Package ‘distrMod’

February 19, 2015

Version 2.5.3
Date 2014-11-25
Title Object Oriented Implementation of Probability Models
Description Object oriented implementation of probability models based on packages 'distr' and 'distrEx'
Author Matthias Kohl, Peter Ruckdeschel
Maintainer Peter Ruckdeschel <Peter.Ruckdeschel@itwm.fraunhofer.de>
Depends R(>= 2.14.0), distr(>= 2.5.2), distrEx(>= 2.4), RandVar(>= 0.6.3), MASS, stats4, methods
Imports sfsmisc, startupmsg
ByteCompile yes
License LGPL-3
Encoding latin1
URL http://distr.r-forge.r-project.org/

R topics documented:

distrMod-package .......................................................... 4
addAlphTrsp2col ...................................................... 10
asBias ................................................................. 11
asBias-class .......................................................... 12
asCov ................................................................. 13
asCov-class .......................................................... 13
asGRisk-class ................................................. 14
asHampel ..................................................... 15
asHampel-class .............................................. 16
asMSE ............................................................. 17
asMSE-class ................................................... 18
asRisk-class ................................................... 19
asRiskwithBias-class ...................................... 20
asSemivar ...................................................... 21
asSemivar-class .............................................. 22
asUnOvShoot .................................................. 23
asUnOvShoot-class ......................................... 24
asymmetricBias .............................................. 25
asymmetricBias-class ...................................... 26
BetaFamily ...................................................... 27
BiasType-class ................................................ 28
BinomFamily .................................................... 29
CauchyLocationScaleFamily ............................... 30
checkL2deriv .................................................. 31
Confint-class .................................................. 32
Confint-methods ............................................. 34
distrModMASK ................................................ 36
distrModOptions ............................................. 37
Estimate-class ................................................ 38
Estimator ...................................................... 41
EvenSymmetric ............................................. 42
EvenSymmetric-class ..................................... 43
existsPIC-methods ........................................ 44
ExpScaleFamily ............................................. 45
fiBias ............................................................ 46
fiBias-class .................................................... 46
fiCov ............................................................ 47
fiCov-class ..................................................... 48
fiHampel ....................................................... 49
fiHampel-class .............................................. 50
fiMSE ............................................................ 51
fiMSE-class ................................................... 51
fiRisk-class ................................................... 52
fiUnOvShoot .................................................. 53
fiUnOvShoot-class ......................................... 54
FunctionSymmetry-class ................................ 55
FunSymmList ................................................ 56
FunSymmList-class ....................................... 57
GammaFamily .................................................. 57
InfoNorm ...................................................... 58
isKerAinKerB ................................................ 59
L2GroupParamFamily-class ............................ 60
L2LocationFamily .......................................... 62
L2LocationFamily-class .................................. 64
## R topics documented:

- `L2LocationScaleFamily` .............................................. 66
- `L2LocationScaleFamily-class` ..................................... 67
- `L2LocationUnknownScaleFamily` ................................... 69
- `L2ParamFamily` .......................................................... 71
- `L2ParamFamily-class` .................................................. 73
- `L2ScaleFamily` ............................................................ 76
- `L2ScaleFamily-class` .................................................... 78
- `L2ScaleUnknownLocationFamily` .................................... 80
- `LnormScaleFamily` ....................................................... 82
- `mceCalc-methods` ....................................................... 83
- `MCEstimate-class` ....................................................... 85
- `MDEstimator` ............................................................. 87
- `meRes` .................................................................. 91
- `MLEstimator` .............................................................. 93
- `modifyModel-methods` ................................................... 96
- `NbinomFamily` ............................................................. 97
- `negativeBias` ............................................................... 99
- `NonSymmetric` ............................................................ 100
- `NonSymmetric-class` .................................................... 100
- `norm` ................................................................ 101
- `NormLocationFamily` .................................................... 102
- `NormLocationScaleFamily` ............................................. 103
- `NormLocationUnknownScaleFamily` ................................ 104
- `NormScaleFamily` .......................................................... 105
- `NormScaleUnknownLocationFamily` ................................ 106
- `NormType` ................................................................ 107
- `NormType-class` ........................................................... 108
- `OddSymmetric` ............................................................. 109
- `OddSymmetric-class` ..................................................... 109
- `onesidedBias-class` ...................................................... 110
- `ParamFamily` ............................................................... 111
- `ParamFamily-class` ....................................................... 115
- `ParamFamParameter` ...................................................... 117
- `ParamFamParameter-class` ............................................. 118
- `PoisFamily` ................................................................. 120
- `positiveBias` ............................................................... 121
- `print-methods` .............................................................. 122
- `ProbFamily-class` ........................................................ 123
- `QFNorm` ................................................................ 124
- `QFNorm-class` ............................................................. 125
- `qqplot` .................................................................. 126
- `RiskType-class` ............................................................ 129
- `SelfNorm` ..................................................................... 130
- `symmetricBias` ............................................................. 130
- `symmetricBias-class` .................................................... 131
- `trafo-methods` ............................................................. 132
- `trafoEst` ...................................................................... 135
distrMod – probability models

Description

Based on the packages distr and distrEx package distrMod provides a flexible framework which allows computation of estimators like maximum likelihood or minimum distance estimators for probability models.

Details

Package: distrMod
Version: 2.5
Date: 2013-09-11
Depends: R(>= 2.6.0), methods, startupmsg, distr(>= 2.2), distrEx(>= 2.2), RandVar(>= 0.6.3), MASS, stats4
LazyLoad: yes
License: LGPL-3
URL: http://distr.r-forge.r-project.org/
SVNRevision: 904

Classes

[*]: there is a generating function with the same name

# ProbFamily classes
# slots: [<name>(<class>)]
name(character), distribution(Distribution),
distrSym(DistributionSymmetry), props(character)

"ProbFamily"
|>"ParamFamily" [*]
additional slots:
param(ParamFamParameter), modifyParam(function),
startPar(function), makeOKPar(function), fam.call(call)
distrMod-package

| |>|"L2ParamFamily" [*]
additional slots:
L2deriv(Euc1RandVarList), L2deriv.fct(function),
L2derivSymm(FunSymmList), L2derivDistr(DistrList),
L2derivDistrSymm(DistrSymmList), FisherInfo(PosSemDefSymmMatrix),
FisherInfo.fct(function)
| |>| |>|"BinomFamily" [*]
| |>| |>|"PoisFamily" [*]
| |>| |>|"BetaFamily" [*]
| |>| |>|"L2GroupParamFamily"
additional slots:
LogDeriv(function)
| |>| |>| |>|"L2ScaleShapeUnion" /VIRTUAL/
| |>| |>| |>|"GammaFamily" [*]
| |>| |>| |>|"L2LocationScaleUnion" /VIRTUAL/
additional slots:
locscalename(character)
| |>| |>| |>| |>|"L2LocationFamily" [*]
| |>| |>| |>| |>|"NormLocationFamily" [*]
| |>| |>| |>| |>|"L2ScaleFamily" [*]
| |>| |>| |>| |>|"NormScaleFamily" [*]
| |>| |>| |>| |>|"ExpScaleFamily" [*]
| |>| |>| |>| |>|"LnormScaleFamily" [*]
| |>| |>| |>| |>|"L2LocationScaleFamily" [*]
| |>| |>| |>| |>|"NormLocationScaleFamily" [*]
| |>| |>| |>| |>|"CauchyLocationScaleFamily" [*]

and a (virtual) class union "L2ScaleUnion" between
"L2LocationScaleUnion" and "L2ScaleShapeUnion"

# ParamFamParameter

"ParamFamParameter" [*] is subclass of class "Parameter" of package "distr".
Additional slots:
main(numeric), nuisance(OptionalNumeric), fixed(OptionalNumeric),
trafo(MatrixOrFunction)

Class unions

"MatrixOrFunction" = union("matrix", "OptionalFunction")
"PrintDetails" = union("Estimate", "Confint",
    "PosSemDefSymmMatrix",
    "ParamFamParameter", "ParamFamily")
Symmetry classes (other classes moved to package "distr")

slots:
type(character), SymmCenter(ANY)

"Symmetry" (from package "distr")
|>"FunctionSymmetry"
|>"NonSymmetric" [*]
|>"EvenSymmetric" [*]
|>"OddSymmetric" [*]

list thereof
"FunSymmList" [*]

Matrix classes (moved to package "distr")

slots:
none

"PosSemDefSymmMatrix" [*] is subclass of class "matrix" of package "base".
|>"PosDefSymmMatrix" [*]

Norm Classes

slots:
name(character), fct(function)

"NormType" [ *]
|>"QFNorm" [ *]

Additional slots:
QuadForm(PosSemDefSymmMatrix)
|>"InfoNorm" [ *]
|>"SelfNorm" [ *]

Bias Classes

slots:
name(character)

"BiasType"
|>"symmetricBias" [ *]
|>"onesidedBias"

Additional slots:
sign(numeric)
|"asymmetricBias" [*
Additional slots:
nu(numeric)

Risk Classes

slots:
type(character)

"RiskType"
|"asRisk"
|>|"asCov" [*
|>|"trAsCov" [*
|>"fiRisk"
|>|"fiCov" [*
|>|"trfiCov" [*
|>|"fiHampel" [*
Additional slots:
bound(numeric)
|>|"fiMSE" [*
|>|"fiBias" [*
|>|"fiUnOvShoot" [*
Additional slots:
width(numeric)

Risk with Bias:
"asRiskwithBias"
slots: biastype(BiasType), normtype(NormType),
|>"asHampel" [*
Additional slots:
bound(numeric)
|>"asBias" [*
|>"asGRisk"
|>|"asMSE" [*
|>|"asUnOvShoot" [*
Additional slots:
width(numeric)
|>|"asSemivar" [*

Estimate Classes

slots:
name(character), estimate(ANY),
samplesize(numeric), asvar(OptionalMatrix), Info(matrix), nui.idx(OptionalNumeric)
fixed.estimate(OptionalNumeric), estimate.call(call), trafo(list[of function, matrix]),
untransformed.estimate(ANY), untransformed.asvar(OptionalMatrix)
criterion.fct(function), method(character),

"Estimate"
|"MCEstimate",
Additional slots:
criterion(numeric)

Confidence interval class

slots:
type(character), confint(array),
estimate.call(call), name.estimate(character),
trafo.estimate(list[of function, matrix]),
nuisance.estimate(OptionalNumeric)
"Confint"

Methods

besides accessor and replacement functions, we have methods

solve, sqrt for matrices checkL2deriv, existsPIC for class L2ParamFamily LogDeriv for class L2GroupParamFamily
validParameter for classes ParamFamily, L2ScaleFamily, L2LocationFamily, and L2LocationScaleFamily
modifyModel for the pairs of classes L2ParamFamily and ParamFamParameter, L2LocationFamily and ParamFamParameter, L2ScaleFamily and ParamFamParameter, L2LocationScaleFamily and ParamFamParameter, GammaFamily and ParamFamParameter, and ExpScaleFamily and ParamFamParameter
mceCalc for the pair of classes numeric and ParamFamily
mleCalc for the pairs of classes numeric and ParamFamily, numeric and BinomFamily, numeric and PoisFamily, numeric and NormLocationFamily, numeric and NormScaleFamily, and numeric and NormLocationScaleFamily
coerce from class MCEstimate to class mle
confint for class Estimate profile for class MCEstimate

Functions

Management of global options:
"distrModOptions", "distrModoptions", "getDistrModOption",
check for ker of matrix: "isKerAinKerB"
particular norms: "EuclideanNorm", "QuadFormNorm"
onesided bias: "positiveBias", "negativeBias",

Estimators:
"Estimator", "MCEstimator", "MLEstimator", "MDEstimator"
special location/scale models:
"L2LocationUnknownScaleFamily", "L2ScaleUnknownLocationFamily"
some special normal models:
"NormScaleUnknownLocationFamily", "NormLocationUnknownScaleFamily",

Start-up-Banner
You may suppress the start-up banner/message completely by setting options("StartupBanner"="off") somewhere before loading this package by library or require in your R-code / R-session.
If option "StartupBanner" is not defined (default) or setting options("StartupBanner"=NULL) or options("StartupBanner"="complete") the complete start-up banner is displayed.
For any other value of option "StartupBanner" (i.e., not in c(NULL,"off","complete")) only the version information is displayed.
The same can be achieved by wrapping the library or require call into either suppressStartupMessages() or onlytypeStartupMessages(. ,atypes="version").
As for general packageStartupMessage's, you may also suppress all the start-up banner by wrapping the library or require call into suppressPackageStartupMessages() from startupmsg-
version 0.5 on.

Demos
Demos are available — see demo(package="distrMod").

Scripts
Example scripts are available — see folder ‘scripts’ in the package folder to package distrMod in your library.

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

Note
Some functions of packages stats, base have intentionally been masked, but completely retain their functionality — see distrModMASK().
If any of the packages stats4, fBasics is to be used together with distrMod, the latter must be attached after any of the first mentioned. Otherwise confint() defined as method in distrMod
may get masked. To re-mask, you may use `confint <- distrMod::confint`. See also `distrMod::mask()`

**Author(s)**

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

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

**References**


A vignette for packages `distr`, `distrSim`, `distrTEst`, and `distrEx` is included into the mere documentation package `distrDoc` and may be called by `require("distrDoc");vignette("distr")`

---

**addAlphTrsp2col**

### addAlphTrsp2col

**Description**

Adds alpha transparency to a given color.

**Usage**

`addAlphTrsp2col(col, alpha=255)`

**Arguments**

- `col` any valid color
- `alpha` transparency; an integer value in [0,255]

**Value**

a color in rgb coordinates

**Author(s)**

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

Examples

```r
addAlphaTrsp2col(rgb(c(1,0.3,0.3), 25)
addAlphaTrsp2col("darkblue", 25)
addAlphaTrsp2col("#AAAAAA", 25)
palette(rainbow(6))
addAlphaTrsp2col(2, 25)
```

---

asBias  Generating function for asBias-class

Description

Generates an object of class "asBias".

Usage

```r
asBias(biastype = symmetricBias(), normtype = NormType())
```

Arguments

- `biastype`: a bias type of class BiasType
- `normtype`: a norm type of class NormType

Value

Object of class "asBias"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

- `asBias-class`

Examples

```r
asBias()
```

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

Class of standardized asymptotic bias; i.e., the neighborhood radius is omitted respectively, set to 1.

Objects from the Class

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

Slots

- `type`: Object of class "character": “asymptotic bias”.
- `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 "asRiskWithBias", directly.
Class "asRisk", by class "asRiskWithBias"
Class "RiskType", by class "asRisk".

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

`asRisk-class, asBias`

Examples

`new("asBias")`
asCov

Generating function for asCov-class

Description
Generates an object of class "asCov".

Usage
asCov()

Value
Object of class "asCov"

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

References

See Also
asCov-class

Examples
ascov()

## The function is currently defined as
function(){ new("asCov") }

asCov-class Asymptotic covariance

Description
Class of asymptotic covariance.

Objects from the Class
Objects can be created by calls of the form new("asCov", ...). More frequently they are created via the generating function asCov.
Slots
  type Object of class "character": “asymptotic covariance”.

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

Methods
  No methods defined with class "asCov" in the signature.

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

References

See Also
  asRisk-class, asCov

Examples
  new("asCov")

asGRisk-class  Convex asymptotic risk

Description
  Class of special convex asymptotic risks.

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

Slots
  type  Object of class "character".
  biastype  Object of class "BiasType": symmetric, one-sided or asymmetric
  normtype  Object of class "NormType": norm in which a multivariate parameter is considered
asHampel

**Extend**

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

**Methods**

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

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**


**See Also**

asRisk-class

---

**asHampel**

*Generating function for asHampel-class*

**Description**

Generates an object of class "asHampel".

**Usage**

`asHampel(bound = Inf, biastype = symmetricBias(), normtype = NormType())`

**Arguments**

bound  positive real: bias bound
biastype  a bias type of class BiasType
normtype  a norm type of class NormType

**Value**

Object of class asHampel

**Author(s)**

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

See Also
asHampel-class

Examples
asHampel()

## The function is currently defined as
function(bound = Inf, biastype = symmetricBias(), normtype = NormType()){
  new("asHampel", bound = bound, biastype = biastype, normtype = normtype) }

asHampel-class  Asymptotic Hampel risk

Description
Class of asymptotic Hampel risk which is the trace of the asymptotic covariance subject to a given bias bound (bound on gross error sensitivity).

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

Slots
type Object of class "character": “trace of asymptotic covariance for given bias bound”.
bound Object of class "numeric": given positive bias bound.
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
bound signature(object = "asHampel"): accessor function for slot bound.
show signature(object = "asHampel")
asMSE

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

References

See Also
asRisk-class, asHampel

Examples
new("asHampel")

asMSE Generating function for asMSE-class

Description
Generates an object of class "asMSE".

Usage
asMSE(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)
Matthias Kohl <Matthias.Kohl@stamats.de>

References
See Also

asMSE-class

Examples

asMSE()

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

asMSE-class Asymptotic mean square error

Description

Class of asymptotic mean square error.

Objects from the Class

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

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 "asMSE" in the signature.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>
asRisk-class

References


See Also

asGRisk-class, asMSE

Examples

new("asMSE")

<table>
<thead>
<tr>
<th>asRisk-class</th>
<th>Asymptotic risk</th>
</tr>
</thead>
</table>

Description

Class of asymptotic risks.

Objects from the Class

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

Slots

type Object of class "character".

Extends

Class "RiskType", directly.

Methods

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

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References

See Also

RISK-RISK

asRiskwithBias-class

Aymptotic risk

Description

Class of asymptotic risks.

Objects from the Class

A “virtual” Class (although it does not contain ”VIRTUAL”): No objects may be created from it.

Slots

type Object of class "character".
biastype Object of class "BiasType".
normtype Object of class "NormType".

Extends

Class "RiskType", directly.

Methods

biastype signature(object = "asRiskwithBias"): accessor function for slot biastype.
biastype<- signature(object = "asRiskwithBias", value = "BiasType"): replacement function for slot biastype.
normtype signature(object = "asRiskwithBias"): accessor function for slot normtype.
normtype<- signature(object = "asRiskwithBias", value = "NormType"): replacement function for slot normtype.
norm signature(object = "asRiskwithBias"): accessor function for slot fct of slot norm.

Author(s)

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

References

asSemivar

See Also

asRisk-class

asSemivar Generating function for asSemivar-class

Description

Generates an object of class "asSemivar".

Usage

asSemivar(sign = 1)

Arguments

sign positive (=1) or negative Bias (= -1)

Value

Object of class "asSemivar"

Author(s)

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

References


See Also

onesidedBias-class

Examples

asSemivar()
asSemivar-class  

Semivariance Risk Type

Description

Class for semi-variance risk.

Objects from the Class

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

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

Methods

sign  signature(object = "asSemivar"): accessor function for slot sign.
sign<-  signature(object = "asSemivar", value = "numeric"): replacement function for slot sign.

Extends

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

Author(s)

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

References


See Also

asGRisk-class, asMSE
Description
Generates an object of class "asUnOvShoot".

Usage
asUnOvShoot(width = 1.960, biastype = symmetricBias())

Arguments
width positive real: half the width of given confidence interval.
biastype a bias type of class BiasType

Value
Object of class "asUnOvShoot"

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

References

See Also
asUnOvShoot-class

Examples
asUnOvShoot()

## The function is currently defined as
function(width = 1.960, biastype = symmetricBias()){
    new("asUnOvShoot", width = width, biastype = biastype) }

Examples
asUnOvShoot()
asUnOvShoot-class

Asymptotic under-/overshoot probability

Description

Class of asymptotic under-/overshoot probability.

Objects from the Class

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

Slots

- **type**: Object of class "character": “asymptotic under-/overshoot probability”.
- **width**: Object of class "numeric": half the width of given confidence interval.
- **biastype**: Object of class "BiasType": symmetric, one-sided or asymmetric

Extends

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

Methods

- **width** signature(object = "asUnOvShoot"): accessor function for slot width.
- **show** signature(object = "asUnOvShoot")

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

- `asGRisk-class`

Examples

`new("asUnOvShoot")`
asymmetricBias

Description

Generates an object of class "asymmetricBias".

Usage

asymmetricBias(name = "asymmetric Bias", nu = c(1,1))

Arguments

name name of the bias type
nu weights for negative and positive bias, respectively

Value

Object of class "asymmetricBias"

Author(s)

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

References


See Also

asymmetricBias-class

Examples

asymmetricBias()

## The function is currently defined as
function(){ new("asymmetricBias", name = "asymmetric Bias", nu = c(1,1)) }
asymmetricBias-class asymmetric Bias Type

Description

Class of asymmetric bias types.

Objects from the Class

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

Slots

- `name` Object of class "character".
- `nu` Object of class "numeric"; to be in (0,1] x (0,1] with maximum 1; weights for negative and positive bias, respectively

Methods

- `nu` signature(object = "asymmetricBias"): accessor function for slot `nu`.
- `nu<-` signature(object = "asymmetricBias", value = "numeric"): replacement function for slot `nu`.

Extends

Class "BiasType", directly.

Author(s)

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

References


See Also

BiasType-class
Examples

```r
asymmetricBias()
# The function is currently defined as
function(){ new("asymmetricBias", name = "asymmetric Bias", nu = c(1,1)) }

aB <- asymmetricBias()
nu(aB)
try(nu(aB) <- -2) ## error
nu(aB) <- c(0.3,1)
```

---

**Description**

Generates an object of class "L2ParamFamily" which represents a Beta family.

**Usage**

```r
BetaFamily(shape1 = 1, shape2 = 1, trafo, withL2derivDistr = TRUE)
```

**Arguments**

- `shape1` positive real: shape1 parameter
- `shape2` positive real: shape2 parameter
- `trafo` matrix: transformation of the parameter
- `withL2derivDistr` logical: shall the distribution of the L2 derivative be computed? Defaults to TRUE; setting it to FALSE speeds up computations.

**Details**

The slots of the corresponding L2 differentiable parameteric family are filled.

**Value**

Object of class "L2ParamFamily"

**Author(s)**

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

**See Also**

- `L2ParamFamily-class`
- `Beta-class`
Examples

```r
(B1 <- BetaFamily())
FisherInfo(B1)
checkL2deriv(B1)
```

<table>
<thead>
<tr>
<th>BiasType-class</th>
<th>Bias Type</th>
</tr>
</thead>
</table>

**Description**

Class of bias types.

