)
clip(IC1)
stand(IC1)
plot(IC1)

Description
The function rlsopticAL computes the optimally robust IC for AL estimators in case of normal location with unknown scale and (convex) contamination neighborhoods. The definition of these estimators can be found in Section 8.2 of Kohl (2005).

Usage
rlsopticAL(r, mean = 0, sd = 1, A.loc.start = 1, a.sc.start = 0,
A.sc.start = 0.5, bUp = 1000, delta = 1e-6, itmax = 100,
check = FALSE, computeIC = TRUE)

Arguments
r non-negative real: neighborhood radius.
mean specified mean.
sd specified standard deviation.
A.loc.start positive real: starting value for the standardizing constant of the location part.
rlsOptIC.AL

- `a.sc.start` real: starting value for centering constant of the scale part.
- `A.sc.start` positive real: starting value for the standardizing constant of the scale part.
- `bUp` positive real: the upper end point of the interval to be searched for the clipping bound `b`.
- `delta` the desired accuracy (convergence tolerance).
- `itmax` the maximum number of iterations.
- `check` logical: should constraints be checked.
- `computeIC` logical: should IC be computed. See details below.

Details

The Lagrange multipliers contained in the expression of the optimally robust IC can be accessed via the accessor functions `cent`, `clip` and `stand`. If `computeIC` is 'FALSE' only the Lagrange multipliers 'A', 'a', and 'b' contained in the optimally robust IC are computed.

Value

If `computeIC` is 'TRUE' an object of class "ContIC" is returned, otherwise a list of Lagrange multipliers

- `A` standardizing matrix
- `a` centering vector
- `b` optimal clipping bound

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

`ContIC-class`, `roblox`

Examples

```r
IC1 <- rlsOptIC.AL(r = 0.1, check = TRUE)
distrExOptions("ErelativeTolerance" = 1e-12)
checkIC(IC1)
distrExOptions("ErelativeTolerance" = .Machine$double.eps^0.25) # default
Risks(IC1)
cent(IC1)
clip(IC1)
stand(IC1)
```
## Description

The function `rlsOptIC.An1` computes the optimally robust IC for An1 estimators in case of normal location with unknown scale and (convex) contamination neighborhoods. The definition of these estimators can be found in Subsection 8.5.3 of Kohl (2005).
Usage
rlsOptIC.An1(r, aUp = 2.5, delta = 1e-06)

Arguments
- r: non-negative real: neighborhood radius.
- aUp: positive real: the upper end point of the interval to be searched for a.
- delta: the desired accuracy (convergence tolerance).

Details
The optimal value of the tuning constant a can be read off from the slot Infos of the resulting IC.

Value
Object of class "IC"

Author(s)
Matthias Kohl <Matthias.Kohl@stamats.de>

References

See Also
IC-class

Examples
IC1 <- rlsOptIC.An1(r = 0.1)
checkIC(IC1)
Risks(IC1)
Infos(IC1)
## don't run to reduce check time on CRAN
## Not run:
plot(IC1)
infoPlot(IC1)

## End(Not run)
Computation of the optimally robust IC for An2 estimators

Description

The function rlsOptIC.An2 computes the optimally robust IC for An2 estimators in case of normal location with unknown scale and (convex) contamination neighborhoods. The definition of these estimators can be found in Subsection 8.5.3 of Kohl (2005).

Usage

rlsOptIC.An2(r, a.start = 1.5, k.start = 1.5, delta = 1e-06, MAX = 100)

Arguments

- **r**: non-negative real: neighborhood radius.
- **a.start**: positive real: starting value for a.
- **k.start**: positive real: starting value for k.
- **delta**: the desired accuracy (convergence tolerance).
- **MAX**: if a or k are beyond the admitted values, MAX is returned.

Details

The computation of the optimally robust IC for An2 estimators is based on optim where MAX is used to control the constraints on a and k. The optimal values of the tuning constants a and k can be read off from the slot Infos of the resulting IC.

Value

Object of class "IC"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

IC-class
Examples

IC1 <- rlsOptIC.AnMad(r = 0.1)
checkIC(IC1)
Risks(IC1)
Infos(IC1)
plot(IC1)
infoPlot(IC1)

**Description**

The function \texttt{rlsOptIC.AnMad} computes the optimally robust IC for AnMad estimators in case of normal location with unknown scale and (convex) contamination neighborhoods. These estimators were considered in Andrews et al. (1972). A definition of these estimators can also be found in Subsection 8.5.3 of Kohl (2005).

