Package ‘EQL’

January 20, 2025

Version 1.0-1

Date 2009-06-18

Title Extended-Quasi-Likelihood-Function (EQL)

Depends ttutils(>= 0.1-0)

Imports lattice(>= 0.17-17)

Author Thorn Thaler <`thorn.thaler@thothal.com`>

Maintainer Thorn Thaler <`thorn.thaler@thothal.com`>

Description Computation of the EQL for a given family of variance functions, Saddlepoint-approximations and related auxiliary functions (e.g. Hermite polynomials).

License GPL-2

Repository CRAN

Date/Publication 2019-08-30 13:04:27 UTC

NeedsCompilation no

Contents

EQL-package 2
approximation 3
cumulants 4
edgeworth 8
eql 10
hermite 14
saddlepoint 16
varianceFamily 17

Index **20**

EQL-package	*Extended Quasi-Likelihood Function (EQL)*

Description

The package **EQL** contains functions for

- computation of the EQL for a given family of variance functions
- Edgeworth approximations
- Saddlepoint approximations
- related auxiliary functions (e.g. Hermite polynomials)

See section 'Index' for a list of exported functions. Section 'Internals' lists the internal functions of the package, which are not exported but may be referenced by `EQL:::.functionName`.

Details

Version:	1.0-0
Date:	2009-06-18
Depends:	ttutils(>= 0.1-0)
Imports:	lattice(>= 0.17-17)
License:	GPL-2
Built:	R 2.8.1; ; 2009-06-22 15:24:08; unix

Index

`approximation`	: Approximation class
`cumulants`	: Cumulant class for the saddlepoint approximation
`edgeworth`	: Edgeworth approximation
`eql`	: Maximization of the EQL function for a particular
	: variance family for a given set of parameters
`extBinomialVarianceFamily`	: Extended binomial variance family ($V(\mu) = \mu^k(1-\mu)^l$)
`gammaCumulants`	: Cumulant functions of the Gamma distribution
`gaussianCumulants`	: Cumulant functions of the normal distribution
`hermite`	: Hermite polynomials
`inverseGaussianCumulants`	: Cumulant functions of the inverse-gaussian distribution
`powerVarianceFamily`	: Power variance family ($V(\mu) = \mu^\theta$)
`saddlepoint`	: Saddlepoint approximation
`varianceFamily`	: Variance family class

Internals

`.eql`	: Computes a single EQL value
`.getFactor`	: Calculates the normalizing factor for the saddlepoint approximation
`.missingFormals`	: Check if a list contains all the arguments of a particular function

Author(s)

Thorn Thaler <thorn.thaler@thothal.com>

Maintainer: Thorn Thaler <thorn.thaler@thothal.com>

See Also

ttutils

`approximation` *An Approximation Class*

Description

An object of class `approximation` stores the approximation nodes together with the approximation itself. Some meta information is saved as well.

Usage

```
approximation(y, approx, n,
              type = c("standardized", "mean", "sum"),
              approx.type = c("Edgeworth", "Saddlepoint"))

## S3 method for class 'approximation'
plot(x, do.annotate = TRUE, ...)
```

Arguments

`y`	a numeric vector or array giving the approximation nodes.
`approx`	a numeric vector or array giving the approximated values at `y`.
`n`	a positive integer giving the number of i.i.d. random variables in the sum.
`type`	a character string giving the type of approximation, i.e. which kind of sum is to be approximated. Must be one of (“standardized”, “mean”, “sum”), representing the shifted and scaled sum, the weighted sum and the raw sum. Can be abbreviated.
`approx.type`	a character string giving the approximation routine used. Must be one of of (“Edgeworth”, “Saddlepoint”) and can be abbreviated.

`x`	an `approximation` object.
`do.annotate`	logical. If `TRUE` (the default) the value of the argument `n` is added to the plot.
`...`	other parameters to be passed through to the plotting function. Giving a named argument for any of • `main` • `sub` • `type` • `xlab` • `ylab` overrides the default values in `plot.approximation`.

Value

An object of class `approximation` contains the following components:

`y`	a numeric vector of values at which the approximation is evaluated (the approximation nodes).
`approx`	a numeric vector containing the approximated values at the approximation nodes `y`.
`type`	a character string giving the type of sum considered, i.e. one of ("standardized", "mean", "sum").
`n`	a positive integer giving the number of i.i.d. random variables in the sum.
`approx.type`	a character string giving the type of approximation.

