Package 'aghq'

January 18, 2021

Type Package

Title Adaptive Gauss Hermite Quadrature for Bayesian Inference

Version 0.1.0

Author Alex Stringer

Maintainer Alex Stringer <alex.stringer@mail.utoronto.ca>

Description Adaptive Gauss Hermite Quadrature for Bayesian inference.
The AGHQ method for normalizing posterior distributions
and making Bayesian inferences based on them. Functions are provided for doing
quadrature and marginal Laplace approximations, and summary methods are provided
for making inferences based on the results.
See Stringer (2021). ``Implementing Adaptive Quadrature for Bayesian Inference:
the aghq Package" <arXiv:2101.04468>.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Depends R (>= 3.5.0)

Imports methods, mvQuad, matrixStats, Matrix, stringr, magrittr,
dplyr, tidyr, tidyselect, rlang, polynom, tibble, purrr

Suggests trustOptim, trust, numDeriv, testthat (>= 2.1.0), knitr,
rmarkdown, ggplot2

VignetteBuilder knitr

Language en-US

NeedsCompilation no

Repository CRAN

Date/Publication 2021-01-18 10:30:02 UTC

R topics documented:

---

aghq *Adaptive Gauss-Hermite Quadrature*

Description

Normalize the log-posterior distribution using Adaptive Gauss-Hermite Quadrature. This function takes in a function and its gradient and Hessian, and returns a list of information about the normalized posterior, with methods for summarizing and plotting.

Usage

```
aghq(ff, k, startingvalue, optresults = NULL, control = default_control(), ...)
```

Arguments

`ff` A list with three elements:

- `fn`: function taking argument `theta` and returning a numeric value representing the log-posterior at `theta`
- `gr`: function taking argument `theta` and returning a numeric vector representing the gradient of the log-posterior at `theta`
- `he`: function taking argument `theta` and returning a numeric matrix representing the hessian of the log-posterior at `theta`

The user may wish to use numDeriv::grad and/or numDeriv::hessian to obtain these. Alternatively, the user may consider the TMB package. This list is deliberately formatted to match the output of TMB::MakeADFun.

k — Integer, the number of quadrature points to use. I suggest at least 3. k = 1 corresponds to a Laplace approximation.

startingvalue — Value to start the optimization. ff$fn(startingvalue), ff$gr(startingvalue), and ff$he(startingvalue) must all return appropriate values without error.

optresults — Optional. A list of the results of the optimization of the log posterior, formatted according to the output of aghq::optimize_theta. The aghq::aghq function handles the optimization for you; passing this list overrides this, and is useful for when you know your optimization is too difficult to be handled by general-purpose software. See the software paper for several examples of this. If you're unsure whether this option is needed for your problem then it probably is not.

control — A list with elements

- method: optimization method to use:
  - 'sparse_trust' (default): trustOptim::trust.optim with method = 'sparse'
  - 'SR1' (default): trustOptim::trust.optim with method = 'SR1'
  - 'trust': trust::trust
  - 'BFGS': optim(...,method = "BFGS")

  Default is 'sparse_trust'.
- optcontrol: optional: a list of control parameters to pass to the internal optimizer you chose. The aghq package uses sensible defaults.

... — Additional arguments to be passed to fffn, ffgr, and ff$he.

Details

When k = 1 the AGHQ method is a Laplace approximation, and you should use the aghq::laplace_approximation function, since some of the methods for aghq objects won't work with only one quadrature point. Objects of class laplace have different methods suited to this case. See ?aghq::laplace_approximation.

Value

An object of class aghq which is a list containing elements:

- normalized_posterior: The output of the normalize_logpost function, which itself is a list with elements:
  - nodesandweights: a dataframe containing the nodes and weights for the adaptive quadrature rule, with the un-normalized and normalized log posterior evaluated at the nodes.
  - thegrid: a NIGrid object from the mvQuad package, see ?mvQuad::createNIGrid.
  - lognormconst: the actual result of the quadrature: the log of the normalizing constant of the posterior.
- marginals: a list of the same length as startingvalue of which element j is the result of calling aghq::marginal_posterior with that j. This is a tbl_df/tbl/data.frame containing the normalized log marginal posterior for theta_j evaluated at the original quadrature points. Has columns "thetaj","logpost_normalized","weights", where j is the j you specified.

- optresults: information and results from the optimization of the log posterior, the result of calling aghq::optimize_theta. This a list with elements:
  - ff: the function list that was provided
  - mode: the mode of the log posterior
  - hessian: the hessian of the log posterior at the mode
  - convergence: specific to the optimizer used, a message indicating whether it converged

See Also

Other quadrature: laplace_approximation(), marginal_laplace(), normalize_logpost(), optimize_theta(), plot.aghq(), print.aghqsummary(), print.aghq(), print.laplacesummary(), print.laplace(), summary.aghq(), summary.laplace()

Examples

```
logfteta2d <- function(eta,y) {
  # eta is now (eta1,eta2)
  # y is now (y1,y2)
  n <- length(y)
  n1 <- ceiling(n/2)
  n2 <- floor(n/2)
  y1 <- y[1:n1]
  y2 <- y[(n1+1):(n1+n2)]
  eta1 <- eta[1]
  eta2 <- eta[2]
  sum(y1) * eta1 - (length(y1) + 1) * exp(eta1) - sum(lgamma(y1+1)) + eta1 +
    sum(y2) * eta2 - (length(y2) + 1) * exp(eta2) - sum(lgamma(y2+1)) + eta2
}
set.seed(84343124)
n1 <- 5
n2 <- 5
n <- n1+n2
y1 <- rpois(n1,5)
y2 <- rpois(n2,5)
objfunc2d <- function(x) logfteta2d(x,c(y1,y2))
funlist2d <- list(
  fn = objfunc2d,
  gr = function(x) numDeriv::grad(objfunc2d,x),
  he = function(x) numDeriv::hessian(objfunc2d,x)
)

thequadrature <- aghq(funlist2d,3,c(0,0))
```

---

compute_moment *Compute moments*

Description

Compute the moment of any function ff using AGHQ.