**Usage**

\texttt{rlsOptIC.AnMad(r, aUp = 2.5, delta = 1e-06)}

**Arguments**

- \texttt{r}: non-negative real: neighborhood radius.
- \texttt{aUp}: positive real: the upper end point of the interval to be searched for a.
- \texttt{delta}: the desired accuracy (convergence tolerance).

**Details**

The optimal value of the tuning constant a can be read off from the slot \texttt{Infos} of the resulting IC.

**Value**

Object of class "IC"

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**


See Also

IC-class

Examples

```r
IC1 <- rlsopticBM(r = 0.1)  
checkIC(IC1)  
Risks(IC1)  
Infos(IC1)  
plot(IC1)  
infoPlot(IC1)
```

---

**rlsopticBM**  
*Computation of the optimally robust IC for BM estimators*

### Description

The function `rlsopticBM` computes the optimally robust IC for BM estimators in case of normal location with unknown scale and (convex) contamination neighborhoods. These estimators were proposed by Bednarski and Mueller (2001). A definition of these estimators can also be found in Section 8.4 of Kohl (2005).

### Usage

```r
rlsopticBM(r, bl.start = 2, bs.start = 1.5, delta = 1e-06, MAX = 100)
```

### Arguments

- `r`  
  non-negative real: neighborhood radius.

- `bl.start`  
  positive real: starting value for `bloc`.

- `bs.start`  
  positive real: starting value for `bsc,0`.

- `delta`  
  the desired accuracy (convergence tolerance).

- `MAX`  
  if `bloc` or `bsc,0` are beyond the admitted values, `MAX` is returned.

### Details

The computation of the optimally robust IC for BM estimators is based on `optim` where `MAX` is used to control the constraints on `bloc` and `bsc,0`. The optimal values of the tuning constants `bloc`, `bsc,0`, `alpha` and `gamma` can be read off from the slot `Infos` of the resulting IC.

### Value

Object of class "IC"

### Author(s)

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


See Also

IC-class

Examples

IC1 <- rlsOptIC.BM(r = 0.1)
checkIC(IC1)
Risks(IC1)
Infos(IC1)
plot(IC1)
infoPlot(IC1)

rlsOptIC.Ha3

Computation of the optimally robust IC for Ha3 estimators

Description

The function rlsOptIC.Ha3 computes the optimally robust IC for Ha3 estimators in case of normal location with unknown scale and (convex) contamination neighborhoods. The definition of these estimators can be found in Subsection 8.5.2 of Kohl (2005).

Usage

rlsOptIC.Ha3(r, a.start = 0.25, b.start = 2.5, c.start = 5,
               delta = 1e-06, MAX = 100)

Arguments

r non-negative real: neighborhood radius.

a.start positive real: starting value for a.

b.start positive real: starting value for b.

c.start positive real: starting value for c.

delta the desired accuracy (convergence tolerance).

MAX if a or b or c are beyond the admitted values, MAX is returned.

Details

The computation of the optimally robust IC for Ha3 estimators is based on optim where MAX is used to control the constraints on a, b and c. The optimal values of the tuning constants a, b and c can be read off from the slot Infos of the resulting IC.
Value

Object of class "IC"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

IC-class

Examples

```r
IC1 <- rlsOptIC.Ha4(r = 0.1, a.start = 2.5, b.start = 2.5, c.start = 5, k.start = 1, delta = 1e-06, MAX = 100)
```

---

**rlsOptIC.Ha4**

Computation of the optimally robust IC for Ha4 estimators

Description

The function rlsOptIC.Ha4 computes the optimally robust IC for Ha4 estimators in case of normal location with unknown scale and (convex) contamination neighborhoods. The definition of these estimators can be found in Subsection 8.5.2 of Kohl (2005).

Usage

```r
rlsOptIC.Ha4(r, a.start = 0.25, b.start = 2.5, c.start = 5, k.start = 1, delta = 1e-06, MAX = 100)
```
Arguments

- `r` non-negative real: neighborhood radius.
- `a.start` positive real: starting value for a.
- `b.start` positive real: starting value for b.
- `c.start` positive real: starting value for c.
- `k.start` positive real: starting value for k.
- `delta` the desired accuracy (convergence tolerance).
- `MAX` if a or b or c or k are beyond the admitted values, `MAX` is returned.

Details

The computation of the optimally robust IC for Ha4 estimators is based on `optim` where `MAX` is used to control the constraints on a, b, c and k. The optimal values of the tuning constants a, b, c and k can be read off from the slot `Infos` of the resulting IC.

