Package ‘distrMod’

November 16, 2022

Version 2.9.0
Date 2022-11-14
Title Object Oriented Implementation of Probability Models
Description Implements S4 classes for probability models based on packages ‘distr’ and ‘distrEx’.
Depends R(>= 3.4), distr(>= 2.8.0), distrEx(>= 2.8.0), RandVar(>= 1.2.0), MASS, stats4, methods
Imports startupmsg, sfsmisc, graphics, stats, grDevices
Suggests ismev, evd,
Enhances RobExtremes
ByteCompile yes
License LGPL-3
Encoding latin1

URL http://distr.r-forge.r-project.org/

LastChangedDate {$LastChangedDate: 2022-06-13 19:41:57 +0200 (Mo, 13 Jun 2022) $}

LastChangedRevision {$LastChangedRevision: 1367 $}

VCS/SVNRevision 1395

NeedsCompilation no

Author Matthias Kohl [aut, cph],
  Peter Ruckdeschel [cre, cph],
  R Core Team [ctb, cph] (for source file 'format.perc')

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

Repository CRAN

Date/Publication 2022-11-16 03:50:18 UTC
**R topics documented:**

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>distrMod-package</td>
<td>4</td>
</tr>
<tr>
<td>.checkEstClassForParamFamily-methods</td>
<td>10</td>
</tr>
<tr>
<td>addAlphTrsp2col</td>
<td>10</td>
</tr>
<tr>
<td>asBias</td>
<td>11</td>
</tr>
<tr>
<td>asBias-class</td>
<td>12</td>
</tr>
<tr>
<td>asCov</td>
<td>13</td>
</tr>
<tr>
<td>asCov-class</td>
<td>14</td>
</tr>
<tr>
<td>asGRisk-class</td>
<td>15</td>
</tr>
<tr>
<td>asHampel</td>
<td>16</td>
</tr>
<tr>
<td>asHampel-class</td>
<td>17</td>
</tr>
<tr>
<td>asMSE</td>
<td>18</td>
</tr>
<tr>
<td>asMSE-class</td>
<td>19</td>
</tr>
<tr>
<td>asRisk-class</td>
<td>20</td>
</tr>
<tr>
<td>asRiskWithBias-class</td>
<td>21</td>
</tr>
<tr>
<td>asSemivar</td>
<td>22</td>
</tr>
<tr>
<td>asSemivar-class</td>
<td>23</td>
</tr>
<tr>
<td>asUnOvShoot</td>
<td>24</td>
</tr>
<tr>
<td>asUnOvShoot-class</td>
<td>25</td>
</tr>
<tr>
<td>asymmetricBias</td>
<td>26</td>
</tr>
<tr>
<td>asymmetricBias-class</td>
<td>27</td>
</tr>
<tr>
<td>BetaFamily</td>
<td>28</td>
</tr>
<tr>
<td>BiasType-class</td>
<td>29</td>
</tr>
<tr>
<td>BinomFamily</td>
<td>30</td>
</tr>
<tr>
<td>CauchyLocationFamily</td>
<td>31</td>
</tr>
<tr>
<td>CauchyLocationScaleFamily</td>
<td>32</td>
</tr>
<tr>
<td>checkL2deriv</td>
<td>33</td>
</tr>
<tr>
<td>Confint-class</td>
<td>34</td>
</tr>
<tr>
<td>confint-methods</td>
<td>36</td>
</tr>
<tr>
<td>distrModMASK</td>
<td>38</td>
</tr>
<tr>
<td>distrModOptions</td>
<td>39</td>
</tr>
<tr>
<td>Estimate-class</td>
<td>40</td>
</tr>
<tr>
<td>Estimator</td>
<td>43</td>
</tr>
<tr>
<td>EvenSymmetric</td>
<td>44</td>
</tr>
<tr>
<td>EvenSymmetric-class</td>
<td>45</td>
</tr>
<tr>
<td>existsPIC-methods</td>
<td>46</td>
</tr>
<tr>
<td>ExpScaleFamily</td>
<td>47</td>
</tr>
<tr>
<td>fitBias</td>
<td>48</td>
</tr>
<tr>
<td>fitBias-class</td>
<td>48</td>
</tr>
<tr>
<td>fitCov</td>
<td>49</td>
</tr>
<tr>
<td>fitCov-class</td>
<td>50</td>
</tr>
<tr>
<td>fitHampel</td>
<td>51</td>
</tr>
<tr>
<td>fitHampel-class</td>
<td>52</td>
</tr>
<tr>
<td>fitMSE</td>
<td>53</td>
</tr>
<tr>
<td>fitMSE-class</td>
<td>53</td>
</tr>
<tr>
<td>fitRisk-class</td>
<td>54</td>
</tr>
<tr>
<td>fitUnOvShoot</td>
<td>55</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>fiUnOvShoot-class</td>
<td>56</td>
</tr>
<tr>
<td>FunctionSymmetry-class</td>
<td>57</td>
</tr>
<tr>
<td>FunSymmList</td>
<td>58</td>
</tr>
<tr>
<td>FunSymmList-class</td>
<td>59</td>
</tr>
<tr>
<td>GammaFamily</td>
<td>59</td>
</tr>
<tr>
<td>InfoNorm</td>
<td>60</td>
</tr>
<tr>
<td>isKerAinKerB</td>
<td>61</td>
</tr>
<tr>
<td>L2GroupParamFamily-class</td>
<td>62</td>
</tr>
<tr>
<td>L2LocationFamily</td>
<td>64</td>
</tr>
<tr>
<td>L2LocationFamily-class</td>
<td>66</td>
</tr>
<tr>
<td>L2LocationScaleFamily</td>
<td>68</td>
</tr>
<tr>
<td>L2LocationScaleFamily-class</td>
<td>69</td>
</tr>
<tr>
<td>L2LocationUnknownScaleFamily</td>
<td>71</td>
</tr>
<tr>
<td>L2ParamFamily</td>
<td>73</td>
</tr>
<tr>
<td>L2ParamFamily-class</td>
<td>76</td>
</tr>
<tr>
<td>L2ScaleFamily</td>
<td>80</td>
</tr>
<tr>
<td>L2ScaleFamily-class</td>
<td>81</td>
</tr>
<tr>
<td>L2ScaleUnknownLocationFamily</td>
<td>83</td>
</tr>
<tr>
<td>LnScaleFamily</td>
<td>85</td>
</tr>
<tr>
<td>LogisticLocationScaleFamily</td>
<td>86</td>
</tr>
<tr>
<td>mceCalc-methods</td>
<td>87</td>
</tr>
<tr>
<td>MCEstimate-class</td>
<td>89</td>
</tr>
<tr>
<td>MCEstimator</td>
<td>91</td>
</tr>
<tr>
<td>MDestimator</td>
<td>93</td>
</tr>
<tr>
<td>meRes</td>
<td>98</td>
</tr>
<tr>
<td>MLEstimator</td>
<td>100</td>
</tr>
<tr>
<td>modifyModel-methods</td>
<td>103</td>
</tr>
<tr>
<td>NbinomFamily</td>
<td>105</td>
</tr>
<tr>
<td>negativeBias</td>
<td>106</td>
</tr>
<tr>
<td>NonSymmetric</td>
<td>107</td>
</tr>
<tr>
<td>NonSymmetric-class</td>
<td>108</td>
</tr>
<tr>
<td>norm</td>
<td>108</td>
</tr>
<tr>
<td>NormLocationFamily</td>
<td>109</td>
</tr>
<tr>
<td>NormLocationScaleFamily</td>
<td>110</td>
</tr>
<tr>
<td>NormLocationUnknownScaleFamily</td>
<td>111</td>
</tr>
<tr>
<td>NormScaleFamily</td>
<td>112</td>
</tr>
<tr>
<td>NormScaleUnknownLocationFamily</td>
<td>113</td>
</tr>
<tr>
<td>NormType</td>
<td>114</td>
</tr>
<tr>
<td>NormType-class</td>
<td>115</td>
</tr>
<tr>
<td>OddSymmetric</td>
<td>116</td>
</tr>
<tr>
<td>OddSymmetric-class</td>
<td>116</td>
</tr>
<tr>
<td>onesidedBias-class</td>
<td>117</td>
</tr>
<tr>
<td>ParamFamily</td>
<td>118</td>
</tr>
<tr>
<td>ParamFamily-class</td>
<td>122</td>
</tr>
<tr>
<td>ParamFamParameter</td>
<td>124</td>
</tr>
<tr>
<td>ParamFamParameter-class</td>
<td>125</td>
</tr>
<tr>
<td>PoisFamily</td>
<td>127</td>
</tr>
<tr>
<td>positiveBias</td>
<td>128</td>
</tr>
</tbody>
</table>
distrMod-package

distrMod – Object Oriented Implementation of 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: 2.9.0
Date: 2022-11-14
Depends: R(>= 3.4), distr(>= 2.8.0), distrEx(>= 2.8.0), RandVar(>= 1.2.0), MASS, stats4,methods
Imports: startupmsg, sfsmisc, graphics, stats, grDevices
Suggests: ismev, evd,
Enhances: RobExtremes
ByteCompile: yes
License: LGPL-3
URL: https://distr.r-forge.r-project.org/
VCS/SVNRevision: 1395
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)
  |
  |>"L2ParamFamily" [*]
  additional slots:
    L2deriv(EuclRandVarList), L2deriv.fct(function),
    L2derivSymm(FunSymmList), L2derivDistr(DistrList),
    L2derivDistrSymm(DistrSymmList), FisherInfo(PosSemDefSymmMatrix),
    FisherInfo.fct(function)
  |>
    |>"BinomFamily" [*]
    |>"PoisFamily" [*]
    |>"BetaFamily" [*]
    |>"NbimFamily" [*
    |>"NbinomwithSizeFamily" [*
    |>"NbinomMeanSizeFamily" [*
    |>"L2GroupParamFamily" 
  additional slots:
    LogDeriv(function)
  |>
    |>"L2ScaleShapeUnion" /VIRTUAL/
  |>
    |>"GammaFamily" [*]
  |>
    |>"L2LocationScaleUnion" /VIRTUAL/
  additional slots:
    locscalename(character)
  |>
    |>"L2LocationFamily" [*]
    |>"NormLocationFamily" [*]
    |>"L2ScaleFamily" [*]
    |>"NormScaleFamily" [*]
    |>"ExpScaleFamily" [*]
    |>"LnormScaleFamily" [*]
    |>"L2LocationScaleFamily" [*]
    |>"NormLocationScaleFamily" [*]
    |>"CauchyLocationScaleFamily" [*]
    |>"LogisticLocationScaleFamily" [*]
  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" [*]
distrMod-package

|"onesidedBias" Additional slots:
  sign(numeric)
|"asymmetricBias" [+] Additional slots:
u(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),
Infos(matrix), nuis.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"
one sided 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 distrModMASK()

Author(s)

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

References

Description

.addAlphTrsp2col

Description

 Adds alpha transparency to a given color.