Author(s)

Thorn Thaler

See Also

`edgeworth`, `saddlepoint`

`cumulants` *Cumulants Class For Saddlepoint Approximations*

Description

A `cumulants` object contains all the cumulant functions that are needed to calculate the saddlepoint approximation.

The predefined functions

- `gammaCumulants`,
- `gaussianCumulants` and
- `inverseGaussianCumulants`

compute the cumulant functions for the normal, gamma and inverse gaussian distribution, respectively.

Usage

```
cumulants(saddlef, cgf = NULL, kappa2f = NULL, rho3f = NULL,
          rho4f = NULL, cgf.deriv = NULL,
          domain = interval(-Inf, Inf), ...)

gammaCumulants(shape, scale)
gaussianCumulants(mu, sigma2)
inverseGaussianCumulants(lambda, nu)

## S3 method for class 'cumulants'
check(object, ...)
```

Arguments

`saddlef`	the saddlepoint function. Corresponds to the inverse of the first derivative of the cumulant generating function (CGF).
`cgf`, `cgf.deriv`	`cgf` is the cumulant generating function. If `NULL` (the default), it will be derived from `cgf.deriv` (the generic derivative function of the cgf).
`kappa2f`	the variance function. If `NULL` (the default), it will be derived from the function `cgf.deriv`.
`rho3f`, `rho4f`	the 3rd and the 4th standardized cumulant function, respectively. If `NULL` (the default), the functions will be derived from `cgf.deriv` if supplied. If neither the cumulants nor `cgf.deriv` are supplied, a warning will be displayed and a flag is set in the output. In this case, saddlepoint approximations cannot make use of the correction term (see `saddlepoint` for further details).
`domain`	an object of type `interval` giving the domain of the random variable. Will be needed to calculate the normalizing factor. See `interval` for further information.
`...`	additional parameters to be passed to the cumulant functions, respectively function check. See section 'Details' for further information.
`shape`, `scale`	shape and scale parameter for the gamma distribution.
`mu`, `sigma2`	mean and variance parameter for the normal distribution.
`lambda`, `nu`	parameters for the inverse Gaussian distribution.
`object`	an object to be tested whether or not it meets the formal requirements.

Details

Basically, there are two ways to specify the cumulant functions using `cumulants`. The first one is to specify each of the following functions seperately:

- `cgf`
- `kappa2f`
- `rho3f`
- `rho4f`

Since the functions may (and probably will) depend on some additional parameters, it is necessary to include these parameters in the respective argument lists. Thus, these additional parameters must be passed to `cumulants` as *named* parameters as well. To be more specific, if one of the above functions has an extra parameter `z`, say, the particular value of `z` must be passed to the function `cumulants` as well (see the example). In any case, the first argument of the cumulant functions must be the value at which the particular function will be evaluated.

The other way to specify the cumulant functions is to specify the generic derivative of the CGF `cgf.deriv`. Its first argument must be the order of the derivative and its second the value at which it should be evaluated, followed by supplementary arguments. `cgf.deriv` must be capable to return the CGF itself, which corresponds to the zeroth derivative.

The function `cumulants` performs a basic check to test if all needed additional parameters are supplied and displays a warning if there are extra arguments in the cumulant functions, which are not specified.

The generic function `check` for the class `cumulants` tests if

- an object has the same fields as an `cumulants` object and
- the cumulant functions are properly vectorized, i.e. if they return a vector whenever the argument is a vector.

Value

`cumulants` returns an object of class `cumulants` containing the following components:

`K`	the cumulant function.
`mu.inv`	the saddlepoint function.
`kappa2`	the variance function.
`rho3`, `rho4`	the 3rd and the 4th standardized cumulant functions.
`domain`	an interval giving the domain of the random variable.
`extra.params`	extra parameter passed to `cumulants`, typically parameters of the underlying distribution.
`type`	character string equating either to “explicit” or “implicit” indicating whether the cumulant functions were passed explicitly or were derived from the generic derivative of the CGF.
`missing`	logical. If `TRUE`, the 3rd and/or the 4th cumulant function were not defined.