Value

Object of class "IC"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

- `IC-class`

Examples

```r
IC1 <- rlsOptIC.Ha4(r = 0.1)
checkIC(IC1)
Risks(IC1)
Infos(IC1)
plot(IC1)
infoPlot(IC1)
```
The function `rlsOptIC.HaMad` computes the optimally robust IC for HaMad estimators in case of normal location with unknown scale and (convex) contamination neighborhoods. These estimators were considered in Andrews et al. (1972). A definition of these estimators can also be found in Subsection 8.5.2 of Kohl (2005).

**Usage**

```r
rlsOptIC.HaMad(r, a.start = 0.25, b.start = 2.5, c.start = 5,
                  delta = 1e-06, MAX = 100)
```

**Arguments**

- `r` non-negative real: neighborhood radius.
- `a.start` positive real: starting value for a.
- `b.start` positive real: starting value for b.
- `c.start` positive real: starting value for c.
- `delta` the desired accuracy (convergence tolerance).
- `MAX` if a or b or c are beyond the admitted values, MAX is returned.

**Details**

The computation of the optimally robust IC for HaMad estimators is based on `optim` where `MAX` is used to control the constraints on a, b and c. The optimal values of the tuning constants a, b, and c can be read off from the slot `Infos` of the resulting IC.

**Value**

Object of class "IC"

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**


See Also

IC-class

Examples

IC1 <- rlsOptIC.Hu1(r = 0.1)
checkIC(IC1)
Risks(IC1)
Infos(IC1)
plot(IC1)
infoPlot(IC1)

Usage

rlsOptIC.Hu1(r, kup = 2.5, delta = 1e-06)

Arguments

r non-negative real: neighborhood radius.
kup positive real: the upper end point of the interval to be searched for k.
delta the desired accuracy (convergence tolerance).

Details

The optimal value of the tuning constant k can be read off from the slot Infos of the resulting IC.

Value

Object of class "IC"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References

sertation.
**rlsOptIC.Hu2**

*Computation of the optimally robust IC for Hu2 estimators*

**Description**

The function `rlsOptIC.Hu2` computes the optimally robust IC for Hu2 estimators in case of normal location with unknown scale and (convex) contamination neighborhoods. These estimators were proposed in Example 6.4.1 of Huber (1981). A definition of these estimators can also be found in Subsection 8.5.1 of Kohl (2005).

**Usage**

```r
c1 <- rlsOptIC.Hu2(r = 0.1) checkIC(c1) Risks(c1) Infos(c1) plot(c1) infoPlot(c1)
```

**Arguments**

- `r` : non-negative real: neighborhood radius.
- `k.start` : positive real: starting value for k.
- `c.start` : positive real: starting value for c.
- `delta` : the desired accuracy (convergence tolerance).
- `MAX` : if k1 or k2 are beyond the admitted values, MAX is returned.

**Details**

The computation of the optimally robust IC for Hu2 estimators is based on `optim` where MAX is used to control the constraints on k and c. The optimal values of the tuning constants k and c can be read off from the slot `Infos` of the resulting IC.

**Value**

Object of class "IC".

**Author(s)**

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


See Also

IC-class

Examples

```r
IC1 <- rlsOptIC.Hu2a(r = 0.1)
checkIC(IC1)
Risks(IC1)
Infos(IC1)
plot(IC1)
infoPlot(IC1)
```

---

**rlsOptIC.Hu2a**

*Computation of the optimally robust IC for Hu2a estimators*

Description

The function `rlsOptIC.Hu2a` computes the optimally robust IC for Hu2a estimators in case of normal location with unknown scale and (convex) contamination neighborhoods. These estimators are a simple modification of Huber (1964), Proposal 2 where we, in addition, admit a clipping from below. The definition of these estimators can be found in Subsection 8.5.1 of Kohl (2005).

Usage

```r
rlsOptIC.Hu2a(r, k1.start = 0.25, k2.start = 2.5, delta = 1e-06, MAX = 100)
```

Arguments

- **r**: non-negative real: neighborhood radius.
- **k1.start**: positive real: starting value for k1.
- **k2.start**: positive real: starting value for k2.
- **delta**: the desired accuracy (convergence tolerance).
- **MAX**: if k1 or k2 are beyond the admitted values, MAX is returned.