**Objects from the Class**

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

**Slots**

- `name` Object of class "character".

**Methods**

- `name` signature(object = "BiasType"): accessor function for slot name.
- `name<-` signature(object = "BiasType", value = "character"): replacement function for slot name.

**Author(s)**

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

**References**


**See Also**

- `RiskType-class`

**Examples**

```r
abB <- positiveBias()
name(abB)
```
BinomFamily

Generating function for Binomial families

Description
Generates an object of class "L2ParamFamily" which represents a Binomial family where the probability of success is the parameter of interest.

Usage
BinomFamily(size = 1, prob = 0.5, trafo)

Arguments

size number of trials
prob probability of success
trafo function in param or matrix: transformation of the parameter

Details
The slots of the corresponding L2 differentiable parameteric family are filled.

Value
Object of class "L2ParamFamily"

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

References

See Also
L2ParamFamily-class, Binom-class

Examples
(B1 <- BinomFamily(size = 25, prob = 0.25))
plot(B1)
FisherInfo(B1)
checkL2deriv(B1)
Description

Generates an object of class "L2LocationScaleFamily" which represents a normal location and scale family.

Usage

CauchyLocationScaleFamily(loc = 0, scale = 1, trafo)

Arguments

loc       location
scale     scale
trafo     function in param or matrix: transformation of the parameter

Details

The slots of the corresponding L2 differentiable parameteric family are filled.

Value

Object of class "L2LocationScaleFamily"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

L2ParamFamily-class, Cauchy-class

Examples

```r
(C1 <- CauchyLocationScaleFamily())
plot(C1)
FisherInfo(C1)
### need smaller integration range:
distrExoptions("ElowerTruncQuantile"=1e-4,"EupperTruncQuantile"=1e-4)
checkL2deriv(C1)
distrExoptions("ElowerTruncQuantile"=1e-7,"EupperTruncQuantile"=1e-7)
```
checkL2deriv

Generic function for checking L2-derivatives

Description

Generic function for checking the L2-derivative of an L2-differentiable family of probability measures.

Usage

checkL2deriv(L2Fam, ...)

Arguments

L2Fam L2-differentiable family of probability measures
...
additional parameters

Details

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

Value

The maximum deviation is returned.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

L2ParamFamily-class

Examples

F1 <- new("L2ParamFamily")
checkL2der1v(F1)
Description

Return value S4 classes for method "confint".

Objects from the Class

Objects could in principle be created by calls of the form new("Confint", ...). The preferred form is to have them created via a call to confint.

Slots

type Object of class "character": type of the confidence interval (asymptotic, bootstrap,...). Can be of length >2. Then in printing, the first element is printed in the gap '[...]' in 'an [...] confidence interval', while the other elements are printed below.

confint Object of class "array": the confidence interval(s).

call.estimate Object of class "call": the estimate(s) for which the confidence intervals are produced.

name.estimate Object of class "character": the name of the estimate(s) for which the confidence intervals are produced.

samplesize.estimate: Object of class "numeric": the sample size of the estimate(s) for which the confidence intervals are produced.

completecases.estimate: Object of class "logical": complete cases at which the estimate was evaluated.

trafo.estimate Object of class "matrix": the trafo/derivative matrix of the estimate(s) for which the confidence intervals are produced.

nuisance.estimate Object of class "OptionalNumeric": the nuisance parameter (if any) at which the confidence intervals are produced.

fixed.estimate Object of class "OptionalNumeric": the fixed part of the parameter (if any) at which the confidence intervals are produced.

Methods

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

confint signature(object = "Confint", method = "missing"): accessor function for slot type.

call.estimate signature(object = "Confint"): accessor function for slot call.estimate.

name.estimate signature(object = "Confint"): accessor function for slot name.estimate.

trafo.estimate signature(object = "Confint"): accessor function for slot trafo.estimate.

samplesize.estimate signature(object = "Confint"): (with additional argument onlycompletecases defaulting to TRUE returns the sample size; in case there are any incomplete cases and argument onlycompletecases is FALSE, the number of these is added to slot samplesize.)
Confint-class

```
completecases.estimate signature(object = "Confint"): accessor function for slot completecases.estimate.
nuisance.estimate signature(object = "Confint"): accessor function for slot nuisance.estimate.
fixed.estimate signature(object = "Confint"): accessor function for slot fixed.estimate.
show signature(object = "Confint"): shows a detailed view of the object; slots nuisance.estimate and fixed.estimate are only shown if non-null, and slot trafo.estimate only if different from a unit matrix.
print signature(object = "Confint"): just as show, but with additional arguments digits.
```

Details for methods 'show', 'print'

Detailedness of output by methods show, print is controlled by the global option show.details to be set by `distrModoptions`.

As method show is used when inspecting an object by typing the object's name into the console, show comes without extra arguments and hence detailedness must be controlled by global options.

Method print may be called with a (partially matched) argument show.details, and then the global option is temporarily set to this value.

More specifically, when show.detail is matched to "minimal" you will be shown only the type of the confidence interval(s) and its/their values. When show.detail is matched to "medium", you will in addition see the type of the estimator(s) for which it is produced, the corresponding call of the estimator, its sample size, and, if present, the value of the corresponding nuisance parameter. Finally, when show.detail is matched to "maximal", additionally you will be shown the fixed part of the parameter (if present) and the transformation of the estimator (if non-trivial, i.e. the identity) in form of its function code respectively of its derivative matrix.

Note

The pretty-printing code for methods show and print has been borrowed from `confint.default` in package `stats`.

Author(s)

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

See Also

Estimator, confint, Estimate-class, trafo-methods

Examples

```
## some transformation
mtrafo <- function(x){
  nms0 <- c("scale","shape")
  nms <- c("shape","rate")
  fval0 <- c(x[2], 1/x[1])
  names(fval0) <- nms
  mat0 <- matrix( c(0, -1/x[1]^2, 1, 0), nrow = 2, ncol = 2,
                  dimnames = list(nms,nms0))
  list(fval = fval0, mat = mat0)}
```
Methods for function `confint` in Package ‘distrMod’

Description

Methods for function `confint` in package `distrMod`; by default uses `confint` and its corresponding S3-methods, but also computes (asymptotic) confidence intervals for objects of class `estimate`. Computes confidence intervals for one or more parameters in a fitted model.

Usage

```r
confint(object, method, ...)  # S4 method for signature 'ANY,missing'
confint(object, method, parm, level = 0.95, ...)  # S4 method for signature 'Estimate,missing'
confint(object, method, level = 0.95)  # S4 method for signature 'mle,missing'
confint(object, method, parm, level = 0.95, ...)  # S4 method for signature 'profile.mle,missing'
```

Arguments

- `parm`: only used in default / signature ANY case: a specification of which parameters are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, all parameters are considered.
- `level`: the confidence level required.
- `method`: not yet used (only as missing; later to allow for various methods)
- `...`: additional argument(s) for methods.

Example:

```r
x <- rgamma(50, scale = 0.5, shape = 3)

## parametric family of probability measures
G <- GammaFamily(scale = 1, shape = 2, trafo = mtrafo)
## MLE
res <- MLEEstimator(x = x, ParamFamily = G)

ci <- confint(res)
print(ci, digits = 4, show.details = "maximal")
print(ci, digits = 4, show.details = "medium")
print(ci, digits = 4, show.details = "minimal")
```
Details

confint is a generic function. Its behavior differs according to its arguments.

**signature ANY,missing:** the default method; uses the S3 generic of package stats, see `confint`.

**signature Estimate,missing:** will return a corresponding confidence interval assuming asymptotic normality, and hence needs suitably filled slot `asvar` in argument object. Besides the actual bounds, organized in an array just as in the S3 generic, the return value also captures the name of the estimator for which it is produced, as well as the corresponding call producing the estimator, and the corresponding `trafo` and `nuisance` slots/parts.

Value

**signature ANY,missing:**
A matrix (or vector) with columns giving lower and upper confidence limits for each parameter. These will be labelled as (1-level)/2 and 1 - (1-level)/2 in % (by default 2.5% and 97.5%).

**signature Estimate,missing:**
An object of class `Confint`

See Also

`confint`, `confint.glm` and `confint.nls` in package MASS, `Confint-class`.

Examples

```r
## for signature ANY examples confer stats::confint
## (empirical) Data
x <- rgamma(50, scale = 0.5, shape = 3)

## parametric family of probability measures
G <- GammaFamily(scale = 1, shape = 2)

## Maximum likelihood estimator
res <- MLEEstimator(x = x, ParamFamily = G)
confint(res)

### for comparison:
require(MASS)
(res1 <- fitdistr(x, "gamma"))
## add a convenient (albeit wrong)
## S3-method for vcov:
## --- wrong as in general cov-matrix
## will not be diagonal
## but for conf-interval this does
## not matter...
vcov.fitdistr <- function(object, ...){
  v <- diag(object$sd^2)
  rownames(v) <- colnames(v) <- names(object$estimate)
  v}

## explicitly transforming to
## MASS parametrization:

```r
mtrafo <- function(x){
  nms0 <- names(c(main(param(G)), nuisance(param(G))))
  nms <- c(“shape”, “rate”)
  fval0 <- c(x[2], 1/x[1])
  names(fval0) <- nms
  mat0 <- matrix(c(0, -1/x[1]^2, 1, 0), nrow = 2, ncol = 2,
                  dimnames = list(nms, nms))
  list(fval = fval0, mat = mat0)
}

G2 <- G
trafo(G2) <- mtrafo
res2 <- MLEstimator(x = x, ParamFamily = G2)

old <- getdistrModOption(“show.details”)
distrModOptions(“show.details” = “minimal”)
res
res1
res2
confint(res)
confint(res1)
confint(res2)
confint(res, level = 0.99)
distrModOptions(“show.details” = old)
```

---

**distrModMASK**

*Masking of/by other functions in package "distrMod"*

### Description

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

### Usage

```r
distrModMASK(library = NULL)
```

### Arguments

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

### Value

- no value is returned
distrModOptions

Author(s)

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

Examples

distrModMASK()

Function to change the global variables of the package 'distrMod'

Description

With distrModOptions you can inspect and change the global variables of the package distrMod.

Usage

distrModOptions(...)  
getdistrModOption(x)  
distrModoptions(...)

Arguments

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

Details

Invoking distrModOptions() with no arguments returns a list with the current values of the options. To access the value of a single option, one should use getdistrModOption("show.details"), e.g., rather than distrModoptions("show.details") which is a list of length one.

Value

distrModOptions() returns a list of the global options of distrMod.  
distrModOptions("show.details") returns the global option show.details as a list of length 1.  
distrModoptions("show.details" = "minimal") sets the value of the global option show.details to "minimal". getdistrModOption("show.details") the current value set for option show.details.

distrModoptions

For compatibility with spelling in package distr, distrModoptions is just a synonym to distrModoptions.
Currently available options

show.details  degree of detailedness for method show for objects of classes of the distrXXX family of packages. Possible values are
  "maximal"  all information is shown
  "minimal"  only the most important information is shown
  "medium"  somewhere in the middle; see actual show-methods for details.
  The default value is "maximal".

Author(s)

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

See Also

options,getOption,distroptions,getdistrOption

Examples

distrModoptions()
distrModoptions("show.details")
distrModoptions("show.details" = "maximal")
distrModOptions("show.details" = "minimal")
  # or
getdistrModOption("show.details")

Description

Class of estimates.

Objects from the Class

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

Slots

name  Object of class "character": name of the estimator.
estimate Object of class "ANY": estimate.
estimate.call Object of class "call": call by which estimate was produced.
Infos  object of class "matrix" with two columns named method and message: additional informations.
asvar object of class "OptionalNumericOrMatrix" which may contain the asymptotic (co)variance of the estimator.

class "numeric" — the samplesize (only complete cases are counted) at which the estimate was evaluated.

class "logical" — complete cases at which the estimate was evaluated.

indices of estimate belonging to the nuisance part.

the fixed and known part of the parameter.

a list with components fct and mat (see below).

object of class "ANY": untransformed estimate.

which may contain the asymptotic (co)variance of the untransformed estimator.

Methods

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

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

estimate signature(object = "Estimate"): accessor function for slot estimate.

untransformed.estimate signature(object = "Estimate"): accessor function for slot untransformed.estimate.

estimate.call signature(object = "Estimate"): accessor function for slot estimate.call.

samplesize signature(object = "Estimate"): (with additional argument onlycompletecases defaulting to TRUE returns the sample size; in case there are any incomplete cases and argument onlycompletecases is FALSE, the number of these is added to slot samplesize.

completecases signature(object = "Estimate"): accessor function for slot completecases.

asvar signature(object = "Estimate"): accessor function for slot asvar.

asvar<- signature(object = "Estimate"): replacement function for slot asvar.

untransformed.asvar signature(object = "Estimate"): accessor function for slot untransformed.asvar.

nuisance signature(object = "Estimate"): accessor function for nuisance part of slot estimate.

main signature(object = "Estimate"): accessor function for main part of slot estimate.

fixed signature(object = "Estimate"): accessor function for slot fixed.

Infos signature(object = "Estimate"): accessor function for slot Infos.

Infos<- signature(object = "Estimate"): replacement function for slot Infos.

addInfo<- signature(object = "Estimate"): function to add an information to slot Infos.

show signature(object = "Estimate")

print signature(object = "Estimate"): just as show, but with additional arguments digits.
Details for methods 'show', 'print'

Detailedness of output by methods show, print is controlled by the global option show.details to be set by distrMod::options.

As method show is used when inspecting an object by typing the object’s name into the console, show comes without extra arguments and hence detailedness must be controlled by global options. Method print may be called with a (partially matched) argument show.details, and then the global option is temporarily set to this value.

More specifically, when show.detail is matched to "minimal" you will be shown only the name/type of the estimator, the value of its main part, and, if present, the corresponding standard errors, as well as, also if present, the value of the nuisance part. When show.detail is matched to "medium", you will in addition see the class of the estimator, its call and its sample-size and, if present, the fixed part of the parameter and the asymptotic covariance matrix. Also the information gathered in the Infos slot is shown. Finally, when show.detail is matched to "maximal", and if, in addition, you estimate non-trivial (i.e. not the identity) transformation of the parameter of the parametric family, you will also be shown this transformation in form of its function and its derivative matrix at the estimated parameter value, as well as the estimator (with standard errors, if present) and (again, if present) the corresponding asymptotic covariance of the untransformed, total (i.e. main and nuisance part) parameter.

trafo realizes partial influence curves; i.e.; we are only interested is some possibly lower dimensional smooth (not necessarily linear or even coordinate-wise) aspect/transformation τ of the parameter θ.

To be coherent with the corresponding nuisance implementation, we make the following convention:

The full parameter θ is split up coordinate-wise in a main parameter θ' and a nuisance parameter θ'' (which is unknown, too, hence has to be estimated, but only is of secondary interest) and a fixed, known part θ'''.

Without loss of generality, we restrict ourselves to the case that transformation τ only acts on the main parameter θ' — if we want to transform the whole parameter, we only have to assume that both nuisance parameter θ'' and fixed, known part of the parameter θ''' have length 0.

To the implementation:

Slot trafo can either contain a (constant) matrix $D_θ$ or a function

\[ \tau: \Theta' \to \tilde{\Theta}, \quad \theta \mapsto \tau(\theta) \]

mapping main parameter θ' to some range $\tilde{\Theta}$.

If slot value trafo is a function, besides τ(θ), it will also return the corresponding derivative matrix \( \frac{\partial}{\partial \theta} \tau(\theta) \). More specifically, the return value of this function theta is a list with entries fval, the function value τ(θ), and mat, the derivative matrix.

In case trafo is a matrix $D$, we interpret it as such a derivative matrix \( \frac{\partial}{\partial \theta} \tau(\theta) \), and, correspondingly, τ(θ) as the linear mapping \( \tau(\theta) = D \theta \).

Note

The pretty-printing code for methods show and print has been borrowed from print.fitdistr in package MASS by B.D. Ripley.
Author(s)

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

See Also

Estimator

Examples

```r
x <- rnorm(100)
Estimator(x, estimator = mean, name = "mean")

x1 <- x; x1[sample(1:100, 10)] <- NA
myEst1 <- Estimator(x1, estimator = mean, name = "mean")
samplesize(myEst1)
samplesize(myEst1, onlycomplete = FALSE)
```

Description

The function `Estimator` provides a general way to compute estimates.

Usage

```r
Estimator(x, estimator, name, Infos, asvar = NULL, nuis.idx, 
trafo = NULL, fixed = NULL, asvar.fct, na.rm = TRUE, ..., 
ParamFamily = NULL, .withEvalAsVar = TRUE)
```

Arguments

- `x` (empirical) data
- `estimator` function: estimator to be evaluated on \( x \).
- `name` optional name for estimator.
- `Infos` character: optional informations about estimator
- `asvar` optionally the asymptotic (co)variance of the estimator
- ` nuis.idx` optionally the indices of the estimate belonging to nuisance parameter
- `fixed` optionally (numeric) the fixed part of the parameter
- `trafo` an object of class `Matrix` or `Function` — a transformation for the main parameter
- `asvar.fct` optionally: a function to determine the corresponding asymptotic variance; if given, `asvar.fct` takes arguments `L2Fam`(the parametric model as object of class `L2ParamFamily`) and `param` (the parameter value as object of class `ParamFamParameter`); arguments are called by name; `asvar.fct` may also process further arguments passed through the `...` argument.
na.rm logical: if TRUE, the estimator is evaluated at complete_cases(x).
...
 further arguments to estimator.
ParamFamily an optional object of class ParamFamily. Passed on to asvar.fct to compute asymptotic variances.
.withEvalAsVar logical: shall slot asVar be evaluated (if asvar.fct is given) or just the call be returned?

Details

The argument criterion has to be a function with arguments the empirical data as well as an object of class "Distribution" and possibly ....

Value

An object of S4-class "Estimate".

Author(s)

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

See Also

Estimate-class

Examples

x <- rnorm(100)
Estimator(x, estimator = mean, name = "mean")

X <- matrix(rnorm(1000), nrow = 10)
Estimator(X, estimator = rowMeans, name = "mean")

---

EvenSymmetric Generating function for EvenSymmetric-class

Description

Generates an object of class "EvenSymmetric".

Usage

EvenSymmetric(SymmCenter = 0)

Arguments

SymmCenter numeric: center of symmetry
Value

Object of class "EvenSymmetric"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

EvenSymmetric-class, FunctionSymmetry-class

Examples

EvenSymmetric()

## The function is currently defined as
function(SymmCenter = 0){
  new("EvenSymmetric", SymmCenter = SymmCenter)
}

EvenSymmetric-class  Class for Even Functions

Description

Class for even functions.

Objects from the Class

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

Slots

type  Object of class "character": contains "even function"
SymmCenter Object of class "numeric": center of symmetry

Extends

Class "FunctionSymmetry", directly.
Class "Symmetry", by class "FunctionSymmetry".

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

EvenSymmetric, FunctionSymmetry-class
**Examples**

```r
new("EvenSymmetric")
```

**Description**

existsPIC-methods to check whether in a given L2 differentiable model at parameter value theta there exist (partial) influence curves to Trafo \( D_\theta \).

**Usage**

```r
existsPIC(object, ...)  # S4 method for signature 'L2ParamFamily'
existsPIC(object, warning = TRUE, tol = .Machine$double.eps)
```

**Arguments**

- `object` L2ParamFamily
- `...` further arguments used by specific methods.
- `warning` logical: should a warning be issued if there exist no (partial) influence curves?
- `tol` the tolerance the linear algebraic operations. Default is .Machine$double.eps.

**Details**

To check the existence of (partial) influence curves and, simultaneously, for bounded (partial) influence curves, by Lemma 1.1.3 in Kohl(2005) [resp. the fact that \( \ker I = \ker I \) for \( J = E(A', 1)'(A', 1)w \) and \( w = \min(1, b/|A', 1|) \)], it suffices to check that \( \ker I \) is a subset of \( \ker D_\theta \). This is done by a call to `isKerAinKerB`.

**Author(s)**

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

**References**


**See Also**

`isKerAinKerB`
ExpScaleFamily

Generating function for exponential scale families

Description

Generates an object of class "L2ScaleFamily" which represents an exponential scale family.

Usage

ExpScaleFamily(scale = 1, trafo)

Arguments

scale scale (= 1/rate)
trafo function in param or matrix: optional transformation of the parameter

Details

The slots of the corresponding L2 differentiable parameteric family are filled. The scale parameter corresponds to 1/rate.

Value

Object of class "L2ScaleFamily"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

L2ParamFamily-class, Exp-class

Examples

(E1 <- ExpScaleFamily())
plot(E1)
Map(L2deriv(E1)[[1]])
checkL2deriv(E1)
fiBias-class

Generating function for fiBias-class

Description
Generates an object of class "fiBias".

Usage
fiBias()

Value
Object of class "fiBias"

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

References

See Also
fibiasMclass

Examples
fibias()

## The function is currently defined as
function(){ new("fiBias") }
fiCov

Slots
   type  Object of class "character": "finite-sample bias".

Extends
   Class "fiRisk", directly.  
   Class "RiskType", by class "fiRisk".

Methods
   No methods defined with class "fiBias" in the signature.

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

References

See Also
   fiRisk-class, fiBias

Examples
   new("fiBias")

---

fiCov  Generating function for fiCov-class

Description
   Generates an object of class "fiCov".

Usage
   asCov()

Value
   Object of class "fiCov"

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

See Also
fiCov-class

Examples

```r
cov()

