Package ‘aghq’

November 3, 2021

Type Package

Title Adaptive Gauss Hermite Quadrature for Bayesian Inference

Version 0.3.0

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 file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.2

Depends R (>= 3.5.0)

Imports methods, mvQuad, Matrix, rlang, polynom, splines, numDeriv

Suggests trustOptim, trust, testthat (>= 2.1.0), parallel

Language en-US

Roxygen list(markdown = TRUE)

NeedsCompilation no

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,
  basegrid = 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.

`basegrid` Optional. Provide an object of class `NIGrid` from the `mvQuad` package, representing the base quadrature rule that will be adapted. This is only for users who want more complete control over the quadrature, and is not necessary if you are fine with the default option which basically corresponds to `mvQuad::createNIGrid(length(theta)`
Note: the `mvQuad` functions used within `aghq` operate on grids in memory, so your `basegrid` object will be changed after you run `aghq`.

`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`.

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
- control: the control parameters passed

See Also

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

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(obj, ...)

## Default S3 method:
compute_moment(obj, ff = function(x) 1, ...)

## S3 method for class 'aghq'
compute_moment(obj, ff = function(x) 1, ...)
```

Arguments

`obj`	Object of class `aghq` output by `aghq::aghq()`, or its `normalized_posterior` element. See ?aghq.
`...`	Used to pass additional argument `ff`.
`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(obj, ...)

## Default S3 method:
compute_pdf_and_cdf(
  obj,
  transformation = NULL,
  finegrid = NULL,
  interpolation = "auto",
  ...
)

## S3 method for class 'list'
compute_pdf_and_cdf(obj, ...)

## S3 method for class 'aghq'
compute_pdf_and_cdf(obj, ...)
```

Arguments

`obj`	Either the output of `aghq::aghq()`, its list of marginal distributions (element `marginals`), or an individual `data.frame` containing one of these marginal distributions as output by `aghq::marginal_posterior()`.

`...`	Used to pass additional arguments.
`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.
`interpolation`	Which method to use for interpolating the marginal posterior, `'polynomial'` (default) or `'spline'`? If `k > 3` then the polynomial may be unstable and you should use the spline, but the spline doesn't work *unless* `k > 3` so it's not the default. See `interpolate_marginal_posterior()`.

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(obj, ...)

## Default S3 method:
compute_quantiles(
  obj,
  q = c(0.025, 0.975),
  transformation = NULL,
  interpolation = "polynomial",
  ...
)

## S3 method for class 'list'
compute_quantiles(obj, q = c(0.025, 0.975), transformation = NULL, ...)

## S3 method for class 'aghq'
compute_quantiles(obj, q = c(0.025, 0.975), transformation = NULL, ...)
```

Arguments

`obj`	Either the output of `aghq::aghq()`, its list of marginal distributions (element `marginals`), or an individual `data.frame` containing one of these marginal distributions as output by `aghq::marginal_posterior()`.
`...`	Used to pass additional arguments.
`q`	Numeric vector of values in (0,1). The quantiles to compute.
`transformation`	Optional. A list containing function `fromtheta()` which accepts and returns numeric vectors, defining a parameter transformation for which you would like the quantiles of. See `?compute_pdf_and_cdf`. This transformation must be monotone and the function checks whether it's increasing or decreasing and returns the transformed quantiles, ordered appropriately.
`interpolation`	Which method to use for interpolating the marginal posterior, `'polynomial'` (default) or `'spline'`? If `k > 3` then the polynomial may be unstable and you should use the spline, but the spline doesn't work *unless* `k > 3` so it's not the default. See `interpolate_marginal_posterior()`.

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)
```

`default_control` *Default control arguments for* `aghq::aghq()`.

Description

Run `default_control()` to print the list of valid control parameters and their defaults, and run with named arguments to change the defaults.

Usage

```
default_control(...)
```

Arguments

`...`	You can provide a named value for any control parameter and its value will be set accordingly. See `?aghq` and examples here.