Details

The computation of the optimally robust IC for Hu2a estimators is based on `optim` where MAX is used to control the constraints on k1 and k2. The optimal values of the tuning constants k1 and k2 can be read off from the slot **Infos** of the resulting IC.
Value
Object of class "IC"

Author(s)
Matthias Kohl<Matthias.Kohl@stamats.de>

References

See Also
IC-class

Examples
IC1 <- rlsOptIC.Hu3(r = 0.1)
checkIC(IC1)
Risks(IC1)
Infos(IC1)
plot(IC1)
infoPlot(IC1)

rlsOptIC.Hu3  Computation of the optimally robust IC for Hu3 estimators

Description
The function rlsOptIC.Hu3 computes the optimally robust IC for Hu3 estimators in case of normal location with unknown scale and (convex) contamination neighborhoods. The definition of these estimators can be found in Subsection 8.5.1 of Kohl (2005).

Usage
rlsOptIC.Hu3(r, k.start = 1, c1.start = 0.1, c2.start = 0.5,
delta = 1e-06, MAX = 100)

Arguments
r non-negative real: neighborhood radius.
k.start positive real: starting value for k.
c1.start positive real: starting value for c1.
c2.start positive real: starting value for c2.
delta the desired accuracy (convergence tolerance).
MAX if k or c1 or c2 are beyond the admitted values, MAX is returned.
**Details**

The computation of the optimally robust IC for Hu2 estimators is based on \texttt{optim} where \texttt{max} is used to control the constraints on \( k, c1 \) and \( c2 \). The optimal values of the tuning constants \( k, c1 \) and \( c2 \) can be read off from the slot \texttt{Infos} of the resulting IC.

**Value**

Object of class "IC"

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**


**See Also**

\texttt{IC-class}

**Examples**

```r
IC1 <- rlsOptIC.Hu3(r = 0.1)
checkIC(IC1)
Risks(IC1)
Infos(IC1)
plot(IC1)
infoPlot(IC1)
```

---

**Description**

The function \texttt{rlsOptIC.HuMad} computes the optimally robust IC for HuMad estimators in case of normal location with unknown scale and (convex) contamination neighborhoods. These estimators were proposed by Andrews et al. (1972), p. 12. A definition of these estimators can also be found in Subsection 8.5.1 of Kohl (2005).

**Usage**

\texttt{rlsOptIC.HuMad(r, kUp = 2.5, delta = 1e-06)}
Arguments

- r: non-negative real: neighborhood radius.
- kUp: positive real: the upper end point of the interval to be searched for k.
- delta: the desired accuracy (convergence tolerance).

Details

The optimal value of the tuning constant k can be read off from the slot Infos of the resulting IC.

Value

Object of class "IC"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

IC-class

Examples

ic1 <- rlsOptIC.Mad(r = 0.1)
checkIC(ic1)
Risks(ic1)
Infos(ic1)
plot(ic1)
infoPlot(ic1)

Computation of the optimally robust IC for M estimators

Description

The function rlsOptIC.M computes the optimally robust IC for M estimators in case of normal location with unknown scale and (convex) contamination neighborhoods. The definition of these estimators can be found in Section 8.3 of Kohl (2005).
Usage

```r
rlsOptIC.M(r, ggLo = 0.5, ggUp = 1.5, a1.start = 0.75, a3.start = 0.25,
brUp = 1000, delta = 1e-05, itmax = 100, check = FALSE)
```

Arguments

- `r` non-negative real: neighborhood radius.
- `ggLo` non-negative real: the lower end point of the interval to be searched for $\gamma$.
- `ggUp` positive real: the upper end point of the interval to be searched for $\gamma$.
- `a1.start` real: starting value for $\alpha_1$.
- `a3.start` real: starting value for $\alpha_3$.
- `bUp` positive real: upper bound used in the computation of the optimal clipping bound $b$.
- `delta` the desired accuracy (convergence tolerance).
- `itmax` the maximum number of iterations.
- `check` logical. Should constraints be checked.

Details

The optimal values of the tuning constants $\alpha_1, \alpha_3, b$ and $\gamma$ can be read off from the slot `Infos` of the resulting IC.

Value

Object of class "IC"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

- `IC-class`
Computation of the optimally robust IC for MM2 estimators

Description
The function `rlsOptIC.MM2` computes the optimally robust IC for MM2 estimators in case of normal location with unknown scale and (convex) contamination neighborhoods. These estimators are based on a proposal of Fraiman et al. (2001), p. 206. A definition of these estimators can also be found in Section 8.6 of Kohl (2005).