Usage

```
compute_moment(normalized_posterior, ff = function(x) 1)
```

Arguments

normalized_posterior	The output of aghq::normalize_logpost. See the documentation for that function.
ff	Any R function which takes in a numeric vector and returns a numeric vector.

Value

A numeric vector containing the moment(s) of ff with respect to the joint distribution being approximated using AGHQ.

See Also

Other summaries: compute_pdf_and_cdf(), compute_quantiles(), interpolate_marginal_posterior(), marginal_posterior()

Examples

```
logfteta2d <- function(eta,y) {
  # eta is now (eta1,eta2)
  # y is now (y1,y2)
  n <- length(y)
  n1 <- ceiling(n/2)
  n2 <- floor(n/2)
  y1 <- y[1:n1]
  y2 <- y[(n1+1):(n1+n2)]
  eta1 <- eta[1]
  eta2 <- eta[2]
  sum(y1) * eta1 - (length(y1) + 1) * exp(eta1) - sum(lgamma(y1+1)) + eta1 +
    sum(y2) * eta2 - (length(y2) + 1) * exp(eta2) - sum(lgamma(y2+1)) + eta2
}
set.seed(84343124)
n1 <- 5
n2 <- 5
n <- n1+n2
y1 <- rpois(n1,5)
y2 <- rpois(n2,5)
objfunc2d <- function(x) logfteta2d(x,c(y1,y2))
funlist2d <- list(
  fn = objfunc2d,
  gr = function(x) numDeriv::grad(objfunc2d,x),
  he = function(x) numDeriv::hessian(objfunc2d,x)
)
```

```
opt_sparsetrust_2d <- optimize_theta(funlist2d,c(1.5,1.5))
norm_sparse_2d_7 <- normalize_logpost(opt_sparsetrust_2d,7,1)

# ff = function(x) 1 should return 1,
# the normalizing constant of the (already normalized) posterior:
compute_moment(norm_sparse_2d_7)
# Compute the mean of theta1 and theta2
compute_moment(norm_sparse_2d_7,ff = function(x) x)
# Compute the mean of lambda1 = exp(theta1) and lambda2 = exp(theta2)
lambdameans <- compute_moment(norm_sparse_2d_7,ff = function(x) exp(x))
lambdameans
# Compare them to the truth:
(sum(y1) + 1)/(length(y1) + 1)
(sum(y2) + 1)/(length(y2) + 1)
# Compute the standard deviation of lambda1
lambda1sd <- sqrt(compute_moment(norm_sparse_2d_7,ff = function(x) (exp(x) - lambdameans[1])^2))[1]
# ...and so on.
```

`compute_pdf_and_cdf` *Density and Cumulative Distribution Function*

Description

Compute the density and cumulative distribution function of the approximate posterior. The density is approximated on a find grid using a polynomial interpolant. The CDF can't be computed exactly (if it could, you wouldn't be using quadrature!), so a fine grid is laid down and the CDF is approximated at each grid point using a simpler integration rule and a polynomial interpolant. This method tends to work well, but won't always.

Usage

```
compute_pdf_and_cdf(margpost, transformation = NULL, finegrid = NULL)
```

Arguments

`margpost`	The output of `aghq::marginal_posterior`. See the documentation for that function.
`transformation`	Optional. A list containing two functions, `fromtheta` and `totheta`, which accept and return numeric vectors, defining a parameter transformation for which you would also like the pdf calculated for. See examples. May also have an element `jacobian`, a function which takes a numeric vector and computes the jacobian of the transformation; if not provided, this is done using `numDeriv::jacobian`.
`finegrid`	Optional, a grid of values on which to compute the CDF. The default makes use of the values in `margpost` but if the results are unsuitable, you may wish to modify this manually.

Value

A tbl_df/tbl/data.frame with columns `theta`, `pdf` and `cdf` corresponding to the value of the parameter and its estimated PDF and CDF at that value.

See Also

Other summaries: `compute_moment()`, `compute_quantiles()`, `interpolate_marginal_posterior()`, `marginal_posterior()`

Examples

```
logfteta2d <- function(eta,y) {
  # eta is now (eta1,eta2)
  # y is now (y1,y2)
  n <- length(y)
  n1 <- ceiling(n/2)
  n2 <- floor(n/2)
  y1 <- y[1:n1]
  y2 <- y[(n1+1):(n1+n2)]
  eta1 <- eta[1]
  eta2 <- eta[2]
  sum(y1) * eta1 - (length(y1) + 1) * exp(eta1) - sum(lgamma(y1+1)) + eta1 +
    sum(y2) * eta2 - (length(y2) + 1) * exp(eta2) - sum(lgamma(y2+1)) + eta2
}
set.seed(84343124)
n1 <- 5
n2 <- 5
n <- n1+n2
y1 <- rpois(n1,5)
y2 <- rpois(n2,5)
objfunc2d <- function(x) logfteta2d(x,c(y1,y2))
funlist2d <- list(
  fn = objfunc2d,
  gr = function(x) numDeriv::grad(objfunc2d,x),
  he = function(x) numDeriv::hessian(objfunc2d,x)
)
opt_sparsetrust_2d <- optimize_theta(funlist2d,c(1.5,1.5))
margpost <- marginal_posterior(opt_sparsetrust_2d,3,1) # margpost for theta1
thepdfandcdf <- compute_pdf_and_cdf(margpost)
with(thepdfandcdf,{
  plot(pdf~theta,type='l')
  plot(cdf~theta,type='l')
})
```

`compute_quantiles` *Quantiles*

Description

Compute marginal quantiles using AGHQ. This function works by first approximating the CDF using `aghq::compute_pdf_and_cdf` and then inverting the approximation numerically.