Details

Valid options are:

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

  Default is 'sparse_trust'.
- `negate`: default `FALSE`. Multiply the functions in `ff` by `-1`? The reason for having this option is for full compatibility with `TMB`: while of course `TMB` allows you to code up your log-posterior any way you like, all of its excellent features including its automatic Laplace approximation and MCMC sampling with `tmbstan` assume you have coded your template to return the **negated** log-posterior. However, by default, `aghq` assumes you have provided the log-posterior **without negation**. Set `negate = TRUE` if you have provided a template which computes the **negated** log-posterior and its derivatives.
- `ndConstruction`: construct a multivariate quadrature rule using a `"product"` rule or a `"sparse"` grid? Default `"product"`. See `?mvQuad::createNIGrid()`.
- `interpolation`: how to interpolate the marginal posteriors. The `'auto'` option (default) chooses for you and should always work well. The `'polynomial'` option uses `polynom::poly.calc()` to construct a global polynomial interpolant and has been observed to be unstable as the number of quadrature points gets larger, which is obviously a bad thing. Try `'spline'` instead, which uses a cubic B-Spline interpolant from `splines::interpSpline()`.
- numhessian: logical, default `FALSE`. Replace the `ff$he` with a numerically-differentiated version, by calling `numDeriv::jacobian` on `ff$gr`. Used mainly for `TMB` with the automatic Laplace approximation, which does not have an automatic Hessian.
- onlynormconst: logical, default `FALSE`. Skip everything after the calculation of the log integral, and just return the numeric value of the log integral. Saves computation time, and most useful in cases where `aghq` is being used as a step in a more complicated procedure.

Value

A list of argument values.

Examples

```
default_control()
default_control(method = "trust")
default_control(negate = TRUE)
```

---

default_control_marglaplace

Default control arguments for `aghq::marginal_laplace()`.

Description

Run `default_control_marglaplace()` to print the list of valid control parameters and their defaults, and run with named arguments to change the defaults.

Usage

```
default_control_marglaplace(...)
```

Arguments

`...`	You can provide a named value for any control parameter and its value will be set accordingly. See `?marginal_laplace` and examples here.

Details

Valid options are:

- `method`: optimization method to use for the `theta` optimization:
  - 'BFGS' (default): `optim(...,method = "BFGS")`
  - 'sparse_trust': `trustOptim::trust.optim`
  - 'SR1': `trustOptim::trust.optim` with `method = 'SR1'`
  - 'sparse': `trust::trust`
- `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'.
- `negate`: default `FALSE`. Multiply the functions in `ff` by `-1`? The reason for having this option is for full compatibility with `TMB`: while of course `TMB` allows you to code up your log-posterior any way you like, all of its excellent features including its automatic Laplace approximation and MCMC sampling with `tmbstan` assume you have coded your template to return the **negated** log-posterior. However, by default, `aghq` assumes you have provided the log-posterior **without negation**. Set `negate = TRUE` if you have provided a template which computes the **negated** log-posterior and its derivatives. **Note** that I don't expect there to be any reason to need this argument for `marginal_laplace`; if you are doing a marginal Laplace approximation using the automatic Laplace approximation provided by `TMB`, you should check out `aghq::marginal_laplace_tmb()`.
- `interpolation`: how to interpolate the marginal posteriors. The `'auto'` option (default) chooses for you and should always work well. The `'polynomial'` option uses `polynom::poly.calc()` to construct a global polynomial interpolant and has been observed to be unstable as the number of quadrature points gets larger, which is obviously a bad thing. Try `'spline'` instead, which uses a cubic B-Spline interpolant from `splines::interpSpline()`.