Usage
`rlsOptIC.MM2(r, c.start = 1.5, d.start = 2, delta = 1e-06, MAX = 100)`

Arguments
- `r`: non-negative real: neighborhood radius.
- `c.start`: positive real: starting value for c.
- `d.start`: positive real: starting value for d.
- `delta`: the desired accuracy (convergence tolerance).
- `MAX`: if a or k are beyond the admitted values, MAX is returned.

Details
The computation of the optimally robust IC for MM2 estimators is based on `optim` where MAX is used to control the constraints on c and d. The optimal values of the tuning constants c and d can be read off from the slot `Infos` of the resulting IC.

Value
Object of class "IC"

Author(s)
Matthias Kohl <Matthias.Kohl@stamats.de>
References

See Also
IC-class

Examples
IC1 <- rlsOptIC.MM2(r = 0.1)
checkIC(IC1)
Risks(IC1)
Infos(IC1)
plot(IC1)
infoPlot(IC1)

rlsOptIC.Tu1

Computation of the optimally robust IC for Tu1 estimators

Description
The function rlsOptIC.Tu1 computes the optimally robust IC for Tu1 estimators in case of normal location with unknown scale and (convex) contamination neighborhoods. The definition of these estimators can be found in Subsection 8.5.4 of Kohl (2005).

Usage
rlsOptIC.Tu1(r, aUp = 10, delta = 1e-06)

Arguments
r non-negative real: neighborhood radius.
aUp positive real: the upper end point of the interval to be searched for a.
delta the desired accuracy (convergence tolerance).

Details
The optimal value of the tuning constant a can be read off from the slot Infos of the resulting IC.

Value
Object of class "IC"
Author(s)
Matthias Kohl <Matthias.Kohl@stamats.de>

References

See Also
IC-class

Examples
rlsoptic.Tu2(r = 0.1)
checkIC(IC1)
Risks(IC1)
Infos(IC1)
plot(IC1)
infoPlot(IC1)

Description
The function rlsOptIC.Tu2 computes the optimally robust IC for Tu2 estimators in case of normal location with unknown scale and (convex) contamination neighborhoods. The definition of these estimators can be found in Subsection 8.5.4 of Kohl (2005).

Usage
rlsOptIC.Tu2(r, a.start = 5, k.start = 1.5, delta = 1e-06, MAX = 100)

Arguments
r          non-negative real: neighborhood radius.
a.start    positive real: starting value for a.
k.start    positive real: starting value for k.
delta      the desired accuracy (convergence tolerance).
MAX        if a or k are beyond the admitted values, MAX is returned.
Details
The computation of the optimally robust IC for Tu2 estimators is based on optim where \texttt{max} is used to control the constraints on a and k. The optimal values of the tuning constant a and k can be read off from the slot \texttt{Infos} of the resulting IC.

Value
Object of class "IC"

Author(s)
Matthias Kohl <Matthias.Kohl@stamats.de>

References

See Also
IC-class

Examples

```r
IC1 <- rlsOptIC.Tu2(r = 0.1)
checkIC(IC1)
Risks(IC1)
Infos(IC1)
plot(IC1)
infoPlot(IC1)
```

---

\textbf{rlsOptIC.TuMad}

Computation of the optimally robust IC for TuMad estimators

Description
The function \texttt{rlsOptIC.TuMad} computes the optimally robust IC for TuMad estimators in case of normal location with unknown scale and (convex) contamination neighborhoods. The definition of these estimators can be found in Subsection 8.5.4 of Kohl (2005).

Usage

```r
rlsOptIC.TuMad(r, aUp = 10, delta = 1e-06)
```
**Arguments**

- `r`  
  non-negative real: neighborhood radius.

- `aUp`  
  positive real: the upper end point of the interval to be searched for a.

- `delta`  
  the desired accuracy (convergence tolerance).

**Details**

The optimal value of the tuning constant a can be read off from the slot `Infos` of the resulting IC.