Usage

```
compute_quantiles(margpost, q = c(0.025, 0.975))
```

Arguments

`margpost`	The output of `aghq::marginal_posterior`. See the documentation for that function.
`q`	Numeric vector of values in (0,1). The quantiles to compute.

Value

A named numeric vector containing the quantiles you asked for, for the variable whose marginal posterior you provided.

See Also

Other summaries: `compute_moment()`, `compute_pdf_and_cdf()`, `interpolate_marginal_posterior()`, `marginal_posterior()`

Examples

```
logfteta2d <- function(eta,y) {
  # eta is now (eta1,eta2)
  # y is now (y1,y2)
  n <- length(y)
  n1 <- ceiling(n/2)
  n2 <- floor(n/2)
  y1 <- y[1:n1]
  y2 <- y[(n1+1):(n1+n2)]
  eta1 <- eta[1]
  eta2 <- eta[2]
  sum(y1) * eta1 - (length(y1) + 1) * exp(eta1) - sum(lgamma(y1+1)) + eta1 +
    sum(y2) * eta2 - (length(y2) + 1) * exp(eta2) - sum(lgamma(y2+1)) + eta2
}
set.seed(84343124)
n1 <- 5
n2 <- 5
n <- n1+n2
y1 <- rpois(n1,5)
y2 <- rpois(n2,5)
objfunc2d <- function(x) logfteta2d(x,c(y1,y2))
funlist2d <- list(
  fn = objfunc2d,
  gr = function(x) numDeriv::grad(objfunc2d,x),
  he = function(x) numDeriv::hessian(objfunc2d,x)
)
```

```
opt_sparsetrust_2d <- optimize_theta(funlist2d,c(1.5,1.5))
margpost <- marginal_posterior(opt_sparsetrust_2d,3,1) # margpost for theta1
etaquant <- compute_quantiles(margpost)
etaquant
# lambda = exp(eta)
exp(etaquant)
# Compare to truth
qgamma(.025,1+sum(y1),1+n1)
qgamma(.975,1+sum(y1),1+n1)
```

---

gcdata *Globular Clusters data for Milky Way mass estimation*

Description

Measurements on star clusters from Eadie and Harris (2016), for use within the Milky Way mass estimation example. Data are documented extensively by that source.

Usage

```
gcdata
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 70 rows and 25 columns.

Source

Eadie GM, Harris WE (2016). "Bayesian mass estimates of the Milky Way: the dark and light sides of parameter assumptions." The Astrophysical Journal, 829(108).

---

gcdatalist *Transformed Globular Clusters data*

Description

GC data prepared for input into the TMB template, for purposes of example. There are a lot of example-specific data preprocessing steps that are not related to the AGHQ method, so for brevity these are done beforehand.

Usage

```
gcdatalist
```

Format

An object of class `list` of length 6.

Source

Eadie GM, Harris WE (2016). "Bayesian mass estimates of the Milky Way: the dark and light sides of parameter assumptions." The Astrophysical Journal, 829(108).

`interpolate_marginal_posterior` *Interpolate the Marginal Posterior*

Description

Build a Lagrange polynomial interpolant of the marginal posterior, for plotting and for computing quantiles.

Usage

```
interpolate_marginal_posterior(margpost)
```

Arguments

`margpost`	The output of `aghq::marginal_posterior`. See the documentation for that function.

Value

A function of `theta` which computes the log interpolated normalized marginal posterior.

See Also

Other summaries: `compute_moment()`, `compute_pdf_and_cdf()`, `compute_quantiles()`, `marginal_posterior()`

`laplace_approximation` *Laplace Approximation*

Description

Wrapper function to implement a Laplace approximation to the posterior. A Laplace approximation is AGHQ with `k = 1` quadrature points. However, the returned object is of a different class `laplace`, and a different summary method is given for it. It is especially useful for high-dimensional problems where the curse of dimensionality renders the use of `k > 1` quadrature points infeasible. The summary method reflects the fact that the user may be using this for a high-dimensional problem, and no plot method is given, because there isn't anything interesting to plot.

Usage

```
laplace_approximation(
  ff,
  startingvalue,
  optresults = NULL,
  control = default_control(),
  ...
)
```

Arguments

ff	A list with three elements: • fn: function taking argument theta and returning a numeric value representing the log-posterior at theta • gr: function taking argument theta and returning a numeric vector representing the gradient of the log-posterior at theta • he: function taking argument theta and returning a numeric matrix representing the hessian of the log-posterior at theta The user may wish to use numDeriv::grad and/or numDeriv::hessian to obtain these. Alternatively, the user may consider the TMB package. This list is deliberately formatted to match the output of TMB::MakeADFun.
startingvalue	Value to start the optimization. ff$fn(startingvalue), ff$gr(startingvalue), and ff$he(startingvalue) must all return appropriate values without error.
optresults	Optional. A list of the results of the optimization of the log posterior, formatted according to the output of aghq::optimize_theta. The aghq::aghq function handles the optimization for you; passing this list overrides this, and is useful for when you know your optimization is too difficult to be handled by general-purpose software. See the software paper for several examples of this. If you're unsure whether this option is needed for your problem then it probably is not.
control	A list with elements • method: optimization method to use: – 'sparse_trust' (default): trustOptim::trust.optim with method = 'sparse' – 'SR1' (default): trustOptim::trust.optim with method = 'SR1' – 'trust': trust::trust – 'BFGS': optim(...,method = "BFGS") Default is 'sparse_trust'. • optcontrol: optional: a list of control parameters to pass to the internal optimizer you chose. The aghq package uses sensible defaults.
...	Additional arguments to be passed to fffn, ffgr, and ff$he.

Value

An object of class laplace with summary and plot methods. This is simply a list with elements lognormconst containing the log of the approximate normalizing constant, and optresults containing the optimization results formatted the same way as optimize_theta and aghq.

See Also

Other quadrature: aghq(), marginal_laplace(), normalize_logpost(), optimize_theta(), plot.aghq(), print.aghqsummary(), print.aghq(), print.laplacesummary(), print.laplace(), summary.aghq(), summary.laplace()

Examples