- numhessian: logical, default `FALSE`. Replace the `ff$he` with a numerically-differentiated version, by calling `numDeriv::jacobian` on `ff$gr`. Used mainly for `TMB` with the automatic Laplace approximation, which does not have an automatic Hessian.
- onlynormconst: logical, default `FALSE`. Skip everything after the calculation of the log integral, and just return the numeric value of the log integral. Saves computation time, and most useful in cases where `aghq` is being used as a step in a more complicated procedure.

Value

A list of argument values.

Examples

```
default_control_marglaplace()
default_control_marglaplace(method = "trust")
default_control_marglaplace(method = "trust",inner_method = "trust")
default_control_marglaplace(negate = TRUE)
```

`default_control_tmb` *Default control arguments for* `aghq::marginal_laplace_tmb()`.

Description

Run `default_control_marglaplace()` to print the list of valid control parameters and their defaults, and run with named arguments to change the defaults.

Usage

```
default_control_tmb(...)
```

Arguments

`...`	You can provide a named value for any control parameter and its value will be set accordingly. See `?marginal_laplace` and examples here.

Details

Valid options are:

- `method`: optimization method to use for the `theta` optimization:
  - 'BFGS' (default): `optim(...,method = "BFGS")`
  - 'sparse_trust': `trustOptim::trust.optim`
  - 'SR1': `trustOptim::trust.optim` with `method = 'SR1'`
  - 'sparse': `trust::trust`
- `negate`: default `TRUE`. Assumes that your `TMB` function template computes the **negated** log-posterior, which it must if you're using `TMB`'s automatic Laplace approximation, which you must be if you're using this function!.

- `interpolation`: how to interpolate the marginal posteriors. The `'auto'` option (default) chooses for you and should always work well. The `'polynomial'` option uses `polynom::poly.calc()` to construct a global polynomial interpolant and has been observed to be unstable as the number of quadrature points gets larger, which is obviously a bad thing. Try `'spline'` instead, which uses a cubic B-Spline interpolant from `splines::interpSpline()`.
- numhessian: logical, default `TRUE`. Replace the `ff$he` with a numerically-differentiated version, by calling `numDeriv::jacobian` on `ff$gr`. Used mainly for `TMB` with the automatic Laplace approximation, which does not have an automatic Hessian.
- onlynormconst: logical, default `FALSE`. Skip everything after the calculation of the log integral, and just return the numeric value of the log integral. Saves computation time, and most useful in cases where `aghq` is being used as a step in a more complicated procedure.

Value

A list of argument values.

Examples

```
default_control_marglaplace()
default_control_marglaplace(method = "trust")
default_control_marglaplace(method = "trust",inner_method = "trust")
default_control_marglaplace(negate = TRUE)
```

`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).

get_log_normconst *Obtain the log-normalizing constant from a fitted quadrature object*

Description

Quick helper S3 method to retrieve the log normalizing constant from an object created using the aghq package. Methods for a list (returned by `aghq::normalize_posterior`) and for objects of class `aghq`, `laplace`, and `marginallaplace`.

Usage

```
get_log_normconst(obj, ...)

## Default S3 method:
get_log_normconst(obj, ...)

## S3 method for class 'numeric'
get_log_normconst(obj, ...)

## S3 method for class 'aghq'
get_log_normconst(obj, ...)

## S3 method for class 'laplace'
get_log_normconst(obj, ...)

## S3 method for class 'marginallaplace'
get_log_normconst(obj, ...)
```

Arguments

obj	A list returned by aghq::normalize_posterior or an object of class aghq, laplace, or marginallaplace.
...	Not used

Value

A number representing the natural logarithm of the approximated normalizing constant.