## The function is currently defined as
function(){ new(“fiCov”) }
```

---

## fiCov-class

### Finite-sample covariance

**Description**
Class of finite-sample covariance.

**Objects from the Class**
Objects can be created by calls of the form `new("fiCov", ...)`. More frequently they are created via the generating function `ficov`.

**Slots**
- `type` Object of class "character": “finite-sample covariance”.

**Extends**
- Class "fiRisk", directly.
- Class "RiskType", by class "fiRisk".

**Methods**
No methods defined with class "fiCov" in the signature.

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

**References**
See Also

fiRisk-class, fiCov

Examples

new("fiCov")

---

fiHampel  Generating function for fiHampel-class

Description

Generates an object of class "fiHampel".

Usage

fiHampel(bound = Inf)

Arguments

bound  positive real: bias bound

Value

Object of class fiHampel

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

fiHampel-class

Examples

fiHampel()

## The function is currently defined as
function(bound = Inf){ new("fiHampel", bound = bound) }
fiHampel-class  

Finite-sample Hampel risk

Description

Class of finite-sample Hampel risk which is the trace of the finite-sample covariance subject to a given bias bound (bound on gross error sensitivity).

Objects from the Class

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

Slots

type  Object of class "character": “trace of finite-sample covariance for given bias bound”.

bound  Object of class "numeric": given positive bias bound.

Extends

Class "fiRisk", directly.
Class "RiskType", by class "fiRisk".

Methods

bound signature(object = "fiHampel"): accessor function for slot bound.

show signature(object = "fiHampel")

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

`fiRisk-class`, `fiHampel`

Examples

`new("fiHampel")`
fIMSE  Generating function for fIMSE-class

Description
Generates an object of class "fIMSE".

Usage
fIMSE()

Value
Object of class "fIMSE"

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

References

See Also
fIMSE-class

Examples
fIMSE()

## The function is currently defined as
function(){ new("fIMSE") }
Slots

type Object of class "character": “finite-sample mean square error”.

Extends

Class "fiRisk", directly.
Class "RiskType", by class "fiRisk".

Methods

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

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

fiRisk-class, fiMSE

Examples

new("fiMSE")

---

fiRisk-class  Finite-sample risk

Description

Class of finite-sample risks.

Objects from the Class

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

Slots

type Object of class "character".

Extends

Class "RiskType", directly.
Methods

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

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

RiskType-class

Description

Generates an object of class "fiUnOvShoot".

Usage

fiUnOvShoot(width = 1.960)

Arguments

width positive real: half the width of given confidence interval.

Value

Object of class "fiUnOvShoot"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References

fiUnOvShoot-class

See Also

fiUnOvShoot-class

Examples

fiUnOvShoot()

## The function is currently defined as
function(width = 1.960){ new("fiUnOvShoot", width = width) }

fiUnOvShoot-class Finite-sample under-/overshoot probability

Description

Class of finite-sample under-/overshoot probability.

Objects from the Class

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

Slots

type Object of class "character": “finite-sample under-/overshoot probability”.
width Object of class "numeric": half the width of given confidence interval.

Extends

Class "fiRisk", directly.
Class "RiskType", by class "fiRisk".

Methods

width signature(object = "fiUnOvShoot"): accessor function for slot width.
show signature(object = "fiUnOvShoot")

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>
FunctionSymmetry-class

References


See Also

fiRisk-class

Examples

new("fiUnOvShoot")

FunctionSymmetry-class

Class of Symmetries for Functions

Description

Class of symmetries for functions.

Objects from the Class

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

Slots

type  Object of class "character": describes type of symmetry.
SymmCenter  Object of class "OptionalNumeric": center of symmetry.

Extends

Class "Symmetry", directly.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

Symmetry-class, OptionalNumeric-class
FunSymmList  Generating function for FunSymmList-class

Description

Generates an object of class "FunSymmList".

Usage

FunSymmList(...)  

Arguments

...  Objects of class "FunctionSymmetry" which shall form the list of symmetry types.

Value

Object of class "FunSymmList"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

FunSymmList-class

Examples

FunSymmList(NonSymmetric(), EvenSymmetric(SymmCenter = 1), OddSymmetric(SymmCenter = 2))

## The function is currently defined as
function (...){
    new("FunSymmList", list(...))
}


FunSymmList-class

List of Symmetries for a List of Functions

Description
Create a list of symmetries for a list of functions

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

Slots
.Data Object of class "list". A list of objects of class "FunctionSymmetry".

Extends
Class "list", from data part.
Class "vector", by class "list".

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

See Also
FunctionSymmetry-class

Examples
new("FunSymmList", list(NonSymmetric(), EvenSymmetric(SymmCenter = 1),
OddSymmetric(SymmCenter = 2)))

GammaFamily
Generating function for Gamma families

Description
Generates an object of class "L2ParamFamily" which represents a Gamma family.

Usage
GammaFamily(scale = 1, shape = 1, trafo, withL2derivDistr = TRUE)
Arguments

- `scale`: positive real: scale parameter
- `shape`: positive real: shape parameter
- `trafo`: matrix: transformation of the parameter
- `withL2derivDistr`: logical: shall the distribution of the L2 derivative be computed? Defaults to TRUE; setting it to FALSE speeds up computations.

Details

The slots of the corresponding L2 differentiable parameteric family are filled.

Value

Object of class "L2ParamFamily"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

`L2ParamFamily-class, Gammad-class`

Examples

```r
(G1 <- GammaFamily())
FisherInfo(G1)
checkL2deriv(G1)
```

---

Description

Generates an object of class "InfoNorm" — used for information-standardized influence curves.

Usage

`InfoNorm()`
isKerAinKerB

Value

Object of class "InfoNorm"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

InfoNorm-class

Examples

InfoNorm()

## The function is currently defined as
function(){ new("InfoNorm") }

---

isKerAinKerB isKerAinKerB

Description

For two matrices A and B checks whether the null space of A is a subspace of the null space of B, in other words, if \( Ax = 0 \) entails \( Bx = 0 \).

Usage

isKerAinKerB(A, B, tol = .Machine$double.eps)

Arguments

A  
a matrix; if A is a vector, A is coerced to a matrix by as.matrix.
B  
a matrix; if B is a vector, B is coerced to a matrix by as.matrix.
tol  
the tolerance for detecting linear dependencies in the columns of A and up to which the two projectors are seen as equal (see below).

Details

via calls to svd, the projectors \( \pi_A \) and \( \pi_B \) onto the respective orthogonal complements of \( \ker(A) \) and \( \ker(B) \) are calculated and then is checked whether \( \pi_B\pi_A = \pi_B \).
Value

logical

Author(s)

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

Examples

```r
ma <- cbind(1,1,c(1,1,1))
D <- t(ma %*% c(0,1,-1))
isKerAinKerB(D,ma)
isKerAinKerB(ma,D)
```

---

**L2GroupParamFamily-class**

*L2 differentiable parametric group family*

---

**Description**

Class of L2 differentiable parametric group families.

**Objects from the Class**

Objects can be created by calls of the form `new("L2GroupParamFamily", ...`). More frequently, this class is just used as an intermediate class to classes of specific group models like `L2LocationFamily-class`, `L2ScaleFamily-class`, and `L2LocationScaleFamily-class`.

**Slots**

- `name` [inherited from class "ProbFamily"] object of class "character": name of the family.
- `distribution` [inherited from class "ProbFamily"] object of class "Distribution": member of the family.
- `distrSymm` [inherited from class "ProbFamily"] object of class "DistributionSymmetry": symmetry of distribution.
- `param` [inherited from class "ParamFamily"] object of class "ParamFamParameter": parameter of the family.
- `fam.call` [inherited from class "ParamFamily"] object of class "call": call by which parametric family was produced.
- `makeOKPar` [inherited from class "ParamFamily"] object of class "function": has argument `param` — the (total) parameter, returns valid parameter; used if `optim` resp. `optimize`— try to use “illegal” parameter values; then `makeOKPar` makes a valid parameter value out of the illegal one.
- `startPar` [inherited from class "ParamFamily"] object of class "function": has argument `x` — the data, returns starting parameter for `optim` resp. `optimize`— a starting estimator in case parameter is multivariate or a search interval in case parameter is univariate.
modifyParam [inherited from class "ParamFamily"] object of class "function": mapping from the parameter space (represented by "param") to the distribution space (represented by "distribution").

props [inherited from class "ProbFamily"] object of class "character": properties of the family.

L2deriv [inherited from class "L2ParamFamily"] object of class "EuclRandVariable": L2 derivative of the family.

L2deriv.fct [inherited from class "L2ParamFamily"] object of class "function": mapping from the parameter space (argument param of class "ParamFamParameter") to a mapping from observation x to the value of the L2derivative; L2deriv.fct is then used from observation x to value of the L2derivative; L2deriv.fct is used by modifyModel to move the L2deriv according to a change in the parameter.

L2derivSymm [inherited from class "L2ParamFamily"] object of class "FunSymmList": symmetry of the maps included in L2deriv.

L2derivDistr [inherited from class "L2ParamFamily"] object of class "UnivarDistrList": list which includes the distribution of L2deriv.

L2derivDistrSymm [inherited from class "L2ParamFamily"] object of class "DistrSymmList": symmetry of the distributions included in L2derivDistr.

FisherInfo.fct [inherited from class "L2ParamFamily"] object of class "function": mapping from the parameter space (argument param of class "ParamFamParameter") to the set of positive semidefinite matrices; FisherInfo.fct is used by modifyModel to move the Fisher information according to a change in the parameter.

FisherInfo [inherited from class "L2ParamFamily"] object of class "PosDefSymmMatrix": Fisher information of the family.

LogDeriv object of class "function": has argument x; the negative logarithmic derivative of the density of the model distribution at the "standard" parameter value.

Extends

Class "L2ParamFamily", directly.
Class "ParamFamily", by class "L2ParamFamily".
Class "ProbFamily", by class "ParamFamily".

Methods

LogDeriv signature(object = "L2GroupParamFamily"): accessor function for slot LogDeriv.

LogDeriv<- signature(object = "L2GroupParamFamily"): replacement function for slot LogDeriv.

Author(s)

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

References

L2LocationFamily

See Also
L2ParamFamily-class, ParamFamily-class

Examples
F1 <- new("L2GroupParamFamily")
plot(F1)

---

Generating function for L2LocationFamily-class

Description
Generates an object of class "L2LocationFamily".

Usage
L2LocationFamily(loc = 0, name, centraldistribution = Norm(),
  locname = "loc", modParam, LogDeriv,
  L2derivDistr.0, FisherInfo.0, distrSymm, L2derivSymm,
  L2derivDistrSymm, trafo, .returnClsName = NULL)

Arguments

loc numeric: location parameter of the model.
name character: name of the parametric family.
centraldistribution object of class "AbscontDistribution": we assume from the beginning, that
centraldistribution is symmetric about its median.

modParam optional function: mapping from the parameter space (represented by "param")
to the distribution space (represented by "distribution").
locname a character vector of length 1 containing the name of the location parameter

LogDeriv function with argument x: the negative logarithmic derivative of the density of
the central distribution; if missing, it is determined numerically using numeric
differentiation.

L2derivDistr.0 object of class "UnivariateDistribution": distribution of the L2derivative at
the central distribution

FisherInfo.0 object of class "PosSemDefSymmMatrix": Fisher information of the model at
the "standard" parameter value
distrSymm object of class "DistributionSymmetry": symmetry of distribution.
L2derivSymm object of class "FunSymmList": symmetry of the maps contained in L2deriv
L2derivDistrSymm object of class "DistrSymmList": symmetry of the distributions contained in L2derivDistr
trafo  matrix or function in `param`: transformation of the parameter

`.returnClsName`  the class name of the return value; by default this argument is `NULL` whereupon the return class will be `L2LocationScaleFamily`; but, internally, this generating function is also used to produce objects of class `NormLocationFamily` and `GumbelLocationFamily` (the latter in package `RobExtremes`.

Details

If `name` is missing, the default “L2 location family” is used. The function `modParam` is optional. If it is missing, it is constructed from `centraldistribution` using the location structure of the model.

Slot `param` is filled accordingly with the argument `trafo` passed to `L2LocationFamily`. In case `L2derivDistr.0` is missing, `L2derivDistr` is computed via `imageDistr`, else `L2derivDistr` is assigned `L2derivDistr.0`, coerced to "UnivariateDistributionList". In case `FisherInfo.0` is missing, Fisher information is computed from `L2deriv` using `E`. If `distrSymm` is missing, it is set to symmetry about `loc`. If `L2derivSymm` is missing, it is set to no symmetry, and if `L2derivDistrSymm` is missing, it is set to no symmetry, too.

Value

Object of class "L2LocationFamily"

Author(s)

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

References


See Also

`L2LocationFamily-class`

Examples

```r
F1 <- L2LocationFamily()
plot(F1)
```
L2LocationFamily-class

$L2$ differentiable parametric group family

Description

Class of $L2$ differentiable parametric group families.

Objects from the Class

Objects can be created by calls of the form `new("L2LocationFamily", ...)`.
More frequently they are created via the generating function `L2LocationFamily`.

Slots

- **name** [inherited from class "ProbFamily"] object of class "character": name of the family.
- **distribution** [inherited from class "ProbFamily"] object of class "Distribution": member of the family.
- **distrSymm** [inherited from class "ProbFamily"] object of class "DistributionSymmetry": symmetry of the distribution.
- **param** [inherited from class "ParamFamily"] object of class "ParamFamParameter": parameter of the family.
- **fam.call** [inherited from class "ParamFamily"] object of class "call": call by which parametric family was produced.
- **makeOKPar** [inherited from class "ParamFamily"] object of class "function": has argument `param` — the (total) parameter, returns valid parameter; used if `optim` resp. `optimize` — try to use "illegal" parameter values; then `makeOKPar` makes a valid parameter value out of the illegal one.
- **startPar** [inherited from class "ParamFamily"] object of class "function": has argument `x` — the data, returns starting parameter for `optim` resp. `optimize` — a starting estimator in case parameter is multivariate or a search interval in case parameter is univariate.
- **modifyParam** [inherited from class "ParamFamily"] object of class "function": mapping from the parameter space (represented by "param") to the distribution space (represented by "distribution").
- **props** [inherited from class "ProbFamily"] object of class "character": properties of the family.
- **L2deriv** [inherited from class "L2ParamFamily"] object of class "EuclRandVariable": $L2$ derivative of the family.
- **L2deriv.fct** [inherited from class "L2ParamFamily"] object of class "function": mapping from the parameter space (argument `param` of class "ParamFamParameter") to a mapping from observation `x` to the value of the $L2$derivative; `L2deriv.fct` is then used from observation `x` to value of the $L2$derivative; `L2deriv.fct` is used by `modifyModel` to move the $L2deriv$ according to a change in the parameter.
- **L2derivSymm** [inherited from class "L2ParamFamily"] object of class "FunSymmList": symmetry of the maps included in `L2deriv`.  

L2LocationFamily-class

L2derivDistr [inherited from class "L2ParamFamily"] object of class "UnivarDistrList": list which includes the distribution of L2deriv.

L2derivDistrSymm [inherited from class "L2ParamFamily"] object of class "DistrSymmList": symmetry of the distributions included in L2derivDistr.

FisherInfo.fct [inherited from class "L2ParamFamily"] object of class "function": mapping from the parameter space (argument param of class "ParamFamParameter") to the set of positive semidefinite matrices; FisherInfo.fct is used by modifyModel to move the Fisher information according to a change in the parameter.

FisherInfo [inherited from class "L2ParamFamily"] object of class "PosDefSymmMatrix": Fisher information of the family.

LogDeriv [inherited from class "L2GroupParamFamily"] object of class "function": has argument x; the negative logarithmic derivative of the density of the model distribution at the "standard" parameter value.

locscalename [inherited from class "L2LocationScaleUnion"] object of class "character": names of location and scale parameter

Extends

Class "L2LocationScaleUnion", directly.
Class "L2GroupParamFamily", by class "L2LocationScaleUnion".
Class "L2ParamFamily", by class "L2GroupParamFamily".
Class "ParamFamily", by class "L2ParamFamily".
Class "ProbFamily", by class "ParamFamily".

Methods

modifyModel signature(model = "L2LocationFamily", param = "ParamFamParameter"): moves the L2-location family model to parameter param

Author(s)

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

References


See Also

L2LocationFamily, ParamFamily-class

Examples

F1 <- new("L2LocationFamily")
plot(F1)
L2LocationScaleFamily  Generating function for L2LocationScaleFamily-class

Description
Generates an object of class "L2LocationScaleFamily".

Usage
L2LocationScaleFamily(loc = 0, scale = 1, name, centraldistribution = Norm(),
locscalename = c("loc", "scale"), modParam, LogDeriv,
L2derivDistr.0, FisherInfo.0, distrSymm, L2derivSymm,
L2derivDistrSymm, trafo, .returnClsName = NULL)

Arguments
loc      numeric: location parameter of the model.
scale    positive number: scale of the model.
name     character: name of the parametric family.
centraldistribution
          object of class "AbscontDistribution": central distribution; we assume by
default, that centraldistribution is symmetric about 0
modParam optional function: mapping from the parameter space (represented by "param")
to the distribution space (represented by "distribution").
locscalename a character vector of length 2 containing the names of the location and scale
              parameter; either unnamed, then order must be c(loc, scale), or named, then
              names must be "loc" and "scale"
LogDeriv  function with argument x: the negative logarithmic derivative of the density
          of the central distribution; if missing, it is determined numerically using numeric
          differentiation.
L2derivDistr.0 list of length 2 of objects of class "UnivariateDistribution": (marginal) dis-
tributions of the coordinates of the L2derivative at the central distribution
FisherInfo.0 object of class "PosSemDefSymmMatrix": Fisher information of the model at
              the "standard" parameter value
distrSymm object of class "DistributionSymmetry": symmetry of distribution.
L2derivSymm object of class "FunSymmList": symmetry of the maps contained in L2deriv
L2derivDistrSymm object of class "DistrSymmList": symmetry of the distributions contained in
L2derivDistr
trafo     matrix or function in param: transformation of the parameter
.returnClsName the class name of the return value; by default this argument is NULL whereupon
          the return class will be L2LocationScaleFamily; but, internally, this generating
          function is also used to produce objects of class NormalLocationScaleFamily,
          CauchyLocationScaleFamily.
**Details**

If name is missing, the default “L2 location and scale family” is used. The function modParam is optional. If it is missing, it is constructed from centraldistribution using the location and scale structure of the model. Slot param is filled accordingly with the argument trafo passed to L2LocationScaleFamily. In case L2derivDistr.0 is missing, L2derivDistr is computed via imageDistr, else L2derivDistr is assigned L2derivDistr.0 coerced to "UnivariateDistributionList". In case FisherInfo.0 is missing, Fisher information is computed from L2deriv using E. If distrSymm is missing, it is set to symmetry about loc. If L2derivSymm is missing, its location and scale components are set to no symmetry, respectively. If L2derivDistrSymm is missing, its location and scale components are set to no symmetry, respectively.

**Value**

Object of class "L2LocationScaleFamily"

**Author(s)**

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

**References**


**See Also**

L2LocationScaleFamily-class

**Examples**

```r
F1 <- L2LocationScaleFamily()
plot(F1)
```

---

**Description**

Class of L2 differentiable parametric group families.

**Objects from the Class**

Objects can be created by calls of the form `new("L2LocationScaleFamily", ...)`. More frequently they are created via the generating function L2LocationScaleFamily.
Slots

name [inherited from class "ProbFamily"] object of class "character": name of the family.
distribution [inherited from class "ProbFamily"] object of class "Distribution": member of the family.
distrSymm [inherited from class "ProbFamily"] object of class "DistributionSymmetry": symmetry of distribution.
param [inherited from class "ParamFamily"] object of class "ParamFamParameter": parameter of the family.
fam.call [inherited from class "ParamFamily"] object of class "call": call by which parametric family was produced.
makeOKPar [inherited from class "ParamFamily"] object of class "function": has argument param — the (total) parameter, returns valid parameter; used if optim resp. optimize—try to use "illegal" parameter values; then makeOKPar makes a valid parameter value out of the illegal one.
startPar [inherited from class "ParamFamily"] object of class "function": has argument x — the data, returns starting parameter for optim resp. optimize—starting estimator in case parameter is multivariate or a search interval in case parameter is univariate.
modifyParam [inherited from class "ParamFamily"] object of class "function": mapping from the parameter space (represented by "param") to the distribution space (represented by "distribution").
props [inherited from class "ProbFamily"] object of class "character": properties of the family.
L2deriv [inherited from class "L2ParamFamily"] object of class "EuclRandVariable": L2 derivative of the family.
L2deriv.fct [inherited from class "L2ParamFamily"] object of class "function": mapping from the parameter space (argument param of class "ParamFamParameter") to a mapping from observation x to the value of the L2derivative; L2deriv.fct is then used from observation x to value of the L2derivative; L2deriv.fct is used by modifyModel to move the L2deriv according to a change in the parameter
L2derivSymm [inherited from class "L2ParamFamily"] object of class "FunSymmList": symmetry of the maps included in L2deriv.
L2derivDistr [inherited from class "L2ParamFamily"] object of class "UnivarDistrList": list which includes the distribution of L2deriv.
L2derivDistrSymm [inherited from class "L2ParamFamily"] object of class "DistrSymmList": symmetry of the distributions included in L2derivDistr.
FisherInfo.fct [inherited from class "L2ParamFamily"] object of class "function": mapping from the parameter space (argument param of class "ParamFamParameter") to the set of positive semidefinite matrices; FisherInfo.fct is used by modifyModel to move the Fisher information according to a change in the parameter
FisherInfo [inherited from class "L2ParamFamily"] object of class "PosDefSymmMatrix": Fisher information of the family.
LogDeriv [inherited from class "L2GroupParamFamily"] object of class "function": has argument x: the negative logarithmic derivative of the density of the model distribution at the "standard" parameter value.
locscalename [inherited from class "L2LocationScaleUnion"] object of class "character": names of location and scale parameter
Extends

Class "L2LocationScaleUnion", directly.
Class "L2GroupParamFamily", by class "L2LocationScaleUnion".
Class "L2ParamFamily", by class "L2GroupParamFamily".
Class "ParamFamily", by class "L2ParamFamily".
Class "ProbFamily", by class "ParamFamily".

Methods

modifyModel signature(model = "L2LocationScaleFamily", param = "ParamFamParameter"): moves the L2-location and scale family model to parameter param

Author(s)

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

References


See Also

L2LocationScaleFamily, ParamFamily-class

Examples

F1 <- new("L2LocationScaleFamily")
plot(F1)

L2LocationUnknownScaleFamily

*Generating function for L2LocationScaleFamily-class in nuisance situation*

Description

Generates an object of class "L2LocationScaleFamily" in the situation where location is main, scale nuisance parameter.

Usage

L2LocationUnknownScaleFamily(loc = 0, scale = 1, name, centraldistribution = Norm(),
locscalenname = c("loc", "scale"), modParam, LogDeriv,
L2derivDistr.0, FisherInfo.0, distrSymm, L2derivSymm,
L2derivDistrSymm, trafo, .returnClsName = NULL)
Arguments

- `loc` numeric: location parameter of the model.
- `scale` positive number: scale of the model.
- `name` character: name of the parametric family.
- `centraldistribution` object of class "AbscontDistribution": central distribution; we assume by default, that centraldistribution is symmetric about 0.
- `modParam` optional function: mapping from the parameter space (represented by "param") to the distribution space (represented by "distribution").
- `locscalenname` a character vector of length 2 containing the names of the location and scale parameter; either unnamed, then order must be `c(loc, scale)`, or named, then names must be "loc" and "scale".
- `LogDeriv` function with argument `x`: the negative logarithmic derivative of the density of the central distribution; if missing, it is determined numerically using numeric differentiation.
- `L2derivDistr.0` list of length 2 of objects of class "UnivariateDistribution": (marginal) distributions of the coordinates of the L2derivative at the central distribution.
- `FisherInfo.0` object of class "PosSemDefSymmMatrix": Fisher information of the model at the "standard" parameter value.
- `distrSymm` object of class "DistributionSymmetry": symmetry of distribution.
- `L2derivSymm` object of class "FunSymmList": symmetry of the maps contained in `L2deriv`.
- `L2derivDistrSymm` object of class "DistrSymmList": symmetry of the distributions contained in `L2derivDistr`.
- `trafo` matrix or function in `param`: transformation of the parameter.
- `.returnClsName` the class name of the return value; by default this argument is NULL whereupon the return class will be `L2LocationScaleFamily`; but, internally, this generating function is also used to produce objects of class `NormalLocationScaleFamily`.

Details

If `name` is missing, the default “L2 location family with unknown scale (as nuisance)” is used. The function `modParam` is optional. If it is missing, it is constructed from `centraldistribution` using the location and scale structure of the model. Slot `param` is filled accordingly with the argument `trafo` passed to `L2LocationUnknownScaleFamily`. In case `L2derivDistr.0` is missing, `L2derivDistr` is computed via `imageDistr`, else `L2derivDistr` is assigned `L2derivDistr.0`, coerced to "UnivariateDistributionList". In case `FisherInfo.0` is missing, Fisher information is computed from `L2deriv` using `E`. If `distrSymm` is missing, it is set to symmetry about `loc`. If `L2derivSymm` is missing, its location and scale components are set to no symmetry, respectively. If `L2derivDistrSymm` is missing, its location and scale components are set to no symmetry, respectively.

Value

Object of class "L2LocationScaleFamily"
Author(s)

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

References


See Also

L2LocationScaleFamily-class

Examples

