Package: QForm (via r-universe)

November 23, 2024

Type Package

Title Fast, safe CDF/PDF estimation and bounding for generalized chi-square random varaibles

Version 0.2.2

Maintainer Ryan R. Christ <rchrist@wustl.edu>

Description QForm provides estimates and upper/lower bounds on p-values for test statistics of the form \deqn{\sum\limits_i f\left(\eta_i \right) A_i + \sigma Z_0}{\sum\limits_i f (\eta_i) A_i + \sigma Z_0} where each \eqn{A_i \sim \chi^2_{a_i}\left(\delta^2_i\right)}{A_i \sim \chi^2_{a_i}\left(\delta^2_i\right)} and \eqn{Z_0 \sim N(0,1)}{Z_0 \sim N(0,1)}, all terms mututally independent.

License GPL-3

Encoding UTF-8

LazyData true

Imports stats, RcppRoll

Suggests future, future.apply

RoxygenNote 7.3.2

Repository https://ryanchrist.r-universe.dev

RemoteUrl https://github.com/ryanchrist/QForm

RemoteRef HEAD

RemoteSha 4f878f579fb4ad98ace1a6a857a802ca27dfceb9

Contents

`plot.QFGaussCDF` *Plot method for a QFGaussCDF object*

Description

Plots the CDF computed by QFGauss.

Usage

```
## S3 method for class 'QFGaussCDF'
plot(x, ...)
```

Arguments

`x`	a QFGaussCDF
`...`	additional parameters

Value

There is nothing returned.

`QFGauss` *Fast CDF/PDF of a Quadratic Form in Gaussians*

Description

Returns the CDF and PDF for the generalized chi-squared distribution. That is, random variables T_f of the form

$$T_f = \sum_i f(\eta_i) A_i + \sigma Z_0$$

where each $A_i \sim \chi^2_{a_i}\left(\delta_i^2\right)$ and $Z_0 \sim N(0,1)$, all are mututally independent. By using the fast Fourier transform and various adjustments for numerical precision, this function is faster and more reliable than Davie's method and related approaches, especially when the returned CDF or PDF is to be evaluated at many points.

Usage

```
QFGauss(
  f.eta,
  delta2 = rep(0, length(f.eta)),
  df = rep(1, length(f.eta)),
  sigma = 0,
  n = 2^16 - 1,
  parallel.sapply = base::sapply
)
```

Arguments

`f.eta`	vector; real-valued coefficients, $f(\eta_i)$, (may be positive or negative)
`delta2`	vector; non-negative real-valued non-centrality parameters for each term (default is 0s). As is standard for chi-squared non-centrality parameters, these are assumed to be already summed across terms when df > 1.
`df`	vector; positive real-valued degrees of freedom for each term (represented in equation above as a_i, default is vector of 1s). If many "redundant" terms with the same $f(\eta_i)$ and δ_i^2 can be collapsed into a single term by setting the corresponding entries in df > 1, very significant speed increases can be achieved.
`sigma`	numeric; standard deviation of optional Gaussian term Z_0, default is 0 (no Gaussian term added)
`n`	integer; number of points at which to evaluate the characteristic function of T_f, must be odd (see Details).
`parallel.sapply`	function; a user-provided version of `sapply`, see Details.

Details

The returned function has three optional, logical arguments. The first is a `density`, which when `TRUE`, prompts the function to evaluate the PDF rather than the CDF. `density` defaults to FALSE. `lower.tail` returns 1 minus the CDF when `TRUE` (not used if `density`==TRUE) and is highly recommended for those interested in the upper tail of T_f. `log.p` returns the desired probabilities in log space.

`parallel.sapply`, by default, is set to `base:sapply`. However, it allows the user to supply a parallelized version of `sapply` (eg: `future_sapply` from the `future.apply` package) to help speed up the calculation of the CDF. This is helpful in cases where `length(f.eta)` is large.

`n` is the number of sub-intervals used in the left-sided Reimann integral approximation of the Fourier transform carried out by `stats::fft`. The default 2^16-1 should work for the vast majority of cases, but n may need to be increased to achieve accurate CDF estimation when T_f has many terms (when `f.eta` is long).

Since `stats::fft` can only evaluate the CDF up to double precision, we extrapolate the tails of T_f. QForm automatically detects the region where the estimated CDF begins to lose precision. A log-linear function is used for tails that go out to infinity and a function of the form $\alpha|x|^\beta$ is used for tails truncated at 0 (when all of the `f.eta` have the same sign). These extrapolated tails, motivated by the form of the characteristic function, provide accurate approximations in most cases when compared against a quad-precision implementation (not yet included in `QForm`).