See Also

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

---

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,
  method = c("auto", "polynomial", "spline")
)
```

Arguments

margpost	The output of aghq::marginal_posterior. See the documentation for that function.
method	The method to use. Default is a k point polynomial interpolant using polynom::poly.calc(). This has been observed to result in unstable behaviour for larger numbers of quadrature points k, which is of course undesirable. If k > 3, you can set method = 'spline' to use splines::interpSpline() instead, which uses cubic B-Splines. These should always be better than a straight polynomial, except don't work when k < 4 which is why they aren't the default. If you try and set method = 'spline' with k < 4 it will be changed back to polynomial, with a warning.

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 `ff$fn`, `ff$gr`, 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()`, `get_log_normconst()`, `marginal_laplace_tmb()`, `marginal_laplace()`, `normalize_logpost()`, `optimize_theta()`, `plot.aghq()`, `print.aghqsummary()`, `print.aghq()`, `print.laplacesummary()`, `print.laplace()`, `print.marginallaplacesummary()`, `summary.aghq()`, `summary.laplace()`, `summary.marginallaplace()`

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 `ff$fn`, `ff$gr`, 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(), get_log_normconst(), laplace_approximation(), marginal_laplace_tmb(), normalize_logpost(), optimize_theta(), plot.aghq(), print.aghqsummary(), print.aghq(), print.laplacesummary(), print.laplace(), print.marginallaplacesummary(), summary.aghq(), summary.laplace(), summary.marginallaplace()

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))
objfunc2dmarg <- function(W,theta) objfunc2d(c(W,theta))
objfunc2dmarggr <- function(W,theta) {
  fn <- function(W) objfunc2dmarg(W,theta)
  numDeriv::grad(fn,W)
}
objfunc2dmarghe <- function(W,theta) {
  fn <- function(W) objfunc2dmarg(W,theta)
  numDeriv::hessian(fn,W)
}

funlist2dmarg <- list(
  fn = objfunc2dmarg,
  gr = objfunc2dmarggr,
  he = objfunc2dmarghe
)
```

`marginal_laplace_tmb` *AGHQ-normalized marginal Laplace approximation from a TMB function template*

Description

Implement the algorithm from `aghq::marginal_laplace()`, but making use of `TMB`'s automatic Laplace approximation. This function takes a function list from `TMB::MakeADFun()` with a non-empty set of `random` parameters, in which the `fn` and `gr` are the unnormalized marginal Laplace approximation and its gradient. It then calls `aghq::aghq()` and formats the resulting object so that its contents and class match the output of `aghq::marginal_laplace()` and are hence suitable for post-processing with `summary`, `aghq::sample_marginal()`, and so on.

Usage

```
marginal_laplace_tmb(
  ff,
  k,
  startingvalue,
  optresults = NULL,
  basegrid = NULL,
  control = default_control_tmb(),
  ...
)
```

Arguments

`ff`	The output of calling `TMB::MakeADFun()` with `random` set to a non-empty subset of the parameters. **VERY IMPORTANT**: `TMB`'s automatic Laplace approximation requires you to write your template implementing the **negated** log-posterior. Therefore, this list that you input here will contain components `fn`, `gr` and `he` that implement the **negated** log-posterior and its derivatives. This is **opposite** to every other comparable function in the `aghq` package, and is done here to emphasize compatibility with `TMB`.
`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.
`basegrid`	Optional. Provide an object of class `NIGrid` from the `mvQuad` package, representing the base quadrature rule that will be adapted. This is only for users who want more complete control over the quadrature, and is not necessary if you are fine with the default option which basically corresponds to `mvQuad::createNIGrid(length(theta)` **Note**: the `mvQuad` functions used within `aghq` operate on grids in memory, so your `basegrid` object will be changed after you run `aghq`.

`control` A list of control parameters. See `?default_control` for details. Valid options are:

- `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'.
- `negate`: default `TRUE`. See `?default_control_tmb`. Assumes that your `TMB` function template computes the **negated** log-posterior, which it must if you're using `TMB`'s automatic Laplace approximation, which you must be if you're using this function!

.

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

Details

Because `TMB` does not yet have the Hessian of the log marginal Laplace approximation implemented, a numerically-differentiated jacobian of the gradient is used via `numDeriv::jacobian()`. You can turn this off (using `ff$he()` instead, which you'll have to modify yourself) using `default_control_tmb(numhessian = FALSE)`.