Usage

.addAlphTrsp2col(col, alpha=255)
asBias

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@uni-oldenburg.de>

Examples

```r
## IGNORE_RDIFF_BEGIN
addAlphTrsp2col(rgb(1,0.3,0.03), 25)
## gives "#FF4C0819" on 32bit and "#FF4D0819" on 64bit
addAlphTrsp2col("darkblue", 25)
addAlphTrsp2col("#AAAAAAA",25)
palette(rainbow(6))
addAlphTrsp2col(2, 25)
## IGNORE_RDIFF_END
```

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

asBias()

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

asBias-class  Standardized Asymptotic Bias

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>
asCov

References

See Also
asRisk-class, asBias

Examples
new("asBias")

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
def 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**

| 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

**Extends**

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")

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**References**


**See Also**

`asRisk-class, asHampel`

**Examples**

`new("asHampel")`
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>

References


See Also

`asGRisk-class, asMSE`

Examples

`new("asMSE")`
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

*RiskType-class*
asRiskwithBias-class  Asymptotic 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@uni-oldenburg.de>

References

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@uni-oldenburg.de>

References

See Also
onesidedBias-class

Examples
asSemivar()
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”.
biasObject 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@uni-oldenburg.de>

References


See Also

asGRisk-class, asMSE
Examples

```r
asUnOvShoot()
```

---

**asUnOvShoot**  
*Generating function for asUnOvShoot-class*

**Description**

Generates an object of class "asUnOvShoot".

**Usage**

```r
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**

```r
asUnOvShoot()
```

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

Generating function for asymmetricBias-class

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@uni-oldenburg.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 \((0,1] \times (0,1]\) 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@uni-oldenburg.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)
```

BetaFamily

Generating function for Beta families

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@uni-oldenburg.de>

See Also

L2ParamFamily-class, Beta-class
BiasType-class

Examples

```r
(B1 <- BetaFamily())
FisherInfo(B1)
## IGNORE_RDIFF_BEGIN
ccheckL2deriv(B1)
## IGNORE_RDIFF_END
```

<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@uni-oldenburg.de>

References


See Also

RiskType-class

Examples

```r
aB <- positiveBias()
name(aB)
```
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)
CauchyLocationFamily

Generating function for Cauchy location families

Description

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

Usage

CauchyLocationFamily(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)

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

References


See Also

L2ParamFamily-class, Cauchy-class

Examples

(C1 <- CauchyLocationFamily())
plot(C1)
FisherInfo(C1)
### need smaller integration range:
checkL2deriv(C1)
CauchyLocationScaleFamily

Generating function for Cauchy location and scale families

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

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>loc</td>
<td>location</td>
</tr>
<tr>
<td>scale</td>
<td>scale</td>
</tr>
<tr>
<td>trafo</td>
<td>function in param or matrix: transformation of the parameter</td>
</tr>
</tbody>
</table>

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
```
(C1 <- CauchyLocationScaleFamily())
## synonymous: C1 <- CauchyFamily()
plot(C1)
FisherInfo(C1)
### need smaller integration range:
distrExoptions("ElowerTruncQuantile"=1e-4,"EupperTruncQuantile"=1e-4)
```
Generic function for checking the L2-derivative of an L2-differentiable family of probability measures.

Usage

```r
checkL2deriv(L2Fam, ...)  
```

Arguments

- `L2Fam`: L2-differentiable family of probability measures
- `x`: argument to be printed
- `...`: additional parameters (ignored for compatibility with S3 generic in case `print.relMatrix`)

Details

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

Value

A list with items `maximum.deviation`, `cent`, `consist`, and `condition` is invisibly returned, where `maximum.deviation` comprises the maximal absolute value of all entries in `cent` and `consist`. `cent` shows the expectation of `L2deriv(L2Fam)` (which should be 0), `consist` shows the difference between the Fisher information and `cov(L2deriv(L2Fam))` (which should be 0), and `condition` is the condition number of the Fisher information.

Note

The return value gives the non-rounded values (which will be machine dependent), whereas on argument `out==TRUE` (the default) we only issue the values up to 5 digits which should be independent of the machine. For the output of relative differences, we adjust accuracy to the size of the maximal (absolute) value of the Fisher information. In case of the consistency condition, at positions where the denominator is 0, we print a "."; this is done through helper S3 method `print.relMatrix`.

Author(s)

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


See Also

`L2ParamFamily-class`

Examples

```r
F1 <- new("L2ParamFamily")
checkL2deriv(F1)
```

Confint-class

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 (only complete cases) 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).
- **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@uni-oldenburg.de>

See Also

Estimator, confint, Estimate-class, trafo-methods
## Examples

```r
## 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)
}

x <- rgamma(50, scale = 0.5, shape = 3)
## parametric family of probability measures
G <- GammaFamily(scale = 1, shape = 2, trafo = mtrafo)
## MLE
res <- MLEstimator(x = x, ParamFamily = G)
## MLE
res <- MLEstimator(x = x, ParamFamily = G)

## MLE
res <- MLEstimator(x = x, ParamFamily = G)

## MLE
res <- MLEstimator(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")
```

### confint-methods

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'
confint(object, method, parm, level = 0.95, ...)
```

#### Arguments

- `object` in default / signature ANY case: a fitted model object, in signature Estimate case, an object of class Estimate
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.

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; its return value is 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: will return an object of class Confint which corresponds to a 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 trao and nuisance slots/parts.

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 <- MLEstimator(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)

## 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")
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

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
Author(s)

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

Examples

distrModOptions()

distrModOptions 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@uni-oldenburg.de>

See Also

`options`, `getOption`, `distroptions`, `getdistrOption`

Examples

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

---

Estimate-class

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

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 $\tau$ 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

$$
\tau: \Theta' \rightarrow \tilde{\Theta}, \quad \theta \mapsto \tau(\theta)
$$

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 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 $\tau(\theta) = D \theta$.

Note

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

Author(s)

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

---

Estimator

Function to compute estimates

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 MatrixorFunction – 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.
EvenSymmetric

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@uni-oldenburg.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**

```r
EvenSymmetric()
```

```r
## 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
**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^*.5)
```

**Arguments**

- `object`: L2ParamFamily
- `...`: further arguments used by specific methods.
- `warning`: logical: should a warning be issued if there exist no (partial) influence curves?

**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 $\ker I = \ker J$ 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@uni-oldenburg.de>

**References**


**See Also**

isKerAinKerB
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])
## IGNORE_RDIFF_BEGIN
checkL2deriv(E1)
## IGNORE_RDIFF_END
fiBias

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
fiBias-class

Examples
fiBias()

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

fiBias-class

Finite-sample Bias

Description
Class of finite-sample bias.

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

## 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")
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") }

Finite-sample mean square error

Description
Class of asymptotic mean square error.

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


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

(G1 <- GammaFamily())
FisherInfo(G1)
## IGNORE_RDIFF_BEGIN
checkL2deriv(G1)
## IGNORE_RDIFF_END

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

```r
## IGNORE_RDIFF_BEGIN
InfoNorm()
## The function is currently defined as
function(){ new("InfoNorm") }
## IGNORE_RDIFF_END
```

---

**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@uni-oldenburg.de>

Examples

```r
ma <- cbind(1,1,c(1,1,7))
D <- t(ma %*% c(0,1,-1))
## IGNORE_RDIFF_BEGIN
## note that results may vary according to BLAS
isKerAinKerB(D,ma)
isKerAinKerB(ma,D)
## IGNORE_RDIFF_END
```
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@uni-oldenburg.de>
References


See Also

L2ParamFamily-class, ParamFamily-class

Examples

```r
F1 <- new("L2GroupParamFamily")
plot(F1)
```

---

**L2LocationFamily**

*Generating function for L2LocationFamily-class*

Description

Generates an object of class "L2LocationFamily".

Usage

```r
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 Classes 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@uni-oldenburg.de>

References


See Also

L2LocationFamily-class

Examples

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 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.
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@uni-oldenburg.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-
butions 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@uni-oldenburg.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": 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 L2deriv; 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 "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@uni-oldenburg.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(),
loc scalename = 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@uni-oldenburg.de>

References


See Also

L2LocationScaleFamily-class

Examples

F1 <- L2LocationUnknownScaleFamily()
plot(F1)

Description

Generates an object of class "L2ParamFamily".

Usage

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

name character string: name of the family
distribution object of class "Distribution": member of the family
distrSymm object of class "DistributionSymmetry": symmetry of distribution.
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
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.
modifyParam function: mapping from the parameter space (represented by "param") to the distribution space (represented by "distribution").
props character vector: properties of the family
L2deriv.fct 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 used by modifyModel to move the L2deriv according to a change in the parameter, and to fill slot L2deriv. More specifically, let us call the parts main and nuisance of the parameter the unknown parameter. If this unknown parameter is one-dimensional, the return value of L2deriv.fct must be a function in argument x, which is vectorized, (i.e., callable for a vector-valued x), and has a one-dimensional, numeric return value. In case the dimension of the unknown parameter is larger than one, the return value must be a list of functions, each of which satisfies the conditions formulated for the case of a one-dimensional parameter of interest. The order of the components of this list is the same as the order of the parameter coordinates in main, followed by the ones in nuisance.
L2derivSymm object of class "FunSymmList": symmetry of the maps contained in L2deriv; a list of symmetry properties of the same length as the return value of L2deriv.fct.
L2derivDistr object of class "UnivarDistrList": distribution of L2deriv; the length of this list of univariate distributions must be of the same length as the return value of L2deriv.fct.
L2derivDistrSymm object of class "DistrSymmList": symmetry of the distributions contained in L2derivDistr; the length of this list of symmetry properties must be of the same length as the return value of L2deriv.fct.
FisherInfo.fct 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 "PosSemDefSymmMatrix": Fisher information of the family

.returnClsName: 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.