**Value**

Object of class "IC"

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**


**See Also**

- `IC-class`

**Examples**

```r
IC1 <- rlsOptIC.TuMad(r = 0.1)  
checkIC(IC1)  
Risks(IC1)  
Infos(IC1)  
plot(IC1)  
infoPlot(IC1)
```

**roblox**  
*Optimally robust estimator for location and/or scale*

**Description**

The function `roblox` computes the optimally robust estimator and corresponding IC for normal location und/or scale and (convex) contamination neighborhoods. The definition of these estimators can be found in Rieder (1994) or Kohl (2005), respectively.
Usage

roblox(x, mean, sd, eps, eps.lower, eps.upper, initial.est, k = 1L,
   fsCor = TRUE, returnIC = FALSE, mad0 = 1e-4, na.rm = TRUE)

Arguments

x vector x of data values, may also be a matrix or data.frame with one row, respectively one column/(numeric) variable.
mean specified mean.
sd specified standard deviation which has to be positive.
eps positive real (0 < eps <= 0.5): amount of gross errors. See details below.
eps.lower positive real (0 <= eps.lower <= eps.upper): lower bound for the amount of gross errors. See details below.
eps.upper positive real (eps.lower <= eps.upper <= 0.5): upper bound for the amount of gross errors. See details below.
initial.est initial estimate for mean and/or sd. If missing median and/or MAD are used.
k positive integer. k-step is used to compute the optimally robust estimator.
fsCor logical: perform finite-sample correction. See function finiteSampleCorrection.
returnIC logical: should IC be returned. See details below.
mad0 scale estimate used if computed MAD is equal to zero
na.rm logical: if TRUE, the estimator is evaluated at complete.cases(x).

Details

Computes the optimally robust estimator for location with scale specified, scale with location specified, or both if neither is specified. The computation uses a k-step construction with an appropriate initial estimate for location or scale or location and scale, respectively. Valid candidates are e.g. median and/or MAD (default) as well as Kolmogorov(-Smirnov) or von Mises minimum distance estimators; 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. (2008), respectively Section 2.2 of Kohl (2005) is returned.

In case of location, respectively scale one additionally has to specify sd, respectively mean where sd and mean have to be a single number.

For sample size <= 2, median and/or MAD are used for estimation.

If eps = 0, mean and/or sd are computed. In this situation it’s better to use function MLEstimator.
Value

Object of class "kStepEstimate".

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

ContIC-class, rlOptIC, rsOptIC, rlsOptIC, AL, kStepEstimate-class, roptest

Examples

ind <- rbinom(100, size=1, prob=0.05)
x <- rnorm(100, mean=ind*3, sd=(1-ind) + ind*9)

## amount of gross errors known
res1 <- roblox(x, eps = 0.05, returnIC = TRUE)
estimate(res1)
## don't run to reduce check time on CRAN
## Not run:
confintr(res1)
confintr(res1, method = symmetricBias())
pIC(res1)
checkIC(pIC(res1))
Risks(pIC(res1))
Infos(pIC(res1))
plot(pIC(res1))
infoPlot(pIC(res1))

## End(Not run)

## amount of gross errors unknown
res2 <- roblox(x, eps.lower = 0.01, eps.upper = 0.1, returnIC = TRUE)
estimate(res2)
## don't run to reduce check time on CRAN
## Not run:
confintr(res2)
confintr(res2, method = symmetricBias())
pIC(res2)
Optimally robust estimation for location and/or scale

Description

The functions `rowroblox` and `colroblox` compute optimally robust estimates for normal location and/or scale and (convex) contamination neighborhoods. The definition of these estimators can be found in Rieder (1994) or Kohl (2005), respectively.

Usage

```r
rowroblox(x, mean, sd, eps, eps.lower, eps.upper, initial.est, k = 1L,
           fsCor = TRUE, mad0 = 1e-4, na.rm = TRUE)
colroblox(x, mean, sd, eps, eps.lower, eps.upper, initial.est, k = 1L,
           fsCor = TRUE, mad0 = 1e-4, na.rm = TRUE)
```
Arguments

- **x**: matrix or data.frame of (numeric) data values.
- **mean**: specified mean. See details below.
- **sd**: specified standard deviation which has to be positive. See also details below.
- **eps**: positive real (0 < eps <= 0.5): amount of gross errors. See details below.
- **eps.lower**: positive real (0 <= eps.lower <= eps.upper): lower bound for the amount of gross errors. See details below.
- **eps.upper**: positive real (eps.lower <= eps.upper <= 0.5): upper bound for the amount of gross errors. See details below.
- **initial.est**: initial estimate for mean and/or sd. If missing median and/or MAD are used.
- **k**: positive integer. k-step is used to compute the optimally robust estimator.
- **fsCor**: logical: perform finite-sample correction. See function `finiteSampleCorrection`.
- **mad0**: scale estimate used if computed MAD is equal to zero
- **na.rm**: logical: if TRUE, the estimator is evaluated at complete cases(x).

