# Package: aghq (via r-universe)

August 20, 2024

**Type** Package

**Title** Adaptive Gauss Hermite Quadrature for Bayesian Inference

**Version** 0.4.1

**Author** Alex Stringer

**Maintainer** Alex Stringer <alex.stringer@uwaterloo.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** GPL (>= 3)

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.2.1

**Depends** R (>= 3.5.0)

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

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

**Language** en-US

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2023-06-02 13:40:02 UTC

## Contents

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,
  transformation = default_transformation(),
  optresults = NULL,
  basegrid = NULL,
  control = default_control(),
  ...
)
```

## Arguments

| | |
|---|---|
| `ff` | A list with three elements:<br>• `fn`: function taking argument `theta` and returning a numeric value representing the log-posterior at `theta`<br>• `gr`: function taking argument `theta` and returning a numeric vector representing the gradient of the log-posterior at `theta`<br>• `he`: function taking argument `theta` and returning a numeric matrix representing the hessian of the log-posterior at `theta`<br>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. |
| `transformation` | Optional. Do the quadrature for parameter `theta`, but return summaries and plots for parameter `g(theta)`. `transformation` is either: a) an `aghqtrans` object returned by `aghq::make_transformation`, or b) a list that will be passed to that function internally. See `?aghq::make_transformation` for details. |
| `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),'GH`
**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_hessian(), get_log_normconst(), get_mode(), get_nodesandweights(), get_numquadpoints(), get_opt_results(), get_param_dim(), laplace_approximation(), marginal_laplace_tmb() marginal_laplace(), nested_quadrature(), 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, ...)

## S3 method for class 'list'
compute_moment(
  obj,
  ff = function(x) 1,
  gg = NULL,
  nn = NULL,
  type = c("raw", "central"),
  method = c("auto", "reuse", "correct"),
  ...
)

## S3 method for class 'aghq'
compute_moment(
  obj,
  ff = function(x) 1,
  gg = NULL,
  nn = NULL,
  type = c("raw", "central"),
  method = c("auto", "reuse", "correct"),
  ...
)

## Default S3 method:
compute_moment(
  obj,
  ff = function(x) 1,
  gg = NULL,
  method = c("auto", "reuse", "correct"),
  ...
)
```

## Arguments

| | |
|---|---|
| obj | Object of class aghq output by aghq::aghq(). See ?aghq. |
| ... | Used to pass additional argument ff. |
| ff | Any R function which takes in a numeric vector and returns a numeric vector. Exactly one of ff or gg must be provided. If both are provided, aghq::compute_moment() will use gg, without warning. |
| gg | The output of, or an object which may be input to aghq::make_moment_function(). See documentation of that function. Exactly one of ff or gg must be provided. If both are provided, aghq::compute_moment() will use gg, without warning. |
| nn | A numeric scalar. Compute the approximate moment of this order, E(theta^nn\|Y). See details. If nn is provided, compute_moment will use it over ff or gg, without warning. |

| | |
|---|---|
| type | Either 'raw' (default) or 'central', see details. |
| method | Method for computing the quadrature points used to approximate moment. One of 'reuse' (default) or 'correct'. See details. The default SHOULD be 'correct'; it is currently set to 'reuse' to maintain compatibility of results with previous versions. This will be switched in a future major release. |

## Details

If multiple of nn, gg, and ff are provided, then compute_moment will use nn, gg, or ff, in that order, without warning.

There are several approximations available. The "best" one is obtained by specifying gg and using method = 'correct'. This recomputes the mode and curvature for the function g(theta)posterior(theta), and takes the ratio of the AGHQ approximation to this function to the AGHQ approximation to the marginal likelihood. This obtains the same relative rate of convergence as the AGHQ approximation to the marginal likelihood. It may take a little extra time, and only works for **positive, scalar-valued** functions g.

method = 'reuse' re-uses the AGHQ adapted points and weights. It's faster than the correct method, because it does not involve any new optimization, it's just a weighted sum. No convergence theory. Seems to work ok in "practice". "Works" for arbitrary g.

Specifying ff instead of gg automatically uses method = 'reuse'. This interface is provided for backwards compatibility mostly. However, one advantage is that it allows for **vector-valued** functions, in which case it just returns the corresponding vector of approximate moments. Also, it only requires the adapted nodes and weights, not the ability to evaluate the log-posterior and its derivatives, although this is unlikely to be a practical concern.