.withMDE: 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 mini-
   mum distance estimator which in turn then serves as starting point for roptest /
   robest (from package ROptEst). If TRUE (default) the latter alternative is used.
   Ignored if ROptEst is not used.

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@uni-oldenburg.de>

References

sertation.

See Also

L2ParamFamily-class

Examples

F1 <- L2ParamFamily()
plot(F1)
Description

Class of L2 differentiable parametric families.

Details

In the E-methods, diagnostics on the involved integrations are available if argument diagnostic is TRUE. Then there is attribute diagnostic attached to the return value, which may be inspected and accessed through showDiagnostic and getDiagnostic.

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. Its map slot must contain a list of functions. Each function in this list must have just one argument x, which is vectorized, (i.e., callable for a vector-valued x), and has a one-dimensional, numeric return value.
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. More specifically, let us call the parts main and nuisance of the parameter the unknown parameter. If this unknown parameter is one-dimensional, the return value of L2deriv.fct must be a function in argument x, which is vectorized, (i.e., callable for a vector-valued x), and has a one-dimensional, numeric return value. In case the dimension of the unknown parameter is larger than one, the return value must be a list of functions, each of which satisfies the conditions formulated for the case of a one-dimensional parameter of interest. The order of the components of this list is the same as the order of the parameter coordinates in main, followed by the ones in nuisance.

L2derivSymm object of class "FunSymmList": symmetry of the maps contained in L2deriv; a list of symmetry properties of the same length as the return value of L2deriv.fct.

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; the length of this list of univariate distributions must be of the same length as the return value of L2deriv.fct.

L2derivDistrSymm object of class "DistrSymmList": symmetry of the distributions contained in L2derivDistr; the length of this list of symmetry properties must be of the same length as the return value of L2deriv.fct.

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.

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.

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.

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

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

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

plot signature(x = "L2ParamFamily"): plot of distribution and L2deriv. More precisely, this method has arguments plot(x, withSweave = getdistrOption("withSweave"), main = FALSE, inner = TRUE, sub = FALSE, col.inner = par("col.main"), cex.inner = 0.8, bmar = par("mar")[1], tmar = par("mar")[3], ..., mfColRow = TRUE, to.draw.arg = NULL, withSubst = TRUE) 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", dimnms) where dimnms is either the row names of the trafo matrix rownames(trafo(x@param)) or if the last expression is NULL a vector "dim<dimnr>", dimnr running through the number of rows of the trafo matrix.

withSubst logical: if TRUE (default) pattern substitution for titles and lables is used; otherwise no substitution is used.

... 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". In all title and axis label arguments, if withSubst is TRUE, the following patterns are substituted:
"%C" class of argument x
"%A" deparsed argument x
"%D" time/date-string when the plot was generated

In addition, argument ... may contain arguments panel.first, panel.last, i.e., hook expressions to be evaluated at the very beginning and at the very end of each panel (within the then valid coordinates). To be able to use these hooks for each panel individually, they may also be lists of expressions (of the same length as the number of panels and run through in the same order as the panels).

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

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@uni-oldenburg.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".

Usage

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@uni-oldenburg.de>

References


See Also

L2ScaleFamily-class

Examples

F1 <- L2ScaleFamily()
plot(F1)
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 [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 = "L2ScaleFamily", param = "ParamFamParameter"): moves the L2-scale family model to parameter param

Author(s)

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

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) 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, CauchyLocationScaleFamily.

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"
LnormScaleFamily

Author(s)
Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.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 parametric family are filled.

Value
Object of class "L2ScaleFamily"

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

Generating function for Logistic location and scale families

Description

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

Usage

LogisticLocationScaleFamily(location = 0, scale = 1, trafo)

LOGISTINT2

Arguments

location 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. LOGISTINT2 is a constant used for the scale part of the Fisher information. More precisely LOGISTINT2 equals to

\[ \int_{-\infty}^{\infty} (\tanh(x/2)x - 1)^2 d\logis(x) \, dx. \]

Value

Object of class "L2LocationScaleFamily"

Author(s)

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

References


See Also

L2ParamFamily-class, Logis-class

Examples

```r
(L1 <- LogisticLocationScaleFamily())
## synonymous: L1 <- LogisticFamily()
plot(L1)
FisherInfo(L1)
### need smaller integration range:
distrExoptions("ElowerTruncQuantile"=1e-4,"EupperTruncQuantile"=1e-4)
checkL2deriv(L1)
distrExoptions("ElowerTruncQuantile"=1e-7,"EupperTruncQuantile"=1e-7)
set.seed(123)
x <- rlogis(100,location=1,scale=2)
CvMMDEstimator(x, L1)
```

---

mceCalc-methods

Methods for functions mceCalc and mleCalc in Package ‘distrMod’

Description

Methods for functions mceCalc and mleCalc in package *distrMod*:

Usage

```r
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, dropZeroDensity = TRUE, 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, ...)
```
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 \texttt{ParamFamily}; the parametric family at which to evaluate the estimator
- **criterion**: a function measuring the “goodness of fit”
- **startPar**: in case \texttt{optim} is used: a starting value for the parameter fit; in case \texttt{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 \texttt{MCEstimate}; may be missing
- **validity.check**: logical: shall return parameter value be checked for validity?
- **dropZeroDensity**: logical of length 1; shall observations with density zero be dropped? Optimizers like \texttt{optim} require finite values, so get problems when negative loglikelihood is evaluated.
- **...**: additional argument(s) for \texttt{optim} / \texttt{optimize}

Details

\texttt{mceCalc} is used internally by function \texttt{MCEstimator} to allow for method dispatch according to argument \texttt{PFam}; similarly, and for the same purpose \texttt{mleCalc} is used internally by function \texttt{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 \texttt{mleCalc} / \texttt{mceCalc} methods, there is the helper function \texttt{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 \texttt{ParamFamParameter}
- **crit.fct**: a function with the named components of theta as arguments returning the criterion value; used for profiling / coercing to class \texttt{mle}
**MCEstimate-class**

- **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

---

**MCEstimate-class**: MCEstimate-class.

**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 `MCEstimators`, `MDEstimators` or `MLEstimators`. More specifically, `MDEstimators`, `CvMMDDEstimators`, and `MLEstimators` return objects of classes `MDEstimate`, `CvMMDEstimate`, and `MLEstimate` respectively, which each are immediate subclasses of `MCEstimate` (without further slots, for internal use in method dispatch).

**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.
- **optimReturn**: object of class "ANY" the return value of the optimizer (or NULL if, e.g., closed form solutions are used).
- **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, MLEstimator, MDEstimator.

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.

optimReturn signature(object = "MCEstimate"): accessor function for slot optimReturn.

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@uni-oldenburg.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, nmsffx = ",", 
.with.checkEstClassForParamFamily = 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?
nmsfx character: a potential suffix to be appended to the estimator name.
.with.checkEstClassForParamFamily logical: Should a the end of the function .checkEstClassForParamFamily; defaults to TRUE; can be switched off for computational time or because this is already checked in a calling wrapper function.

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 "MCEstimate" 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@uni-oldenburg.de>

See Also

ParamFamily-class, ParamFamily, MCEstimate-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

# IGNORE_RDIFF_BEGIN
MCEstimator(x = x, ParamFamily = G, criterion = TotalVarDist,
            crit.name = "Total variation distance")
# IGNORE_RDIFF_END

## Hellinger minimum distance estimator
## Note: you can also use function MDEstimator!

# discretize Gamma distribution
distroptions(DistrResolution = 1e-8)
MCEstimator(x = x, ParamFamily = G, criterion = HellingerDist,
            crit.name = "Hellinger Distance", startPar = c(1,2))
distroptions(DistrResolution = 1e-6)

## or smooth empirical distribution (takes some time!)
#MCEstimator(x = x, ParamFamily = G, criterion = TotalVarDist,
#             asis.smooth.discretize = "smooth", crit.name = "Total variation distance")

# Hellinger minimum distance estimator
## Note: you can also use function MDEstimator!

# discretize Gamma distribution
distroptions(DistrResolution = 1e-8)
MCEstimator(x = x, ParamFamily = G, criterion = HellingerDist,
            crit.name = "Hellinger Distance", startPar = c(1,2))
distroptions(DistrResolution = 1e-6)