Value

If `k > 1`, an object of class `marginallaplace` (and inheriting from class aghq) of the same structure as that returned by `aghq::marginal_laplace()`, with `plot` and `summary` methods, and suitable for input into `aghq::sample_marginal()` for drawing posterior samples.

See Also

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

`marginal_posterior` *Marginal Posteriors*

Description

Compute the marginal posterior for a given parameter using AGHQ. This function is mostly called within `aghq()`.

Usage

```
marginal_posterior(
  optresults,
  k,
  j,
  basegrid = NULL,
  ndConstruction = "product"
)
```

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.
basegrid	Optional. Provide an object of class NIGrid from the mvQuad package, representing the base quadrature rule that will be adapted. This is only for users who want more complete control over the quadrature, and is not necessary if you are fine with the default option which basically corresponds to mvQuad::createNIGrid(length(theta)
ndConstruction	Create a multivariate grid using a product or sparse construction? Passed directly to mvQuad::createNIGrid(), see that function for further details. Note that the use of sparse grids within aghq is currently **experimental** and not supported by tests. In particular, calculation of marginal posteriors is known to fail currently.

Value

a 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_po

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,
  basegrid = NULL,
  ndConstruction = "product",
  ...
)
```

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.

basegrid	Optional. Provide an object of class NIGrid from the mvQuad package, representing the base quadrature rule that will be adapted. This is only for users who want more complete control over the quadrature, and is not necessary if you are fine with the default option which basically corresponds to mvQuad::createNIGrid(length(theta)
ndConstruction	Create a multivariate grid using a product or sparse construction? Passed directly to mvQuad::createNIGrid(), see that function for further details. Note that the use of sparse grids within aghq is currently **experimental** and not supported by tests. In particular, calculation of marginal posteriors is known to fail currently.
...	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(), get_log_normconst(), laplace_approximation(), marginal_laplace_tmb(), marginal_laplace(), optimize_theta(), plot.aghq(), print.aghqsummary(), print.aghq(), print.laplacesummary(), print.laplace(), print.marginallaplacesummary(), summary.aghq(), summary.laplace(), summary.marginallaplace()

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 = default_control(method = "trust"))
opt_bfgs <- optimize_theta(funlist,1.5,control = default_control(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()`, `get_log_normconst()`, `laplace_approximation()`, `marginal_laplace_tmb()`, `marginal_laplace()`, `normalize_logpost()`, `plot.aghq()`, `print.aghqsummary()`, `print.aghq()`, `print.laplacesummary()`, `print.laplace()`, `print.marginallaplacesummary()`, `summary.aghq()`, `summary.laplace()`, `summary.marginallaplace()`

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 = default_control(method = "trust"))
optimize_theta(funlist,1.5,control = default_control(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(), get_log_normconst(), laplace_approximation(), marginal_laplace_tmb(), marginal_laplace(), normalize_logpost(), optimize_theta(), print.aghqsummary(), print.aghq(), print.laplacesummary(), print.laplace(), print.marginallaplacesummary(), summary.aghq(), summary.laplace(), summary.marginallaplace()

Other quadrature: aghq(), get_log_normconst(), laplace_approximation(), marginal_laplace_tmb(), marginal_laplace(), normalize_logpost(), optimize_theta(), print.aghqsummary(), print.aghq(), print.laplacesummary(), print.laplace(), print.marginallaplacesummary(), summary.aghq(), summary.laplace(), summary.marginallaplace()

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()`, `get_log_normconst()`, `laplace_approximation()`, `marginal_laplace_tmb()`, `marginal_laplace()`, `normalize_logpost()`, `optimize_theta()`, `plot.aghq()`, `print.aghqsummary()`, `print.laplacesummary()`, `print.laplace()`, `print.marginallaplacesummary()`, `summary.aghq()`, `summary.laplace()`, `summary.marginallaplace()`

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()`, `get_log_normconst()`, `laplace_approximation()`, `marginal_laplace_tmb()`, `marginal_laplace()`, `normalize_logpost()`, `optimize_theta()`, `plot.aghq()`, `print.aghq()`, `print.laplacesummary()`, `print.laplace()`, `print.marginallaplacesummary()`, `summary.aghq()`, `summary.laplace()`, `summary.marginallaplace()`

Other quadrature: `aghq()`, `get_log_normconst()`, `laplace_approximation()`, `marginal_laplace_tmb()`, `marginal_laplace()`, `normalize_logpost()`, `optimize_theta()`, `plot.aghq()`, `print.aghq()`, `print.laplacesummary()`, `print.laplace()`, `print.marginallaplacesummary()`, `summary.aghq()`, `summary.laplace()`, `summary.marginallaplace()`

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()`, `get_log_normconst()`, `laplace_approximation()`, `marginal_laplace_tmb()`, `marginal_laplace()`, `normalize_logpost()`, `optimize_theta()`, `plot.aghq()`, `print.aghqsummary()`, `print.aghq()`, `print.laplacesummary()`, `print.marginallaplacesummary()`, `summary.aghq()`, `summary.laplace()`, `summary.marginallaplace()`

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(), get_log_normconst(), laplace_approximation(), marginal_laplace_tmb(), marginal_laplace(), normalize_logpost(), optimize_theta(), plot.aghq(), print.aghqsummary(), print.aghq(), print.laplace(), print.marginallaplacesummary(), summary.aghq(), summary.laplace(), summary.marginallaplace()

Other quadrature: aghq(), get_log_normconst(), laplace_approximation(), marginal_laplace_tmb(), marginal_laplace(), normalize_logpost(), optimize_theta(), plot.aghq(), print.aghqsummary(), print.aghq(), print.laplace(), print.marginallaplacesummary(), summary.aghq(), summary.laplace(), summary.marginallaplace()

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)
```