Our current tail extrapolation scheme can become unstable or fail in cases where the target distribution is extremely skewed. In these cases, one of the tails decays too rapidly to be estimated with the given number of FFT grid points (set by QFGauss optional argument `n`). `QFGaussBounds` cannot currently calculate bounds for a cdf returned by `QFGauss` that has a missing tail. While we plan to address extremely skewed cases in future versions by deploying a second FFT when needed, for now, we recommend that users who really care about estimation of the thin tail or obtaining bounds with `QFGaussBounds` to try increasing the number of FFT grid points, `n`, passed to `QFGauss`.

A note on unbounded densities: The density of T_f is guaranteed to be bounded if length(f.eta) > 2 and there is no trouble in density estimation posed by asymptotes. In the length(f.eta)==2 case, if the two components of f.eta are of opposite signs, then the density of T_f may have an asymptote at some value t. While the density in the neighborhood around that t should be accurately calculated, due to the FFT and spline interpolation approach used, the density reported at t may be reported as some finite rather than as Inf. In the length(f.eta)==1 case, QFGauss resorts to dchisq and the density at 0 is accurately reported as Inf.

Value

A function that evaluates the CDF or PDF of T_f.

See Also

QFGaussBounds, TestQFGauss

Examples

```
f.eta <- c(-12, -7, 5, 7, -9, 10, 8)
delta2 <- c(2, 10, -4, 3, 8, -5, -12)^2
df <- c(1.1,5.2,0.4,10,1,2.5,1)

cdf <- QFGauss(f.eta, delta2, df)

# Inspect computed CDF
plot(cdf)

# Plot computed CDF at desired points
x <- seq(-1500, 2000, len = 1e3)
plot(x, cdf(x), type = "l", ylab = expression(CDF), xlab = expression(T[f]),
     main = expression(CDF~of~T[f])) # CDF
plot(x, cdf(x,density = TRUE), type = "l", ylab = expression(PDF),
     xlab = expression(T[f]), main = expression(PDF~of~T[f])) # PDF

# Compare computed CDF to empirical CDF of target distribution based on 10,000 samples
TestQFGauss(cdf)

# QFGauss can be accelerated by passing it a parallel version of sapply
## Not run:
# In this example we use only 2 parallel workers but more may be added
require(future.apply); plan(tweak(multiprocess,workers = 2))
f.eta <- 5 * rnorm(500)
system.time(cdf <- QFGauss(f.eta))
system.time(cdf <- QFGauss(f.eta, parallel.sapply = future_sapply))

## End(Not run)
```

QFGaussBounds — *Bounds on the CDF of a Quadratic Form in Gaussians*

Description

Returns a function for calculating upper and lower bounds on the CDF for random variables $Q_f = T_f + R_f$ where only the CDF of T_f is known. These random variables have the form

Usage

```
QFGaussBounds(
  cdf,
  f = "identity",
  max.abs.eta,
  sum.eta,
  sum.etasq,
  sum.eta.deltasq = 0,
  sum.etasq.deltasq = 0,
  include.saddlepoint = FALSE
)
```

Arguments

`cdf`	function; the cdf of T_f returned by `QForm::QForm`
`f`	character or QFormFunction object; the function f for the Q_f of interest.
`max.abs.eta`	vector; element-wise upper bound on the absolute value of the η_i in R_f (see Details)
`sum.eta`	vector; element-wise sum of the η_i in R_f (see Details)
`sum.etasq`	vector; element-wise sum of the η_i^2 in R_f (see Details)
`sum.eta.deltasq`	vector; element-wise sum of the $\eta_i \delta_i^2$ in R_f (see Details)
`sum.etasq.deltasq`	vector; element-wise sum of the $\eta_i^2 \delta_i^2$ in R_f (see Details)
`include.saddlepoint`	logical; if TRUE also return saddlepoint approximation based estimate of Q_f alongside bounds. Currently only available when f = "identity." Default is FALSE.

Details

$$T_f = \sum_{i \in \mathcal{T}} f(\eta_i) A_i + \sigma Z_0$$

,

$$R_f = \sum_{i \in \mathcal{R}} f(\eta_i) A_i$$

, where each $A_i \sim \chi^2_{a_i}\left(\delta_i^2\right)$ and $Z_0 \sim N(0,1)$, all mututally independent, and $a_i = 1$ for all $i \in \mathcal{R}$. We aim to remove this final restriction in future work.

If `max.abs.eta < .Machine$double.eps`, then the contribution of R_f to Q_f is ignored for numerical stability and the function returned is simply wrapper for the provided CDF of T_f. If this is not desired, a user may want to consider rescaling Q_f to avoid this behavior. Currently only $f = "identity"$ is supported, but future versions will allow one to select f from a list or specify their own function with its corresponding bounds through a QFormFunction object.