## or smooth empirical distribution (takes some time!)
#MCEstimator(x = x, ParamFamily = G, criterion = HellingerDist,
#             asis.smooth.discretize = "smooth", crit.name = "Hellinger distance")
```

---

MDEstimator

**Function to compute minimum distance estimates**
Description

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

Usage

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, nmsffx = "", .with.checkEstClassForParamFamily = TRUE)

CvMMDEstimator(x, ParamFamily, muDatOrMod = c("Mod", "Dat", "Other"), mu = NULL, paramDepDist = FALSE, startPar = NULL, Infos, trafo = NULL, penalty = 1e20, validity.check = TRUE, asvar.fct = .CvMMDCovariance, na.rm = TRUE, ..., .withEvalAsVar = TRUE, nmsffx = "", .with.checkEstClassForParamFamily = TRUE)

KolmogorovMDEstimator(x, ParamFamily, paramDepDist = FALSE, startPar = NULL, Infos, trafo = NULL, penalty = 1e20, validity.check = TRUE, asvar.fct, na.rm = TRUE, ..., .withEvalAsVar = TRUE, nmsffx = "", .with.checkEstClassForParamFamily = TRUE)

TotalVarMDEstimator(x, ParamFamily, paramDepDist = FALSE, startPar = NULL, Infos, trafo = NULL, penalty = 1e20, validity.check = TRUE, asvar.fct, na.rm = TRUE, ..., .withEvalAsVar = TRUE, nmsffx = "", .with.checkEstClassForParamFamily = TRUE)

HellingerMDEstimator(x, ParamFamily, paramDepDist = FALSE, startPar = NULL, Infos, trafo = NULL, penalty = 1e20, validity.check = TRUE, asvar.fct, na.rm = TRUE, ..., .withEvalAsVar = TRUE, nmsffx = "", .with.checkEstClassForParamFamily = TRUE)

CvMDist2(e1,e2,... )

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
muDatOrMod a character string specifying whether as integration measure mu in Cramer-von-Mises distance, the empirical cdf (corresponding to argument value "Dat") or the current model distribution (corresponding to argument value "Mod") or a given integration (probability) measure / distribution mu (corresponding to argument value "Other") is to be used; must be one of "Dat" (default) or "Mod" or "Other". You can specify just the initial letter; the default is "Mod".
mu optional integration (probability) measure for CvM MDE. defaults to NULL and is ignored in options muDatOrMod in "Dat" and "Mod"; in case "Other", it must be of class UnivariateDistribution.
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

trafo: an object of class Matrix or Function – 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).

... for the estimators: further arguments to criterion or optimize or optim, respectively; for CvMDist2, these can be used e.g. by E().

.withEvalAsVar: logical: shall slot asVar be evaluated (if asvar.fct is given) or just the call be returned?

nmsffx: character: a potential suffix to be appended to the estimator name.

e1: object of class "Distribution" or class "numeric"

e2: object of class "Distribution"

.with.checkEstClassForParamFamily: logical: Should a the end of the function .checkEstClassForParamFamily; defaults to TRUE; can be switched off for computational time or because this is already checked in a calling wrapper function.

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.

The functions CvMMDEstimator, KolmogorovMDEstimator, TotalVarMDEstimator, and HellingerMDEstimator are aliases where the distance is fixed. More specifically, CvMMDEstimator uses Cramer-von-Mises distance, see CvMDist with integration measure mu either equal to the empirical cdf or to the current best fitting model distribution; the alternative is selected by argument muDatOrMod). As it is asymptotically linear, asymptotic variances are available. In case of alternative "Dat", this variance is computed by means of helper function .CvMMDCovarianceWithMux, case of alternative "Mod" we use helper function .CvMMDCovariance. In both case one may use these helper function to get hand on the respective influence function. For covariances computed by .CvMMDCovariance, diagnostics on the involved integrations are available if argument diagnostic is TRUE. Then there is
attribute diagnostic attached to the return value, which may be inspected and accessed through `showDiagnostic` and `getDiagnostic`.

KolmogorovMDEstimator uses Kolmogorov distance, see `KolmogorovDist`, TotalVarMDEstimator, uses total variation distance, see `TotalVarDist` and HellingerMDEstimator uses Hellinger distance, see `HellingerDist`.

Function CvMDist2 calls `CvMDist` and computes the Cramer-von-Mises distance between distributions e1 and e2 with integration measure mu equal to e2; it is used in alternative "Mod" in CvMMDist.

Value

The estimators return an object of S4-class "MCEstimate" which inherits from class "Estimate". CvMDist2 returns the respective distance.

Theoretical Background

It should be noted that CvMMDist results in an asymptotically linear (hence asymptotically normal) estimator with an influence function which is always bounded; HellingerMDEstimator adapts, for growing sample size, the MLE estimator, hence is asymptotically efficient, while for finite sample size is bias robust. KolmogorovMDEstimator is square-root-n consistent but, due to the faceted level sets of the distance fails to be asymptotically normal. In the terminology of Donoho/Liu, TotalVarMDEstimator and HellingerMDEstimator rely on strong distances, while CvMMDist and KolmogorovMDEstimator use weak distances, so the latter ensure protection against larger classes of contamination (simply because the distribution balls based on the respective distances contain more elements).

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

References


See Also

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

Examples

```r
## (empirical) Data
set.seed(123)
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)
## or
KolmogorovMDEstimator(x = x, ParamFamily = G)

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

## these examples take too much time for R CMD check --as-cran
## von Mises minimum distance estimator with default mu = Mod
MDEstimator(x = x, ParamFamily = G, distance = CvMDist, asvar.fct = .CvMMDCovarianceWithMux)
## or
CvMMDEstimator(x = x, ParamFamily = G)
## or
CvMMDEstimator(x = x, ParamFamily = G, muDatOrMod="Mod")

## or with data based integration measure:
CvMMDEstimator(x = x, ParamFamily = G, muDatOrMod="Dat")

## von Mises minimum distance estimator with mu = N(0,1)
MDEstimator(x = x, ParamFamily = G, distance = CvMDist, mu = Norm())
## or, with asy Var
MDEstimator(x = x, ParamFamily = G, distance = CvMDist, mu = Norm(), asvar.fct = function(L2Fam, param, ...){
  .CvMMDCovariance(L2Fam=L2Fam, param=param, mu=Norm(), N = 400)
})
## synomymous to
CvMMDEstimator(x = x, ParamFamily = G, muDatOrMod="Other", mu = Norm())

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

---

**meRes**

help functions for mceCalc and mleCalc

### 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,
optReturn = NULL)
generic.criterion.fct(theta, Data, ParamFam, criterion.ff, fun, ...)
```

### 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 ParamFamParameter; the parameter value
- `crit.fct`: a function to fill slot minuslog1 when an object of class MCEstimate is coerced to class mle (from package stats4); to this end 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.
optReturn object of class “ANY” the return value of the optimizer (or NULL if, e.g., closed form solutions are 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 \( \text{fun} \) 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 \( \text{dim}(\text{object})[2] \), else \( \text{length}(\text{object}) \).

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

Author(s)

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

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

```r
MLEstimator(x, ParamFamily, startPar = NULL,
            Infos, trafo = NULL, penalty = 1e20,
            validity.check = TRUE, na.rm = TRUE, ...
            , .withEvalAsVar = TRUE, dropZeroDensity = TRUE, nmsffx = ",",
            .with.checkEstClassForParamFamily = 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?
- `dropZeroDensity` logical of length 1; shall observations with density zero be dropped? Optimizers like optim require finite values, so get problems when negative loglikelihood is evaluated.
- `nmsffx` character: a potential suffix to be appended to the estimator name.
- `.with.checkEstClassForParamFamily` logical: Should a the end of the function .checkEstClassForParamFamily; defaults to TRUE; can be switched off for computational time or because this is already checked in a calling wrapper function.
Details

The function uses \texttt{mleCalc} for method dispatch; this method by default calls \texttt{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 \texttt{<Matthias.Kohl@stamats.de>},
Peter Ruckdeschel \texttt{<peter.ruckdeschel@uni-oldenburg.de>}

See Also

\texttt{ParamFamily-class,ParamFamily,MCEstimator,MCEstimate-class,fitdistr,mle}

Examples

\begin{verbatim}
# 1. Binomial data
# (empirical) data
# seed for reproducibility:
set.seed(20200306)
x <- rbinom(100, size=25, prob=.25)

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

# 2. Poisson data

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

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

# 3. Normal (Gaussian) location and scale
# (empirical) data
# seed for reproducibility:
set.seed(20200306)
x <- rnorm(100)

## ML-estimate
\end{verbatim}
MLEstimator(x, NormLocationScaleFamily())
## compare:
c(mean(x), sd(x))

#############################
## 4. Gamma model
#############################
## (empirical) data
# seed for reproducibility:
set.seed(20200306)
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
cconfint(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
  fval0
}
modifyModel-methods

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

modifyModel(model, param,...)
## S4 method for signature 'ParamFamily,ParamFamParameter'
modifyModel(model,param,
.withCall = TRUE, ...)

```r
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
res2

## some profiling
par(mfrow=c(1,2))
plot(profile(res2))
par(mfrow=c(1,1))

#############################
## 5. Cauchy Location Scale model
#############################
(C <- CauchyLocationScaleFamily())
loc.true <- 1
scl.true <- 2

## (empirical) data
# seed for reproducibility:
set.seed(20200306)
x <- rcauchy(50, location = loc.true, scale = scl.true)

## Maximum likelihood estimator
(res <- MLEstimator(x = x, ParamFamily = C))
## Asymptotic (CLT-based) confidence interval
confint(res)
```
modifyModel-methods

## 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, ...)

## S4 method for signature 'ExpScaleFamily,ParamFamParameter'
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.
NbinomFamily

Generating function for Nbinomial families

Description

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@uni-oldenburg.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@uni-oldenburg.de>

References

NonSymmetric

See Also

onesidedBias-class

Examples

negativeBias()

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

---

NonSymmetric 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()

## 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")

norm  
Norm functions

Description
Functions to determine certain norms.

Usage
  EuclideanNorm(x)
  QuadFormNorm(x, A)
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"

Arguments

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

Value

the columnwise evaluated norms

Author(s)

Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.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)))
NormLocationScaleFamily

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())
## synonymous: N1 <- NormFamily()
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 parameteric family are filled.

Value

Object of class "L2LocationScaleFamily"

Author(s)

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

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

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

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

(N1 <- NormScaleFamily())
plot(N1)
FisherInfo(N1)
checkL2deriv(N1)
See Also

L2ParamFamily-class, Norm-class

Examples

```r
(N1 <- NormScaleUnknownLocationFamily())
plot(N1)
FisherInfo(N1)
checkL2deriv(N1)
```

Description

Generates an object of class "NormType".

Usage

```r
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@uni-oldenburg.de>

References


See Also

NormType-class

Examples

```r
## IGNORE_RDIFF_BEGIN
NormType()
## IGNORE_RDIFF_END
```

```
NormType-class

Norm Type

Description

Class of norm types.

Objects from the Class

Could be generated by \code{new("NormType")}; more frequently one will use the generating function \code{NormType}

Slots

\code{name} Object of class "character".

\code{fct} Object of class "function" — the norm to be evaluated.

Methods

\code{name} \code{signature(object = "NormType")}: accessor function for slot \code{name}.

\code{name<-} \code{signature(object = "NormType", value = "character")}: replacement function for slot \code{name}.

\code{fct} \code{signature(object = "NormType")}: accessor function for slot \code{fct}.

\code{fct<-} \code{signature(object = "NormType", value = "function")}: replacement function for slot \code{fct}.

Author(s)

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

References


See Also

\code{BiasType-class}

Examples

```r
## IGNORE_RDIFF_BEGIN
EuclNorm <- NormType("EuclideanNorm", EuclideanNorm)
\code{fct(EuclNorm)}
\code{name(EuclNorm)}
## IGNORE_RDIFF_END
```
OddSymmetric-class

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**

```r
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`. 
onesidedBias-class

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

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@uni-oldenburg.de>

References


See Also

*BiasType-class*

Examples

```r
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
```

---

**ParamFamily**

Generating function for ParamFamily-class

Description

Generates an object of class "ParamFamily".

Usage

```r
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 MCEstimating 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@uni-oldenburg.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), ## sd known!
                 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, .returnClsName = "ParamWithScaleFamParameter"),
                 distribution = Norm(mean = 0, sd = 1), ## mean known!
                 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, .returnClsName = "ParamWithScaleFamParameter"),
                   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,
ParamFamily-class

Parametric family of probability measures.