```r
F1 <- L2LocationUnknownScaleFamily()
plot(F1)
```

L2ParamFamily

Generating function for L2ParamFamily-class

Description

Generates an object of class "L2ParamFamily".

Usage

```r
L2ParamFamily(name, distribution = Norm(), distrSymm,
  main = main(param), nuisance = nuisance(param),
  fixed = fixed(param), trafo = trafo(param),
  param = ParamFamParameter(name = paste("Parameter of", name),
    main = main, nuisance = nuisance,
    fixed = fixed, trafo = trafo),
  props = character(0),
  startPar = NULL, makeOKPar = NULL,
  modifyParam = function(theta){ Norm(mean=theta) },
  L2deriv.fct = function(param) {force(theta <- param@main)
    return(function(x) (x-theta))},
  L2derivSymm, L2derivDistr, L2derivDistrSymm,
  FisherInfo.fct, FisherInfo = FisherInfo.fct(param),
  returnClsName = NULL, .withMDE = TRUE)
```
**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>character string: name of the family</td>
</tr>
<tr>
<td>distribution</td>
<td>object of class &quot;Distribution&quot;: member of the family</td>
</tr>
<tr>
<td>distrSymm</td>
<td>object of class &quot;DistributionSymmetry&quot;: symmetry of distribution.</td>
</tr>
<tr>
<td>main</td>
<td>numeric vector: main parameter</td>
</tr>
<tr>
<td>nuisance</td>
<td>numeric vector: nuisance parameter</td>
</tr>
<tr>
<td>fixed</td>
<td>numeric vector: fixed part of the parameter</td>
</tr>
<tr>
<td>trafo</td>
<td>function in param or matrix: transformation of the parameter</td>
</tr>
<tr>
<td>param</td>
<td>object of class &quot;ParamFamParameter&quot;: parameter of the family</td>
</tr>
<tr>
<td>startPar</td>
<td>startPar is a function in the observations x returning initial information for MCEstimator used by optimize resp. optim; i.e; if (total) parameter is of length 1, startPar returns a search interval, else it returns an initial parameter value.</td>
</tr>
<tr>
<td>makeOKPar</td>
<td>makeOKPar is a function in the (total) parameter param; used if optim resp. optimize—try to use “illegal” parameter values; then makeOKPar makes a valid parameter value out of the illegal one; if NULL slot makeOKPar of ParamFamily is used to produce it.</td>
</tr>
<tr>
<td>modifyParam</td>
<td>function: mapping from the parameter space (represented by &quot;param&quot;) to the distribution space (represented by &quot;distribution&quot;).</td>
</tr>
<tr>
<td>props</td>
<td>character vector: properties of the family</td>
</tr>
<tr>
<td>L2deriv.fct</td>
<td>function: mapping from the parameter space (argument param of class &quot;ParamFamParameter&quot;) to a mapping from observation x to the value of the L2derivative; L2deriv.fct is used by modifyModel to move the L2deriv according to a change in the parameter.</td>
</tr>
<tr>
<td>L2derivSymm</td>
<td>object of class &quot;FunSymmList&quot;: symmetry of the maps contained in L2deriv</td>
</tr>
<tr>
<td>L2derivDistr</td>
<td>object of class &quot;UnivarDistrList&quot;: distribution of L2deriv</td>
</tr>
<tr>
<td>L2derivDistrSymm</td>
<td>object of class &quot;DistrSymmList&quot;: symmetry of the distributions contained in L2derivDistr</td>
</tr>
<tr>
<td>FisherInfo.fct</td>
<td>function: mapping from the parameter space (argument param of class &quot;ParamFamParameter&quot;) to the set of positive semidefinite matrices; FisherInfo.fct is used by modifyModel to move the Fisher information according to a change in the parameter.</td>
</tr>
<tr>
<td>FisherInfo</td>
<td>object of class &quot;PosSemDefSymmMatrix&quot;: Fisher information of the family</td>
</tr>
<tr>
<td>.returnClsName</td>
<td>the class name of the return value; by default this argument is NULL whereupon the return class will be L2ParamFamily; but, internally, this generating function is also used to e.g. produce objects of class BinomialFamily, PoisFamily, GammaFamily, BetaFamily.</td>
</tr>
<tr>
<td>.withMDE</td>
<td>logical of length 1: Tells R how to use the function from slot startPar in case of a kStepEstimator—use it as is or to compute the starting point for a minimum distance estimator which in turn then serves as starting point for roptest / robust (from package ROptEst). If TRUE (default) the latter alternative is used. Ignored if ROptEst is not used.</td>
</tr>
</tbody>
</table>
**Details**

If `name` is missing, the default “L2 differentiable parametric family of probability measures” is used. In case `distrSymm` is missing it is set to `NoSymmetry()`. If `param` is missing, the parameter is created via `main`, `nuisance` and `trafo` as described in `ParamFamParameter`. In case `L2derivSymm` is missing, it is filled with an object of class `FunSymmList` with entries `NonSymmetric()`. In case `L2derivDistr` is missing, it is computed via `imageDistr`. If `L2derivDistrSymm` is missing, it is set to an object of class `DistrSymmList` with entries `NoSymmetry()`. In case `FisherInfo` is missing, it is computed from `L2deriv` using `E`.

**Value**

Object of class "L2ParamFamily"

**Author(s)**

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

**References**


**See Also**

`L2ParamFamily-class`

**Examples**

```r
F1 <- L2ParamFamily()
plot(F1)
```

**Description**

Class of L2 differentiable parametric families.

**Objects from the Class**

Objects can be created by calls of the form `new("L2ParamFamily", ...)`. More frequently they are created via the generating function `L2ParamFamily`.
Slots

name [inherited from class "ProbFamily"] object of class "character": name of the family.
distribution [inherited from class "ProbFamily"] object of class "Distribution": member of the family.
distrSymm [inherited from class "ProbFamily"] object of class "DistributionSymmetry": symmetry of distribution.
param [inherited from class "ParamFamily"] object of class "ParamFamParameter": parameter of the family.
fam.call [inherited from class "ParamFamily"] object of class "call": call by which parametric family was produced.
makeOKPar [inherited from class "ParamFamily"] object of class "function": has argument param — the (total) parameter, returns valid parameter; used if optim resp. optimize — try to use "illegal" parameter values; then makeOKPar makes a valid parameter value out of the illegal one.
startPar [inherited from class "ParamFamily"] object of class "function": has argument x — the data, returns starting parameter for optim resp. optimize — a starting estimator in case parameter is multivariate or a search interval in case parameter is univariate.
modifyParam [inherited from class "ParamFamily"] object of class "function": mapping from the parameter space (represented by "param") to the distribution space (represented by "distribution").
props [inherited from class "ProbFamily"] object of class "character": properties of the family.
L2deriv object of class "EuclRandVariable": L2 derivative of the family.
L2deriv.fct object of class "function": mapping from the parameter space (argument param of class "ParamFamParameter") to a mapping from observation x to the value of the L2derivative; L2deriv.fct is then used from observation x to value of the L2derivative; L2deriv.fct is used by modifyModel to move the L2deriv according to a change in the parameter
L2derivSymm [ object of class "FunSymmList": symmetry of the maps included in L2deriv.
L2derivDistr object of class "OptionalDistrListOrCall" (i.e., NULL or an object of class "DistrList" or the respective call to generate the latter object): if non-null and non-call, a list which includes the distribution of L2deriv.
L2derivDistrSymm object of class "DistrSymmList": symmetry of the distributions included in L2derivDistr.
FisherInfo.fct object of class "function": mapping from the parameter space (argument param of class "ParamFamParameter") to the set of positive semidefinite matrices; FisherInfo.fct is used by modifyModel to move the Fisher information according to a change in the parameter
FisherInfo object of class "PosDefSymmMatrix": Fisher information of the family.
.withEvalL2derivDistr logical of length one: if TRUE slot L2derivDistr gets evaluated, otherwise it is only kept as call.

Extends

Class "ParamFamily", directly.
Class "ProbFamily", by class "ParamFamily".
Methods

L2deriv signature(object = "L2ParamFamily"): accessor function for L2deriv.

\texttt{L2deriv} signature(object = "L2ParamFamily", param = "ParamFamParameter"): returns the L2derivative at param, i.e. evaluates slot function L2deriv.fct at param.

L2derivSymm signature(object = "L2ParamFamily"): accessor function for L2derivSymm.

L2derivDistr signature(object = "L2ParamFamily"): accessor function for L2derivDistr.

L2derivDistrSymm signature(object = "L2ParamFamily"): accessor function for L2derivDistrSymm.

FisherInfo signature(object = "L2ParamFamily"): accessor function for FisherInfo.

\texttt{FisherInfo} signature(object = "L2ParamFamily", param = "ParamFamParameter"): returns the Fisher Information at param, i.e. evaluates slot function FisherInfo.fct at param.

checkL2deriv signature(object = "L2ParamFamily"): check centering of L2deriv and compute precision of Fisher information.

\texttt{E} signature(object = "L2ParamFamily", fun = "EuclRandVariable", cond = "missing"): expectation of fun under the distribution of object.

\texttt{E} signature(object = "L2ParamFamily", fun = "EuclRandMatrix", cond = "missing"): expectation of fun under the distribution of object.

\texttt{E} signature(object = "L2ParamFamily", fun = "EuclRandVarList", cond = "missing"): expectation of fun under the distribution of object.

\texttt{plot} signature(x = "L2ParamFamily"): plot of distribution and L2deriv. More precisely, this method has arguments

\begin{verbatim}
plot(x, withSweave = getdistrOption("withSweave"), main = FALSE, 
     where
     x object of class "L2ParamFamily"
     withSweave logical: if TRUE (for working with Sweave) no extra device is opened and height/width
     are not set
     main logical: is a main title to be used? or
     just as argument main in plot.default.
     inner logical: do panels have their own titles? or
     character vector of / cast to length 'number of plotted panels' with the corresponding
     panel titles. For further information, see also plot and the description of argument main
     in plot.default.
     sub logical: is a sub-title to be used? or
     just as argument sub in plot.default.
     tmar top margin – useful for non-standard main title sizes
     bmar bottom margin – useful for non-standard sub title sizes
     cex.inner magnification to be used for inner titles relative to the current setting of cex; as in
     par; can be a vector of length 2; in this case the first component is for the distribution
     panels, the second for the L2-derivative-panels.
     col.inner character or integer code; color for the inner title
     mfColRow shall default partition in panels be used — defaults to TRUE
     to.draw.arg Either NULL (default; everything is plotted) or a vector of either integers (the
     indices of the subplots to be drawn) or characters — the names of the subplots to be
     drawn: these names are to be chosen among c("d","p","q", dimms) where dimms
     is either the row names of the trafo matrix rownames(trafo(x@param)) or if the last
\end{verbatim}
expression is NULL a vector "dim<dimnr>", dimnr running through the number of rows of the trafo matrix.

... additional arguments for plot — see plot, plot.default, plot.stepfun

if ... contains argument ylim, this may either be as in plot.default (i.e. a vector of length 2) or a vector of length 4, where the first two elements are the values for ylim in panels "d.c" and "d.d", and the last two elements are the values for ylim resp. xlim in panels "p", "p.c", "p.d" and "q", "q.c", "q.d".

modifyModel signature(model = "L2ParamFamily", param = "ParamFamParameter"): moves the L2-parametric Family model to parameter param

Author(s)

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

References


See Also

L2ParamFamily, ParamFamily-class

Examples

F1 <- new("L2ParamFamily")
plot(F1)

## selection of subpanels for plotting
F2 <- L2LocationScaleFamily()
layout(matrix(c(1,2,3,3), nrow=2, byrow=TRUE))
plot(F2,mfColRow = FALSE,
  to.draw.arg=c("p","q","loc"))
plot(F2,mfColRow = FALSE, inner=list("empirical cdf","pseudo-inverse",
  "L2-deriv, loc.part"), to.draw.arg=c("p","q","loc"))

L2ScaleFamily Generating function for L2ScaleFamily-class

Description

Generates an object of class "L2ScaleFamily".
**L2ScaleFamily**

**Usage**