The returned bounds function takes a vector of observed values `q` at which to calculate bounds as it's main argument. If `q` is not known exactly, but only a lower bound `ql` and an upper bound `qu` are known, then those may provided instead of `q` and the returned bounds on the CDF will be valid for a `q` in `[ql,qu]`. If `q` is provided, `ql` and `qu` are ignored. The returned bounds function itself returns a matrix with four columns: `c("lower.bound","upper.bound","one.minus.lower.bound","one.minus.upper.bound")` The first and second columns are lower and upper bounds on the CDF at `q` respectively; the third and fourth columns are equal to one minus the first two columns but calculated separately by the function internally in order to maintain numerical precision in the upper tail. Thus, it is strongly recommneded that users interested in upper tail p-values use the third and fourth columns rather than the first and second.

The returned bounds function can also take a parallel version of sapply from a given R parallelization package via the optional argument `parallel.sapply`. This can substantially speed up computation for long `q`. See Example below and `QFGauss` for more details.

`QFGaussBounds` cannot currently calculate bounds for a cdf returned by `QFGauss` that has a missing tail. See `QFGauss` for more details.

Value

A vectorized function which evaluates upper and lower bounds on the CDF of Q_f.

See Also

`QFGauss`, `TestQFGaussBounds`

Examples

```
f.eta <- c(-12, -7, 5, 7, -9, 10, 8)
delta2 <- c(2, 10, -4, 3, 8, -5, -12)^2

cdf <- QFGauss(f.eta, delta2)

bounds <- QFGaussBounds(cdf = cdf, f = "identity",
                        max.abs.eta = 10, sum.eta = 5, sum.etasq = 200)

## Not run:
# Evaluate the bounds at a set of points
xx <- seq(-1e3, 1e3, len = 6)
## This may take 5 - 10 secs.
system.time(y <- bounds(xx))

x <- seq(-1e3, 1e3, len = 1e3)
plot(x, cdf(x), type = "l", ylab = expression(CDF), xlab = expression(T[f]),
     main = expression(Bounds~on~CDF~of~T[f])) # CDF
points(xx, y[,1], col = "blue")
points(xx, y[,2], col = "red")

# Generate diagnostic plots for bounds (currently TestQFGaussBounds only
# works for cases where the QFGauss produeced CDF has all df = 1.)
TestQFGaussBounds(QFGauss(c(1,5,-4,-3,10),c(2,-1,4,-5,5)^2),2)

# The function returned by QFGaussBounds can be accelerated by passing it a
# parallel version of sapply.
# In this example we use only 2 parallel workers but more may be added
require(future.apply); plan(tweak(multiprocess,workers = 2))
system.time(y <- bounds(xx, parallel.sapply = future_sapply))

## End(Not run)
```

`QForm` *QForm: A package for fast, safe CDF/PDF estimation for generalized chi-square random varaibles and screening with p-value bounds for quadratic forms.*

Description

QForm returns the CDF and PDF for the generalized chi-squared distribution with numerical accuracy deep into the tail (see `QFGauss`). It can also provide reliable upper and lower bounds on the CDF when only some of the chi-square terms are known (see `QFGaussBounds`). By using the fast Fourier transform in combination with novel concentration inequalities and various adjustments for numerical precision, QForm function is faster and more reliable than Davie's method and related approaches, especially when the returned CDF or PDF is to be evaluated at many points.

Details

Initially motivated by genome-wide association studies (GWAS), QForm is aimed at obtaining upper and lower bounds on p-values for test statistics with a limiting distribution of the form of $Q_f = T_f + R_f$ where only the CDF of T_f is known. These random variables have the form

$$T_f = \sum_{i \in \mathcal{T}} f\left(\eta_i\right) A_i + \sigma Z_0$$

,

$$R_f = \sum_{i \in \mathcal{R}} f\left(\eta_i\right) A_i$$

, where each $A_i \sim \chi^2_{a_i}\left(\delta_i^2\right)$ and $Z_0 \sim N(0,1)$, all mututally independent, and $a_i = 1$ for all $i \in \mathcal{R}$. We aim to remove this final restriction in future work.

In the genomics literature, SKAT and related methods have limiting distributions of this form. In the machine learning and kernel methods literature, other popular test statistics share this limiting distribution, among them the Hilbert-Schimidt Information Criterion (HSIC). Approximate methods have emerged in the genomics (eg: FastSKAT) and kernel methods literature, have emerged based on the idea of using a top-k SVD to obtain T_f and then attempt to approximate the contribution from R_f using a single random variable that matches some of the moments of R_f. However, we've found that in several applications, such approximations of R_f can lead to p-value estimates that are off by orders of magnitude. We take a concentration-inequality based approach to bounding the potential contribution of R_f to the overall distribution of Q_f, allowing us to obtain exact upper and lower bounds on the p-value that can allow users to quickly discard observations (eg: genomic loci) that could never be significant while concentrating further computational resources on more precisely evaluating the p-value at loci that could still potentially be interesting/significant.