```
logfteta2d <- function(eta,y) {
  # eta is now (eta1,eta2)
  # y is now (y1,y2)
  n <- length(y)
  n1 <- ceiling(n/2)
  n2 <- floor(n/2)
  y1 <- y[1:n1]
  y2 <- y[(n1+1):(n1+n2)]
  eta1 <- eta[1]
  eta2 <- eta[2]
  sum(y1) * eta1 - (length(y1) + 1) * exp(eta1) - sum(lgamma(y1+1)) + eta1 +
    sum(y2) * eta2 - (length(y2) + 1) * exp(eta2) - sum(lgamma(y2+1)) + eta2
}
set.seed(84343124)
n1 <- 5
n2 <- 5
n <- n1+n2
y1 <- rpois(n1,5)
y2 <- rpois(n2,5)
objfunc2d <- function(x) logfteta2d(x,c(y1,y2))
funlist2d <- list(
  fn = objfunc2d,
  gr = function(x) numDeriv::grad(objfunc2d,x),
  he = function(x) numDeriv::hessian(objfunc2d,x)
)

thequadrature <- aghq(funlist2d,3,c(0,0))
```

---

marginal_laplace *Marginal Laplace approximation*

Description

Implement the marginal Laplace approximation of Tierney and Kadane (1986) for finding the marginal posterior (theta | Y) from an unnormalized joint posterior (W, theta, Y) where W is high dimensional and theta is low dimensional. See the AGHQ software paper for a detailed example, or Stringer et. al. (2020).

Usage

```
marginal_laplace(
  ff,
  k,
  startingvalue,
  optresults = NULL,
  control = default_control_marglaplace(),
  ...
)
```

Arguments

ff	A function list similar to that required by aghq. However, each function now takes arguments W and theta. Explicitly, this is a list containing elements: • fn: function taking arguments W and theta and returning a numeric value representing the log-joint posterior at W, theta • gr: function taking arguments W and theta and returning a numeric vector representing the gradient with respect to W of the log-joint posterior at W, theta • he: function taking arguments W and theta and returning a numeric matrix representing the hessian with respect to W of the log-joint posterior at W, theta
k	Integer, the number of quadrature points to use. I suggest at least 3. k = 1 corresponds to a Laplace approximation.
startingvalue	A list with elements W and theta, which are numeric vectors to start the optimizations for each variable. If you're using this method then the log-joint posterior should be concave and these optimizations should not be sensitive to starting values.
optresults	Optional. A list of the results of the optimization of the log posterior, formatted according to the output of aghq::optimize_theta. The aghq::aghq function handles the optimization for you; passing this list overrides this, and is useful for when you know your optimization is too difficult to be handled by general-purpose software. See the software paper for several examples of this. If you're unsure whether this option is needed for your problem then it probably is not.
control	A list with elements • method: optimization method to use for the theta optimization: – 'sparse_trust' (default): trustOptim::trust.optim – 'sparse': trust::trust – 'BFGS': optim(...,method = "BFGS") • inner_method: optimization method to use for the W optimization; same options as for method Default inner_method is 'sparse_trust' and default method is 'BFGS'.
...	Additional arguments to be passed to fffn, ffgr, and ff$he.

Value

If k > 1, an object of class marginallaplace, which includes the result of calling aghq::aghq on the Laplace approximation of (theta|Y), ... See software paper for full details. If k = 1, an object of class laplace which is the result of calling aghq::laplace_approximation on the Laplace approximation of (theta|Y).

See Also

Other quadrature: aghq(), laplace_approximation(), normalize_logpost(), optimize_theta(), plot.aghq(), print.aghqsummary(), print.aghq(), print.laplacesummary(), print.laplace(), summary.aghq(), summary.laplace()

---

marginal_posterior *Marginal Posteriors*

Description

Compute the marginal posterior for a given parameter using AGHQ.

Usage

```
marginal_posterior(optresults, k, j)
```

Arguments

optresults	The results of calling aghq::optimize_theta(): see return value of that function.
k	Integer, the number of quadrature points to use. I suggest at least 3. k = 1 corresponds to a Laplace approximation.
j	Integer between 1 and the dimension of the parameter space. Which index of the parameter vector to compute the marginal posterior for.

Value

a tbl_df/tbl/data.frame containing the normalized log marginal posterior for theta_j evaluated at the original quadrature points. Has columns "thetaj","logpost_normalized","weights", where j is the j you specified.

See Also

Other summaries: compute_moment(), compute_pdf_and_cdf(), compute_quantiles(), interpolate_marginal_poste

Examples