```r
L2ScaleFamily(scale = 1, loc = 0, name, centraldistribution = Norm(),
               locscalename = c("loc", "scale"), modParam, LogDeriv,
               L2derivDistr.0, FisherInfo.0, distrSymm, L2derivSymm,
               L2derivDistrSymm, trafo, .returnClsName = NULL)
```

**Arguments**

- `scale` positive number: scale parameter of the model
- `loc` numeric: location parameter of the model
- `name` character: name of the parametric family.
- `centraldistribution` object of class "AbscontDistribution": central distribution; we assume from the beginning, that centraldistribution is symmetric about 0
- `locscalename` a character vector of length 1 or 2 containing the names of the scale resp. of location and scale parameter; if length is 2, locscalename is either unnamed, then order must be c(scale,loc), or named, then names must be "loc" and "scale".
- `modParam` optional function: mapping from the parameter space (represented by "param") to the distribution space (represented by "distribution").
- `LogDeriv` function with argument x: the negative logarithmic derivative of the density of the central distribution; if missing, it is determined numerically using numeric differentiation.
- `L2derivDistr.0` object of class "UnivariateDistribution": distribution of the L2derivative at the central distribution
- `FisherInfo.0` object of class "PosSemDefSymmMatrix": Fisher information of the model at the "standard" parameter value
- `distrSymm` object of class "DistributionSymmetry": symmetry of distribution.
- `L2derivSymm` object of class "FunSymmList": symmetry of the maps contained in L2deriv
- `L2derivDistrSymm` object of class "DistrSymmList": symmetry of the distributions contained in L2derivDistr
- `trafo` matrix or function in param: transformation of the parameter
- `.returnClsName` the class name of the return value; by default this argument is NULL whereupon the return class will be L2ScaleFamily; but, internally, this generating function is also used to produce objects of class NormScaleFamily, ExpScaleFamily, and LnormScaleFamily.

**Details**

If name is missing, the default “L2 scale family” is used. The function modParam is optional. If it is missing, it is constructed from centraldistribution using the scale structure of the model. Slot param is filled accordingly with the argument trafo passed to L2ScaleFamily. In case
L2derivDistr.0 is missing, L2derivDistr is computed via imageDistr, else L2derivDistr is assigned L2derivDistr.0, coerced to "UnivariateDistributionList". In case FisherInfo.0 is missing, Fisher information is computed from L2deriv using E. If distrSymm is missing, it is set to symmetry about loc. If L2derivSymm is missing, it is set to no symmetry, and if L2derivDistrSymm is missing, it is set to no symmetry.

Value

Object of class "L2ScaleFamily"

Author(s)

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

References


See Also

L2ScaleFamily-class

Examples

F1 <- L2ScaleFamily()
plot(F1)

L2ScaleFamily-class  L2 differentiable parametric group family

Description

Class of L2 differentiable parametric group families.

Objects from the Class

Objects can be created by calls of the form new("L2ScaleFamily", ...). More frequently they are created via the generating function L2ScaleFamily.
Slots

  name [inherited from class "ProbFamily"] object of class "character": name of the family.
  distribution [inherited from class "ProbFamily"] object of class "Distribution": member of
the family.
  distrSymm [inherited from class "ProbFamily"] object of class "DistributionSymmetry": sym-
metry of distribution.
  param [inherited from class "ParamFamily"] object of class "ParamFamParameter": parameter of
the family.
  fam.call [inherited from class "ParamFamily"] object of class "call": call by which parametric
family was produced.
  makeOKPar [inherited from class "ParamFamily"] object of class "function": has argument param
—the (total) parameter, returns valid parameter; used if optim resp. optimize—try to use
"illegal" parameter values; then makeOKPar makes a valid parameter value out of the illegal
one.
  startPar [inherited from class "ParamFamily"] object of class "function": has argument x —
the data, returns starting parameter for optim resp. optimize—a starting estimator in case
parameter is multivariate or a search interval in case parameter is univariate.
  modifyParam [inherited from class "ParamFamily"] object of class "function": mapping from
the parameter space (represented by "param") to the distribution space (represented by "distribution").
  props [inherited from class "ProbFamily"] object of class "character": properties of the family.
  L2deriv [inherited from class "L2ParamFamily"] object of class "EuclRandVariable": L2 deriva-
tive of the family.
  L2deriv.fct [inherited from class "L2ParamFamily"] object of class "function": mapping from
the parameter space (argument param of class "ParamFamParameter") to a mapping from
observation x to the value of the L2derivative; L2deriv.fct is then used from observation
x to value of the L2derivative; L2deriv.fct is used by modifyModel to move the L2deriv
according to a change in the parameter
  L2derivSymm [inherited from class "L2ParamFamily"] object of class "FunSymmList": symmetry
of the maps included in L2deriv.
  L2derivDistr [inherited from class "L2ParamFamily"] object of class "UnivarDistrList": list
which includes the distribution of L2deriv.
  L2derivDistrSymm [inherited from class "L2ParamFamily"] object of class "DistrSymmList": sym-
metry of the distributions included in L2derivDistr.
  FisherInfo.fct [inherited from class "L2ParamFamily"] object of class "function": mapping
from the parameter space (argument param of class "ParamFamParameter") to the set of
positive semidefinite matrices; FisherInfo.fct is used by modifyModel to move the Fisher
information according to a change in the parameter
  FisherInfo [inherited from class "L2ParamFamily"] object of class "PosDefSymmMatrix": Fisher
information of the family.
  LogDeriv [inherited from class "L2GroupParamFamily"] object of class "function": has argu-
ment x; the negative logarithmic derivative of the density of the model distribution at the
"standard" parameter value.
  locscalename [inherited from class "L2LocationScaleUnion"] object of class "character":
names of location and scale parameter
Extends

Class "L2LocationScaleUnion", directly.
Class "L2GroupParamFamily", by class "L2LocationScaleUnion".
Class "L2ParamFamily", by class "L2GroupParamFamily".
Class "ProbFamily", by class "ParamFamily".

Methods

modifyModel signature(model = "L2ScaleFamily", param = "ParamFamParameter"):
   moves the L2-scale family model to parameter param

Author(s)

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

References


See Also

L2ScaleFamily, ParamFamily-class

Examples

F1 <- new("L2ScaleFamily")
plot(F1)

L2ScaleUnknownLocationFamily

Generating function for L2LocationScaleFamily-class in nuisance situation

Description

Generates an object of class "L2LocationScaleFamily" in the situation where scale is main, location nuisance parameter.

Usage

L2ScaleUnknownLocationFamily(loc = 0, scale = 1, name, centraldistribution = Norm(),
locscalename = c("loc", "scale"), modParam, LogDeriv,
L2derivDistr.0, FisherInfo.0, distrSymm, L2derivSymm,
L2derivDistrSymm, trafo, .returnClsName = NULL)
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>loc</td>
<td>numeric: location parameter of the model.</td>
</tr>
<tr>
<td>scale</td>
<td>positive number: scale of the model.</td>
</tr>
<tr>
<td>name</td>
<td>character: name of the parametric family.</td>
</tr>
<tr>
<td>centraldistribution</td>
<td>object of class &quot;AbscontDistribution&quot;: central distribution; we assume by default, that centraldistribution is symmetric about 0</td>
</tr>
<tr>
<td>modParam</td>
<td>optional function: mapping from the parameter space (represented by &quot;param&quot;) to the distribution space (represented by &quot;distribution&quot;).</td>
</tr>
<tr>
<td>locscalename</td>
<td>a character vector of length 2 containing the names of the location and scale parameter; either unnamed, then order must be c(loc, scale), or named, then names must be &quot;loc&quot; and &quot;scale&quot;</td>
</tr>
<tr>
<td>LogDeriv</td>
<td>function with argument x: the negative logarithmic derivative of the density of the central distribution; if missing, it is determined numerically using numeric differentiation.</td>
</tr>
<tr>
<td>L2derivDistr.0</td>
<td>list of length 2 of objects of class &quot;UnivariateDistribution&quot;: (marginal) distributions of the coordinates of the L2derivative at the central distribution</td>
</tr>
<tr>
<td>FisherInfo.0</td>
<td>object of class &quot;PosSemDefSymmMatrix&quot;: Fisher information of the model at the &quot;standard&quot; parameter value</td>
</tr>
<tr>
<td>distrSymm</td>
<td>object of class &quot;DistributionSymmetry&quot;: symmetry of distribution.</td>
</tr>
<tr>
<td>L2derivSymm</td>
<td>object of class &quot;FunSymmList&quot;: symmetry of the maps contained in L2deriv</td>
</tr>
<tr>
<td>L2derivDistrSymm</td>
<td>object of class &quot;DistrSymmList&quot;: symmetry of the distributions contained in L2derivDistr</td>
</tr>
<tr>
<td>trafo</td>
<td>matrix or function in param: transformation of the parameter</td>
</tr>
<tr>
<td>.returnClsName</td>
<td>the class name of the return value; by default this argument is NULL whereupon the return class will be L2LocationScaleFamily; but, internally, this generating function is also used to produce objects of class NormalLocationScaleFamily, CauchyLocationScaleFamily.</td>
</tr>
</tbody>
</table>

Details

If name is missing, the default “L2 scale family with unknown location (as nuisance)” is used. The function modParam is optional. If it is missing, it is constructed from centraldistribution using the location and scale structure of the model. Slot param is filled accordingly with the argument trafo passed to L2ScaleUnknownLocationFamily. In case L2derivDistr.0 is missing, L2derivDistr is computed via imageDistr, else L2derivDistr is assigned L2derivDistr.0, coerced to "UnivariateDistributionList". In case FisherInfo.0 is missing, Fisher information is computed from L2deriv using E. If distrSymm is missing, it is set to symmetry about loc. If L2derivSymm is missing, its location and scale components are set to no symmetry, respectively. If L2derivDistrSymm is missing, its location and scale components are set to no symmetry, respectively.

Value

Object of class "L2LocationScaleFamily"
Author(s)

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

References


See Also

L2LocationScaleFamily-class

Examples

F1 <- L2ScaleUnknownLocationFamily()
plot(F1)

---

LnormScaleFamily Generating function for lognormal scale families

Description

Generates an object of class "L2ScaleFamily" which represents a lognormal scale family.

Usage

LnormScaleFamily(meanlog = 0, sdlog = 1, trafo)

Arguments

meanlog mean of the distribution on the log scale
sdlog standard deviation of the distribution on the log scale
trafo matrix: transformation of the parameter

Details

The slots of the corresponding L2 differentiable parameteric family are filled.

Value

Object of class "L2ScaleFamily"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>
mceCalc-methods

References


See Also

L2ParamFamily-class, Lnorm-class

Examples

(L1 <- LnormScaleFamily())
plot(L1)
Map(L2deriv(L1)[[1]])
checkL2deriv(L1)

mceCalc-methods

Methods for functions mceCalc and mleCalc in Package ‘distrMod’

Description

Methods for functions mceCalc and mleCalc in package distrMod:

Usage

mceCalc(x, PFam, ...)
mleCalc(x, PFam, ...)
## S4 method for signature 'numeric,ParamFamily'
mceCalc(x, PFam, criterion,
    startPar = NULL, penalty = 1e20, crit.name,
    Infos = NULL, validity.check = TRUE,
    withthetaPar = FALSE,...)
## S4 method for signature 'numeric,ParamFamily'
mleCalc(x, PFam, startPar = NULL,
    penalty = 1e20, Infos = NULL,
    validity.check = TRUE,...)
## S4 method for signature 'numeric,BinomFamily'
mleCalc(x, PFam, ...)
## S4 method for signature 'numeric,PoisFamily'
mleCalc(x, PFam, ...)
## S4 method for signature 'numeric,NormLocationFamily'
mleCalc(x, PFam, ...)
## S4 method for signature 'numeric,NormScaleFamily'
mleCalc(x, PFam, ...)
## S4 method for signature 'numeric,NormLocationScaleFamily'
mleCalc(x, PFam, ...)
Arguments

- **x**: numeric; data at which to evaluate the estimator
- **pfam**: an object of class ParamFamily; the parametric family at which to evaluate the estimator
- **criterion**: a function measuring the “goodness of fit”
- **startPar**: in case optim is used: a starting value for the parameter fit; in case optimize is used: a vector containing a search interval for the (one-dim) parameter
- **penalty**: numeric; penalizes non-permitted parameter values
- **crit.name**: character; the name of the criterion; may be missing
- **withthetaPar**: logical; shall Parameter theta be transmitted?
- **Infos**: matrix; info slot to be filled in object of class MCEstimate; may be missing
- **validity.check**: logical: shall return parameter value be checked for validity?
- **...**: additional argument(s) for optim / optimize

Details

mceCalc is used internally by function MCEstimator to allow for method dispatch according to argument PFam; similarly, and for the same purpose mleCalc is used internally by function MLEstimator. This way we / or any other developer can write particular methods for special cases where we may avoid using numerical optimization without interfering with existing code. For programming one’s own mleCalc / mceCalc methods, there is the helper function meRes to produce consistent return values.

Value

a list with components

- **estimate**: — the estimate as a named vector of numeric
- **criterion**: — the criterion value (i.e.: a numeric of length 1); e.g. the neg. log likelihood
- **est.name**: — the name of the estimator
- **param**: — estimate coerced to class ParamFamParameter
- **crit.fct**: — a function with the named components of theta as arguments returning the criterion value; used for profiling / coercing to class mle
- **method**: — a character reporting how the estimate was obtained, i.e., by optim, by optimize or by explicit calculations
- **crit.name**: character; the name of the criterion; may be ""
- **Infos**: matrix; info slot to be filled in object of class MCEstimate; may be NULL
- **samplesize**: numeric; sample size of x
Description

Class of minimum criterion estimates.

Objects from the Class

Objects can be created by calls of the form `new("MCEstimate", ...)`. More frequently they are created via the generating functions MCEstimator, MDEstimator or MLEstimator.

Slots

name Object of class "character": name of the estimator.
estimate Object of class "ANY": estimate.
estimate.call Object of class "call": call by which estimate was produced.
criterion Object of class "numeric": minimum value of the considered criterion.
criterion.fct Object of class "function": the considered criterion function; used for compatibility with class "mle" from package `stats4`: should be a function returning the criterion; i.e. a numeric of length 1 and should have as arguments all named components of argument
untransformed.estimate

method Object of class "character": the method by which the estimate was calculated, i.e.; "optim", "optimize", or "explicit calculation"; used for compatibility with class "mle" from package `stats4`, could be any character value.

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

optimwarn object of class "character" warnings issued during optimization.

startPar — object of class "ANY": filled either with NULL (no starting value used) or with "numeric" — the value of the starting parameter.

asvar object of class "OptionalMatrix" which may contain the asymptotic (co)variance of the estimator.
samplesize object of class "numeric" — the samplesize at which the estimate was evaluated.
nuis.idx object of class "OptionalNumeric": indices of estimate belonging to the nuisance part

fixed object of class "OptionalNumeric": the fixed and known part of the parameter.

trafo object of class "list": a list with components fct and mat (see below).

untransformed.estimate Object of class "ANY": untransformed estimate.

untransformed.asvar object of class "OptionalNumericOrMatrix" which may contain the asymptotic (co)variance of the untransformed estimator.

completecases object of class "logical" — complete cases at which the estimate was evaluated.

startPar object of class "ANY": usually filled with argument startPar of generating function MCEstimator, MDEstimator, MLEstimator.
### Extends

Class "Estimate", directly.

### Methods

- **criterion** signature(object = "MCEstimate"): accessor function for slot criterion.
- **criterion<-** signature(object = "MCEstimate"): replacement function for slot criterion.
- **optimwarn** signature(object = "MCEstimate"): accessor function for slot optimwarn.
- **startPar** signature(object = "MCEstimate"): accessor function for slot startPar.
- **criterion.fct** signature(object = "MCEstimate"): accessor function for slot criterion.fct.
- **show** signature(object = "Estimate")
- **coerce** signature(from = "MCEstimate", to = "mle"): create a "mle" object from a "MCEstimate" object
- **profile** signature(fitted = "MCEstimate"): coerces fitted to class "mle" and then calls the corresponding profile-method from package stats4; for details we confer to the corresponding man page.

### Author(s)

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

### See Also

- `Estimate-class`, `MCEstimator`, `MDEstimator`, `MLEstimator`

### Examples

```r
## (empirical) Data
x <- rgamma(50, scale = 0.5, shape = 3)

## parametric family of probability measures
G <- GammaFamily(scale = 1, shape = 2)

MDEstimator(x, G)
(m <- MLEstimator(x, G))
m.mle <- as(m,"mle")
par(mfrow=c(1,2))
profileM <- profile(m)
## plot-profile throws an error
```
MCEstimator  

*Function to compute minimum criterion estimates*

**Description**

The function MCEstimator provides a general way to compute estimates for a given parametric family of probability measures which can be obtained by minimizing a certain criterion. For instance, the negative log-Likelihood in case of the maximum likelihood estimator or some distance between distributions like in case of minimum distance estimators.

**Usage**

```r
MCEstimator(x, ParamFamily, criterion, crit.name, 
startPar = NULL, Infos, trafo = NULL, 
penalty = 1e20, validity.check = TRUE, asvar.fct, na.rm = TRUE, 
..., .withEvalAsVar = TRUE)
```

**Arguments**

- `x` (empirical) data
- `ParamFamily` object of class "ParamFamily"
- `criterion` function: criterion to minimize; see Details section.
- `crit.name` optional name for criterion.
- `startPar` initial information used by optimize resp. optim; i.e.; if (total) parameter is of length 1, `startPar` is a search interval, else it is an initial parameter value; if NULL slot `startPar` of `ParamFamily` is used to produce it; in the multivariate case, `startPar` may also be of class Estimate, in which case slot untransformed.estimate is used.
- `Infos` character: optional informations about estimator
- `trafo` an object of class MatrixorFunction – a transformation for the main parameter
- `penalty` (non-negative) numeric: penalizes non valid parameter-values
- `validity.check` logical: shall return parameter value be checked for validity? Defaults to yes (TRUE)
- `asvar.fct` optionally: a function to determine the corresponding asymptotic variance; if given, `asvar.fct` takes arguments `L2Fam((the parametric model as object of class L2ParamFamily)))` and `param` (the parameter value as object of class `ParamFamParameter`); arguments are called by name; asvar.fct may also process further arguments passed through the ... argument
- `na.rm` logical: if TRUE, the estimator is evaluated at `complete.cases(x)`.
- `...` further arguments to criterion or optimize or optim, respectively.
- `.withEvalAsVar` logical: shall slot asVar be evaluated (if asvar.fct is given) or just the call be returned?
Details

The argument criterion has to be a function with arguments the empirical data as well as an object of class "Distribution" and possibly .... Uses mceCalc for method dispatch.

Value

An object of S4-class "MCEestimate" which inherits from class "Estimate".

Note

The criterion function may be called together with a parameter thetaPar which is the current parameter value under consideration, i.e.: the value under which the model distribution is considered. Hence, if desired, particular criterion functions could make use of this information, by, say computing the criterion differently for different parameter values.

Author(s)

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

See Also

ParamFamily-class,ParamFamily,MCEestimate-class

Examples

```r
## (empirical) Data
x <- rgamma(50, scale = 0.5, shape = 3)

## parametric family of probability measures
G <- GammaFamily(scale = 1, shape = 2)

## Maximum Likelihood estimator
## Note: you can directly use function MLEstimator!
negLogLikelihood <- function(x, Distribution){
  res <- -sum(log(Distribution@d(x)))
  names(res) <- "Negative Log-Likelihood"
  return(res)
}
MCEstimator(x = x, ParamFamily = G, criterion = negLogLikelihood)

## Kolmogorov-Smirnov minimum distance estimator
## Note: you can also use function MDEstimator!
MCEstimator(x = x, ParamFamily = G, criterion = KolmogorovDist,
             crit.name = "Kolmogorov distance")

## Total variation minimum distance estimator
## Note: you can also use function MDEstimator!
## discretize Gamma distribution
MCEstimator(x = x, ParamFamily = G, criterion = TotalVarDist,
            crit.name = "Total variation distance")
```
## MDEstimator

Function to compute minimum distance estimates

### Description

The function MDEstimator provides a general way to compute minimum distance estimates.

### Usage

```r
MDEstimator(x, ParamFamily, distance = KolmogorovDist, dist.name, 
paramDepDist = FALSE, startPar = NULL, Infos, trafo = NULL, 
penalty = 1e20, validity.check = TRUE, asvar.fct, na.rm = TRUE, 
..., .withEvalAsVar = TRUE)
```

### Arguments

- **x** (empirical) data
- **ParamFamily** object of class "ParamFamily"
- **distance** (generic) function: to compute distance between (empirical) data and objects of class "Distribution".
- **dist.name** optional name of distance
- **paramDepDist** logical: will computation of distance be parameter dependent (see also note below)? if TRUE, distance function must be able to digest a parameter thetaPar; otherwise this parameter will be eliminated if present in ...-argument.
- **startPar** initial information used by optimize resp. optim; i.e: if (total) parameter is of length 1, startPar is a search interval, else it is an initial parameter value; if NULL slot startPar of ParamFamily is used to produce it; in the multivariate case, startPar may also be of class Estimate, in which case slot untransformed.estimate is used.
- **Infos** character: optional informations about estimator
MDEstimator

trafo

penalty

validity.check

asvar.fct

na.rm

... withEvalAsVar

Details

The argument distance has to be a (generic) function with arguments the empirical data as well as an object of class "Distribution" and possibly ...; e.g. KolmogorovDist (default), TotalVarDist or HellingerDist. Uses mceCalc for method dispatch.

Value

An object of S4-class "MCEstimate" which inherits from class "Estimate".

Note

The distance function may be called together with a parameter thetapar which is the current parameter value under consideration, i.e.: the value under which the model distribution is considered. Hence, if desired, particular distance functions could make use of this information, by, say computing the distance differently for different parameter values.

Author(s)

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

References


See Also

ParamFamily-class, ParamFamily, MCEstimator, MCEstimate-class, fitdistr
Examples

```r
## (empirical) Data
x <- rgamma(50, scale = 0.5, shape = 3)

## parametric family of probability measures
G <- GammaFamily(scale = 1, shape = 2)

## Kolmogorov-Smirnov minimum distance estimator
MDEstimator(x = x, ParamFamily = G, distance = KolmogorovDist)

## von Mises minimum distance estimator with default mu
MDEstimator(x = x, ParamFamily = G, distance = CvMDist)

## Total variation minimum distance estimator
## gamma distributions are discretized
MDEstimator(x = x, ParamFamily = G, distance = TotalVarDist)
## or smoothing of empirical distribution (takes some time!)
## MDEstimator(x = x, ParamFamily = G, distance = TotalVarDist, asis.smooth.discretize = "smooth")

## Hellinger minimum distance estimator
## gamma distributions are discretized
distroptions(DistrResolution = 1e-10)
MDEstimator(x = x, ParamFamily = G, distance = HellingerDist, startPar = c(1,2))
distroptions(DistrResolution = 1e-6) # default
## or smoothing of empirical distribution (takes some time!)
## MDEstimator(x = x, ParamFamily = G, distance = HellingerDist, asis.smooth.discretize = "smooth")
```

Description

helper functions to produce consistent lists to be digested in functions `mceCalc` and `mleCalc`

Usage

```r
meRes(x, estimate, criterion.value, param, crit.fct, method = "explicit solution",
       crit.name = "Maximum Likelihood", Infos, warns = ",", startPar = NULL)
get.criterion.fct(theta, Data, ParamFam, criterion.ff, fun, ...)
## S4 method for signature 'numeric'
samplesize(object)
```
Arguments

- **x**: numeric; the data at which to evaluate the estimate
- **estimate**: numeric; the estimate
- **criterion.value**: numeric; the value of the criterion
- **param**: object of class `ParamFamilyParameter`; the parameter value
- **crit.fct**: a function to fill slot `minuslogl` when an object of class `MCEstimate` is coerced to class `mle` (from package `stats4`); this way function `get.criterion.fct` (also see details below) is helpful (at least if the dimension of the estimator is larger than 1).
- **method**: character; describes how the estimate was obtained
- **crit.name**: character; name of the criterion
- **Infos**: optional matrix of characters in two columns; information to be attached to the estimate
- **warns**: collected warnings in optimization
- **samplesize**: numeric; the sample size at which the estimator was evaluated
- **theta**: the parameter value as named numeric vector
- **Data**: numeric; the data at which to evaluate the MCE
- **ParamFam**: an object of class `ParamFamily`; the parametric family at which to evaluate the MCE
- **criterion.ff**: the criterion function used in the MCE
- **fun**: wrapper to the criterion function used in the MCE (with certain checking whether parameter value is permitted and possibly penalizing if not; see code to , for example.)
- **startPar**: value of argument StartPar — starting parameter used.
- **...**: further arguments to be passed to `optim`/`optimize`
- **object**: numeric; the data at which to evaluate the estimate

Details

`get.criterion.fct` produces a function `criterion.fct` to fill slot `minuslogl` when an object of class `MCEstimate` is coerced to class `mle` (from package `stats4`); this way we may use profiling methods introduced there also for objects of our classes. More specifically, we produce a function where all coordinates/components of `theta` appear as separate named arguments, which then calls `fun` with these separate arguments again stacked to one (named) vector argument;

`samplesize` determines the samplesize of argument `object`, i.e.; if `object` has an attribute `dim`, it returns `dim(object)[2]`, else `length(object)`.

Value

- **meRes**: a list of prescribed structure to be digested in functions `mceCalc` and `mleCalc` by the internal helper function `process.meCalcRes`.
- **get.criterion.fct**: a function; see details below;
- **samplesize**: numeric
Function to compute maximum likelihood estimates

Description

The function MLEstimator provides a general way to compute maximum likelihood estimates for a given parametric family of probability measures. This is done by calling the function MCEstimator which minimizes the negative log-Likelihood.

Usage

MLEstimator(x, ParamFamily, startPar = NULL,
            Infos, trafo = NULL, penalty = 1e20,
            validity.check = TRUE, na.rm = TRUE, ..., 
            .withEvalAsVar = TRUE)

Arguments

- **x**: (empirical) data
- **ParamFamily**: object of class "ParamFamily"
- **startPar**: initial information used by optimize resp. optim; i.e; if (total) parameter is of length 1, startPar is a search interval, else it is an initial parameter value; if NULL slot startPar of ParamFamily is used to produce it; in the multivariate case, startPar may also be of class Estimate, in which case slot untransformed.estimate is used.
- **Infos**: character: optional informations about estimator
- **trafo**: an object of class MatrixOrFunction – a transformation for the main parameter
- **penalty**: (non-negative) numeric: penalizes non valid parameter-values
- **validity.check**: logical: shall return parameter value be checked for validity? Defaults to yes (TRUE)
- **na.rm**: logical: if TRUE, the estimator is evaluated at complete.cases(x).
- **...**: further arguments to criterion or optimize or optim, respectively.
- **.withEvalAsVar**: logical: shall slot asVar be evaluated (if asvar.fct is given) or just the call be returned?

Details

The function uses mleCalc for method dispatch; this method by default calls mceCalc using the negative log-likelihood as criterion which should be minimized.
Value
An object of S4-class "MCEstimate" which inherits from class "Estimate".

Author(s)
Matthias Kohl <Matthias.Kohl@stamats.de>,
Peter Ruckdeschel <Peter.Ruckdeschel@itwm.fraunhofer.de>

See Also
ParamFamily-class, ParamFamily, MCEstimator, MCEstimate-class, fitdistr.mle

Examples

### Binomial data
```r
# (empirical) data
x <- rbinom(100, size=25, prob=.25)

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

### Poisson data
```r
# 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, 488), 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
MLEstimator(x, PoisFamily())
```

### Normal (Gaussian) location and scale
```r
# (empirical) data
x <- rnorm(100)

# ML-estimate
MLEstimator(x, NormLocationScaleFamily())

# compare:
c(mean(x), sd(x))
```

### Gamma model
```r
# (empirical) data
```
MLEstimator

x <- rgamma(50, scale = 0.5, shape = 3)

## parametric family of probability measures
G <- GammaFamily(scale = 1, shape = 2)

## Maximum likelihood estimator
(res <- MLEstimator(x = x, ParamFamily = G))

## Asymptotic (CLT-based) confidence interval
confint(res)

## some profiling
par(mfrow=c(1,2))
plot(profile(res))
par(mfrow=c(1,1))

## implementation of ML-estimator of package MASS
require(MASS)
(res1 <- fitdistr(x, "gamma"))

## comparison
## shape
estimate(res)[2]
## rate
1/estimate(res)[1]

## minor differences due to the fact that by default, fitdistr uses
## BFGS, while we use Nelder-Mead instead

## log-likelihood
res1$loglik
## negative log-likelihood
criterion(res)

## explicitly transforming to
## MASS parametrization:
mtrafo <- function(x){
  nms0 <- names(c(main(param(G)), nuisance(param(G))))
  nms <- c("shape","rate")
  fval0 <- c(x[2], 1/x[1])
  names(fval0) <- nms
  mat0 <- matrix( c(0, -1/x[1]^2, 1, 0), nrow = 2, ncol = 2,
                  dimnames = list(nms, nms0))
  list(fval = fval0, mat = mat0))

G2 <- G
trafo(G2) <- mtrafo
res2 <- MLEstimator(x = x, ParamFamily = G2)

old <- getdistrModOption("show.details")
distrModoptions("show.details" = "minimal")
res1
Methods for function `modifyModel` in Package `distrMod`

Description

Methods for function `modifyModel` in package `distrMod`; `modifyModel` moves a model from one parameter value to another.

Usage

```r
modifyModel(model, param,...)
## S4 method for signature 'ParamFamily,ParamFamParameter'
modifyModel(model,param,
    .withCall = TRUE, ...)
## S4 method for signature 'L2ParamFamily,ParamFamParameter'
modifyModel(model,param,
    .withCall = TRUE, .withL2derivDistr = TRUE, ...)
## S4 method for signature 'L2LocationFamily,ParamFamParameter'
modifyModel(model,param, ...)
## S4 method for signature 'L2ScaleFamily,ParamFamParameter'
modifyModel(model,param, ...)
## S4 method for signature 'L2LocationScaleFamily,ParamFamParameter'
modifyModel(model,
    param, ...)
## S4 method for signature 'GammaFamily,ParamFamParameter'
modifyModel(model,param, ...)
```
modifyModel(model, param, ...)  