Specifying a numeric value nn will return the moment E(theta^nn|Y). This automatically does some internal shifting to get the evaluations away from zero, to avoid the inherent problem of multi-modal "posteriors" that occurs when the posterior mode is near zero, and account for the fact that some of the new adapted quadrature points may be negative. So, the actual return value is E(theta^nn + a|Y) - a for a cleverly-chosen value a.

Finally, type='raw' computes raw moments E(g(theta)|Y), where type='central' computes central moments, E(g(theta - E(g(theta)|Y))|Y). See examples.

## Value

A numeric vector containing the moment(s) of ff with respect to the joint distribution being approximated using 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)
)
quad <- aghq(funlist2d,7,c(0,0))
```

---

`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 = default_transformation(),
  finegrid = NULL,
  interpolation = "auto",
  ...
)

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

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

## 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. Calculate pdf/cdf for a transformation of the parameter whose posterior was normalized using adaptive quadrature. transformation is either: a) an aghqtrans object returned by aghq::make_transformation, or b) a list that will be passed to that function internally. See ?aghq::make_transformation for details. |
| 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_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,
  q = c(0.025, 0.975),
  transformation = default_transformation(),
  ...
)

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

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

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

## 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(). |
| q | Numeric vector of values in (0,1). The quantiles to compute. |
| transformation | Optional. Calculate marginal quantiles for a transformation of the parameter whose posterior was normalized using adaptive quadrature. transformation is either: a) an aghqtrans object returned by aghq::make_transformation, or b) a list that will be passed to that function internally. See ?aghq::make_transformation for details. Note that since g has to be monotone anyways, this just returns sort(g(q)) instead of q. |
| ... | Used to pass additional arguments. |
| 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_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)
```

---

`correct_marginals` *Correct the posterior marginals of a fitted aghq object*

---

## Description

The default method of computation for aghq objects computes approximate marginals using an outdated algorithm with no known theoretical properties. The updated algorithm computes pointwise approximate marginals that satisfy the same rate of convergence as the original approximate marginal likelihood. This function takes a fitted aghq object and recomputes its marginals using the updated algorithm

## Usage

```
correct_marginals(quad, ...)
```

## Arguments

| | |
|---|---|
| quad | An object of class aghq returned by `aghq::aghq()`. |
| ... | Not used |

## Value

An object of class aghq equal to the provided object, but with its `marginals` component replaced with one calculated using `method='correct'`. See `marginal_posterior`.

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.
- method_summaries: default 'reuse', method to use to compute moments and marginals. Choosing 'correct' corresponds to the approximations suggested in the *Stochastic Convergence...* paper, which attain the same rate of convergence as the approximation to the marginal likelihood. See ?compute_moment.

## 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.
- method_summaries: default 'reuse', method to use to compute moments and marginals. Choosing 'correct' corresponds to the approximations suggested in the *Stochastic Convergence...* paper, which attain the same rate of convergence as the approximation to the marginal likelihood. See ?compute_moment.

## 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.
- method_summaries: default 'reuse', method to use to compute moments and marginals. Choosing 'correct' corresponds to the approximations suggested in the *Stochastic Convergence...* paper, which attain the same rate of convergence as the approximation to the marginal likelihood. See ?compute_moment.

## Value

A list of argument values.

## Examples

```
default_control_tmb()
default_control_tmb(method = "trust")
```

## default_transformation *Default transformation*

### Description

Default (identity) transformation object. Default argument in package functions which accept transformations, and useful for user inspection.

### Usage

```
default_transformation()
```

### See Also

Other transformations: `make_transformation()`, `validate_transformation()`

### Examples

```
default_transformation()
```

## 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_hessian *Obtain the Hessian from an aghq object*

### Description

Quick helper method to retrieve the Hessian from an aghq object. Just calls `aghq::get_opt_results`.

### Usage

```
get_hessian(obj, ...)
```

### Arguments

| | |
|---|---|
| `obj` | Object of class `aghq` returned by `aghq::aghq`. |
| `...` | Not used |

### Value

A numeric matrix of dimension `dim(theta)` x `dim(theta)` containing the negative Hessian of the log-posterior evaluated at the mode.