`gammaCumulants`, `gaussianCumulants` and `inverseGaussianCumulants` return a `cumulants` object representing the cumulant functions of the particular distribution.

Note

If it happens that one of the cumulant functions `f`, say, does not need any extra arguments while the others do, one have to define these extra arguments for `f` nonetheless. The reason is that `cumulants` passes any additional arguments to all defined cumulant functions and it would end up in an error, if a function is not capable of dealing with additional arguments.

Hence, it is good practice to define all cumulant functions for the same set of arguments, needed or not. An alternative is to add `...` to the argument list in order to absorb any additional arguments.

The functions must be capable of handling vector input properly.

Supplementary arguments *must not* be named similar to the arguments of `cumulants` (especially any abbreviation must be avoided), for the argument matching may match an argument (thought to be an extra argument for one of the cumulant function) to an argument of `cumulants`. The same problem may arise, if additional cumulant function parameters are not named.

Author(s)

Thorn Thaler

References

Reid, N. (1991). Approximations and Asymptotics. *Statistical Theory and Modelling*, London: Chapman and Hall.

See Also

`edgeworth`, `saddlepoint`

Examples

```
# Define cumulant functions for the normal distribution

saddlef <- function(x, mu, sigma2) (x-mu)/sigma2
cgf <- function(x, mu, sigma2) mu*x+sigma2*x^2/2

## Not run:

# cgf, saddlef, kappa2, rho3 and rho4 must have the same argument lists!
# Functions are _not_ properly vectorized!
kappa2 <- function(x, sigma2) sigma2
rho3 <- function(x) 0
rho4 <- function(x) 0

cc <- cumulants(saddlef, cgf, kappa2, rho3, rho4, mu=0, sigma2=1)

check(cc) # FALSE


## End(Not run)

kappa2 <- function(x, mu, sigma2)
    rep(sigma2, length(x))
rho3 <- function(x, mu, sigma2)   # or function(x, ...)
    rep(0, length(x))
rho4 <- function(x, mu, sigma2)   # or function(x, ...)
    rep(0, length(x))

cc <- cumulants(saddlef, cgf, kappa2, rho3, rho4, mu=0, sigma2=1)

cc$K(1:2)       # 0.5 2
cc$kappa2(1:2)  # 1 1
```

```
cc$mu.inv(1:2)  # 1 2
cc$rho3(1:2)    # 0 0
cc$rho4(1:2)    # 0 0

check(cc) # TRUE

# The same using the generic derivative of the cgf

K.deriv <- function(n, x, mu, sigma2) {
  if (n <= 2) {
    switch(n + 1,
           return(mu * x + sigma2 * x ^ 2 / 2), # n == 0
           return(mu + sigma2 * x),             # n == 1
           return(rep(sigma2, length(x))))      # n == 2
  } else {
    return(rep(0, length(x)))                   # n >= 3
  }
}

cc <- cumulants(saddlef, cgf.deriv=K.deriv, mu=0, sigma2=1)

cc$K(1:2)       # 0.5 2
cc$kappa2(1:2)  # 1 1
cc$mu.inv(1:2)  # 1 2
cc$rho3(1:2)    # 0 0
cc$rho4(1:2)    # 0 0

check(cc) # TRUE

# The same using a predefined function
cc <- gaussianCumulants(0, 1)

cc$K(1:2)       # 0.5 2
cc$kappa2(1:2)  # 1 1
cc$mu.inv(1:2)  # 1 2
cc$rho3(1:2)    # 0 0
cc$rho4(1:2)    # 0 0

check(cc) # TRUE
```

edgeworth *Edgeworth Approximation*

Description

Computes the Edgeworth expansion of either the standardized mean, the mean or the sum of i.i.d. random variables.