```
## A 2d example ##
logfteta2d <- function(eta,y) {
  # eta is now (eta1,eta2)
  # y is now (y1,y2)
  n <- length(y)
  n1 <- ceiling(n/2)
  n2 <- floor(n/2)
  y1 <- y[1:n1]
  y2 <- y[(n1+1):(n1+n2)]
  eta1 <- eta[1]
  eta2 <- eta[2]
  sum(y1) * eta1 - (length(y1) + 1) * exp(eta1) - sum(lgamma(y1+1)) + eta1 +
    sum(y2) * eta2 - (length(y2) + 1) * exp(eta2) - sum(lgamma(y2+1)) + eta2
}
set.seed(84343124)
n1 <- 5
n2 <- 5
n <- n1+n2
y1 <- rpois(n1,5)
y2 <- rpois(n2,5)
objfunc2d <- function(x) logfteta2d(x,c(y1,y2))
funlist2d <- list(
  fn = objfunc2d,
  gr = function(x) numDeriv::grad(objfunc2d,x),
  he = function(x) numDeriv::hessian(objfunc2d,x)
)
opt_sparsetrust_2d <- optimize_theta(funlist2d,c(1.5,1.5))

# Now actually do the marginal posteriors
marginal_posterior(opt_sparsetrust_2d,3,1)
marginal_posterior(opt_sparsetrust_2d,3,2)
marginal_posterior(opt_sparsetrust_2d,7,2)
```

`normalize_logpost` *Normalize the joint posterior using AGHQ*

Description

This function takes in the optimization results from `aghq::optimize_theta()` and returns a list with the quadrature points, weights, and normalization information. Like `aghq::optimize_theta()`, this is designed for use only within `aghq::aghq`, but is exported for debugging and documented in case you want to modify it somehow, or something.

Usage

```
normalize_logpost(optresults, k, whichfirst = 1, ...)
```

Arguments

optresults	The results of calling aghq::optimize_theta(): see return value of that function.
k	Integer, the number of quadrature points to use. I suggest at least 3. k = 1 corresponds to a Laplace approximation.
whichfirst	Integer between 1 and the dimension of the parameter space, default 1. The user shouldn't have to worry about this: it's used internally to re-order the parameter vector before doing the quadrature, which is useful when calculating marginal posteriors.
...	Additional arguments to be passed to optresults$ff, see ?optimize_theta.

Value

If k > 1, a list with elements:

- nodesandweights: a dataframe containing the nodes and weights for the adaptive quadrature rule, with the un-normalized and normalized log posterior evaluated at the nodes.
- thegrid: a NIGrid object from the mvQuad package, see ?mvQuad::createNIGrid.
- lognormconst: the actual result of the quadrature: the log of the normalizing constant of the posterior.

If k = 1, then the method returns a numeric value representing the log of the normalizing constant computed using a Laplace approximation.

See Also

Other quadrature: aghq(), laplace_approximation(), marginal_laplace(), optimize_theta(), plot.aghq(), print.aghqsummary(), print.aghq(), print.laplacesummary(), print.laplace(), summary.aghq(), summary.laplace()

Examples

```
# Same setup as optimize_theta
logfteta <- function(eta,y) {
  sum(y) * eta - (length(y) + 1) * exp(eta) - sum(lgamma(y+1)) + eta
}
set.seed(84343124)
y <- rpois(10,5) # Mode should be sum(y) / (10 + 1)
truemode <- log((sum(y) + 1)/(length(y) + 1))
objfunc <- function(x) logfteta(x,y)
funlist <- list(
  fn = objfunc,
  gr = function(x) numDeriv::grad(objfunc,x),
  he = function(x) numDeriv::hessian(objfunc,x)
)
opt_sparsetrust <- optimize_theta(funlist,1.5)
opt_trust <- optimize_theta(funlist,1.5,control = list(method = "trust"))
opt_bfgs <- optimize_theta(funlist,1.5,control = list(method = "BFGS"))
```

```
# Quadrature with 3, 5, and 7 points using sparse trust region optimization:
norm_sparse_3 <- normalize_logpost(opt_sparsetrust,3,1)
norm_sparse_5 <- normalize_logpost(opt_sparsetrust,5,1)
norm_sparse_7 <- normalize_logpost(opt_sparsetrust,7,1)

# Quadrature with 3, 5, and 7 points using dense trust region optimization:
norm_trust_3 <- normalize_logpost(opt_trust,3,1)
norm_trust_5 <- normalize_logpost(opt_trust,5,1)
norm_trust_7 <- normalize_logpost(opt_trust,7,1)

# Quadrature with 3, 5, and 7 points using BFGS optimization:
norm_bfgs_3 <- normalize_logpost(opt_bfgs,3,1)
norm_bfgs_5 <- normalize_logpost(opt_bfgs,5,1)
norm_bfgs_7 <- normalize_logpost(opt_bfgs,7,1)
```

`optimize_theta` *Obtain function information necessary for performing quadrature*

Description

This function computes the two pieces of information needed about the log posterior to do adaptive quadrature: the mode, and the hessian at the mode. It is designed for use within `aghq::aghq`, but is exported in case users need to debug the optimization process and documented in case users want to write their own optimizations.

Usage

```
optimize_theta(ff, startingvalue, control = default_control(), ...)
```

Arguments

`ff`	A list with three elements: • `fn`: function taking argument `theta` and returning a numeric value representing the log-posterior at `theta` • `gr`: function taking argument `theta` and returning a numeric vector representing the gradient of the log-posterior at `theta` • `he`: function taking argument `theta` and returning a numeric matrix representing the hessian of the log-posterior at `theta` The user may wish to use `numDeriv::grad` and/or `numDeriv::hessian` to obtain these. Alternatively, the user may consider the `TMB` package. This list is deliberately formatted to match the output of `TMB::MakeADFun`.
`startingvalue`	Value to start the optimization. `ff$fn(startingvalue)`, `ff$gr(startingvalue)`, and `ff$he(startingvalue)` must all return appropriate values without error.
`control`	A list with elements • `method`: optimization method to use:

  - 'sparse_trust' (default): `trustOptim::trust.optim` with `method = 'sparse'`
  - 'SR1' (default): `trustOptim::trust.optim` with `method = 'SR1'`
  - 'trust': `trust::trust`
  - 'BFGS': `optim(...,method = "BFGS")`

  Default is 'sparse_trust'.

- `optcontrol`: optional: a list of control parameters to pass to the internal optimizer you chose. The `aghq` package uses sensible defaults.

`...` Additional arguments to be passed to `ff$fn`, `ff$gr`, and `ff$he`.

Value

A list with elements

- `ff`: the function list that was provided
- `mode`: the mode of the log posterior
- `hessian`: the hessian of the log posterior at the mode
- `convergence`: specific to the optimizer used, a message indicating whether it converged

See Also

Other quadrature: `aghq()`, `laplace_approximation()`, `marginal_laplace()`, `normalize_logpost()`, `plot.aghq()`, `print.aghqsummary()`, `print.aghq()`, `print.laplacesummary()`, `print.laplace()`, `summary.aghq()`, `summary.laplace()`

Examples