Arguments  

model an object of class ParamFamily — the model to move.  
param an object of class ParamFamParameter — the parameter to move to.  
.withCall logical: shall slot fam.call be updated?  
.withL2derivDistr logical: shall slot L2derivDistr be updated or just the call to do the updated be stored?  
... additional argument(s) for methods; not used so far  

Details  
modifyModel is merely used internally for moving the model along modified parameter values during a model fit.  
It generally simply copies the original model and only modifies the affected slots, i.e. distribution, the distribution of the observations, param, the parameter, L2deriv, the L2-derivative at the parameter, L2FisherInfo, the Fisher information at the parameter, the symmetry slots distrSymm, L2derivSymm, and L2derivDistrSymm, and, finally, L2derivDistr the (marginal) distribution(s) of the L2derivative. By default, also slot fam.call is updated.  
In case model is of class L2LocationFamily, L2ScaleFamily, or L2LocationScaleFamily, symmetry slots are updated to be centered about the median of the (central) distribution (assuming the latter is symmetric about the median); as an intermediate step, these methods call the general modifyModel-method for signature L2ParamFamily; in this call, however, slot fam.call is not updated (this is the reason for argument .withCall); this is then done in the individual parts of the corresponding method.  

Value  

a corresponding instance of the model in argument model with moved parameters.  

Description  
NbinomFamily  
Generating function for Nbinomial families  

Generates an object of class "L2ParamFamily" which represents a Nbinomial family where the probability of success is the parameter of interest.  

Usage  

NbinomFamily(size = 1, prob = 0.5, trafo)  
NbinomwithSizeFamily(size = 1, prob = 0.5, trafo, withL2derivDistr = TRUE)  
NbinomMeanSizeFamily(size = 1, mean = 0.5, trafo, withL2derivDistr = TRUE)
Arguments

size number of trials
prob probability of success
mean alternative parameter for negative binomial parameter
trafo function in param or matrix: transformation of the parameter
withL2derivDistr logical: shall the distribution of the L2 derivative be computed? Defaults to TRUE; setting it to FALSE speeds up computations.

Details

The slots of the corresponding L2 differentiable parameteric family are filled. NbinomFamily assumes size to be known; while for NbinomwithSizeFamily it is a second (unknown) parameter; for NbinomMeanSizeFamily is like NbinomwithSizeFamily but uses the size,mean parametrization instead of the size,prob one.

Value

Object of class "L2ParamFamily"

Author(s)

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

References


See Also

L2ParamFamily-class, Nbinom-class

Examples

(N1 <- NbinomFamily(size = 25, prob = 0.25))
plot(N1)
FisherInfo(N1)
checkL2deriv(N1)
(N1.w <- NbinomwithSizeFamily(size = 25, prob = 0.25))
plot(N1.w)
FisherInfo(N1.w)
checkL2deriv(N1.w)
(N2.w <- NbinomMeanSizeFamily(size = 25, mean = 75))
plot(N2.w)
FisherInfo(N2.w)
checkL2deriv(N2.w)
negativeBias  Generating function for onesidedBias-class

Description

Generates an object of class "onesidedBias".

Usage

negativeBias(name = "negative Bias")

Arguments

name  name of the bias type

Value

Object of class "onesidedBias"

Author(s)

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

References


See Also

onesidedBias-class

Examples

negativeBias()

## The function is currently defined as

function(){ new("onesidedBias", name = "negative Bias", sign = -1) }
NonSymmetric-class

Generating function for NonSymmetric-class

Description

Generates an object of class "NonSymmetric".

Usage

NonSymmetric()

Value

Object of class "NonSymmetric"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

nonsymmetric-class, functionsymmetry-class

Examples

NonSymmetric()

```r
## The function is currently defined as
function(){
  new("NonSymmetric")
}
```

---

NonSymmetric-class  

Class for Non-symmetric Functions

Description

Class for non-symmetric functions.

Objects from the Class

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

Slots

type  Object of class "character": contains “non-symmetric function"

SymmCenter  Object of class "NULL"
Extends

Class "FunctionSymmetry", directly.
Class "Symmetry", by class "FunctionSymmetry".

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

NonSymmetric

Examples

new("NonSymmetric")

<table>
<thead>
<tr>
<th>norm</th>
<th>Norm functions</th>
</tr>
</thead>
</table>

Description

Functions to determine certain norms.

Usage

EuclideanNorm(x)
QuadFormNorm(x, A)

Arguments

x    vector or matrix; norm is determined columnwise
A    pos. semidefinite Matrix

Value

the columnwise evaluated norms

Author(s)

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

See Also

onesidedBias-class

Examples

mm <- matrix(rnorm(20),2,10)
EuclideanNorm(mm)
QuadFormNorm(mm, A = PosSemDefSymmMatrix(matrix(c(3,1,1,1),2,2)))
NormLocationFamily  Generating function for normal location families

Description
Generates an object of class "L2LocationFamily" which represents a normal location family.

Usage
NormLocationFamily(mean = 0, sd = 1, trafo)

Arguments
- mean  mean
- sd    standard deviation
- trafo function in param or matrix: transformation of the parameter

Details
The slots of the corresponding L2 differentiable parameteric family are filled.

Value
Object of class "L2LocationFamily"

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

References

See Also
L2ParamFamily-class, Norm-class

Examples
(N1 <- NormLocationFamily())
plot(N1)
L2derivDistr(N1)
NormLocationScaleFamily

Generating function for normal location and scale families

Description
Generates an object of class "L2LocationScaleFamily" which represents a normal location and scale family.

Usage
NormLocationScaleFamily(mean = 0, sd = 1, trafo)

Arguments
mean mean
sd standard deviation
trafo function in param or matrix: transformation of the parameter

Details
The slots of the corresponding L2 differentiable parameteric family are filled.

Value
Object of class "L2LocationScaleFamily"

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

References

See Also
L2ParamFamily-class, Norm-class

Examples
(N1 <- NormLocationScaleFamily())
plot(N1)
FisherInfo(N1)
checkL2deriv(N1)
NormLocationUnknownScaleFamily

Generating function for normal location families with unknown scale as nuisance

Description
Generates an object of class "L2LocationScaleFamily" which represents a normal location family with unknown scale as nuisance.

Usage
NormLocationUnknownScaleFamily(mean = 0, sd = 1, trafo)

Arguments
mean mean
sd standard deviation
trafo function in param or matrix: transformation of the parameter

Details
The slots of the corresponding L2 differentiable parametric family are filled.

Value
Object of class "L2LocationScaleFamily"

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

References

See Also
L2ParamFamily-class, Norm-class

Examples
(N1 <- NormLocationUnknownScaleFamily())
plot(N1)
FisherInfo(N1)
checkL2deriv(N1)
NormScaleFamily

Generating function for normal scale families

Description

Generates an object of class "L2ScaleFamily" which represents a normal scale family.

Usage

NormScaleFamily(sd = 1, mean = 0, trafo)

Arguments

sd  standard deviation
mean mean
trafo function in param or matrix: transformation of the parameter

Details

The slots of the corresponding L2 differentiable parameteric family are filled.

Value

Object of class "L2ScaleFamily"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

L2ParamFamily-class, Norm-class

Examples

(N1 <- NormScaleFamily())
plot(N1)
FisherInfo(N1)
checkL2deriv(N1)
NormScaleUnknownLocationFamily

Generating function for normal scale families with unknown location as nuisance

Description

Generates an object of class "L2LocationScaleFamily" which represents a normal scale family with unknown location as nuisance.

Usage

```R
NormScaleUnknownLocationFamily(sd = 1, mean = 0, trafo)
```

Arguments

- `mean`: mean
- `sd`: standard deviation
- `trafo`: function in `param` or matrix: transformation of the parameter

Details

The slots of the corresponding L2 differentiable parameteric family are filled.

Value

Object of class "L2LocationScaleFamily"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

- `L2ParamFamily-class`
- `Norm-class`

Examples

```R
(N1 <- NormScaleUnknownLocationFamily())
plot(N1)
FisherInfo(N1)
checkL2deriv(N1)
```
Description

Generates an object of class "NormType".

Usage

```
NormType(name = "EuclideanNorm", fct = EuclideanNorm)
```

Arguments

- **name**: slot name of the class
- **fct**: slot fct of the class

Value

Object of class "NormType"

Author(s)

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

References


See Also

- `NormType-class`

Examples

```
NormType()
```
NormType-class

Norm Type

Description

Class of norm types.

Objects from the Class

Could be generated by `new("NormType")`: more frequently one will use the generating function
`NormType`

Slots

name Object of class "character".
fct Object of class "function" — the norm to be evaluated.

Methods

name signature(object = "NormType"): accessor function for slot name.
name<- signature(object = "NormType", value = "character"): replacement function for slot name.
fct signature(object = "NormType"): accessor function for slot fct.
fct<- signature(object = "NormType", value = "function"): replacement function for slot fct.

Author(s)

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

References


See Also

BiasType-class

Examples

EuclNorm <- NormType("EuclideanNorm",EuclideanNorm)
fct(EuclNorm)
name(EuclNorm)
OddSymmetric

Generating function for OddSymmetric-class

Description
Generates an object of class "OddSymmetric".

Usage
OddSymmetric(SymmCenter = 0)

Arguments
SymmCenter numeric: center of symmetry

Value
Object of class "OddSymmetric"

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

See Also
OddSymmetric-class, FunctionSymmetry-class

Examples
OddSymmetric()

## The function is currently defined as
function(SymmCenter = 0){
    new("OddSymmetric", SymmCenter = SymmCenter)
}

OddSymmetric-class Class for Odd Functions

Description
Class for odd functions.

Objects from the Class
Objects can be created by calls of the form new("OddSymmetric"). More frequently they are created via the generating function OddSymmetric.
Slots

- **type**: Object of class "character": contains “odd function"
- **SymmCenter**: Object of class "numeric": center of symmetry

Extends

- Class "FunctionSymmetry", directly.
- Class "Symmetry", by class "FunctionSymmetry".

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

- OddSymmetric, FunctionSymmetry-class

Examples

```r
new("OddSymmetric")
```

---

**onesidedBias-class**  
onesided Bias Type

Description

Class of onesided bias types.

Objects from the Class

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

Slots

- **name**: Object of class "character".
- **sign**: Object of class "numeric", to be in \{-1,1\} — whether bias is to be positive or negative

Methods

- **sign**: signature(object = "onesidedBias"): accessor function for slot sign.
- **sign<-**: signature(object = "onesidedBias", value = "numeric"): replacement function for slot sign.

Extends

Class "BiasType", directly.
Author(s)

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

References


See Also

BiasType-class

Examples

positiveBias()
## The function is currently defined as
function(){ new("onesidedBias", name = "positive Bias", sign = 1) }

negativeBias()
## The function is currently defined as
function(){ new("onesidedBias", name = "negative Bias", sign = -1) }

pB <- positiveBias()

sign(pB)
try(sign(pB) <- -2) ## error
sign(pB) <- -1

Surf

ParamFamily

Generating function for ParamFamily-class

Description

Generates an object of class "ParamFamily".

Usage

ParamFamily(name, distribution = Norm(), distrSymm, modifyParam,
   main = main(param), nuisance = nuisance(param),
   fixed = fixed(param), trafo = trafo(param),
   param = ParamFamParameter(name = paste("Parameter of",
                               name), main = main, nuisance = nuisance,
                               fixed = fixed, trafo = trafo),
   props = character(0),
   startPar = NULL, makeOKPar = NULL)
Arguments

name character string: name of family
distribution object of class "Distribution": member of the family
distrSymm object of class "DistributionSymmetry": symmetry of distribution.
startPar startPar is a function in the observations x returning initial information for MCEstimator used by optimize resp. optim; i.e; if (total) parameter is of length 1, startPar returns a search interval, else it returns an initial parameter value.
makeOKPar makeOKPar is a function in the (total) parameter param; used if optim resp. optimize--- try to use “illegal” parameter values; then makeOKPar makes a valid parameter value out of the illegal one; if NULL slot makeOKPar of ParamFamily is used to produce it.
main numeric vector: main parameter
nuisance numeric vector: nuisance parameter
fixed numeric vector: fixed part of the parameter
trafo function in param or matrix: transformation of the parameter
param object of class "ParamFamParameter": parameter of the family
modifyParam function: mapping from the parameter space (represented by "param") to the distribution space (represented by "distribution").
props character vector: properties of the family

Details

If name is missing, the default “"parametric family of probability measures"" is used. In case distrSymm is missing it is set to NoSymmetry(). If param is missing, the parameter is created via main, nuisance and trafo as described in ParamFamParameter. One has to specify a function which represents a mapping from the parameter space to the corresponding distribution space; e.g., in case of normal location a simple version of such a function would be function(theta){ norm(mean = theta) }.

Value

Object of class "ParamFamily"

Author(s)

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

See Also

ParamFamily-class
## Examples

```r
## "default" (normal location)
F1 <- ParamFamily(modifyParam = function(theta) { Norm(mean = theta) })
plot(F1)

# Some examples:
## 1. Normal location family
theta <- 0
names(theta) <- "mean"
NL <- ParamFamily(name = "Normal location family",
                 param = ParamFamParameter(name = "location parameter", main = theta),
                 distribution = Norm(mean = 0, sd = 1),
                 startPar = function(x,...) c(min(x),max(x)),
                 distrSymm <- SphericalSymmetry(SymmCenter = 0),
                 modifyParam = function(theta) { Norm(mean = theta, sd = 1) },
                 props = paste(c("The normal location family is invariant under",
                                "the group of transformations 'g(x) = x + mean'",
                                "with location parameter 'mean'"), collapse = " "))
NL

## 2. Normal scale family
theta <- 1
names(theta) <- "sd"
NS <- ParamFamily(name = "Normal scale family",
                 param = ParamFamParameter(name = "scale parameter", main = theta),
                 distribution = Norm(mean = 0, sd = 1),
                 startPar = function(x,...) c(0,-min(x)+max(x)),
                 distrSymm <- SphericalSymmetry(SymmCenter = 0),
                 modifyParam = function(theta) { Norm(mean = 0, sd = theta) },
                 props = paste(c("The normal scale family is invariant under",
                                "the group of transformations 'g(y) = sd*y'",
                                "with scale parameter 'sd'"), collapse = " "))
NS

## 3. Normal location and scale family
theta <- c(0, 1)
names(theta) <- c("mean", "sd")
NLS <- ParamFamily(name = "Normal location and scale family",
                   param = ParamFamParameter(name = "location and scale parameter", main = theta),
                   distribution = Norm(mean = 0, sd = 1),
                   startPar = function(x,...) c(median(x),mad(x)),
                   makeOKPar = function(param) {param[2]<-abs(param[2]); return(param)},
                   distrSymm <- SphericalSymmetry(SymmCenter = 0),
                   modifyParam = function(theta) {
                       Norm(mean = theta[1], sd = theta[2])
                   },
```
props = paste(c("The normal location and scale family is",
  "invariant under the group of transformations",
  "g(x) = sd*x + mean' with location parameter",
  "'mean' and scale parameter 'sd'"),
  collapse = " ")

NLS

## 4. Binomial family
theta <- 0.3
names(theta) <- "prob"
B <- ParamFamily(name = "Binomial family",
  param = ParamFamParameter(name = "probability of success",
    main = theta),
  startPar = function(x, ...) c(0,1),
  distribution = Binom(size = 15, prob = 0.3), # size known!
  modifyParam = function(theta){ Binom(size = 15, prob = theta) },
  props = paste(c("The Binomial family is symmetric with respect",
    "to prob = 0.5; i.e.,",
    "d(Binom(size, prob))(k)=d(Binom(size,1-prob))(size-k)"),
    collapse = " "))
B

## 5. Poisson family
theta <- 7
names(theta) <- "lambda"
P <- ParamFamily(name = "Poisson family",
  param = ParamFamParameter(name = "positive mean",
    main = theta),
  startPar = function(x, ...) c(0,max(x)),
  distribution = Pois(lambda = 7),
  modifyParam = function(theta){ Pois(lambda = theta) })
P

## 6. Exponential scale family
theta <- 2
names(theta) <- "scale"
ES <- ParamFamily(name = "Exponential scale family",
  param = ParamFamParameter(name = "scale parameter",
    main = theta,
    .returnClsName = "ParamWithScaleFamParameter"),
  startPar = function(x, ...) c(0,max(x)-min(x)),
  distribution = Exp(rate = 1/2),
  modifyParam = function(theta){ Exp(rate = 1/theta) },
  props = paste(c("The Exponential scale family is invariant under",
    "the group of transformations 'g(y) = scale*y'",
    "with scale parameter 'scale = 1/rate'"),
    collapse = " "))
ES

## 7. Lognormal scale family
theta <- 2
names(theta) <- "scale"
LS <- ParamFamily(name = "Lognormal scale family",
  param = ParamFamParameter(name = "scale parameter",
    main = theta,
.returnClsName = "ParamWithScaleFamParameter",
startPar = function(x,...) c(0,max(x)-min(x)),
distribution = Lnorm(meanlog = log(2), sdlog = 2),## sdlog known!
modifyParam = function(theta){
  Lnorm(meanlog = log(theta), sdlog = 2)
},
props = paste(c("The Lognormal scale family is invariant under",
  "the group of transformations 'g(y) = scale*y'",
  "with scale parameter 'scale = exp(meanlog)'"),
collapse = " "))
 LS

## 8. Gamma family
theta <- c(1, 2)
names(theta) <- c("scale","shape")
G <- ParamFamily(name = "Gamma family",
  param = ParamFamParameter(name = "scale and shape", main = theta,
    withPosRestr = TRUE,            
  .returnClsName = "ParamWithScaleAndShapeFamParameter"),
startPar = function(x,...) c(E <- mean(x); V <- var(x); c(V/E, E^2/V)),
makeOKPar = function(param) abs(param),
distribution = Gammad(scale = 1, shape = 2),
modifyParam = function(theta){
  Gammad(scale = theta[1], shape = theta[2])
},
props = paste(c("The Gamma family is scale invariant via the",
  "parametrization 
    '(nu,shape)=(log(scale),shape)'"),
collapse = " "))
G

---

**Description**

Class of parametric families of probability measures.

**Objects from the Class**

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

**Slots**

- `name` [inherited from class "ProbFamily"] object of class "character": name of the family.
- `distribution` [inherited from class "ProbFamily"] object of class "Distribution": member of the family.
- `distrSymm` [inherited from class "ProbFamily"] object of class "DistributionSymmetry": symmetry of distribution.
param object of class "ParamFamParameter": parameter of the family.

fam.call object of class "call": call by which parametric family was produced.

makeOKPar object of class "function": has argument param — the (total) parameter, returns valid parameter; used if optim resp. optimize— try to use “illegal” parameter values; then makeOKPar makes a valid parameter value out of the illegal one.

startPar object of class "function": has argument x — the data, returns starting parameter for optim resp. optimize— a starting estimator in case parameter is multivariate or a search interval in case parameter is univariate.

modifyParam object of class "function": mapping from the parameter space (represented by "param") to the distribution space (represented by "distribution").

props [inherited from class "ProbFamily"] object of class "character": properties of the family.

.withMDE object of class "logical" (of length 1): Tells R how to use the function from slot startPar in case of a kStepEstimator — use it as is or to compute the starting point for a minimum distance estimator which in turn then serves as starting point for rptest / robest (from package ROptEst). If TRUE (default) the latter alternative is used. Ignored if ROptEst is not used.

.withEvalAsVar object of class "logical" (of length 1): Tells R whether in determining kStepEstimators one evaluates the asymptotic variance or just produces a call to do so.

**Extends**

Class "ProbFamily", directly.

**Methods**

- **main** signature(object = "ParamFamily"): wrapped accessor function for slot main of slot param.
- **nuisance** signature(object = "ParamFamily"): wrapped accessor function for slot nuisance of slot param.
- **fixed** signature(object = "ParamFamily"): wrapped accessor function for slot fixed of slot param.
- **trafo** signature(object = "ParamFamily", param = "missing"): wrapped accessor function for slot trafo of slot param.
- **param** signature(object = "ParamFamily"): accessor function for slot param.
- **modifyParam** signature(object = "ParamFamily"): accessor function for slot modifyParam.
- **fam.call** signature(object = "ParamFamily"): accessor function for slot fam.call.
- **plot** signature(x = "ParamFamily"): plot of slot distribution.
- **show** signature(object = "ParamFamily")

**Details for methods 'show', 'print'**

Detailedness of output by methods show, print is controlled by the global option show.details to be set by distrModoptions.

As method show is used when inspecting an object by typing the object’s name into the console, show comes without extra arguments and hence detailedness must be controlled by global options.
Method print may be called with a (partially matched) argument show.details, and then the
global option is temporarily set to this value.

For class ParamFamily, this becomes relevant for slot param. For details therefore confer to
ParamFamParameter-class.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

Distribution-class

Examples

F1 <- new("ParamFamily") # prototype
plot(F1)

Description

Generates an object of class "ParamFamParameter".

Usage

ParamFamParameter(name, main = numeric(0), nuisance, fixed, trafo,
                   ..., .returnClsName = NULL)

Arguments

  name        (optional) character string: name of parameter
  main        numeric vector: main parameter
  nuisance    (optional) numeric vector: nuisance paramter
  fixed       (optional) numeric vector: fixed part of the parameter
  trafo       (optional) MatrixorFunction: transformation of the parameter
  ...         (optional) additional arguments for further return classes, e.g.
               \withPosRestr (only use case so far) for class ParamWithShapeFamParameter
  .returnClsName character or NULL; if non-null, the generated object will be of class .returnClsName,
                    which must be a subclass of ParamFamParameter.
Details

If name is missing, the default ""parameter of a parametric family of probability measures"" is used. If nuisance is missing, the nuisance parameter is set to NULL. The number of columns of trafo have to be equal and the number of rows have to be not larger than the sum of the lengths of main and nuisance. If trafo is missing, no transformation to the parameter is applied; i.e., trafo is set to an identity matrix.

Value

Object of class "ParamFamParameter" (or, if non-null, of class .returnClsName)

Author(s)

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

See Also

ParamFamParameter-class

Examples

ParamFamParameter(main = 0, nuisance = 1, fixed = 2,
                   trafo = function(x) list(fval = sin(x),
                                  mat = matrix(cos(x),1,1))
                   )

ParamFamParameter-class

Parameter of a parametric family of probability measures

Description

Class of the parameter of parametric families of probability measures.