Usage

```
edgeworth(x, n, rho3, rho4, mu, sigma2, deg=3,
          type = c("standardized", "mean", "sum"))
```

Arguments

`x`	a numeric vector or array giving the values at which the approximation should be evaluated.
`n`	a positive integer giving the number of i.i.d. random variables in the sum.
`rho3`	a numeric value giving the standardized 3rd cumulant. May be missing if `deg <= 1`.
`rho4`	a numeric value giving the standardized 4th cumulant. May be missing if `deg <= 2`.
`mu`	a numeric value giving the mean. May be missing if `type = "standardized"`, since it is only needed for transformation purposes.
`sigma2`	a positive numeric value giving the variance. May be missing if `type= "standardized"`.
`deg`	an integer value giving the order of the approximation: • `deg=1`: corresponds to a normal approximation • `deg=2`: takes 3rd cumulant into account • `deg=3`: allows for the 4th cumulant as well. The default value is 3.
`type`	determines which sum should be approximated. Must be one of (“standardized”, “mean”, “sum”), representing the shifted and scaled sum, the weighted sum and the raw sum. Can be abbreviated.

Details

The Edgeworth approximation (EA) for the density of the standardized mean $Z = \frac{S_n - n\mu}{\sqrt{n\sigma^2}}$, where

- $S_n = Y_1 + \ldots + Y_n$ denotes the sum of i.i.d. random variables,
- μ denotes the expected value of Y_i,
- σ^2 denotes the variance of Y_i

is given by:

$$f_Z(s) = \varphi(s)[1 + \frac{\rho_3}{6\sqrt{n}} H_3(s) + \frac{\rho_4}{24n} H_4(s) + \frac{\rho_3^2}{72n} H_6(s)],$$

with φ denoting the density of the standard normal distribution and ρ_3 and ρ_4 denoting the 3rd and the 4th standardized cumulants of Y_i respectively. $H_n(x)$ denotes the nth Hermite polynomial (see `hermite` for details).

The EA for the mean and the sum can be obtained by applying the transformation theorem for densities. In this case, the expected value `mu` and the variance `sigma2` must be given to allow for an appropriate transformation.

Value

edgeworth returns an object of the class approximation. See approximation for further details.

Author(s)

Thorn Thaler

References

Reid, N. (1991). Approximations and Asymptotics. *Statistical Theory and Modelling*, London: Chapman and Hall.

See Also

approximation,hermite,saddlepoint

Examples

```
# Approximation of the mean of n iid Chi-squared(2) variables

n <- 10
df <- 2
mu <- df
sigma2 <- 2*df
rho3 <- sqrt(8/df)
rho4 <- 12/df
x <- seq(max(df-3*sqrt(2*df/n),0), df+3*sqrt(2*df/n), length=1000)
ea <- edgeworth(x, n, rho3, rho4, mu, sigma2, type="mean")
plot(ea, lwd=2)

# Mean of n Chi-squared(2) variables is n*Chi-squared(n*2) distributed
lines(x, n*dchisq(n*x, df=n*mu), col=2)
```

---

eql *The Extended Quasi-Likelihood Function*

Description

Computes the Extended Quasi Likelihood (EQL) function for a given set of variance functions from a particular variance family.

Usage

```
eql(formula, param.space, family = powerVarianceFamily(),
    phi.method = c("pearson", "mean.dev"), include.model = TRUE,
    smooth.grid = 10, do.smooth = dim(family) == 1,
    verbose = 1, ...)
```

```
## S3 method for class 'eql'
plot(x, do.points = (dim(x) == 1 && sum(!x$is.smoothed) <= 20),
     do.ci = TRUE, alpha = 0.95, do.bw = TRUE,
     show.max = TRUE, ...)
```

Arguments

`formula`	an object of class `formula` (or one that can be coerced to that class): a symbolic description of the model to be used to determine the parameters of the variance function.
`param.space`	a list of parameters for which the EQL value should be evaluated. If provided as a named list, the names must equal the names of the parameters defined by the variance family.
`family`	an object of class `varianceFamily` giving a parameterized family of variance functions. See `varianceFamily` for further details.
`phi.method`	a character string giving the name of the method used to estimate the dispersion parameter ϕ. Must be one of ("pearson", "mean.dev") representing the estimation of ϕ by the mean Pearson's statistic or by the mean deviance, respectively.
`include.model`	logical. If `TRUE` (the default) the final model is included in the output.
`x`	an object of class `eql`.
`do.smooth`, `smooth.grid`	`do.smooth` is a logical value and `smooth.grid` is an integer value giving the number of nodes for the smoothing process. If `do.smooth` is `TRUE`, smoothing is carried out by cubic splines on an equidistant grid with an amount of nodes equals to `smooth.grid` between two adjacent EQL values. Smoothing is currently only available for one-dimensional variance families, i.e. families that depend only on one parameter.
`verbose`	the amount of feedback requested: '0' or `FALSE` means no feedback, '1' or `TRUE` means some feedback (the default), and '2' means to show all available feedback. For the default setting, a progress bar will be displayed to give a rough estimation of the remaining calculation time. Full feedback prints the EQL value for each parameter combination.
`...`	further arguments to be passed to the `glm` routine and the plotting routine, respectively.
`do.points`, `show.max`	logical. If `do.points` is `TRUE`, the computed EQL values are marked in the plot. If `show.max` is `TRUE`, the maximum of the EQL function is emphasized in the plot.
`do.ci`, `alpha`	`do.ci` is a logical value, if `TRUE` a α confidence interval (respectively confidence ellipsoid) is added to the plot.
`do.bw`	logical. If `TRUE` (the default) a "black and white" plot is produced, otherwise colours are used.