```
# Poisson/Exponential example
logfteta <- function(eta,y) {
  sum(y) * eta - (length(y) + 1) * exp(eta) - sum(lgamma(y+1)) + eta
}

y <- rpois(10,5) # Mode should be (sum(y) + 1) / (length(y) + 1)

objfunc <- function(x) logfteta(x,y)
funlist <- list(
  fn = objfunc,
  gr = function(x) numDeriv::grad(objfunc,x),
  he = function(x) numDeriv::hessian(objfunc,x)
)

optimize_theta(funlist,1.5)
optimize_theta(funlist,1.5,control = list(method = "trust"))
optimize_theta(funlist,1.5,control = list(method = "BFGS"))
```

`plot.aghq` *Plot method for AGHQ objects*

Description

Plot the marginal pdf and cdf from an aghq object.

Usage

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

Arguments

`x`	The return value of `aghq::aghq`.
`...`	not used.

Value

Silently plots.

See Also

Other quadrature: `aghq()`, `laplace_approximation()`, `marginal_laplace()`, `normalize_logpost()`, `optimize_theta()`, `print.aghqsummary()`, `print.aghq()`, `print.laplacesummary()`, `print.laplace()`, `summary.aghq()`, `summary.laplace()`

Other quadrature: `aghq()`, `laplace_approximation()`, `marginal_laplace()`, `normalize_logpost()`, `optimize_theta()`, `print.aghqsummary()`, `print.aghq()`, `print.laplacesummary()`, `print.laplace()`, `summary.aghq()`, `summary.laplace()`

Examples

```
logfteta2d <- function(eta,y) {
  # eta is now (eta1,eta2)
  # y is now (y1,y2)
  n <- length(y)
  n1 <- ceiling(n/2)
  n2 <- floor(n/2)
  y1 <- y[1:n1]
  y2 <- y[(n1+1):(n1+n2)]
  eta1 <- eta[1]
  eta2 <- eta[2]
  sum(y1) * eta1 - (length(y1) + 1) * exp(eta1) - sum(lgamma(y1+1)) + eta1 +
    sum(y2) * eta2 - (length(y2) + 1) * exp(eta2) - sum(lgamma(y2+1)) + eta2
}
set.seed(84343124)
n1 <- 5
```

```
n2 <- 5
n <- n1+n2
y1 <- rpois(n1,5)
y2 <- rpois(n2,5)
objfunc2d <- function(x) logfteta2d(x,c(y1,y2))
funlist2d <- list(
  fn = objfunc2d,
  gr = function(x) numDeriv::grad(objfunc2d,x),
  he = function(x) numDeriv::hessian(objfunc2d,x)
)

thequadrature <- aghq(funlist2d,3,c(0,0))
plot(thequadrature)
```

`print.aghq` *Print method for AGHQ objects*

Description

Pretty print the object– just gives some basic information and then suggests the user call `summary(...)`.

Usage

```
## S3 method for class 'aghq'
print(x, ...)
```

Arguments

`x`	An object of class aghq.
`...`	not used.

Value

Silently prints summary information.

See Also

Other quadrature: `aghq()`, `laplace_approximation()`, `marginal_laplace()`, `normalize_logpost()`, `optimize_theta()`, `plot.aghq()`, `print.aghqsummary()`, `print.laplacesummary()`, `print.laplace()`, `summary.aghq()`, `summary.laplace()`

Examples

```
logfteta2d <- function(eta,y) {
  # eta is now (eta1,eta2)
  # y is now (y1,y2)
  n <- length(y)
```

```
  n1 <- ceiling(n/2)
  n2 <- floor(n/2)
  y1 <- y[1:n1]
  y2 <- y[(n1+1):(n1+n2)]
  eta1 <- eta[1]
  eta2 <- eta[2]
  sum(y1) * eta1 - (length(y1) + 1) * exp(eta1) - sum(lgamma(y1+1)) + eta1 +
    sum(y2) * eta2 - (length(y2) + 1) * exp(eta2) - sum(lgamma(y2+1)) + eta2
}
set.seed(84343124)
n1 <- 5
n2 <- 5
n <- n1+n2
y1 <- rpois(n1,5)
y2 <- rpois(n2,5)
objfunc2d <- function(x) logfteta2d(x,c(y1,y2))
funlist2d <- list(
  fn = objfunc2d,
  gr = function(x) numDeriv::grad(objfunc2d,x),
  he = function(x) numDeriv::hessian(objfunc2d,x)
)

thequadrature <- aghq(funlist2d,3,c(0,0))
thequadrature
```

`print.aghqsummary` *Print method for AGHQ summary objects*

Description

Print the summary of an aghq object. Almost always called by invoking `summary(...)` interactively in the console.

Usage