Details

Computes the optimally robust estimator for location with scale specified, scale with location specified, or both if neither is specified. The computation uses a k-step construction with an appropriate initial estimate for location or scale or location and scale, respectively. Valid candidates are e.g. median and/or MAD (default) as well as Kolmogorov(-Smirnov) or Cram\'er von Mises minimum distance estimators; cf. Rieder (1994) and Kohl (2005). In case package Biobase from Bioconductor is installed as is suggested, median and/or MAD are computed using function `rowmedians`.

These functions are optimized for the situation where one has a matrix and wants to compute the optimally robust estimator for every row, respectively column of this matrix. In particular, the amount of cross errors is assumed to be constant for all rows, respectively columns.

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. (2008), respectively Section 2.2 of Kohl (2005) is returned.

In case of location, respectively scale one additionally has to specify sd, respectively mean where sd and mean can be a single number, i.e., identical for all rows, respectively columns, or a vector with length identical to the number of rows, respectively columns.

For sample size <= 2, median and/or MAD are used for estimation.

If eps = 0, mean and/or sd are computed.
Value

Object of class "kStepEstimate".

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

roblox, kStepEstimate-class

Examples

```r
ind <- rbinom(200, size=1, prob=0.05)
X <- matrix(rnorm(200, mean=ind*3, sd=(1-ind) + ind*9), nrow = 2)
rowRoblox(X)
rowRoblox(X, k = 3)
rowRoblox(X, eps = 0.05)
rowRoblox(X, eps = 0.05, k = 3)

X1 <- t(X)
colRoblox(X1)
colRoblox(X1, k = 3)
colRoblox(X1, eps = 0.05)
colRoblox(X1, eps = 0.05, k = 3)

X2 <- rbind(rnorm(100, mean = -2, sd = 3), rnorm(100, mean = -1, sd = 4))
rowRoblox(X2, sd = c(3, 4))
rowRoblox(X2, eps = 0.03, sd = c(3, 4))
rowRoblox(X2, sd = c(3, 4), k = 4)
rowRoblox(X2, eps = 0.03, sd = c(3, 4), k = 4)

X3 <- cbind(rnorm(100, mean = -2, sd = 3), rnorm(100, mean = 1, sd = 2))
colRoblox(X3, mean = c(-2, 1))
colRoblox(X3, eps = 0.02, mean = c(-2, 1))
colRoblox(X3, mean = c(-2, 1), k = 4)
colRoblox(X3, eps = 0.02, mean = c(-2, 1), k = 4)
```
rsOptIC

Computation of the optimally robust IC for AL estimators

Description

The function rsOptIC computes the optimally robust IC for AL estimators in case of normal scale and (convex) contamination neighborhoods. The definition of these estimators can be found in Rieder (1994) or Kohl (2005), respectively.

Usage

rsOptIC(r, mean = 0, sd = 1, bUp = 1000, delta = 1e-06, itmax = 100, computeIC = TRUE)

Arguments

r  
non-negative real: neighborhood radius.
mean  
specified mean.
sd  
specified standard deviation.
bUp  
positive real: the upper end point of the interval to be searched for the clipping bound b.
delta  
the desired accuracy (convergence tolerance).
itmax  
the maximum number of iterations.
computeIC  
logical: should IC be computed. See details below.

Details

If 'computeIC' is 'FALSE' only the Lagrange multipliers 'A', 'a', and 'b' contained in the optimally robust IC are computed.

Value

If 'computeIC' is 'TRUE' an object of class "ContIC" is returned, otherwise a list of Lagrange multipliers

- A  standardizing constant
- a  centering constant
- b  optimal clipping bound

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References

showdown

See Also

ContIC-class, roblox

Examples

```r
IC1 <- rsOptIC(r = 0.1)
distrExOptions("ErelativeTolerance" = 1e-12)
checkIC(IC1)
distrExOptions("ErelativeTolerance" = .Machine$double.eps^0.25) # default
Risks(IC1)
cent(IC1)
clip(IC1)
stand(IC1)
plot(IC1)
```

showdown  

Estimator Showdown by Monte-Carlo Study.

Description

The function `showdown` can be used to perform Monte-Carlo studies comparing a competitor with rmx estimators in case of normal location and scale. In addition, maximum likelihood (ML) estimators (mean and sd) and median and MAD are computed. The comparison is based on the empirical MSE.