Details

The EQL function as defined by *Nelder and Pregibon* (see 'References') is given by:

$$Q^+_\theta(y,\mu) = -\frac{1}{2}\log[2\pi\phi V_\theta(y)] - \frac{1}{2\phi}D_\theta(y,\mu),$$

where $D_\theta()$ and $V_\theta()$ denote the deviance function and the variance function, respectively, determined by the particular choice of the variance family.

The goal is to maximize the EQL function over μ and the not necessarily one-dimensional space of parameters θ. The function `eql` takes a particular finite set of candidate parameters and computes the corresponding EQL value for each of these parameters and returns the maximum EQL value for the given set. That implies that the function is only capable of capturing local maxima. If the maximum occurs at the boundary of the set, the set of parameters may be badly chosen and one should consider a larger set with the found maximum as an interior point.

The `plot` function is an important tool to investigate the structure of the EQL function. Confidence intervals and confidence ellipsoids give an idea of plausible parameter values for the variance function. The contour plot used for two-dimensional variance families is generated using the package **lattice**, which in turn relies on so called `trellis` plots. Hence, for two-dimensional families the `plot` function does not only generate the plot, but also returns the plot object to allow for further modifications of the plot. This is not true for one-dimensional variance models, which are plotted using the R standard graphical engine.

For large parameter sets the computation may take a long time. If no feedback is chosen, the function seems to be hung up, because the function does not provide any textual feedback while computing. Hence, a minimal feedback (including a progress bar) should be chosen to have an idea of the remaining calculation time.

An explicitely given deviance function speeds up calculation. A rather large amount of the total calculation time is used to determine the numerical values of the integral in the deviance function.

Value

`eql` returns an object of class `eql`, which contains the following components:

`eql`	a numerical vector with the computed eql values for the given set of parameter values. For one-dimensional variance families (i.e. those families with only one parameter), a smoothing operation can be performed to obtain intermediate values.
`param`	a `data.frame` containing the values of the parameters at which the eql function was evaluated.
`eql.max`	the maximum value of the eql function in the considered range.
`param.max`	a `data.frame` containing the values of the parameters at which the maximum is obtained.
`dim`	an integer value giving the dimension of the parameters in the underlying variance family.
`smooth`	a logical value indicating whether a smoothing operation was performed.
`is.smoothed`	a vector of logical values of the same length as `eql` indicating if the particular EQL value was obtained by smoothing or was calculated directly.

`smooth.grid`	an integer value giving the number of points used in the smoothing process or `NULL` if no smoothing was performed.
`model`	if `include.model` is `TRUE`, the GLM for which the maximum EQL value was archieved, `NULL` otherwise.

Note

The EQL for variance functions with $V_\theta(0) = 0$ becomes infinite. Hence, if there are exact zeros in the data, one should provide a variance family, which do not equate to zero at the origin. *Nelder and Pregibon* propose some adjustment of $V(y)$ at the origin, which leads to a modified variance function.

The predefined families `powerVarianceFamily` and `extBinomialVarianceFamily` are, however, *not* capable of dealing with exact zeros, for there is no general mechanism to modify the variance function for all possible values of the particular variance family.

The confidence interval for one-dimensional variance families is not calculated exactly, but depends on the amount of EQL values available. Hence, if one is interested in a confidence interval, one should allow for smoothing.