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.

diss.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.

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

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)

ParamFamParameter Generating function for ParamFamParameter-class

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 parameter
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@uni-oldenburg.de>

See Also

ParamFamParameter-class

Examples

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

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 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” 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@uni-oldenburg.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
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@uni-oldenburg.de>

References


See Also

onesidedBias-class

Examples

positiveBias()

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

Methods for print to the S4 classes in package distrMod;

Usage

## 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

## set options to maximal detailedness
show.old <- getdistrModOption("show.details")
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"]}
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))
}
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)

---

**ProbFamily-class**

*Family of probability measures*

**Description**

Class of families of probability measures.

**Objects from the Class**

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

**Slots**

- `name` Object of class "character": name of the family.
- `distribution` Object of class "Distribution": member of the family.
- `distrSymm` Object of class "DistributionSymmetry": symmetry of distribution.
- `props` Object of class "character": properties of the family.

**Methods**

- `name` signature(object = "ProbFamily"): accessor function for slot name.
- `name<-` signature(object = "ProbFamily"): replacement function for slot name.
- `distribution` signature(object = "ProbFamily"): accessor function for slot distribution.
- `distrSymm` signature(object = "ProbFamily"): accessor function for slot distrSymm.
- `props` signature(object = "ProbFamily"): accessor function for slot props.
- `props<-` signature(object = "ProbFamily"): replacement function for slot props.
- `addProp<-` signature(object = "ProbFamily"): add a property to slot props.
- `r` signature(object = "ProbFamily"): wrapped accessor to slot r of slot "Distribution".


**d** signature(object = "ProbFamily"): wrapped accessor to slot d of slot "Distribution".

**p** signature(object = "ProbFamily"): wrapped accessor to slot p of slot "Distribution".

**q** signature(object = "ProbFamily"): wrapped accessor to slot q of slot "Distribution".

**q.l** signature(object = "ProbFamily"): wrapped accessor to slot q of slot "Distribution" – for compatibility with RStudio or Jupyter IRKernel / synonymous to q.

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

Distribution-class

---

### QFNorm

**Generating function for QFNorm-class**

#### Description

Generates an object of class "QFNorm".

#### Usage

```r
QFNorm(name = "norm based on quadratic form",
       QuadForm = PosSemDefSymmMatrix(matrix(1)))
```

#### Arguments

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

#### Value

Object of class "QFNorm"

**Author(s)**

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

**References**


**See Also**

QFNorm-class
QFNorm-class

## IGNORE_RDIFF_BEGIN
QFNorm()

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

## IGNORE_RDIFF_END

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@uni-oldenburg.de>

References

See Also

NormType-class

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, as well as to estimate objects. In this context, `qqplot` produces a QQ plot of data (argument `x`) against a (model) distribution. If the second argument is of class 'Estimate', `qqplot` looks at the estimate.call-slot and checks whether it can use an argument ParamFamily to conclude on the model distribution. Graphical parameters may be given as arguments to `qqplot`. In all title and label arguments, if `withSubst` is TRUE, the following patterns are substituted:

"%C" class of argument `x`

"%A" deparsed argument `x`

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

Usage

```r
qqplot(x, y, ...)
## S4 method for signature 'ANY,UnivariateDistribution'
qqplot(x, y,
    n = length(x), withIdLine = TRUE,
    withConf = TRUE, withConf.pw = withConf, withConf.sim = withConf,
    plot.it = TRUE, datax = FALSE, xlab = deparse(substitute(x)),
    ylab = deparse(substitute(y)),
    ..., width = 10, height = 5.5, withSweave = getdistrOption("withSweave"),
    mfColRow = TRUE, n.CI = n, with.lab = FALSE, lab.pts = NULL,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
    withConf.pw = withConf, withConf.sim = withConf,
**Arguments**

- **x**: data to be checked for compatibility with distribution/model y.
- **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 in?
- **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`)?
- **datax**: logical; shall data be plotted on x-axis?
- **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
- **with.lab**: logical; shall observation labels be plotted in?
- **lab.pts**: character or NULL; observation labels to be used
- **attr.pre**: logical; do graphical attributes for plotted data refer to indices prior (TRUE) or posterior to selection via arguments `which.lbs`, `which.Order`, `which.nonlbs` (FALSE)?
- **which.lbs**: integer or NULL; which observations shall be labelled
- **which.Order**: integer or NULL; which of the ordered (remaining) observations shall be labelled
which.nonlbs  indices of the observations which should be plotted but not labelled; either an
integer vector with the indices of the observations to be plotted into graph or
NULL — then all non-labelled observations are plotted.

order.trafo  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 distribu-
tion?
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
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

added.points.CI logical; should CIs be plotted through additional points (and not only through
data points)?
cex.pch  magnification factor for the plotted symbols (for backward compatibility); it is
ignored once col.pch is specified.
col.pch  color for the plotted symbols (for backward compatibility); it is ignored once
col.pch is specified.
cex.pts  size of the points of the second argument plotted, can be a vector; if argument
attr.pre is TRUE, it is recycled to the length of all observations and determines
the sizes of all plotted symbols, i.e., the selection is done within this argument;
in this case argument col.npts is ignored. If attr.pre is FALSE, cex.pts is
recycled to the number of the observations selected for labelling and refers
to the index ordering after the selection. Then argument cex.npts determi-
nes the sizes of the shown but non-labelled observations as given in argument
which.nonlbs.
col.pts  color of the points of the second argument plotted, can be a vector as in cex.pts
(with col.npts as counterpart).
pch.pts  symbol of the points of the second argument plotted, can be a vector as in
cex.pts (with pch.npts as counterpart).
qqplot

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.
qqplot signature(x = "ANY", y = "Estimate"): produces a QQ plot of a dataset x against the theoretical quantiles of the model distribution of the model that can be reconstructed from the estimator y; more specifically, it tries to get hand at the argument 'ParamFamily' of the estimator's call; if this is available, internally this model is shifted to the estimated parameter by a call to modifyModel, and then this shifted model is used in a call to the (x = "ANY", y = "UnivariateDistribution")-method. 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.
- crit: A matrix with the lower and upper confidence bounds (computed by qqbounds).
- err: logical vector of length 2.

(elements crit and err are taken from the return value(s) of qqbounds).

Author(s)

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

References


See Also

- qqplot from package stats – the standard QQ plot function, qqplot from package distr for comparisons of distributions, and qqbounds, used by qqplot to produce confidence intervals.

Examples

```r
set.seed(123)
x <- rnorm(40, mean=15, sd=30)
qqplot(x, Chisq(df=15))
NF <- NormLocationScaleFamily(mean=15, sd=30)
qqplot(x, NF, with.lab=TRUE, which.Order=1:5, cex.lbs=1.3)
mlE <- MLEstimator(x, NF)
qqplot(x, mlE)
```
returnlevelplot

Methods for Function returnlevelplot in Package 'distrMod'

Description

We generalize the return level plot (which is one of the diagnostical plots provided package ismev, e.g., in function gev.diag), see also Coles' book below, to be applicable to distribution and probability model objects. In this context, returnlevelplot produces a rescaled QQ plot of data (argument x) against a (model) distribution. Graphical parameters may be given as arguments to returnlevelplot. In all title and label arguments, if withSubst is TRUE, the following patterns are substituted:

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

Usage

returnlevelplot(x, y, ...)  
## S4 method for signature 'ANY,UnivariateDistribution'
returnlevelplot(x,y,  
  n = length(x), withIdLine = TRUE,  
  withConf = TRUE, withConf.pw = withConf, withConf.sim = withConf,  
  plot.it = TRUE, datax = FALSE, MaxOrPOT = c("Max","POT"), npy = 365,  
  threshold = if(is(y,"GPareto")) NA else 0,  
  xlab = deparse(substitute(x)),  
  ylab = deparse(substitute(y)),  
  main = "",  
  ..., width = 10, height = 5.5, withSweave = getdistrOption("withSweave"),  
  mfColRow = TRUE, n.CI = n, with.lab = FALSE, lab.pts = NULL, which.lbs = NULL,  
  which.Order = NULL, which.nonlbs = NULL, attr.pre = FALSE, 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"), added.points.CI = TRUE,  
  cex.pch = par("cex"), col.pch = par("col"),  
  cex.pts = 1, col.pts = par("col"), pch.pts = 19,  
  cex.npts = 1, col.npts = grey(.5), pch.npts = 20,  
  cex.lbs = par("cex"), col.lbs = par("col"), adj.lbs = par("adj"),  
  alpha.trsp = NA, jit.fac = 0, jit.tol = .Machine$double.eps,  
  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,
returnlevelplot

debug = FALSE, withSubst = TRUE)
## S4 method for signature 'ANY,ProbFamily'
returnlevelplot(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)), ...)  
## S4 method for signature 'ANY,Estimate'
returnlevelplot(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)), ...)  

Arguments

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

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 in?

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 returnlevelplot)?

datax logical; shall data be plotted on x-axis?

MaxOrPOT a character string specifying whether it is used for block maxima ("Max") or for points over threshold ("POT"); must be one of "Max" (default) or "POT". You can specify just the initial letter.

npy number of observations per year/block.

threshold numerical; in case of MaxOrPOT="POT", this captures the (removed) threshold. If it is NA, it is reconstructed from the distribution y.

main Main title

xlab x-label

ylab y-label

... further parameters for method returnlevelplot 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
with.lab  logical; shall observation labels be plotted in?
lab.pts   character or NULL; observation labels to be used
attr.pre  logical; do graphical attributes for plotted data refer to indices prior (TRUE) or posterior to selection via arguments which.lbs, which.Order, which.nonlbs (FALSE)?
which.lbs integer or NULL; which observations shall be labelled
which.nonlbs indices of the observations which should be plotted but not labelled; either an integer vector with the indices of the observations to be plotted into graph or NULL — then all non-labelled observations are plotted.
which.Order 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
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
added.points.CI logical; should CIs be plotted through additional points (and not only through data points)?
cex.pch   magnification factor for the plotted symbols (for backward compatibility); it is ignored once col.pts is specified.
col.pch   color for the plotted symbols (for backward compatibility); it is ignored once col.pts is specified.
cxn.pts size of the points of the second argument plotted, can be a vector; if argument attr.pre is TRUE, it is recycled to the length of all observations and determines the sizes of all plotted symbols, i.e., the selection is done within this argument; in this case argument col.npts is ignored. If attr.pre is FALSE, cxn.pts is recycled to the number of the observations selected for labelling and refers to the index ordering after the selection. Then argument cxn.npts determines the sizes of the shown but non-labelled observations as given in argument which.nonlbs.

col.npts color of the points of the second argument plotted, can be a vector as in cxn.pts (with col.npts as counterpart).

pch.npts symbol of the points of the second argument plotted, can be a vector as in cxn.pts (with pch.npts as counterpart).

cex.npts size of the non-labelled points of the data argument plotted (may be a vector).

cex.lbs magnification factor for the plotted observation labels

col.lbs color for the plotted observation labels

adj.lbs adj parameter for the plotted observation labels

alpha.trsp alpha transparency to be added ex post to colors col.pch and col.lbs; 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.

jit.tol threshold for jittering: if distance between points is smaller than jit.tol, points are considered replicates.

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

debug logical; if TRUE additional output to debug confidence bounds.

withSubst logical; if TRUE (default) pattern substitution for titles and axis lables is used; otherwise no substitution is used.
Details

`returnlevelplot` signature(`x = "ANY", y = "UnivariateDistribution"`): produces a return level plot of a dataset `x` against the theoretical quantiles of distribution `y`.

`returnlevelplot` signature(`x = "ANY", y = "ProbFamily"`): produces a return level 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.

`returnlevelplot` signature(`x = "ANY", y = "Estimate"`): produces a return level plot of a dataset `x` against the theoretical quantiles of the model distribution of the model that can be reconstructed from the estimator `y`; more specifically, it tries to get hand at the argument `ParamFamily` of the estimator's call; if this is available, internally this model is shifted to the estimated parameter by a call to `modifyModel`, and then this shifted model is used in a call to the `(x = "ANY", y = "UnivariateDistribution")`-method. Passed through the `...` argument, all arguments valid for signature(`x = "ANY", y = "UnivariateDistribution"`) are also valid for this signature.

Value

As for function `returnlevelplot` 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 `NA`s.
- `crit`: A matrix with the lower and upper confidence bounds (computed by `qqbounds`).
- `err`: logical vector of length 2. (elements `crit` and `err` are taken from the return value(s) of `qqbounds`).

Note

The confidence bands given in our version of the return level plot differ from the ones given in package `ismev`. We use non-parametric bands, hence also allow for non-parametric deviances from the model, whereas in package `ismev` they are based on profiling, hence only check for variability within the parametric class.

Author(s)

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

References


See Also

`qqplot` from package `stats` – the standard QQ plot function, `qqplot` from package `distr` for comparisons of distributions, `qqplot` from this package and `qqbounds`, used by `returnlevelplot` to produce confidence intervals.

Examples