Objects from the Class

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

Slots

main Object of class "numeric": main parameter.
nuisance Object of class "OptionalNumeric": optional nuisance parameter.
fixed Object of class "OptionalNumeric": optional fixed part of the parameter.
trafo Object of class "MatrixorFunction": transformation of the parameter.
name Object of class "character": name of the parameter.

withPosRestr (for ParamWithShapeFamParameter and ParamWithScaleAndShapeFamParameter): Object of class "logical": Is shape restricted to be positive?

Extends
Class "Parameter", directly.
Class "OptionalParameter", by class "Parameter".

Methods

main signature(object = "ParamFamParameter"): accessor function for slot main.
main<- signature(object = "ParamFamParameter"): replacement function for slot main.
nuisance signature(object = "ParamFamParameter"): accessor function for slot nuisance.
nuisance<- signature(object = "ParamFamParameter"): replacement function for slot nuisance.
fixed signature(object = "ParamFamParameter"): accessor function for slot fixed.
fixed<- signature(object = "ParamFamParameter"): replacement function for slot fixed.
trafo signature(object = "ParamFamParameter"): accessor function for slot trafo.
trafo<- signature(object = "ParamFamParameter"): replacement function for slot trafo.
length signature(x = "ParamFamParameter"): sum of the lengths of main and nuisance.
dimension signature(x = "ParamFamParameter"): length of main.
withPosRestr signature(object = "ParamWithShapeFamParameter"): accessor function for slot trafo.
withPosRestr<- signature(object = "ParamWithShapeFamParameter"): replacement function for slot trafo.
show signature(object = "ParamFamParameter")
show signature(object = "ParamWithShapeFamParameter")
show signature(object = "ParamWithScaleAndShapeFamParameter")

Details for methods 'show', 'print'

Detailedness of output by methods show, print is controlled by the global option show.details to be set by distrMod.options.

As method show is used when inspecting an object by typing the object's name into the console, show comes without extra arguments and hence detailedness must be controlled by global options.

Method print may be called with a (partially matched) argument show.details, and then the global option is temporarily set to this value.

More specifically, when show.detail is matched to "minimal" only class and name as well as main and nuisance part of the parameter are shown. When show.detail is matched to "medium", and if you estimate non-trivial (i.e. not the identity) transformation of the parameter of the parametric family, you will in addition be shown the derivative matrix, if the transformation is given in form of this matrix, while, if the transformation is in function form, you will only be told this. Finally, when show.detail is matched to "maximal", and you have a non-trivial transformation in function form, you will also be shown the code to this function.
Author(s)
Matthias Kohl <Matthias.Kohl@stamats.de>,
Peter Ruckdeschel <Peter.Ruckdeschel@itwm.fraunhofer.de>

See Also
Parameter-class

Examples
new("ParamFamParameter")

PoisFamily Generating function for Poisson families

Description
Generates an object of class "L2ParamFamily" which represents a Poisson family.

Usage
PoisFamily(lambda = 1, trafo)

Arguments
lambda positive mean
trafo function in param or matrix: transformation of the parameter

Details
The slots of the corresponding L2 differentiable parameteric family are filled.

Value
Object of class "L2ParamFamily"

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

References

See Also
L2ParamFamily-class, Pois-class
positiveBias

Examples

(P1 <- PoisFamily(lambda = 4.5))
plot(P1)
FisherInfo(P1)
checkL2deriv(P1)

positiveBias | Generating function for onesidedBias-class

Description
Generates an object of class "onesidedBias".

Usage
positiveBias(name = "positive Bias")

Arguments

name | name of the bias type

Value
Object of class "onesidedBias"

Author(s)
Peter Ruckdeschel <Peter.Ruckdeschel@itwm.fraunhofer.de>

References

See Also
onesidedBias-class

Examples

positiveBias()

## The function is currently defined as
function(){ new("onesidedBias", name = "positive Bias", sign = 1) }
Methods for print to the S4 classes in package `distrMod`:

## Usage

```r
## S4 method for signature 'ShowDetails'
print(x, digits =getOption("digits"),
      show.details = c("maximal", "minimal", "medium"))
```

### Arguments

- **x**: object of class `ShowDetails`, a class union of classes `OptionalNumeric`, `OptionalMatrix`, `MatrixorFunction`, `Estimate`, `MCEstimate`.
- **digits**: unchanged w.r.t. default method of package base: a non-null value for `digits` specifies the minimum number of significant digits to be printed in values. The default, 'NULL', uses `getOption(digits)'. (For the interpretation for complex numbers see `signif`). Non-integer values will be rounded down, and only values greater than or equal to 1 and no greater than 22 are accepted.
- **show.details**: a character, controlling the degree of detailedness of the output; currently the following values are permitted: "maximal", "minimal", "medium"; for the meaning for the actual class, confer to the corresponding class help file.

## Details

This method provides sort of a "show with extra arguments", in form of a common print method for the mentioned S4 classes. Essentially this print method just temporarily sets the global options according to the optional arguments `digits` and `show.details`, calls `show` and then re-sets the options to their global settings.

## Examples

```r
## set options to maximal detailedness
distrModOptions("show.details" = "maximal")
## define a model
NS <- NormLocationScaleFamily(mean=2, sd=3)
## generate data out of this situation
x <- r(distribution(NS))(30)

## want to estimate mu/sigma, sigma^2
--> new trafo slot:
trafo(NS) <- function(param){
  mu <- param["mean"]
  sd <- param["sd"]
```

```
family <- c(mu/sd, sd^2)
nfamily <- c("mu/sig", "sig^2")
names(family) <- nfamily
mat <- matrix(c(1/sd, 0, -mu/sd^2*2, 2*sdb, 2, 2)
dimnames(mat) <- list(nfamily, c("mean", "sd"))
return(list(family=family, mat=mat))
}
print(param(NS))
print(param(NS), show.details = "minimal")
print(param(NS), show.details = "medium")

# Maximum likelihood estimator
res <- MLEstimator(x = x, ParamFamily = NS)
print(res) # equivalent to 'show(res)' or 'res'
print(res, digits = 4)
print(res, show.details = "minimal")
print(res, show.details = "medium")
distrModoptions("show.details" = show.old)
\textbf{d} signature(object = "ProbFamily"): wrapped accessor to slot \texttt{d} of slot "Distribution".
\textbf{p} signature(object = "ProbFamily"): wrapped accessor to slot \texttt{p} of slot "Distribution".
\textbf{q} signature(object = "ProbFamily"): wrapped accessor to slot \texttt{q} of slot "Distribution".

\textbf{Author(s)}
Matthias Kohl <Matthias.Kohl@stamats.de>

\textbf{See Also}
\emph{Distribution-class}

---

\textbf{QFNorm} \quad \textit{Generating function for QFNorm-class}

\textbf{Description}
Generates an object of class "QFNorm".

\textbf{Usage}
\begin{verbatim}
QFNorm(name = "norm based on quadratic form",
       QuadForm = PosSemDefSymmMatrix(matrix(1)))
\end{verbatim}

\textbf{Arguments}
\begin{itemize}
  \item \textbf{name} \quad slot name of the class
  \item \textbf{QuadForm} \quad slot QuadForm of the class
\end{itemize}

\textbf{Value}
Object of class "QFNorm"

\textbf{Author(s)}
Peter Ruckdeschel <Peter.Ruckdeschel@itwm.fraunhofer.de>

\textbf{References}

\textbf{See Also}
\emph{QFNorm-class}
QFNorm-class

Examples

QFNorm()

### The function is currently defined as
function()
{
  new("QFNorm")
}

QFNorm-class

Norm classes for norms based on quadratic forms

Description

Classes for norms based on quadratic forms

Objects from the Class

could be created by a call to new, but normally one would use the generating functions QFNorm, InfoNorm, and SelfNorm

Slots

  name Object of class "character".
  fct Object of class "function".
  QuadForm Object of class "PosSemDefSymmMatrix".

Extends

"QFNorm" extends class "NormType", directly, and "InfoNorm" and "SelfNorm" each extend class "QFNorm", directly (and do not have extra slots).

Methods

  QuadForm signature(object = "QFNorm"): accessor function for slot QuadForm.
  QuadForm<- signature(object = "QFNorm"): replacement function for slot QuadForm.

Author(s)

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

References

qqplot

Methods for Function qqplot in Package 'distrMod'

Description

We generalize function qqplot from package stats to be applicable to distribution and probability model objects. In this context, qqplot produces a QQ plot of data (argument x) against a (model) distribution. Graphical parameters may be given as arguments to qqplot.

Usage

qqplot(x, y, ...)
qqplot(x, y, ...,
    n = length(x), withIdLine = TRUE,
    withConf = TRUE, withConf.pw = withConf, withConf.sim = withConf,
    plot.it = TRUE, xlab = deparse(substitute(x)), ylab = deparse(substitute(y)),
    ..., width = 10, height = 5.5, withSweave = getdistrOption("withSweave"),
    mfColRow = TRUE, n.CI = n, withLab = FALSE, lab.pts = NULL, which.lbs = NULL,
    which.Order = NULL, order.traf = NULL,
    col.IdL = "red", lty.IdL = 2, lwd.IdL = 2, alpha.CI = .95,
    exact.pCI = (n<100), exact.sCI = (n<100), nosym.pCI = FALSE,
    col.pCI = "orange", lty.pCI = 3, lwd.pCI = 2, pch.pCI = par("pch"),
    cex.pCI = par("cex"),
    col.sCI = "tomato2", lty.sCI = 4, lwd.sCI = 2, pch.sCI = par("pch"),
    cex.sCI = par("cex"),
    cex.pch = par("cex"), col.pch = par("col"),
    cex.lbl = par("cex"), col.lbl = par("col"), adj.lbl = NULL,
    alpha.trsp = NA, jit.fac = 0,
    check.NotInSupport = TRUE, col.NotInSupport = "red",
    with.legend = TRUE, legend.bg = "white",
    legend.pos = "topleft", legend.cex = 0.8,
    legend.pref = "", legend.postf = "", legend.alpha = alpha.CI)

Arguments

x data to be checked for compatibility with distribution/model y.
qqplot

y object of class "UnivariateDistribution" or of class "ProbFamily".
n numeric; assumed sample size (by default length of x).
withIdLine logical; shall line \( y = x \) be plotted?
withConf logical; shall confidence lines be plotted?
withConf.pw logical; shall pointwise confidence lines be plotted?
withConf.sim logical; shall simultaneous confidence lines be plotted?
plot.it logical; shall be plotted at all (inherited from qqplot)?
xlab x-label
ylab y-label
... further parameters for method qqplot with signature ANY, UnivariateDistribution or with function plot
width width (in inches) of the graphics device opened
height height (in inches) of the graphics device opened
withSweave logical: if TRUE (for working with Sweave) no extra device is opened and height/width are not set
mfColRow shall default partition in panels be used — defaults to TRUE
n.CI numeric; number of points to be used for confidence interval
withLab logical; shall observation labels be plotted in?
lab.pts character or NULL; observation labels to be used
which.lbs integer or NULL; which observations shall be labelled
which.Order integer or NULL; which of the ordered (remaining) observations shall be labelled
order.traf function or NULL; an optional trafo by which the observations are ordered (as order(trafo(obs)).
col.IdL color for the identity line
lty.IdL line type for the identity line
lwd.IdL line width for the identity line
alpha.CI confidence level
exact.pCI logical; shall pointwise CIs be determined with exact Binomial distribution?
exact.sCI logical; shall simultaneous CIs be determined with exact Kolmogorov distribution?
nosym.pCI logical; shall we use (shortest) asymmetric CIs?
col.pCI color for the pointwise CI
lty.pCI line type for the pointwise CI
lwd.pCI line width for the pointwise CI
pch.pCI symbol for points (for discrete mass points) in pointwise CI
cex.pCI magnification factor for points (for discrete mass points) in pointwise CI
col.sCI color for the simultaneous CI
lty.sCI line type for the simultaneous CI
qqplot

- lwd.sci: line width for the simultaneous CI
- pch.sci: symbol for points (for discrete mass points) in simultaneous CI
- cex.sci: magnification factor for points (for discrete mass points) in simultaneous CI
- cex.pch: magnification factor for the plotted symbols
- col.pch: color for the plotted symbols
- cex.lbl: magnification factor for the plotted observation labels
- col.lbl: color for the plotted observation labels
- adj.lbl: adj parameter for the plotted observation labels
- alpha.trsp: alpha transparency to be added ex post to colors col.pch and col.lbl; if one-dim and NA all colors are left unchanged. Otherwise, with usual recycling rules alpha.trsp gets shorted/prolongated to length the data-symbols to be plotted. Coordinates of this vector alpha.trsp with NA are left unchanged, while for the remaining ones, the alpha channel in rgb space is set to the respective coordinate value of alpha.trsp. The non-NA entries must be integers in \([0,255]\) (0 invisible, 255 opaque).
- jit.fac: jittering factor used for discrete distributions
- check.NotInSupport: logical; shall we check if all x-quantiles lie in support(y)?
- col.NotInSupport: logical; if preceding check TRUE color of x-quantiles if not in support(y)
- with.legend: logical; shall a legend be plotted?
- legend.bg: background color for the legend
- legend.pos: position for the legend
- legend.cex: magnification factor for the legend
- legend.pref: character to be prepended to legend text
- legend.postf: character to be appended to legend text
- legend.alpha: nominal coverage probability

Details

qqplot signature(x = "ANY", y = "UnivariateDistribution"): produces a QQ plot of a dataset x against the theoretical quantiles of distribution y.

qqplot signature(x = "ANY", y = "ProbFamily"): produces a QQ plot of a dataset x against the theoretical quantiles of the model distribution of model y. Passed through the ... argument, all arguments valid for signature(x = "ANY", y = "UnivariateDistribution") are also valid for this signature.

Value

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

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

Class of risks; e.g., estimator risks.

Objects from the Class

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

Slots

- type: Object of class "character": type of risk.

Methods

- type signature(object = "RiskType"): accessor function for slot type.
- show signature(object = "RiskType")

Author(s)

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

Generates an object of class "SelfNorm" — used for self-standardized influence curves.

Usage

SelfNorm()

Value

Object of class "SelfNorm"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

SelfNorm-class

Examples

SelfNorm()

```r
## The function is currently defined as
function(){ new("SelfNorm") }
```

Description

Generates an object of class "symmetricBias".

Usage

symmetricBias(name = "symmetric Bias")
Arguments

name name of the bias type

Value

Object of class "symmetricBias"

Author(s)

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

References


See Also

symmetricBias-class

Examples

symmetricBias()

## The function is currently defined as
function()
{
  new("symmetricBias", name = "symmetric Bias")
}

Description

Class of symmetric bias types.

Objects from the Class

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

Slots

name Object of class "character".
Methods

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

Extends

Class "BiasType", directly.

Author(s)

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

References


See Also

BiasType-class

Examples

symmetricBias()

## The function is currently defined as
function(){ new("symmetricBias", name = "symmetric Bias") }

Description

Methods for function trafo in package distrMod; there are accessor (trafo) and replacement (trafo<-) versions.

Usage

trafo(object, param, ...)

## S4 method for signature 'Estimate,missing'
trafo(object, param)

## S4 method for signature 'ParamFamParameter,missing'
trafo(object, param)

## S4 method for signature 'ParamWithScaleAndShapeFamParameter,missing'
trafo(object, param)
trafo-methods

```r
## S4 method for signature 'ParamFamily,missing'
trafo(object, param)
## S4 method for signature 'ParamFamily,ParamFamParameter'
trafo(object, param)
## S4 method for signature 'Estimate,ParamFamParameter'
trafo(object, param)
trafo.fct(object)
trafo(object) <- value
```

**Arguments**

- **object**: an object of either class `Estimate`, `ParamFamParameter`, `ParamFamily`
- **param**: an object of class `ParamFamParameter`; the parameter value at which to evaluate the transformation
- **value**: a matrix or a function; if it is a matrix, dimensions must be consistent to the parametric setting; if it is function, it should take one argument `param` of class `ParamFamParameter` and return a list of length two with named components `fval` (the function value, see below) and `mat` (a matrix — with the same dimensions consistency conditions as above).
- ... additional argument(s) for methods; not used so far.

**Details**

`trafo` is a slot of class `ParamFamParameter`, which in turn is a slot of class `ParamFamily`. It also sort of arises in class `Estimate`, i.e., all slots can be identified by the information contained in an instance thereof.

`trafo` realizes partial influence curves; i.e., we are only interested in some possibly lower dimensional smooth (not necessarily linear or even coordinate-wise) aspect/transformation $\tau$ of the parameter $\theta$.

To be coherent with the corresponding *nuisance* implementation, we make the following convention:

The full parameter $\theta$ is split up coordinate-wise in a main parameter $\theta'$ and a nuisance parameter $\theta''$ (which is unknown, too, hence has to be estimated, but only is of secondary interest) and a fixed, known part $\theta'''$.

Without loss of generality, we restrict ourselves to the case that transformation $\tau$ only acts on the main parameter $\theta'$ — if we want to transform the whole parameter, we only have to assume that both nuisance parameter $\theta''$ and fixed, known part of the parameter $\theta'''$ have length 0.

To the implementation:

Slot `trafo` can either contain a (constant) matrix $D_\theta$ or a function $\tau: \Theta' \rightarrow \tilde{\Theta}$, $\theta \mapsto \tau(\theta)$ mapping main parameter $\theta'$ to some range $\tilde{\Theta}$.

If slot value `trafo` is a function, besides $\tau(\theta)$, it will also return the corresponding derivative matrix $\frac{\partial}{\partial \theta} \tau(\theta)$. More specifically, the return value of this function `theta` is a list with entries `fval`, the function value $\tau(\theta)$, and `mat`, the derivative matrix.
In case \( \text{trafo} \) is a matrix \( D \), we interpret it as such a derivative matrix \( \frac{\partial}{\partial \theta} \tau(\theta) \), and, correspondingly, \( \tau(\theta) \) as the linear mapping \( \tau(\theta) = D \theta \).

According to the signature, \textit{method trafo} will return different return value types. For signature \texttt{Estimate,missing}: it will return a list with entries \texttt{fct}, the function \( \tau \), and \texttt{mat}, the matrix \( \frac{\partial}{\partial \theta} \tau(\theta) \). function \( \tau \) will then return the list \texttt{list(fval, mat)} mentioned above.

\texttt{Estimate,ParamFamParameter}: as signature \texttt{Estimate,missing}.

\texttt{ParamFamParameter,missing}: it will just return the corresponding matrix.

\texttt{ParamFamily,missing}: is just wrapper to signature \texttt{ParamFamParameter,missing}.

\texttt{ParamFamily,ParamFamParameter}: as signature \texttt{Estimate,missing}.

\textbf{Value}

The return value depends on the signature. For \texttt{trafo.fct}, we return the corresponding function \( \tau() \) (see below). For \texttt{trafo}, we have:

\texttt{signature Estimate,missing}: a list of length two with components \texttt{fct} and \texttt{mat} (see below)

\texttt{signature Estimate,ParamFamParameter}: a list of length two with components \texttt{fct} and \texttt{mat} (see below)

\texttt{signature ParamFamParameter,missing}: a matrix (see below)

\texttt{signature ParamFamily,missing}: a matrix (see below)

\texttt{signature ParamFamily,ParamFamParameter}: a list of length two with components \texttt{fct} and \texttt{mat} (see below)

\textbf{Examples}

```r
## Gaussian location and scale
NS <- NormLocationScaleFamily(mean=2, sd=3)
## generate data out of this situation
x <- r(distribution(NS))(30)

## want to estimate mu/sigma, sigma^2
## -> new trafo slot:
trafo(NS) <- function(param){
  mu <- param["mean"]
  sd <- param["sd"]
  fval <- c(mu/sd, sd^2)
  nfval <- c(\"mu/sig\", \"sig^2\")
  names(fval) <- nfval
  mat <- matrix(\c{1/sd,0,-mu/sd^2,2*sd},2,2)
  dimnames(mat) <- list(nfval,\c{\"mean\","sd\"})
  return(list(fval=fval, mat=mat))
}

## Maximum likelihood estimator
```
trafoEst

Function trafoEst in Package ‘distrMod’

Description

trafoEst takes a $\tau$ like function (compare trafo-methods) and transforms an existing estimator by means of this transformation.

Usage

trafoEst(fct, estimator)

Arguments

- `fct`: a $\tau$ like function, i.e., a function in the main part $\theta$ of the parameter returning a list `list(fval, mat)` where `fval` is the function value $\tau(\theta)$ of the transformation, and `mat`, its derivative matrix at $\theta$.
- `estimator`: an object of class `Estimator`.

Details

The disadvantage of this proceeding is that the transformation is not accounted for in determining the estimate (e.g. in a corresponding optimality); it simply transforms an existing estimator, without reapplying it to data. This becomes important in optimally robust estimation.

Value

exactly the argument `estimator`, but with modified slots `estimate`, `asvar`, and `trafo`.

Examples

```r
(res <- MLEstimator(x = x, ParamFamily = NS))
## confidence interval
confint(res)
```

```r
# Gaussian location and scale
NS <- NormLocationScaleFamily(mean = 2, sd = 3)
# generate data out of this situation
x <- r(distribution(NS))(30)

# want to estimate mu/sigma, sigma^2
# -> without new trafo slot:
mtrafo <- function(param){
  mu <- param["mean"]
  sd <- param["sd"]
  fval <- c(mu/sd, sd^2)
  nfval <- c("mu/sig", "sig^2")
  names(fval) <- nfval
}
```
mat <- matrix(c(1/sd, 0, -mu/sd^2, 2/sd), 2, 2)
dimnames(mat) <- list(nfval, c("mean", "sd"))
return(list(fval=fval, mat=mat))}

## Maximum likelihood estimator in the original problem
res0 <- MLEstimator(x = x, ParamFamily = NS)
## transformation
res <- trafoEst(mtrafo, res0)
## confidence interval
confint(res)

---

**trAsCov**

*Generating function for trAsCov-class*

**Description**

Generates an object of class "trAsCov".

**Usage**

`trAsCov()`

**Value**

Object of class "trAsCov"

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**


**See Also**

`trAsCov-class`

**Examples**

`trAsCov()`