The function `eql` does not use a direct maximization routine, but rather do a simple maximation over a finite set. Hence, all obtained values including confidence intervals and confidence ellipsoids have a "local flavour" and should not be regarded as global solutions.

The confidence bounds are determined rather empirically and do heavily depend on the amount of parameter values under consideration.

Author(s)

Thorn Thaler

References

Nelder, J.A. and Pregibon, D. (1987). An extended quasi-likelihood function. *Biometrika*, **74**, 221–232.

See Also

`varianceFamily`, `glm`

Examples

```
## Power Variance Family
# Data from Box and Cox (1964)
x <- (-1:1)
y <- c(674,370,292,338,266,210,170,118,90,1414,1198,634,1022,620,438,
       442,332,220,3636,3184,2000,1568,1070,566,1140,884,360)
yarn.raw <- data.frame(expand.grid(x3=x, x2=x, x1=x), cycles=y)
yarn <- data.frame(x1=yarn.raw$x1, x2=yarn.raw$x2, x3=yarn.raw$x3,
   cycles=yarn.raw$cycles)
attach(yarn)

ps.power <- list(theta=seq(1, 4, length = 20))
```

```
eq.power <- eql(cycles~x1+x2+x3, param.space=ps.power,
   family=powerVarianceFamily("log"), smooth.grid=500)
plot(eq.power)

## Not run:
## Extended Binomial Variance Family
# Data from McCullagh & Nelder: GLM, p. 329
# (zeros replaced by 'NA')

site <- rep(1:9, each=10)
variety <- rep(1:10, 9)
resp <- c(0.05,NA,NA,0.10,0.25,0.05,0.50,1.30,1.50,1.50,
   NA,0.05,0.05,0.30,0.75,0.30,3,7.50,1,12.70,1.25,1.25,
   2.50,16.60,2.50,2.50,NA,20,37.50,26.25,2.50,0.50,0.01,
   3,2.50,0.01,25,55,5,40,5.50,1,6,1.10,2.50,8,16.50,
   29.50,20,43.50,1,5,5,5,5,5,10,5,50,75,5,0.10,5,5,
   50,10,50,25,50,75,5,10,5,5,25,75,50,75,75,75,17.50,
   25,42.50,50,37.50,95,62.50,95,95,95) / 100

ps.binomial <- list(seq(1, 2.2, length=32), seq(1, 3, length=32))
eq.binomial <- eql(resp~site*variety, param.space=ps.binomial,
   family=extBinomialVarianceFamily())
plot(eq.binomial)

## End(Not run)
```

hermite *Hermite Polynomials*

Description

Computes the Hermite polynomial $H_n(x)$.

Usage

```
hermite(x, n, prob = TRUE)
```

Arguments

x	a numeric vector or array giving the values at which the Hermite polynomial should be evaluated.
n	an integer vector or array giving the degrees of the Hermite polynomials. If `length(x) != 1`, n must be either of the same length as x or a single value.
prob	logical. If TRUE (the default) the probabilistic version of the Hermite polynomial is evaluated, otherwise the physicists' Hermite polynomials are used. See the 'Details' section below for further information.

Details

The Hermite polynomials are given by:

- $H_{n+1}(x) = xH_n(x) - nH_{n-1}(x)$, with $H_0(x) = 1$ and $H_1(x) = x$, (Probabilists' version $H_n^{Pr}(x)$)
- $H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x)$, with $H_0(x) = 1$ and $H_1(x) = 2x$. (Physicists' version $H_n^{Ph}(x)$)

and the relationship between the two versions is given by

$$H_n^{Ph}(x) = 2^{n/2} H_n^{Pr}(\sqrt{2}x).$$

The term 'probabilistic' is motivated by the fact that in this case the Hermite polynomial $H_n(x)$ can be as well defined by

$$H_n(x) = (-1)^n \frac{1}{\varphi(x)} \varphi^{(n)}(x),$$

where $\varphi(x)$ denotes the density function of the standard normal distribution and $\varphi^{(k)}(x)$ denotes the kth derivative of $\varphi(x)$ with respect to x.

If the argument `n` is a vector it must be of the same length as the argument `x` or the length of the argument `x` must be equal to one. The Hermite polynomials are then evaluated either at x_i with degree n_i or at x with degree n_i, respectively.