```r
set.seed(20190331)
returnlevelplot(r(Norm(15,sqrt(30)))(40), Chisq(df=15))
### more could be seen after installing RobExtremes and ismev
#

## IGNORE_RDIFF_BEGIN
## at R CMD check --as-cran, it does not find package cluster
## when trying to attach package rrcov
## so remove this from testing
if(require(RobExtremes) && require(ismev)){
  data(portpirie)
  gevfit <- gev.fit(portpirie[,2]) ## taken from example from ismev::gev.fit
  GEVF <- GEVFamily(scale=gevfit$mle[2],shape=gevfit$mle[3],loc=gevfit$mle[1])
  erg <- returnlevelplot(portpirie[,2], GEVF)
  print(names(erg))
  print(names(erg$plotArgs))
  print(names(erg$IdLineArgs))
  returnlevelplot(portpirie[,2], GEVF, datax=TRUE)
}
## IGNORE_RDIFF_END
```

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

\texttt{type} signature(object = "RiskType"): accessor function for slot type.
\texttt{show} signature(object = "RiskType")

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

---

\texttt{SelfNorm}  \hspace{1cm} \textit{Generating function for SelfNorm-class}

Description

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

Usage

\texttt{SelfNorm()}

Value

Object of class "SelfNorm"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

\texttt{SelfNorm-class}

Examples

```r
## IGNORE_RDIFF_BEGIN
SelfNorm()

## The function is currently defined as
function(){ new("SelfNorm") }
## IGNORE_RDIFF_END
```
**symmetricBias**

Generating function for symmetricBias-class

**Description**

Generates an object of class "symmetricBias".

**Usage**

```r
symmetricBias(name = "symmetric Bias")
```

**Arguments**

- `name` name of the bias type

**Value**

Object of class "symmetricBias"

**Author(s)**

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

**References**


**See Also**

- `symmetricBias-class`

**Examples**

```r
symmetricBias()
```

### The function is currently defined as
```r
function(){ new("symmetricBias", name = "symmetric Bias") }
```
symmetricBias-class  symmetric Bias Type

Description

Class of symmetric bias types.

Objects from the Class

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

Slots

\begin{itemize}
\item name  Object of class "character".
\end{itemize}

Methods

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

Extends

Class "BiasType", directly.

Author(s)

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

References


See Also

\texttt{BiasType-class}

Examples

\begin{verbatim}
symmetricBias()
## The function is currently defined as
function(){ new("symmetricBias", name = "symmetric Bias") }
\end{verbatim}
Methods for function `trafo` in package `distrMod`. There are accessor (`trafo`) and replacement (`trafo<-`) versions.

**Usage**

```r
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)
## 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.

As usual, `trafo` also is the accessor and replacement method for this slot. Its corresponding return value depends on the signature for which the accessor / replacement method is used. More specifically, for `trafo`, we have methods for the following signatures:
signature Estimate,missing: returns a list of length two with components fct and mat (see below)
signature Estimate,ParamFamParameter: returns a list of length two with components fct and mat (see below)
signature ParamFamParameter,missing: returns a matrix (see below)
signature ParamFamily,missing: returns a matrix (see below)
signature ParamFamily,ParamFamParameter: returns a list of length two with components fct and mat (see below)

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 \(\theta\).

For the this function \(\tau()\), we provide an accessor \(\text{trafo}.\text{fct}\) for signature ParamFamily-method returning this function.

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 \(\text{trafo}\) 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 \(\text{trafo}\) 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 \(\text{theta}\) is a list with entries \(\text{fval}\), the function value \(\tau(\theta)\), and \(\text{mat}\), 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, method \(\text{trafo}\) will return different return value types. For signature

Estimate,missing: it will return a list with entries \(\text{fct}\), the function \(\tau\), and \(\text{mat}\), the matrix \(\frac{\partial}{\partial \theta} \tau(\theta)\). function \(\tau\) will then return the list \(\text{list}(\text{fval}, \text{mat})\) mentioned above.

Estimate,ParamFamParameter: as signature Estimate,missing.

ParamFamParameter,missing: it will just return the corresponding matrix.

ParamFamily,missing: is just wrapper to signature ParamFamParameter,missing.

ParamFamily,ParamFamParameter: as signature Estimate,missing.
### 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
(res <- MLEstimator(x = x, ParamFamily = NS))
## confidence interval
confint(res)
```

---

**Description**

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

**Usage**

`trafoEst(fct, estimator)`

**Arguments**

- **fct**: a τ-like function, i.e., a function in the main part θ of the parameter returning a list list(fval, mat) where fval is the function value τ(θ) of the transformation, and mat, its derivative matrix at θ.
- **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
## 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>
trAsCov-class

References


See Also

trAsCov-class

Examples

trAsCov()

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

---

trAsCov-class Trace of asymptotic covariance

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`.

**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**

```r
ew("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 upto 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

* Beta model
  BetaFamily, 28
* Cauchy location and scale model
  CauchyLocationFamily, 31
  CauchyLocationScaleFamily, 32
* Gamma model
  GammaFamily, 59
* Hampel risk
  asHampel, 16
  asHampel-class, 17
  fiHampel, 51
  fiHampel-class, 52
* Logistic location and scale model
  LogisticLocationScaleFamily, 86
* Negative Binomial model
  NbinomFamily, 105
* Poisson model
  PoisFamily, 127
* S4 distribution class
  addAlphTrsp2col, 10
* algebra
  isKerAinKerB, 61
* array
  isKerAinKerB, 61
* asymptotic bias
  asBias, 11
  asBias-class, 12
* asymptotic covariance
  asCov, 13
  asCov-class, 14
  InfoNorm, 60
  NormType, 114
  QFNorm, 131
  SelfNorm, 144
  trAsCov, 150
  trAsCov-class, 151
* asymptotic mean square error
  asMSE, 18
* asymptotic risk
  asRisk-class, 20
  asRiskwithBias-class, 21
  asSemivar-class, 23
  asUnOvShoot, 24
  asUnOvShoot-class, 25
  asymmetricBias-class, 27
  BiasType-class, 29
  NormType-class, 115
  onesidedBias-class, 117
  QFNorm-class, 132
  symmetricBias-class, 146
* bias
  asSemivar, 22
  asSemivar-class, 23
  asymmetricBias, 26
  asymmetricBias-class, 27
  BiasType-class, 29
  negativeBias, 106
  norm, 108
  NormType-class, 115
  onesidedBias-class, 117
  positiveBias, 128
  symmetricBias, 145
  symmetricBias-class, 146
* binomial model
  BinomFamily, 30
* bounded influence curve
  existsPIC-methods, 46
* classes
  asBias-class, 12
  asCov-class, 14
  asGRisk-class, 15
  asHampel-class, 17
  asHampel-class, 17
  asMSE-class, 19
  asRisk-class, 20
  asRiskwithBias-class, 21
  asSemivar-class, 23
  asUnOvShoot-class, 25
  asymmetricBias-class, 27
INDEX