```
## S3 method for class 'aghqsummary'
print(x, ...)
```

Arguments

`x`	The result of calling `summary(...)` on an object of class aghq.
`...`	not used.

Value

Silently prints summary information.

See Also

Other quadrature: aghq(), laplace_approximation(), marginal_laplace(), normalize_logpost(), optimize_theta(), plot.aghq(), print.aghq(), print.laplacesummary(), print.laplace(), summary.aghq(), summary.laplace()

Other quadrature: aghq(), laplace_approximation(), marginal_laplace(), normalize_logpost(), optimize_theta(), plot.aghq(), print.aghq(), print.laplacesummary(), print.laplace(), summary.aghq(), summary.laplace()

Examples

```
logfteta2d <- function(eta,y) {
  # eta is now (eta1,eta2)
  # y is now (y1,y2)
  n <- length(y)
  n1 <- ceiling(n/2)
  n2 <- floor(n/2)
  y1 <- y[1:n1]
  y2 <- y[(n1+1):(n1+n2)]
  eta1 <- eta[1]
  eta2 <- eta[2]
  sum(y1) * eta1 - (length(y1) + 1) * exp(eta1) - sum(lgamma(y1+1)) + eta1 +
    sum(y2) * eta2 - (length(y2) + 1) * exp(eta2) - sum(lgamma(y2+1)) + eta2
}
set.seed(84343124)
n1 <- 5
n2 <- 5
n <- n1+n2
y1 <- rpois(n1,5)
y2 <- rpois(n2,5)
objfunc2d <- function(x) logfteta2d(x,c(y1,y2))
funlist2d <- list(
  fn = objfunc2d,
  gr = function(x) numDeriv::grad(objfunc2d,x),
  he = function(x) numDeriv::hessian(objfunc2d,x)
)

thequadrature <- aghq(funlist2d,3,c(0,0))
# Summarize and automatically call its print() method when called interactively:
summary(thequadrature)
```

---

print.laplace *Print method for AGHQ objects*

Description

Pretty print the object– just gives some basic information and then suggests the user call `summary(...)`.

Usage

```
## S3 method for class 'laplace'
print(x, ...)
```

Arguments

x	An object of class aghq.
...	not used.

Value

Silently prints summary information.

See Also

Other quadrature: aghq(), laplace_approximation(), marginal_laplace(), normalize_logpost(), optimize_theta(), plot.aghq(), print.aghqsummary(), print.aghq(), print.laplacesummary(), summary.aghq(), summary.laplace()

Examples

```
logfteta2d <- function(eta,y) {
  # eta is now (eta1,eta2)
  # y is now (y1,y2)
  n <- length(y)
  n1 <- ceiling(n/2)
  n2 <- floor(n/2)
  y1 <- y[1:n1]
  y2 <- y[(n1+1):(n1+n2)]
  eta1 <- eta[1]
  eta2 <- eta[2]
  sum(y1) * eta1 - (length(y1) + 1) * exp(eta1) - sum(lgamma(y1+1)) + eta1 +
    sum(y2) * eta2 - (length(y2) + 1) * exp(eta2) - sum(lgamma(y2+1)) + eta2
}
set.seed(84343124)
n1 <- 5
n2 <- 5
n <- n1+n2
y1 <- rpois(n1,5)
y2 <- rpois(n2,5)
objfunc2d <- function(x) logfteta2d(x,c(y1,y2))
funlist2d <- list(
  fn = objfunc2d,
  gr = function(x) numDeriv::grad(objfunc2d,x),
  he = function(x) numDeriv::hessian(objfunc2d,x)
)

thequadrature <- aghq(funlist2d,3,c(0,0))
thequadrature
```

`print.laplacesummary` *Print method for laplacesummary objects*

Description

Print the summary of an `laplace` object. Almost always called by invoking `summary(...)` interactively in the console.

Usage

```
## S3 method for class 'laplacesummary'
print(x, ...)
```

Arguments

`x`	The result of calling `summary(...)` on an object of class `laplace`.
`...`	not used.

Value

Silently prints summary information.

See Also

Other quadrature: `aghq()`, `laplace_approximation()`, `marginal_laplace()`, `normalize_logpost()`, `optimize_theta()`, `plot.aghq()`, `print.aghqsummary()`, `print.aghq()`, `print.laplace()`, `summary.aghq()`, `summary.laplace()`

Other quadrature: `aghq()`, `laplace_approximation()`, `marginal_laplace()`, `normalize_logpost()`, `optimize_theta()`, `plot.aghq()`, `print.aghqsummary()`, `print.aghq()`, `print.laplace()`, `summary.aghq()`, `summary.laplace()`

Examples

```
logfteta2d <- function(eta,y) {
  # eta is now (eta1,eta2)
  # y is now (y1,y2)
  n <- length(y)
  n1 <- ceiling(n/2)
  n2 <- floor(n/2)
  y1 <- y[1:n1]
  y2 <- y[(n1+1):(n1+n2)]
  eta1 <- eta[1]
  eta2 <- eta[2]
  sum(y1) * eta1 - (length(y1) + 1) * exp(eta1) - sum(lgamma(y1+1)) + eta1 +
    sum(y2) * eta2 - (length(y2) + 1) * exp(eta2) - sum(lgamma(y2+1)) + eta2
}
set.seed(84343124)
```

```
n1 <- 5
n2 <- 5
n <- n1+n2
y1 <- rpois(n1,5)
y2 <- rpois(n2,5)
objfunc2d <- function(x) logfteta2d(x,c(y1,y2))
funlist2d <- list(
  fn = objfunc2d,
  gr = function(x) numDeriv::grad(objfunc2d,x),
  he = function(x) numDeriv::hessian(objfunc2d,x)
)

thelaplace <- laplace_approximation(funlist2d,c(0,0))
# Summarize and automatically call its print() method when called interactively:
summary(thelaplace)
```

`sample_marginal` *Sample from the mixture-of-Gaussians approximation to the marginal posterior of the "inner" variables from a marginal Laplace approximation.*

Description

Draws samples from the mixture-of-Gaussians approximation to the variables that were marginalized over in a marginal Laplace approximation fit using `aghq::marginal_laplace`. Computes the Choleskies once and then draws as many samples as you ask, quickly.

Usage