Value

the Hermite polynomial (either the probabilists' or the physicists' version) evaluated at `x`.

Author(s)

Thorn Thaler

References

Fedoryuk, M.V. (2001). Hermite polynomials. *Encyclopaedia of Mathematics*, Kluwer Academic Publishers.

Examples

```
2^(3/2)*hermite(sqrt(2)*5, 3)    # = 940
hermite(5, 3, FALSE)             # = 940
hermite(2:4, 1:3)                # H_1(2), H_2(3), H_3(4)
hermite(2:4, 2)                  # H_2(2), H_2(3), H_2(4)
hermite(2, 1:3)                  # H_1(2), H_2(2), H_3(2)
## Not run:
hermite(1:3, 1:4)                # Error!

## End(Not run)
```

`saddlepoint` *Saddlepoint Approximation*

Description

Computes the (normalized) saddlepoint approximation of the mean of n i.i.d. random variables.

Usage

```
saddlepoint(x, n, cumulants, correct = TRUE, normalize = FALSE)
```

Arguments

`x`	a numeric vector or array with the values at which the approximation should be evaluated.
`n`	a positive integer giving the number of i.i.d. random variables in the sum.
`cumulants`	a `cumulants` object giving the cumulant functions and the saddlepoint function. See `cumulants` for further information.
`correct`	logical. If `TRUE` (the default) the correction term involving the 3rd and the 4th standardized cumulant functions is included.
`normalize`	logical. If `TRUE` the renormalized version of the saddlepoint approximation is calculated. The renormalized version does neither make use of the 3rd nor of the 4th cumulant function so setting `correct=TRUE` will result in a warning. The default is `FALSE`.

Details

The saddlepoint approximation (SA) for the density of the mean $Z = S_n/n$ of i.i.d. random variables Y_i with $S_n = \sum_{i=1}^n Y_i$ is given by:

$$f_Z(z) \approx c\sqrt{\frac{n}{2\pi K''_Y(s)}}\exp\{n[K_Y(s) - sz]\},$$

where c is an appropriatly chosen correction term, which is based on higher cumulants. The function $K_Y(\cdot)$ denotes the cumulant generating function and s denotes the *saddlepoint* which is the solution of the saddlepoint function:

$$K'(s) = z.$$

For the renormalized version of the SA one chooses c such that $f_Z(z)$ integrates to one, otherwise it includes the 3rd and the 4th standardized cumulant.

The saddlepoint approximation is an improved version of the Edgeworth approximation and makes use of ‘exponential tilted’ densities. The weakness of the Edgeworth method lies in the approximation in the tails of the density. Thus, the saddlepoint approximation embed the original density in the “conjugate exponential family” with parameter θ. The mean of the embeded density depends on θ which allows for evaluating the Edgeworth approximation at the mean, where it is known to give reasonable results.

Value

`saddlepoint` returns an object of class `approximation`. See function `approximation` for further details.

Author(s)

Thorn Thaler

References

Reid, N. (1991). Approximations and Asymptotics. *Statistical Theory and Modelling*, London: Chapman and Hall.

See Also

`approximation`, `cumulants`, `edgeworth`

Examples

```
# Saddlepoint approximation for the density of the mean of n Gamma
# variables with shape=1 and scale=1
n <- 10
shape <- scale <- 1
x <- seq(0, 3, length=1000)
sp <- saddlepoint(x, n, gammaCumulants(shape, scale))
plot(sp, lwd=2)

# Mean of n Gamma(1,1) variables is n*Gamma(n,1) distributed
lines(x, n*dgamma(n*x, shape=n*shape, scale=scale), col=2)
```

`varianceFamily` *Variance Family Class For The EQL-Method*

Description

`varianceFamily` provides a class for a parameterized family of variance functions to be used with `eql`.

The predefined functions `powerVarianceFamily` and `extBinomialVarianceFamily` compute the variance family defined by the parametric variance functions $V_\theta(\mu) = \mu^\theta$ and $V_{k,l}(\mu) = \mu^k(1-\mu)^l$, respectively.