---

print.marginallaplacesummary

Summary statistics for models using marginal Laplace approximations

---

Description

The `summary.marginallaplace` calls `summary.aghq`, but also computes summary statistics of the random effects, by drawing from their approximate posterior using `aghq::sample_marginal` with the specified number of samples.

Usage

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

Arguments

`x`	Object of class `marginallaplacesummary` returned by calling `summary` on an object of class `marginallaplace`.
`...`	not used.

Value

Nothing. Prints contents.

See Also

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

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))
objfunc2dmarg <- function(W,theta) objfunc2d(c(W,theta))
objfunc2dmarggr <- function(W,theta) {
  fn <- function(W) objfunc2dmarg(W,theta)
  numDeriv::grad(fn,W)
}
objfunc2dmarghe <- function(W,theta) {
  fn <- function(W) objfunc2dmarg(W,theta)
  numDeriv::hessian(fn,W)
}

funlist2dmarg <- list(
  fn = objfunc2dmarg,
  gr = objfunc2dmarggr,
  he = objfunc2dmarghe
)

themarginallaplace <- aghq::marginal_laplace(funlist2dmarg,3,list(W = 0,theta = 0))
summary(themarginallaplace)
```

`sample_marginal` *Exact independent samples from an approximate posterior distribution*

Description

Draws samples from an approximate marginal distribution for general posteriors approximated using `aghq`, or from the mixture-of-Gaussians approximation to the variables that were marginalized over in a marginal Laplace approximation fit using `aghq::marginal_laplace` or `aghq::marginal_laplace_tmb`.

Usage

```
sample_marginal(quad, ...)