BiasType-class, 29
Confint-class, 34
Estimate-class, 40
EvenSymmetric-class, 45
fiBias-class, 48
fiCov-class, 50
fiHampel-class, 52
fiMSE-class, 53
fiRisk-class, 54
fiUnOvShoot-class, 56
FunctionSymmetry-class, 57
FunSymmList-class, 59
L2GroupParamFamily-class, 62
L2LocationFamily-class, 66
L2LocationScaleFamily-class, 69
L2ParamFamily-class, 76
L2ScaleFamily-class, 81
MCEstimate-class, 89
NonSymmetric-class, 108
NormType-class, 115
OddSymmetric-class, 116
onesidedBias-class, 117
ParamFamily-class, 122
ParamFamParameter-class, 125
ProbFamily-class, 130
QFNorm-class, 132
RiskType-class, 143
symmetricBias-class, 146
trAsCov-class, 151
trFiCov-class, 153
* confidence interval
  Confint-class, 34
* convex risk
  asGRisk-class, 15
* distribution
  addAlphTrsp2col, 10
distrModMASK, 38
distrModOptions, 39
ParamFamily, 118
qqplot, 133
returnlevelplot, 138
* documentation
  distrModMASK, 38
* estimate
  Estimate-class, 40
MCEstimate-class, 89
* even function
  EvenSymmetric, 44
  EvenSymmetric-class, 45
* existence of influence curves
  existsPIC-methods, 46
* exponential scale model
  ExpScaleFamily, 47
* family
  ProbFamily-class, 130
* finite-sample bias
  fiBias, 48
fiBias-class, 48
* finite-sample covariance
  fiCov, 49
fiCov-class, 50
trFiCov, 152
trFiCov-class, 153
* finite-sample risk
  fiRisk-class, 54
fiUnOvShoot, 55
fiUnOvShoot-class, 56
* global options
  distrModOptions, 39
* hplot
  qqplot, 133
  returnlevelplot, 138
* influence curve
  existsPIC-methods, 46
* info file
  distrModMASK, 38
* ker
  isKerAinKerB, 61
* location and scale model
  CauchyLocationFamily, 31
CauchyLocationScaleFamily, 32
LogisticLocationScaleFamily, 86
* location model
  NormLocationFamily, 109
NormLocationScaleFamily, 110
NormLocationUnknownScaleFamily, 111
NormScaleUnknownLocationFamily, 113
* lognormal scale model
  LnormScaleFamily, 85
* masking
  distrModMASK, 38
* math
  EvenSymmetric, 44
  FunSymmList, 58
NonSymmetric, 107
NormType, 114
OddSymmetric, 116
QFNorm, 131
* mean square error
asMSE-class, 19
fiMSE, 53
fiMSE-class, 53
* methods
.checkEstClassForParamFamily-methods, 10
* misc
distrModOptions, 39
* models
BetaFamily, 28
BinomFamily, 30
CauchyLocationFamily, 31
CauchyLocationScaleFamily, 32
checkL2deriv, 33
confint-methods, 36
ExpScaleFamily, 47
GammaFamily, 59
L2GroupParamFamily-class, 62
L2LocationFamily, 64
L2LocationFamily-class, 66
L2LocationScaleFamily, 68
L2LocationScaleFamily-class, 69
L2LocationUnknownScaleFamily, 71
L2ParamFamily, 73
L2ParamFamily-class, 76
L2ScaleFamily, 80
L2ScaleFamily-class, 81
L2ScaleUnknownLocationFamily, 83
LnormScaleFamily, 85
LogisticLocationScaleFamily, 86
mceCalc-methods, 87
meRes, 98
modifyModel-methods, 103
NbinomFamily, 105
NormLocationFamily, 109
NormLocationScaleFamily, 110
NormLocationUnknownScaleFamily, 111
NormScaleFamily, 112
NormScaleUnknownLocationFamily, 113
ParamFamily, 118
ParamFamily-class, 122
ParamFamParameter, 124
PoisFamily, 127
print-methods, 129
ProbFamily-class, 130
trafo-methods, 147
trafoEst, 149
validParameter-methods, 154
* normal location model
NormLocationFamily, 109
NormLocationScaleFamily, 110
NormLocationUnknownScaleFamily, 111
NormScaleUnknownLocationFamily, 113
* normal scale model
NormScaleFamily, 112
* odd function
OddSymmetric, 116
OddSymmetric-class, 116
* options
distrModOptions, 39
* package
distrMod-package, 4
* parametric family
ParamFamily-class, 122
* parameter
ParamFamParameter, 124
ParamFamily-class, 125
* parametric family
checkL2deriv, 33
L2GroupParamFamily-class, 62
L2LocationFamily, 64
L2LocationFamily-class, 66
L2LocationScaleFamily, 68
L2LocationScaleFamily-class, 69
L2LocationUnknownScaleFamily, 71
L2ParamFamily, 73
L2ParamFamily-class, 76
L2ScaleFamily, 80
L2ScaleFamily-class, 81
L2ScaleUnknownLocationFamily, 83
LnormScaleFamily, 85
LogisticLocationScaleFamily, 86
mceCalc-methods, 87
meRes, 98
modifyModel-methods, 103
NbinomFamily, 105
NormLocationFamily, 109
NormLocationScaleFamily, 110
NormLocationUnknownScaleFamily, 111
NormScaleFamily, 112
NormScaleUnknownLocationFamily, 113
ParamFamily, 118
ParamFamily-class, 122
PoisFamily, 127
print-methods, 129
ProbFamily-class, 130
trafo-methods, 147
trafoEst, 149
validParameter-methods, 154
* partial influence curve
existsPIC-methods, 46
* probability measure
ProbFamily-class, 130
* programming
distrModMASK, 38
* projector
  isKerAinKerB, 61
* pseudo inverse
  isKerAinKerB, 61
* risk
  asBias, 11
  asBias-class, 12
  asCov, 13
  asCov-class, 14
  asGRisk-class, 15
  asHampel, 16
  asHampel-class, 17
  asMSE, 18
  asMSE-class, 19
  asRisk-class, 20
  asRiskwithBias-class, 21
  asSemivar-class, 23
  asUnOvShoot, 24
  asUnOvShoot-class, 25
  asymmetricBias, 26
  asymmetricBias-class, 27
  BiasType-class, 29
  fiBias, 48
  fiBias-class, 48
  fiCov, 49
  fiCov-class, 50
  fiHampel, 51
  fiHampel-class, 52
  fiMSE, 53
  fiMSE-class, 53
  fiRisk-class, 54
  fiUnOvShoot, 55
  fiUnOvShoot-class, 56
  InfoNorm, 60
  negativeBias, 106
  norm, 108
  NormType, 114
  NormType-class, 115
  onesidedBias-class, 117
  positiveBias, 128
  QFNorm, 131
  QFNorm-class, 132
  RiskType-class, 143
  SelfNorm, 144
  symmetricBias, 145
  symmetricBias-class, 146
  trAsCov, 150
  trAsCov-class, 151
  trFiCov, 152
  trFiCov-class, 153
* robust
  asBias, 11
  asCov, 13
  asHampel, 16
  asMSE, 18
  asSemivar, 22
  asUnOvShoot, 24
  asymmetricBias, 26
  existsPIC-methods, 46
  fiBias, 48
  fiCov, 49
  fiHampel, 51
  fiMSE, 53
  fiUnOvShoot, 55
  InfoNorm, 60
  MDEstimator, 93
  negativeBias, 106
  norm, 108
  positiveBias, 128
  SelfNorm, 144
  symmetricBias, 145
  trAsCov, 150
  trFiCov, 152
* scale model
  ExpScaleFamily, 47
  LnNormScaleFamily, 85
  NormScaleFamily, 112
* semivariance
  asSemivar, 22
  asSemivar-class, 23
* symmetry
  EvenSymmetric, 44
  EvenSymmetric-class, 45
  FunctionSymmetry-class, 57
  FunSymmList, 58
  FunSymmList-class, 59
  NonSymmetric, 107
  NonSymmetric-class, 108
  OddSymmetric, 116
  OddSymmetric-class, 116
* univar
  Estimator, 43
  MCEstimator, 91
  MDEstimator, 93
  MLEstimator, 100
  .checkEstClassForParamFamily
get.criterion.fct (meRes), 98
getDiagnostic, 76, 96
getdistrModOption (distrModOptions), 39
getdistrOption, 40
ggetOption, 40
gev.diag, 138
HellingerDist, 96
HellingerMDEstimator (MDEstimator), 93
InfoNorm, 60
InfoNorm-class (QFNorm-class), 132
Infos (Estimate-class), 40
Infos<-,Estimate-method (Estimate-class), 40
 Infos<-,Estimate-method (Estimate-class), 40
isKerAinKerB, 46, 61
KolmogorovDist, 96
KolmogorovMDEstimator (MDEstimator), 93
L2deriv (L2ParamFamily-class), 76
L2deriv,L2ParamFamily,missing-method (L2ParamFamily-class), 76
L2deriv,L2ParamFamily,ParamFamParameter-method (L2ParamFamily-class), 76
L2derivDistr (L2ParamFamily-class), 76
L2derivDistr,L2ParamFamily-method (L2ParamFamily-class), 76
L2derivDistrSymm (L2ParamFamily-class), 76
L2derivDistrSymm,L2ParamFamily-method (L2ParamFamily-class), 76
L2GroupParamFamily-class, 62
L2LocationFamily, 64, 67
L2LocationFamily-class, 66
L2LocationScaleFamily, 68, 71
L2LocationScaleFamily-class, 69
L2LocationUnknownScaleFamily, 71
L2ParamFamily, 73, 79
L2ParamFamily-class, 76
L2ScaleFamily, 80, 83
L2ScaleFamily-class, 81
L2ScaleUnknownLocationFamily, 83
length,ParamFamParameter-method
 (ParamFamParameter-class), 125
LnormScaleFamily, 85
LogDeriv (L2GroupParamFamily-class), 62
LogDeriv,L2GroupParamFamily-method (L2GroupParamFamily-class), 62
LogDeriv<-,L2GroupParamFamily-method (L2GroupParamFamily-class), 62
LogisticFamily
 (LogisticLocationScaleFamily), 86
LogisticLocationScaleFamily, 86
LOGISTINT2
 (LogisticLocationScaleFamily), 86
main (ParamFamParameter-class), 125
main,Estimate-method (Estimate-class), 40
main,ParamFamily-method (ParamFamily-class), 122
main,ParamFamParameter-method (ParamFamParameter-class), 125
main,ParamWithScaleAndShapeFamParameter-method (ParamFamily-class), 125
main<-,ParamFamParameter-method (ParamFamParameter-class), 125
makeOKPar (ParamFamily-class), 122
makeOKPar,ParamFamily-method (ParamFamily-class), 122
MASKING (distrModMASK), 38
mceCalc, 92, 95, 98, 99, 101
mceCalc (mceCalc-methods), 87
mceCalc,numeric,ParamFamily-method (mceCalc-methods), 87
mceCalc-methods, 87
MCEstimate-class, 89
MCEstimator, 90, 91, 97, 101
MDEstimator-class (MCEstimate-class), 89
MDEstimator, 90, 93
meRes, 88, 98
method (MCEstimate-class), 89
method,MCEstimate-method (MCEstimate-class), 89
mle, 101
mleCalc, 98, 99, 101
INDEX

mleCalc (mceCalc-methods), 87
mleCalc, numeric, BinomFamily-method (mceCalc-methods), 87
mleCalc, numeric, NormLocationFamily-method (mceCalc-methods), 87
mleCalc, numeric, NormLocationScaleFamily-method (mceCalc-methods), 87
mleCalc, numeric, NormScaleFamily-method (mceCalc-methods), 87
mleCalc, numeric, ProbFamily-method (mceCalc-methods), 87
mleCalc-methods (mceCalc-methods), 87
MLEstimate-class (MCEstimate-class), 89
MLEstimator, 90, 100
modifyModel (modifyModel-methods), 103
modifyModel, ExpScaleFamily, ParamFamParameter-method (modifyModel-methods), 103
modifyModel, GammaFamily, ParamFamParameter-method (modifyModel-methods), 103
modifyModel, L2LocationFamily, ParamFamParameter-method (modifyModel-methods), 103
modifyModel, L2LocationScaleFamily, ParamFamParameter-method (modifyModel-methods), 103
modifyModel, L2ScaleFamily, ParamFamParameter-method (modifyModel-methods), 103
modifyModel, ParamFamily, ParamFamParameter-method (modifyModel-methods), 103
modifyModel-methods, 103
modifyParam (ParamFamily-class), 122
modifyParam, ParamFamily-method (ParamFamily-class), 122