Our implementation features two main new functions. First, we do not rely on `CompQuadForm`, which implements Davie's method but as such has difficult-to-tune parameters and can often fail for pvalues smaller than 1e-16. Davie's method is based on a more general integral transform that relates the CDF of a random variable to its characteristic function, but predates the fast fourier transform. We make use of the same identity as Davies, but by combining it with the FFT, obtain the CDF of random variables of the form of Q_f at many points in parallel (implemented in `QFGauss`).

Given the CDF produced by `QFGauss`, we apply a set of analytic and numerical intergration routines to T_f to calculate our p-value bounds for Q_f (implemented in `QFGaussBounds`).

QForm functions

`QFGauss` `TestQFGauss` `QFGaussBounds` `TestQFGaussBounds`

`TestQFGauss` *Test function for a QFGaussCDF object*

Description

Compares the CDF inferred by QFGauss to an empirical CDF.

Usage

```
TestQFGauss(cdf, n.samps = 10000)
```

Arguments

`cdf`	a QFGaussCDF
`n.samps`	number of draws from the target distribution with which to construct the empirical CDF

Details

Four plots are produced. The top-left plot overlays the CDF computed by QFGauss (in black) and the empirical CDF (in red) based on 10,000 samples. The top-right plot shows the distance between the empirical and QFGauss-computed CDF and the corresponding ks.test p-value (two-sided alternative). The ks.test p-value will be `NA` if `cdf` is missing one of its tails (see `QFGauss` for details). The two bottom plots allow comparison of the empirical CDF (in red) with the computed CDF (in black) in each tail.

Value

Nothing is returned.

See Also

QFGauss, TestQFGaussBounds

Examples

```
TestQFGauss(QFGauss(c(1,3,4,-3)))
```

TestQFGaussBounds *Test function for a QFGaussBounds*

Description

Compares the CDF bounds inferred by QFGaussBounds to a truncated approximation of the CDF and a naive quadrature-based implementation of the bounds.

Usage

```
TestQFGaussBounds(
  fullcdf,
  k = min(20, floor(length(attr(fullcdf, "f.eta"))/2)),
  n.bound.points = 16,
  lower.tail.end = 20,
  upper.tail.end = 20,
  parallel.sapply = base::sapply
)
```

Arguments

`fullcdf`	QFGaussCDF; the target CDF including all terms; currently TestQFGaussBounds only works for cases where the QFGauss produeced CDF has all df = 1.
`k`	numeric; the number of truncated terms provided to QFGaussBounds from which to bound fullcdf
`n.bound.points`	numeric; the number of points at which to evaluate the bounds for plotting
`lower.tail.end`	numeric; the -log_10 lower tail probability at which to start each x-axis (default = 20)
`upper.tail.end`	numeric; the -log_10 upper tail probability at which to end each x-axis (default = 20)
`parallel.sapply`	function; a user-provided version of `sapply`, see Details.

Details

Here, `fullcdf` is taken to be the CDF of the target random variable Q_f (see documentation of `QFGauss` for definitions). Four plots are produced. The top-left plot overlays the target CDF of Q_f, `fullcdf`, computed by `QFGauss` (in black) and a truncated approximation to that CDF (in orange) based on simply adding the expectation of the remainder term R_f to the top-k truncated version of `fullcdf`, T_f. By "top-k" here we mean taking the terms of Q_f with the largest magnitude coefficients, `f.eta`, and using that to define T_f, which is what is done in `TestQFGaussBounds` internally. The green line is a similar approximation but where R_f is approximated with a moment-matching gaussian. The upper and lower bounds on `fullcdf` computed by `QFGaussBounds` are plotted as red and blue circles respectively. The upper and lower bounds on `fullcdf` computed by a naive quadrature-based implementation of the bounds are plotted as red and blue Xs respectively. The top-right plot shows the difference between the truncated approximation of the CDF and `fullcdf` in log space. It may be interpreted as follows. The x-axis plots the -log_10 p-value one would have reported based on the truncated approximation alone. The y-axis is the difference between the true -log_10 p-value and the approximate -log_10 p-value. The difference in p-values in the upper tail is plotted with a solid line. The difference in p-values in the lower tail is plotted with a dashed line. This plot effectively shows how far one might be misled by the truncated approximation shifted by the expectation of the remainder terms. The two bottom plots allow comparison of the empirical CDF (in red) with the computed CDF (in black) in each tail.

Value

There is nothing returned.

See Also

`QFGaussBounds`, `TestQFGauss`

Examples

```
TestQFGaussBounds(QFGauss(c(1,5,-4,-3),c(2,-1,4,-5)^2),2)
```

Index