```
sample_marginal(quad, M, ...)
```

Arguments

`quad`	The result of running `aghq::marginal_laplace`, object of class `marginallaplace` from which to draw samples.
`M`	Numeric, integer saying how many samples to draw
`...`	Not used

Details

This method samples from the posterior and returns a vector that is ordered the same as the "W" variables in your marginal Laplace approximation. If your model contains more than one type of variable that is being marginalized over, it is up to you to subset these out of W. See the `aghq` package software paper for several examples.

Value

A list containing elements:

- samps: d x M matrix where d = dim(W) and each column is a sample from pi(W|Y,theta)
- theta: M x S tibble where S = dim(theta) containing the value of theta for each sample

---

summary.aghq *Summary statistics computed using AGHQ*

Description

The summary.aghq method computes means, standard deviations, and quantiles and the associated print method prints these along with diagnostic and other information about the quadrature.

Usage

```
## S3 method for class 'aghq'
summary(object, ...)
```

Arguments

object	The return value from aghq::aghq.
...	not used.

Value

A list of class aghqsummary, which has a print method. Elements:

- mode: the mode of the log posterior
- hessian: the hessian of the log posterior at the mode
- covariance: the inverse of the hessian of the log posterior at the mode
- cholesky: the upper cholesky trinagle of the hessian of the log posterior at the mode
- quadpoints: the number of quadrature points used in each dimension
- dim: the dimension of the parameter space
- summarytable: a table containing the mean, median, mode, standard deviation and quantiles of each parameter, computed according to the posterior normalized using AGHQ

See Also

Other quadrature: aghq(), laplace_approximation(), marginal_laplace(), normalize_logpost(), optimize_theta(), plot.aghq(), print.aghqsummary(), print.aghq(), print.laplacesummary(), print.laplace(), summary.laplace()

Examples

```
logfteta2d <- function(eta,y) {
  # eta is now (eta1,eta2)
  # y is now (y1,y2)
  n <- length(y)
  n1 <- ceiling(n/2)
  n2 <- floor(n/2)
  y1 <- y[1:n1]
  y2 <- y[(n1+1):(n1+n2)]
  eta1 <- eta[1]
  eta2 <- eta[2]
  sum(y1) * eta1 - (length(y1) + 1) * exp(eta1) - sum(lgamma(y1+1)) + eta1 +
    sum(y2) * eta2 - (length(y2) + 1) * exp(eta2) - sum(lgamma(y2+1)) + eta2
}
set.seed(84343124)
n1 <- 5
n2 <- 5
n <- n1+n2
y1 <- rpois(n1,5)
y2 <- rpois(n2,5)
objfunc2d <- function(x) logfteta2d(x,c(y1,y2))
funlist2d <- list(
  fn = objfunc2d,
  gr = function(x) numDeriv::grad(objfunc2d,x),
  he = function(x) numDeriv::hessian(objfunc2d,x)
)

thequadrature <- aghq(funlist2d,3,c(0,0))
# Summarize and automatically call its print() method when called interactively:
summary(thequadrature)
# or, compute the summary and save for further processing:
ss <- summary(thequadrature)
str(ss)
```

`summary.laplace` *Summary method for Laplace Approximation objects*

Description

Summary method for objects of class `laplace`. Similar to the method for objects of class `aghq`, but assumes the problem is high-dimensional and does not compute or print any large objects or summaries. See `summary.aghq` for further information.

Usage

```
## S3 method for class 'laplace'
summary(object, ...)
```

Arguments

object	An object of class laplace.
...	not used.

Value

Silently prints summary information.

See Also

Other quadrature: aghq(), laplace_approximation(), marginal_laplace(), normalize_logpost(), optimize_theta(), plot.aghq(), print.aghqsummary(), print.aghq(), print.laplacesummary(), print.laplace(), summary.aghq()

Other quadrature: aghq(), laplace_approximation(), marginal_laplace(), normalize_logpost(), optimize_theta(), plot.aghq(), print.aghqsummary(), print.aghq(), print.laplacesummary(), print.laplace(), summary.aghq()

Examples

```
logfteta2d <- function(eta,y) {
  # eta is now (eta1,eta2)
  # y is now (y1,y2)
  n <- length(y)
  n1 <- ceiling(n/2)
  n2 <- floor(n/2)
  y1 <- y[1:n1]
  y2 <- y[(n1+1):(n1+n2)]
  eta1 <- eta[1]
  eta2 <- eta[2]
  sum(y1) * eta1 - (length(y1) + 1) * exp(eta1) - sum(lgamma(y1+1)) + eta1 +
    sum(y2) * eta2 - (length(y2) + 1) * exp(eta2) - sum(lgamma(y2+1)) + eta2
}
set.seed(84343124)
n1 <- 5
n2 <- 5
n <- n1+n2
y1 <- rpois(n1,5)
y2 <- rpois(n2,5)
objfunc2d <- function(x) logfteta2d(x,c(y1,y2))
funlist2d <- list(
  fn = objfunc2d,
  gr = function(x) numDeriv::grad(objfunc2d,x),
  he = function(x) numDeriv::hessian(objfunc2d,x)
)

thelaplace <- laplace_approximation(funlist2d,c(0,0))
# Summarize and automatically call its print() method when called interactively:
summary(thelaplace)
```

Index