Usage

```
varianceFamily(varf, devf = NULL, link = "log", initf = NULL,
                validmuf = NULL, name = "default")

powerVarianceFamily(link = "log")
extBinomialVarianceFamily(link = "logit")
```

Arguments

`varf`	the parameterized variance function.
`devf`	the deviance function. If `NULL` (the default) it will be determined numerically from the variance function `varf`.
`link`	the link function.
`initf`	a function returning an object of class `expression`. The `expression` object should give a sequence of initializing commands for the `glm` routine such as setting the starting values. If `NULL` (the default), a very rudimentary initialize function is chosen, which may not be appropriate. See `family` for further details.
`validmuf`	a function giving `TRUE` if its argument is a valid value for μ and `FALSE` otherwise. If `NULL` (the default), all μ are supposed to be valid.
`name`	a character string giving the name of the variance family.

Details

The purpose of the function `varianceFamily` is to provide a convenient way to specify families of variance functions. An extended glm `family` object for a particular choice of a parameter vector can be obtained via the class member `family`.

The minimal specification for a `varianceFamily` object is the variance function $V_\theta(\mu)$ with θ describing the vector of family parameters. If not given explicitly, the deviance function is determined numerically.

The family parameter of `powerVarianceFamily` is ‘theta’, while the names of the parameters of `extBinomialVarianceFamily` are ‘k’ and ‘l’.

Value

`varianceFamily` returns an object of class `varianceFamily` containing the following components:

`family`	a function which computes an `extFamily` object, which is an extension of the `family` object known from classical `glm`. `extFamily` inherits from class `family` and contains an additional field holding the value of the particular parameters at which the family was evaluated.
`name`	a character string giving the name of the variance family.
`params`	a list of the parameters of the variance family.
`type.dev`	a character string. Equals either “explicit” or “numerical” depending on how the deviance function was determined.

Note

Those arguments passed to `varianceFunction` that are functions, are supposed to accept the variance family’s parameter as an argument. The idea is that any of these functions may give different results for different values of the family’s parameters. Even if any of these functions do not depend on these parameters, they must be contained in the function’s argument list.

Author(s)

Thorn Thaler

References

Nelder, J.A. and Pregibon, D. (1987). An Extended Quasi-Likelihood Function. *Biometrika*, **74**, 221–232.

See Also

family, eql

Examples

```
# The extended binomial variance family
# (the deviance is determined numerically)

# init does not depend on k and l but it must accept
# these parameters anyways
init <- function(k, l) {
  return(expression({
    mustart <- (weights * y + 0.5)/(weights + 1)
    n <- rep.int(1, nobs)}))
}
validmuf <- function(mu, k, l) {
  return(all(mu > 0) && all(mu < 1))
}
varf <- function(y, k, l)  y^k*(1-y)^l
suppressWarnings(vf <- varianceFamily(varf=varf, link="log", initf=init,
                        validmuf=validmuf,
                        name="Extended-Binomial-Family"))
vf$family(1,1) # corresponds to binomial()

y <- runif(10, 0, 1)
mu <- runif(10, 0, 1)

all.equal(vf$family(1,1)$dev.resids(y,mu,1),     # TRUE
          binomial()$dev.resids(y,mu,1))
```

Index

∗ **package**
    EQL-package, 2

approximation, *2*, 3, *10*, *17*

check.cumulants *(*cumulants*)*, 4
cumulants, *2*, 4, *16*, *17*

edgeworth, *2*, *4*, *7*, 8, *17*
EQL *(*EQL-package*)*, 2
eql, *2*, 10, *17*, *19*
EQL-package, 2
extBinomialVarianceFamily, *2*, *13*
extBinomialVarianceFamily
        *(*varianceFamily*)*, 17

family, *18*, *19*

gammaCumulants, *2*
gammaCumulants *(*cumulants*)*, 4
gaussianCumulants, *2*
gaussianCumulants *(*cumulants*)*, 4
glm, *11*, *13*, *18*

hermite, *2*, *9*, *10*, 14

interval, *5*
inverseGaussianCumulants, *2*
inverseGaussianCumulants *(*cumulants*)*, 4

plot.approximation *(*approximation*)*, 3
plot.eql *(*eql*)*, 10
powerVarianceFamily, *2*, *13*
powerVarianceFamily *(*varianceFamily*)*, 17

saddlepoint, *2*, *4*, *5*, *7*, *10*, 16

ttutils, *3*

varianceFamily, *2*, *11*, *13*, 17