## S3 method for class 'aghq'
sample_marginal(quad, M, transformation = NULL, interpolation = "auto", ...)

## S3 method for class 'marginallaplace'
sample_marginal(
  quad,
  M,
  transformation = NULL,
  interpolation = "auto",
  numcores = getOption("mc.cores", 1L),
  ...
)
```

Arguments

`quad`	Object from which to draw samples. An object inheriting from class `marginallaplace` (the result of running `aghq::marginal_laplace` or `aghq::marginal_laplace_tmb`), or an object inheriting from class `aghq` (the result of running `aghq::aghq()`). Can also provide a `data.frame` returned by `aghq::compute_pdf_and_cdf` in which case samples are returned for `transparam` if `transformation` is provided, and for `param` if `transformation = NULL`.
`...`	Used to pass additional arguments.
`M`	Numeric, integer saying how many samples to draw
`transformation`	Optional. A list containing function `fromtheta()` which accepts and returns numeric vectors, defining a parameter transformation for which you would like samples to be taken. See `?compute_pdf_and_cdf`. Note that unlike there, where this operation is a bit more complicated, here all is done is samples are taken on the original scale and then `transformation$fromtheta()` is called on them before returning.
`interpolation`	Which method to use for interpolating the marginal posteriors (and hence to draw samples using the inverse CDF method), `'auto'` (choose for you), `'polynomial'` or `'spline'`? If `k > 3` then the polynomial may be unstable and you should use the spline, but the spline doesn't work *unless* `k > 3` so it's not the default. The default of `'auto'` figures this out for you. See `interpolate_marginal_posterior()`.
`numcores`	Integer, default `getOption('mc.cores')`. If greater than 1, the Cholesky decompositions of the Hessians are computed in parallel using `parallel::mcapply`,

for the Gaussian approximation involved for objects of class marginallaplace. This step is slow so may be sped up by parallelization, if the matrices are sparse (and hence the operation is just slow, but not memory-intensive). Uses the parallel package so is not available on Windows.

Details

For objects of class aghq or their marginal distribution components, sampling is done using the inverse CDF method, which is just compute_quantiles(quad$marginals[[1]],runif(M)).

For marginal Laplace approximations (aghq::marginal_laplace()): this method samples from the posterior and returns a vector that is ordered the same as the "W" variables in your marginal Laplace approximation. See Algorithm 1 in Stringer et. al. (2021, https://arxiv.org/abs/2103.07425) for the algorithm; the details of sampling from a Gaussian are described in the reference(s) therein, which makes use of the (sparse) Cholesky factors. These are computed once for each quadrature point and stored.

For the marginal Laplace approximations where the "inner" model is handled entirely by TMB (aghq::marginal_laplace_tmb), the interface here is identical to above, with the order of the "W" vector being determined by TMB. See the names of ffenvlast.par, for example (where ff is your template obtained from a call to TMB::MakeADFun.

Value

If run on a marginallaplace object, 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
- thetasamples: A list of S numeric vectors each of length M where the jth element is a sample from pi(theta_{j}|Y). These are samples from the **marginals**, NOT the **joint**. Sampling from the joint is a much more difficult problem and how to do so in this context is an active area of research.

If run on an aghq object, then a list with just the thetasamples element. It still returns a list to maintain output consistency across inputs.

If, for some reason, you don't want to do the sampling from pi(theta|Y), you can manually set quad$marginals = NULL. Note that this sampling is typically *very* fast and so I don't know why you would need to not do it but the option is there if you like.

If, again for some reason, you just want samples from one marginal distribution using inverse CDF, you can just do compute_quantiles(quad$marginals[[1]],runif(M)).

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))
objfunc2dmarg <- function(W,theta) objfunc2d(c(W,theta))
objfunc2dmarggr <- function(W,theta) {
  fn <- function(W) objfunc2dmarg(W,theta)
  numDeriv::grad(fn,W)
}
objfunc2dmarghe <- function(W,theta) {
  fn <- function(W) objfunc2dmarg(W,theta)
  numDeriv::hessian(fn,W)
}

funlist2dmarg <- list(
  fn = objfunc2dmarg,
  gr = objfunc2dmarggr,
  he = objfunc2dmarghe
)
```