```r
## The function is currently defined as
function(){ new("trAsCov") }
```
Description

Class of trace of asymptotic covariance.

Objects from the Class

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

Slots

- `type`: Object of class "character": “trace of asymptotic covariance”.

Extends

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

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

- `asRisk-class`, `trAsCov`

Examples

- `new("trAsCov")`
trFiCov

Generating function for trFiCov-class

Description

Generates an object of class "trFiCov".

Usage

trFiCov()

Value

Object of class "trFiCov"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

trFiCov-class

Examples

trFiCov()

## The function is currently defined as
function(){ new("trFiCov") }

trFiCov-class

Trace of finite-sample covariance

Description

Class of trace of finite-sample covariance.

Objects from the Class

Objects can be created by calls of the form new("trFiCov", ...). More frequently they are created via the generating function trFiCov.
validParameter-methods

Slots

type Object of class "character": “trace of finite-sample covariance”.

Extends

Class "fiRisk", directly.
Class "RiskType", by class "fiRisk".

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

fiRisk-class.trFiCov

Examples

new("trFiCov")

validParameter-methods

Methods for function validParameter in Package 'distrMod'

Description

Methods for function validParameter in package distrMod to check whether a new parameter (e.g. "proposed" by an optimization) is valid.

Usage

validParameter(object, ...)  
## S4 method for signature 'ParamFamily'  
validParameter(object, param)  
## S4 method for signature 'L2ScaleUnion'  
validParameter(object, param, tol=.Machine$double.eps)  
## S4 method for signature 'L2ScaleFamily'  
validParameter(object, param, tol=.Machine$double.eps)  
## S4 method for signature 'L2LocationFamily'  
validParameter(object, param)  
## S4 method for signature 'L2LocationScaleFamily'  
validParameter(object, param, tol=.Machine$double.eps)  
## S4 method for signature 'BinomFamily'
validParameter(object, param, tol=.Machine$double.eps)
## S4 method for signature 'PoisFamily'
validParameter(object, param, tol=.Machine$double.eps)
## S4 method for signature 'L2ScaleShapeUnion'
validParameter(object, param, tol=.Machine$double.eps)

**Arguments**

- **object**  
an object of class `ParamFamily`

- **param**  
either a numeric vector or an object of class `ParamFamParameter`

- **tol**  
accuracy up to which the conditions have to be fulfilled

- **...**  
additional argument(s) for methods.

**Details**

- **method for signature**
  
  `ParamFamily` checks if all parameters are finite by `is.finite` if their length is between 1 and the joint length of main and nuisance parameter of `object`, and finally, if a call to `modifyParam(object)` with argument `param` would throw an error.
  
  `L2ScaleUnion` checks if the parameter is finite by `is.finite`, and if it is strictly larger than 0 (upto argument `tol`).

  - **L2ScaleFamily** checks if the parameter length is 1, and otherwise uses `L2ScaleUnion-method`.
  
  - **L2LocationFamily** checks if the parameter is finite by `is.finite`, if its length is 1
  
  - **L2LocationScaleFamily** checks if the parameter length is 1 or 2 (e.g. if one features as nuisance parameter), and also uses `L2ScaleUnion-method`.

  - **BinomFamily** checks if the parameter is finite by `is.finite`, if its length is 1, and if it is strictly larger than 0 and strictly smaller than 1 (upto argument `tol`)

  - **PoisFamily** checks if the parameter is finite by `is.finite`, if its length is 1, and if it is strictly larger than 0 (upto argument `tol`)

  - **L2ScaleShapeUnion** uses `L2ScaleUnion-method`, checks if parameter length is 1 or 2 (e.g. if one features as nuisance parameter), and if shape is strictly larger than 0 (upto argument `tol`)

**Value**

- logical of length 1 — valid or not

**Examples**

```r
NS <- NormLocationScaleFamily()
validParameter(NS, c(scale=0.1, loc=2))
validParameter(NS, c(scale=0.1, loc=2))
validParameter(NS, c(scale=0, loc=2))
validParameter(NS, c(mean=2, sd=2))
```
Index

*Topic algebra
  isKerAinKerB, 59
*Topic array
  isKerAinKerB, 59
*Topic classes
  asBias-class, 12
  asCov-class, 13
  asGRisk-class, 14
  asHampel-class, 16
  asMSE-class, 18
  asRisk-class, 19
  asRiskWithBias-class, 20
  asSemivari-class, 22
  asUnOvShoot-class, 24
  asymmetricBias-class, 26
  BiasType-class, 28
  Confint-class, 32
  Estimate-class, 38
  EvenSymmetric-class, 43
  fiBias-class, 46
  fiCov-class, 48
  fiHampel-class, 50
  fiMSE-class, 51
  fiRisk-class, 52
  fiUnOvShoot-class, 54
  FunctionSymmetry-class, 55
  FunSymmList-class, 57
  L2GroupParamFamily-class, 60
  L2LocationFamily-class, 64
  L2LocationScaleFamily-class, 67
  L2ParamFamily-class, 73
  L2ScaleFamily-class, 78
  MCEstimate-class, 85
  NonSymmetric-class, 100
  NormType-class, 108
  OddSymmetric-class, 109
  onesidedBias-class, 110
  ParamFamily-class, 115
  ParamFamParameter-class, 118

  ProbFamily-class, 123
  QFNorm-class, 125
  RiskType-class, 129
  symmetricBias-class, 131
  trAsCov-class, 137
  trFiCov-class, 138
*Topic distribution
  addAlphTrsp2col, 10
  distrModMASK, 36
  distrModOptions, 37
  ParamFamily, 111
  qqplot, 126
*Topic documentation
  distrModMASK, 36
*Topic hplot
  qqplot, 126
*Topic math
  EvenSymmetric, 42
  FunSymmList, 56
  NonSymmetric, 100
  NormType, 107
  OddSymmetric, 109
  QFNorm, 124
*Topic misc
  distrModOptions, 37
*Topic models
  BetaFamily, 27
  BinomFamily, 29
  CauchyLocationScaleFamily, 30
  checkL2deriv, 31
  confint-methods, 34
  ExpScaleFamily, 45
  GammaFamily, 57
  L2GroupParamFamily-class, 60
  L2LocationFamily, 62
  L2LocationFamily-class, 64
  L2LocationScaleFamily, 66
  L2LocationScaleFamily-class, 67
  L2LocationUnknownScaleFamily, 69
biastype<-, asRiskwithBias-method (asRiskwithBias-class), 20
BinomFamily, 29
bound (asHampel-class), 16
bound, asHampel-method (asHampel-class), 16
bound, fiHampel-method (fiHampel-class), 50
call.estimate (Confint-class), 32
call.estimate, Confint-method (Confint-class), 32
CauchyLocationScaleFamily, 30
checkL2deriv, 31
checkL2deriv, L2ParamFamily-method (L2ParamFamily-class), 73
criterion, MCEstimate, mle-method (MCEstimate-class), 85
completecases (Estimate-class), 38
completecases, Estimate-method (Estimate-class), 38
completecases.estimate (Confint-class), 32
completecases.estimate, Confint-method (Confint-class), 32
confint, 32–35
confint (confint-methods), 34
criterion, ANY, missing-method (confint-methods), 34
confint, Confint, missing-method (Confint-class), 32
confint, Estimate, missing-method (confint-methods), 34
criterion, MCEstimate, mle-method (MCEstimate-class), 85
criterion, MCEstimate-method (MCEstimate-class), 85
criterion.<-, MCEstimate-method (MCEstimate-class), 85
criterion<-, MCEstimate-method (MCEstimate-class), 85
d, ProbFamily-method (ProbFamily-class), 123
dimension, ParamFamParameter-method (ParamFamParameter-class), 118
distribution (ProbFamily-class), 123
distribution, ProbFamily-method (ProbFamily-class), 123
distrMod (distrMod-package), 4
distrMod-package, 4
distrModMASK, 36
distrModOptions, 37
distrModOptions, 33, 40, 116, 119
distrModOptions (distrModOptions), 37
disrptions, 38
distrSymm (ProbFamily-class), 123
distrSymm, ProbFamily-method (ProbFamily-class), 123
E, L2ParamFamily, EuclRandMatrix, missing-method (L2ParamFamily-class), 73
E, L2ParamFamily, EuclRandVariable, missing-method (L2ParamFamily-class), 73
E, L2ParamFamily, EuclRandVarList, missing-method (L2ParamFamily-class), 73
estimate (Estimate-class), 38
euclidenanNorm (norm), 101
EvenSymmetric, 42, 43
EvenSymmetric-class, 43
existsPIC (existsPIC-methods), 44
ExpScaleFamily, 45
fam.call (ParamFamily-class), 115
fam.call, ParamFamily-method (ParamFamily-class), 115
fct (NormType-class), 108
INDEX

fct, NormType-method (NormType-class), 108
fct<- (NormType-class), 108
fct<-, NormType-method (NormType-class), 108
fiBias, 46, 47
fiBias-class, 46
fiCov, 47, 49
fiCov-class, 48
fiHampel, 49, 50
fiHampel-class, 50
fiMSE, 51, 52
fiMSE-class, 51
fiRisk-class, 52
FisherInfo (L2ParamFamily-class), 73
FisherInfo, L2ParamFamily, missing-method
(L2ParamFamily-class), 73
FisherInfo, L2ParamFamily, ParamFamParameter-method
(L2ParamFamily-class), 73
fitdistr, 90, 94
fiUnOvShoot, 53
fiUnOvShoot-class, 54
fixed (ParamFamParameter-class), 118
fixed, Estimate-method (Estimate-class), 38
fixed, ParamFamily-method
(ParamFamily-class), 115
fixed, ParamFamParameter-method
(ParamFamParameter-class), 118
fixed, ParamWithScaleAndShapeFamParameter-method
(ParamFamParameter-class), 118
fixed, estimate (Confint-class), 32
fixed, estimate, Confint-method
(Confint-class), 32
fixed< (ParamFamParameter-class), 118
fixed<-, ParamFamParameter-method
(ParamFamParameter-class), 118
FunctionSymmetry-class, 55
FunSymmList, 56
FunSymmList-class, 57
GammaFamily, 57
get.criterion.fct (meRes), 91
getDistrModOption (distrModOptions), 37
getDistrOption, 38
ggetOption, 38
InfoNorm, 58
InfoNorm-class (QFNorm-class), 125
Infos (Estimate-class), 38
Infos, Estimate-method (Estimate-class), 38
Infos< (Estimate-class), 38
Infos<-, Estimate-method
(Estimate-class), 38
isKerAinKerB, 44, 59
L2deriv (L2ParamFamily-class), 73
L2deriv, L2ParamFamily, missing-method
(L2ParamFamily-class), 73
L2deriv, L2ParamFamily, ParamFamParameter-method
(L2ParamFamily-class), 73
L2derivDistr (L2ParamFamily-class), 73
L2derivDistr, L2ParamFamily-method
(L2ParamFamily-class), 73
L2derivDistrSymm (L2ParamFamily-class), 73
L2derivDistrSymm, L2ParamFamily-method
(L2ParamFamily-class), 73
L2derivSymm (L2ParamFamily-class), 73
L2derivSymm, L2ParamFamily-method
(L2ParamFamily-class), 73
L2GroupParamFamily-class, 60
L2LocationFamily, 62, 65
L2LocationFamily-class, 64
L2LocationScaleFamily, 66, 69
L2LocationScaleFamily-class, 67
L2LocationUnknownScaleFamily, 69
L2ParamFamily, 71, 76
L2ParamFamily-class, 73
L2ScaleFamily, 76, 80
L2ScaleFamily-class, 78
L2ScaleUnknownLocationFamily, 80
length, ParamFamParameter-method
(ParamFamParameter-class), 118
LnormScaleFamily, 82
LogDeriv (L2GroupParamFamily-class), 60
LogDeriv, L2GroupParamFamily-method
(L2GroupParamFamily-class), 60
LogDeriv< (L2GroupParamFamily-class), 60
LogDeriv<-, L2GroupParamFamily-method
(L2GroupParamFamily-class), 60
main (ParamFamParameter-class), 118
main, Estimate-method (Estimate-class), 38
main,ParamFamily-method
  (ParamFamily-class), 115
main,ParamFamParameter-method
  (ParamFamParameter-class), 118
main,ParamWithScaleAndShapeFamParameter-method
  (ParamWithScaleAndShapeFamParameter-class), 118
main<-,ParamFamParameter-method
  (ParamFamParameter-class), 118
makeOKPar (ParamFamily-class), 115
makeOKPar,ParamFamily-method
  (ParamFamily-class), 115
MASKING (distrModMASK), 36
mceCalc (mceCalc-methods), 83
mceCalc,numeric,ParamFamily-method
  (mceCalc-methods), 83
mceCalc-methods, 83
MCEstimate-class, 85
MCEstimator, 86, 87, 90, 94
MDEstimator, 86, 89
meRes, 84, 91
method (MCEstimate-class), 85
method,MCEstimate-method
  (MCEstimate-class), 85
mle, 94
mleCalc, 91–93
mleCalc (mceCalc-methods), 83
mleCalc,numeric,BinomFamily-method
  (mceCalc-methods), 83
mleCalc,numeric,NormLocationFamily-method
  (mceCalc-methods), 83
mleCalc,numeric,NormLocationScaleFamily-method
  (mceCalc-methods), 83
mleCalc,numeric,NormScaleFamily-method
  (mceCalc-methods), 83
mleCalc,numeric,ParamFamily-method
  (mceCalc-methods), 83
mleCalc,numeric,PoisFamily-method
  (mceCalc-methods), 83
mleCalc-methods (mceCalc-methods), 83
MLEstimator, 86, 93
modifyModel (modifyModel-methods), 96
modifyModel,ExpScaleFamily,ParamFamParameter-method
  (modifyModel-methods), 96
modifyModel,GammaFamily,ParamFamParameter-method
  (modifyModel-methods), 96
modifyModel,L2LocationFamily,ParamFamParameter-method
  (modifyModel-methods), 96
modifyModel,L2LocationScaleFamily,ParamFamParameter-method
  (modifyModel-methods), 96
modifyModel,L2ParamFamily,ParamFamParameter-method
  (modifyModel-methods), 96
modifyModel,L2ScaleFamily,ParamFamParameter-method
  (modifyModel-methods), 96
modifyModel,ParamFamily,ParamFamParameter-method
  (modifyModel-methods), 96
modifyModel-methods, 96
modifyParam (ParamFamily-class), 115
modifyParam,ParamFamily-method
  (ParamFamily-class), 115
NA, 128
name,BiasType-method (BiasType-class), 28
name,Estimate-method (Estimate-class), 38
name,NormType-method (NormType-class), 108
name,ProbFamily-method
  (ProbFamily-class), 123
name.estimate (Confint-class), 32
name.estimate,Confint-method
  (Confint-class), 32
name<-,BiasType-method
  (BiasType-class), 28
name<-,Estimate-method
  (Estimate-class), 38
name<-,NormType-method
  (NormType-class), 108
name<-,ProbFamily-method
  (ProbFamily-class), 123
NbinomFamily, 97
NbinomMeanSizeFamily (NbinomFamily), 97
NbinomMeanSizeFamily (NbinomFamily), 97
negativeBias, 99
NonSymmetric, 100, 101
NonSymmetric-class, 100
norm, 101
norm (asRiskwithBias-class), 20
norm,asRiskwithBias-method
  (asRiskwithBias-class), 20
NormLocationFamily, 102
NormLocationScaleFamily, 103
NormLocationUnknownScaleFamily, 104
NormScaleFamily, 105
NormScaleUnknownLocationFamily, 106
NormType, 107, 108
normtype (asRiskwithBias-class), 20
normtype,asRiskwithBias-method (asRiskwithBias-class), 20
NormType-class, 108
normtype<- (asRiskwithBias-class), 20
normtype<-,asRiskwithBias-method (asRiskwithBias-class), 20
nu (asymmetricBias-class), 26
nu,asymmetricBias-method (asymmetricBias-class), 26
nu<-(asymmetricBias-class), 26
nu<-,asymmetricBias-method (asymmetricBias-class), 26
nuisance (ParamFamParameter-class), 118
nuisance,Estimate-method (Estimate-class), 38
nuisance,ParamFamily-method (ParamFamily-class), 115
nuisance,ParamFamParameter-method (ParamFamParameter-class), 118
nuisance,ParamWithScaleAndShapeFamParameter-method (ParamFamParameter-class), 118
nuisance,estimate (Confint-class), 32
nuisance,estimate,Confint-method (Confint-class), 32
nuisance<-(ParamFamParameter-class), 118
nuisance<-,ParamFamParameter-method (ParamFamParameter-class), 118

OddSymmetric, 109, 110
OddSymmetric-class, 109
onesidedBias-class, 110
optimwarn (MCEstimate-class), 85
optimwarn,MCEstimate-method (MCEstimate-class), 85
options, 38

p,ProbFamily-method (ProbFamily-class), 123
par, 75
param,ParamFamily-method (ParamFamily-class), 115
ParamFamily, 88, 90, 94, 111
ParamFamily-class, 115
ParamFamParameter, 73, 112, 117
ParamFamParameter-class, 118

ParamWithScaleAndShapeFamParameter-class (ParamFamParameter-class), 118
ParamWithScaleFamParameter-class (ParamFamParameter-class), 118
ParamWithShapeFamParameter-class (ParamFamParameter-class), 118
plot, 75, 76
plot (L2ParamFamily-class), 73
plot,L2ParamFamily,missing-method (L2ParamFamily-class), 73
plot,ParamFamily,missing-method (ParamFamily-class), 115
plot-methods (L2ParamFamily-class), 73
plot.default, 75, 76
plot.stepfun, 76
PoisFamily, 120
positiveBias, 121
print,Confint-method (Confint-class), 32
print,Estimate-method (Estimate-class), 38
print,ShowDetails-method (print-methods), 122
print-methods, 122
ProbFamily-class, 123
profile, 86
profile,MCEstimate-method (MCEstimate-class), 85
props (ProbFamily-class), 123
props,ProbFamily-method (ProbFamily-class), 123
props<-(ProbFamily-class), 123
props<-,ProbFamily-method (ProbFamily-class), 123

q,ProbFamily-method (ProbFamily-class), 123
QFNorm, 124
QFNorm-class, 125
qqbounds, 129
qqplot, 126, 126, 127–129
qqplot,ANY,ProbFamily-method (qqplot), 126
qqplot,ANY,UnivariateDistribution-method (qqplot), 126
qqplot-methods (qqplot), 126
QuadForm (QFNorm-class), 125
QuadForm,QFNorm-method (QFNorm-class), 125
QuadForm<-(QFNorm-class), 125
QuadForm<-, QFNorm-method (QFNorm-class), 125
QuadFormNorm (norm), 101
r, ProbFamily-method (ProbFamily-class), 123
RiskType-class, 129
samplesize (Estimate-class), 38
samplesize, Estimate-method (Estimate-class), 38
samplesize, numeric-method (meRes), 91
samplesize, estimate (Confint-class), 32
samplesize, estimate, Confint-method (Confint-class), 32
SelfNorm, 130
SelfNorm-class (QFNorm-class), 125
show, asHampel-method (asHampel-class), 16
show, asUnOvShoot-method (asUnOvShoot-class), 24
show, Conffint-method (Confint-class), 32
show, Estimate-method (Estimate-class), 38
show, fiHampel-method (fiHampel-class), 50
show, fiUnOvShoot-method (fiUnOvShoot-class), 54
show, MCEstimate-method (MCEstimate-class), 85
show, ParamFamily-method (ParamFamily-class), 115
show, ParamFamParameter-method (ParamFamParameter-class), 118
show, ParamWithScaleAndShapeFamParameter-method (ParamFamParameter-class), 118
show, ParamWithShapeFamParameter-method (ParamFamParameter-class), 118
show, RiskType-method (RiskType-class), 129
show, details (distrModOptions), 37
sign<-, onesidedBias-method (onesidedBias-class), 110
sign<-, asSemivar-method (asSemivar-class), 22
sign<-, onesidedBias-method (onesidedBias-class), 110
startPar (ParamFamParameter-class), 115
startPar, MCEstimate-method (MCEstimate-class), 85
startPar, ParamFamily-method (ParamFamily-class), 115
svd, 59
symmetricBias, 130
symmetricBias-class, 131
trafo (trafo-methods), 132
trafo, Estimate, missing-method (trafo-methods), 132
trafo, Estimate, ParamFamParameter-method (trafo-methods), 132
trafo, ParamFamily, missing-method (trafo-methods), 132
trafo, ParamFamily, ParamFamParameter-method (trafo-methods), 132
trafo, ParamFamily, ParamFamParameter-method missing-method (trafo-methods), 132
trafo-methods, 132
trafo, estimate (Confint-class), 32
trafo, estimate, Confint-method (Confint-class), 32
trafo, fct (trafo-methods), 132
trafo, fct, ParamFamily-method (trafo-methods), 132
trafo, fct-methods (trafo-methods), 132
trafo<-, (trafo-methods), 132
trafo<-, ParamFamily-method (trafo-methods), 132
trafo<-, ParamFamParameter-method (trafo-methods), 132
trafoEst, 135
trAsCov, 136, 137
trAsCov-class, 137
trFiCov, 138, 139
trFiCov-class, 138
type, Conffint-method (Conffint-class), 32
type, RiskType-method (RiskType-class), 129
untransformed.asvar (Estimate-class), 38
untransformed.asvar, Estimate-method (Estimate-class), 38
untransformed estimate
  (Estimate-class), 38
untransformed estimate, Estimate-method
  (Estimate-class), 38

validParameter
  (validParameter-methods), 139
validParameter, BinomFamily-method
  (validParameter-methods), 139
validParameter, L2LocationFamily-method
  (validParameter-methods), 139
validParameter, L2LocationScaleFamily-method
  (validParameter-methods), 139
validParameter, L2ScaleFamily-method
  (validParameter-methods), 139
validParameter, L2ScaleShapeUnion-method
  (validParameter-methods), 139
validParameter, L2ScaleUnion-method
  (validParameter-methods), 139
validParameter, ParamFamily-method
  (validParameter-methods), 139
validParameter, PoisFamily-method
  (validParameter-methods), 139
validParameter-methods, 139

width (asUnOvShoot-class), 24
width, asUnOvShoot-method
  (asUnOvShoot-class), 24
width, fiUnOvShoot-method
  (fiUnOvShoot-class), 54
withPosRestr (ParamFamParameter-class), 118
withPosRestr, ParamWithShapeFamParameter-method
  (ParamFamParameter-class), 118
withPosRestr<- (ParamFamParameter-class), 118
withPosRestr<-, ParamWithShapeFamParameter-method
  (ParamFamParameter-class), 118