NA, 137, 142
name, BiasType-method (BiasType-class), 29
name, Estimate-method (Estimate-class), 40
name, NormType-method (NormType-class), 115
name, ProbFamily-method (ProbFamily-class), 130
name.estimate (Confint-class), 34
name.estimate, Confint-method (Confint-class), 34

name<-, BiasType-method (BiasType-class), 29
name<-, Estimate-method (Estimate-class), 40
name<-, NormType-method (NormType-class), 115
name<-, ProbFamily-method (ProbFamily-class), 130
NbinomFamily, 105
NbinomMeanSizeFamily (NbinomFamily), 105
NbinomwithSizeFamily (NbinomFamily), 105
negativeBias, 106
NonSymmetric, 107, 108
NonSymmetric-class, 108
norm, 108
norm (asRiskwithBias-class), 21
norm, asRiskwithBias-method (asRiskwithBias-class), 21
NormFamily (NormLocationScaleFamily), 110
NormLocationFamily, 109
NormLocationScaleFamily, 110
NormLocationUnknownScaleFamily, 111
NormScaleFamily, 112
NormScaleUnknownLocationFamily, 113
NormType, 114, 115
normtype (asRiskwithBias-class), 21
normtype, asRiskwithBias-method (asRiskwithBias-class), 21
NormType-class, 115
normtype<-, (asRiskwithBias-class), 21
normtype<-, asRiskwithBias-method (asRiskwithBias-class), 21
nu (asymmetricBias-class), 27
nu, asymmetricBias-method (asymmetricBias-class), 27
nu<-, (asymmetricBias-class), 27
nu<-, asymmetricBias-method (asymmetricBias-class), 27
nuisance (ParamFamParameter-class), 125
nuisance, Estimate-method (Estimate-class), 40
nuisance, ParamFamily-method (ParamFamily-class), 122
nuisance, ParamFamParameter-method (ParamFamParameter-class), 125
nuisance, ParamWithScaleAndShapeFamParameter-method (ParamFamParameter-class), 125
nuisance.estimate (Confint-class), 34
nuisance.estimate,Confint-method (Confint-class), 34
nuisance<- (ParamFamParameter-class), 125
nuisance<-,ParamFamParameter-method (ParamFamParameter-class), 125

OddSymmetric, 116, 117
OddSymmetric-class, 116
onesidedBias-class, 117
optimReturn (MCEstimate-class), 89
optimReturn,MCEstimate-method (MCEstimate-class), 89
optimwarn (MCEstimate-class), 89
optimwarn,MCEstimate-method (MCEstimate-class), 89

options, 40

p,ProbFamily-method (ProbFamily-class), 130
par, 78
param,ParamFamily-method (ProbFamily-class), 122
ParamFamily, 92, 97, 101, 118
ParamFamily-class, 122
ParamFamParameter, 75, 119, 124
ParamFamParameter-class, 125
ParamWithScaleAndShapeFamParameter-class (ParamFamParameter-class), 125
ParamWithScaleFamParameter-class (ParamFamParameter-class), 125
ParamWithShapeFamParameter-class (ParamFamParameter-class), 125
plot, 78
plot (L2ParamFamily-class), 76
plot,ParamFamily,missing-method (L2ParamFamily-class), 76
plot,ParamFamily,missing-method (ParamFamily-class), 122
plot-methods (L2ParamFamily-class), 76
plot.default, 78
plot.stepfun, 78
PoisFamily, 127
positiveBias, 128
print,Confint-method (Confint-class), 34
print,Estimate-method (Estimate-class), 40
print,ShowDetails-method (print-methods), 129
print-methods, 129

QFNorm, 131
QFNorm-class, 132
qquadbounds, 137, 143
qqplot, 133, 134, 137, 143
qqplot,ANY,Estimate-method (qqplot), 133
qqplot,ANY,ProbFamily-method (qqplot), 133
qqplot,ANY,UnivariateDistribution-method (qqplot), 133
qqplot-methods (qqplot), 133
quadNorm<-, (QFNorm-class), 132
QuadForm (QFNorm-class), 132
QuadForm,QFNorm-method (QFNorm-class), 132
QuadForm<-, (QFNorm-class), 132
QuadForm<-,QFNorm-method (QFNorm-class), 132
QuadFormNorm (norm), 108

r,ProbFamily-method (ProbFamily-class), 130
returnlevelplot, 138, 139, 142
returnlevelplot,ANY,Estimate-method (returnlevelplot), 138
returnlevelplot,ANY,ProbFamily-method (returnlevelplot), 138
returnlevelplot,ANY,UnivariateDistribution-method (returnlevelplot), 138
returnlevelplot-methods (returnlevelplot), 138
RiskType-class, 143
samplesize (Estimate-class), 40
samplesize,Estimate-method
(Estimate-class), 40
samplesize,numeric-method (meRes), 98
samplesize,estimate (Confint-class), 34
samplesize,estimate,Confint-method
(Confint-class), 34
SelfNorm, 144
SelfNorm-class (QFNorm-class), 132
show,asHampel-method (asHampel-class), 17
show,asUnOvShoot-method
(asUnOvShoot-class), 25
show,Confint-method (Confint-class), 34
show,Estimate-method (Estimate-class), 40
show,fiHampel-method (fiHampel-class), 52
show,fiUnOvShoot-method
(fiUnOvShoot-class), 56
show,MCEstimate-method
(MCEstimate-class), 89
show,ParamFamily-method
(ParamFamily-class), 122
show,ParamFamParameter-method
(ParamFamParameter-class), 125
show,ParamWithScaleAndShapeFamParameter-method
(ParamFamParameter-class), 125
show,RiskType-method (RiskType-class), 143
show,details (distrModOptions), 39
showDiagnostic, 76, 96
sign (onesidedBias-class), 117
sign,asSemivar-method
(asSemivar-class), 23
sign,onesidedBias-method
(onesidedBias-class), 117
sign<-, asSemivar-method
(asSemivar-class), 23
sign<-, onesidedBias-method
(onesidedBias-class), 117
startPar (ParamFamily-class), 122
startPar, MCEstimate-method
(MCEstimate-class), 89
startPar, ParamFamily-method
(ParamFamily-class), 122
svd, 62
symmetricBias, 145
symmetricBias-class, 146
TotalVarDist, 96
TotalVarMDEstimator (MDEstimator), 93
trafo (trafo-methods), 147
trafo,Estimate,missing-method
(trafo-methods), 147
trafo,Estimate,ParamFamParameter-method
(trafo-methods), 147
trafo,ParamFamily,missing-method
(trafo-methods), 147
trafo,ParamFamily,ParamFamParameter-method
(trafo-methods), 147
trafo,ParamFamParameter,missing-method
(trafo-methods), 147
trafo,ParamWithScaleAndShapeFamParameter,missing-method
(trafo-methods), 147
trafo-methods, 147
trafo.estimate (Confint-class), 34
trafo.estimate,Confint-method
(Confint-class), 34
trafo.fct (trafo-methods), 147
trafo.fct,ParamFamily-method
(trafo-methods), 147
trafo.fct-methods (trafo-methods), 147
trafo<-, (trafo-methods), 147
trafo<-, ParamFamily-method
(trafo-methods), 147
trafo<-, ParamFamParameter-method
(trafo-methods), 147
trafoEst, 149
trAsCov, 150, 152
trAsCov-class, 151
trFiCov, 152, 153
trFiCov-class, 153
type,Confint-method (Confint-class), 34
type,RiskType-method (RiskType-class), 143
untransformed.asvar (Estimate-class), 40
untransformed.asvar,Estimate-method
(Estimate-class), 40
untransformed.estimate
(Estimate-class), 40
untransformed.estimate,Estimate-method
(Estimate-class), 40

INDEX
validParameter
  (validParameter-methods), 154
validParameter,BinomFamily-method
  (validParameter-methods), 154
validParameter,L2LocationFamily-method
  (validParameter-methods), 154
validParameter,L2LocationScaleFamily-method
  (validParameter-methods), 154
validParameter,L2ScaleFamily-method
  (validParameter-methods), 154
validParameter,L2ScaleShapeUnion-method
  (validParameter-methods), 154
validParameter,L2ScaleUnion-method
  (validParameter-methods), 154
validParameter,ParamFamily-method
  (validParameter-methods), 154
validParameter,PoisFamily-method
  (validParameter-methods), 154
validParameter-methods, 154

width (asUnOvShoot-class), 25
width,asUnOvShoot-method
  (asUnOvShoot-class), 25
width,fiUnOvShoot-method
  (fiUnOvShoot-class), 56
withPosRestr (ParamFamParameter-class), 125
withPosRestr,ParamWithShapeFamParameter-method
  (ParamFamParameter-class), 125
withPosRestr<-
  (ParamFamParameter-class), 125
withPosRestr<-,ParamWithShapeFamParameter-method
  (ParamFamParameter-class), 125