`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(), get_log_normconst(), laplace_approximation(), marginal_laplace_tmb(), marginal_laplace(), normalize_logpost(), optimize_theta(), plot.aghq(), print.aghqsummary(), print.aghq(), print.laplacesummary(), print.laplace(), print.marginallaplacesummary(), summary.laplace(), summary.marginallaplace()

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(), get_log_normconst(), laplace_approximation(), marginal_laplace_tmb(), marginal_laplace(), normalize_logpost(), optimize_theta(), plot.aghq(), print.aghqsummary(), print.aghq(), print.laplacesummary(), print.laplace(), print.marginallaplacesummary(), summary.aghq(), summary.marginallaplace()

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

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)
```

`summary.marginallaplace`

Summary statistics for models using marginal Laplace approximations

Description

The `summary.marginallaplace` calls `summary.aghq`, but also computes summary statistics of the random effects, by drawing from their approximate posterior using `aghq::sample_marginal` with the specified number of samples.

Usage

```
## S3 method for class 'marginallaplace'
summary(object, M = 1000, max_print = 30, ...)
```

Arguments

`object`	Object inheriting from **both** classes `aghq` and `marginallaplace`, for example as returned by `aghq::marginal_laplace` or `aghq::marginal_laplace_tmb`.
`M`	Number of samples to use to compute summary statistics of the random effects. Default `1000`. Lower runs faster, higher is more accurate.
`max_print`	Sometimes there are a lot of random effects. If there are more random effects than `max_print`, the random effects aren't summarized, and a note is printed to this effect. Default `30`.
`...`	not used.

Value

A list containing an object of class `aghqsummary` (see `summary.aghq`) plus the following objects:

See Also

Other quadrature: `aghq()`, `get_log_normconst()`, `laplace_approximation()`, `marginal_laplace_tmb()`, `marginal_laplace()`, `normalize_logpost()`, `optimize_theta()`, `plot.aghq()`, `print.aghqsummary()`, `print.aghq()`, `print.laplacesummary()`, `print.laplace()`, `print.marginallaplacesummary()`, `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))
objfunc2dmarg <- function(W,theta) objfunc2d(c(W,theta))
objfunc2dmarggr <- function(W,theta) {
  fn <- function(W) objfunc2dmarg(W,theta)
  numDeriv::grad(fn,W)
}
objfunc2dmarghe <- function(W,theta) {
  fn <- function(W) objfunc2dmarg(W,theta)
  numDeriv::hessian(fn,W)
}

funlist2dmarg <- list(
  fn = objfunc2dmarg,
  gr = objfunc2dmarggr,
  he = objfunc2dmarghe
)

themarginallaplace <- aghq::marginal_laplace(funlist2dmarg,3,list(W = 0,theta = 0))
summary(themarginallaplace)
```

`validate_control` *Validate a control list*

Description

This function checks that the names and value types for a supplied `control` list are valid and are unlikely to cause further errors within aghq and related functions. Users should not have to worry about this and should just use `default_control()` and related constructors.

Usage

```
validate_control(control, type = c("aghq", "marglaplace", "tmb"), ...)
```

Arguments

`control`	A list.
`type`	One of `c('aghq','marglapace','tmb')`. The type of control object to validate. Will basically validate against the arguments required by `aghq`, `marginal_laplace`, and `marginal_laplace_tmb`, respectively.
`...`	Not used.

Details

To users reading this: just use `default_control()`, `default_control_marglaplace()`, or `default_control_tmb()` as appropriate, to ensure that your control arguments are correct. This function just exists to provide more descriptive error messages in the event that an incompatible list is provided.

Value

Logical, TRUE if the list of control arguments is valid, else FALSE.

Examples

```
validate_control(default_control())
validate_control(default_control_marglaplace(),type = "marglaplace")
validate_control(default_control_tmb(),type = "tmb")
```

Index