Usage

```r
showdown(n, M, eps, contD, seed = 123, estfun, estMean, estsd,
    eps.lower = 0, eps.upper = 0.05, steps = 3L, fsCor = TRUE,
    plot1 = FALSE, plot2 = FALSE, plot3 = FALSE)
```

Arguments

- `n` integer; sample size, should be at least 3.
- `M` integer; Monte-Carlo replications.
- `eps` amount of contamination in \([0, 0.5]\).
- `contD` object of class "UnivariateDistribution"; contaminating distribution.
- `seed` random seed.
- `estfun` function to compute location and scale estimator; see details below.
- `estMean` function to compute location estimator; see details below.
- `estSd` function to compute scale estimator; see details below.
- `eps.lower` used by rmx estimator.
- `eps.upper` used by rmx estimator.
- `steps` integer; steps used for estimator construction.
fsCor logical; use finite-sample correction.
plot1 logical; plot cdf of ideal and real distribution.
plot2 logical; plot 20 (or M if M < 20) randomly selected samples.
plot3 logical; generate boxplots of the results.

Details

Normal location and scale with mean = 0 and sd = 1 is used as ideal model (without restriction due to equivariance).

Since there is no estimator which yields reliable results if 50 percent or more of the observations are contaminated, we use a modification where we re-simulate all samples including at least 50 percent contaminated data.

If estfun is specified it has to compute and return a location and scale estimate (vector of length 2). One can also specify the location and scale estimator separately by using estMean and estSd where estMean computes and returns the location estimate and estSd the scale estimate.

We use function rowRoblox for the computation of the rmx estimator.

Value

Data.frame including empirical MSE (standardized by sample size n) and relMSE with respect to the rmx estimator.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

rowRoblox

Examples

library(MASS)
## compare with Huber's Proposal 2
showdown(n = 20, M = 100, eps = 0.02, contD = Norm(mean = 3, sd = 3),
estfun = function(x){ unlist(hubers(x)) },
plot1 = TRUE, plot2 = TRUE, plot3 = TRUE)
## compare with Huber M estimator with MAD scale

```r
showdown(n = 20, M = 100, eps = 0.02, contD = Norm(mean = 3, sd = 3),
  estfun = function(x)( unlist(huber(x)) ),
  plot1 = TRUE, plot2 = TRUE, plot3 = TRUE)
```
Index

*Topic package
  RobLox-package, 2
*Topic robust
  finiteSampleCorrection, 4
  r1OptIC, 5
  rlsOptIC.AL, 6
  rlsOptIC.An1, 8
  rlsOptIC.An2, 10
  rlsOptIC.AnMad, 11
  rlsOptIC.BM, 12
  rlsOptIC.Ha3, 13
  rlsOptIC.Ha4, 14
  rlsOptIC.HaMad, 16
  rlsOptIC.Hu1, 17
  rlsOptIC.Hu2, 18
  rlsOptIC.Hu2a, 19
  rlsOptIC.Hu3, 20
  rlsOptIC.HuMad, 21
  rlsOptIC.M, 22
  rlsOptIC.MM2, 24
  rlsOptIC.Tu1, 25
  rlsOptIC.Tu2, 26
  rlsOptIC.TuMad, 27
  roblox, 28
  rowRoblox and colRoblox, 31
  rsOptIC, 34
  showdown, 35
  rlsOptIC.BM, 12
  rlsOptIC.Ha3, 13
  rlsOptIC.Ha4, 14
  rlsOptIC.HaMad, 16
  rlsOptIC.Hu1, 17
  rlsOptIC.Hu2, 18
  rlsOptIC.Hu2a, 19
  rlsOptIC.Hu3, 20
  rlsOptIC.HuMad, 21
  rlsOptIC.M, 22
  rlsOptIC.MM2, 24
  rlsOptIC.Tu1, 25
  rlsOptIC.Tu2, 26
  rlsOptIC.TuMad, 27
  RobLox (RobLox-package), 2
  roblox, 5–7, 28, 33, 35
  RobLox-package, 2
  rOptest, 30
  rowRoblox, 5, 36
  rowRoblox (rowRoblox and colRoblox), 31
  rowRoblox and colRoblox, 31
  rsOptIC, 30, 34
  showdown, 35

colRoblox, 5
colRoblox (rowRoblox and colRoblox), 31

finiteSampleCorrection, 4, 29, 32

MLEestimator, 29

r1OptIC, 5, 30
rlsOptIC.AL, 6, 30
rlsOptIC.An1, 8
rlsOptIC.An2, 10
rlsOptIC.AnMad, 11