## See Also

Other quadrature: aghq(), get_log_normconst(), get_mode(), get_nodesandweights(), get_numquadpoints(), get_opt_results(), get_param_dim(), laplace_approximation(), marginal_laplace_tmb(), marginal_laplace(), nested_quadrature(), normalize_logpost(), optimize_theta(), plot.aghq(), print.aghqsummary(), print.aghq(), print.laplacesummary(), print.laplace(), print.marginallaplacesummary summary.aghq(), summary.laplace(), summary.marginallaplace()

---

# 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(), get_hessian(), get_mode(), get_nodesandweights(), get_numquadpoints(), get_opt_results(), get_param_dim(), laplace_approximation(), marginal_laplace_tmb(), marginal_laplace(), nested_quadrature(), normalize_logpost(), optimize_theta(), plot.aghq(), print.aghqsummary(), print.aghq(), print.laplacesummary(), print.laplace(), print.marginallaplacesummary summary.aghq(), summary.laplace(), summary.marginallaplace()

---

`get_mode` *Obtain the mode from an aghq object*

---

## Description

Quick helper method to retrieve the mode from an aghq object. Just calls `aghq::get_opt_results`.

## Usage

```
get_mode(obj, ...)
```

## Arguments

| | |
|---|---|
| `obj` | Object of class aghq returned by `aghq::aghq`. |
| `...` | Not used |

## Value

A numeric vector of length `dim(theta)` containing the posterior mode.

## See Also

Other quadrature: aghq(), get_hessian(), get_log_normconst(), get_nodesandweights(), get_numquadpoints(), get_opt_results(), get_param_dim(), laplace_approximation(), marginal_laplace_tmb() marginal_laplace(), nested_quadrature(), normalize_logpost(), optimize_theta(), plot.aghq(), print.aghqsummary(), print.aghq(), print.laplacesummary(), print.laplace(), print.marginallaplacesummary summary.aghq(), summary.laplace(), summary.marginallaplace()

get_nodesandweights *Obtain the nodes and weights table from a fitted quadrature object*

## Description

Quick helper S3 method to retrieve the quadrature nodes and weights 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_nodesandweights(obj, ...)

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

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

## S3 method for class 'data.frame'
get_nodesandweights(obj, ...)

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

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

## S3 method for class 'marginallaplace'
get_nodesandweights(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()`, `get_hessian()`, `get_log_normconst()`, `get_mode()`, `get_numquadpoints()`, `get_opt_results()`, `get_param_dim()`, `laplace_approximation()`, `marginal_laplace_tmb()`, `marginal_laplace()`, `nested_quadrature()`, `normalize_logpost()`, `optimize_theta()`, `plot.aghq()`, `print.aghqsummary()`, `print.aghq()`, `print.laplacesummary()`, `print.laplace()`, `print.marginallaplacesummary`
`summary.aghq()`, `summary.laplace()`, `summary.marginallaplace()`

## get_numquadpoints *Obtain the number of quadrature nodes used from an aghq object*

### Description

Quick helper S3 method to retrieve the number of quadrature points used when creating an aghq object.

### Usage

```
get_numquadpoints(obj, ...)
```

### Arguments

| | |
|---|---|
| obj | Object of class aghq returned by aghq::aghq. |
| ... | Not used |

### Value

A numeric vector of length 1 containing k, the number of quadrature points used.

### See Also

Other quadrature: aghq(), get_hessian(), get_log_normconst(), get_mode(), get_nodesandweights(), get_opt_results(), get_param_dim(), laplace_approximation(), marginal_laplace_tmb(), marginal_laplace(), nested_quadrature(), normalize_logpost(), optimize_theta(), plot.aghq(), print.aghqsummary(), print.aghq(), print.laplacesummary(), print.laplace(), print.marginallaplacesummary summary.aghq(), summary.laplace(), summary.marginallaplace()

## get_opt_results *Obtain the optimization results from an aghq object*

### Description

Quick helper S3 method to retrieve the mode and Hessian from an aghq object. The full results of calling aghq::optimize_theta are stored in obj$optresults.

### Usage

```
get_opt_results(obj, ...)

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

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

### Arguments

| | |
|---|---|
| obj | Object of class aghq returned by aghq::aghq. |
| ... | Not used |

### Value

A named list with elements:

- mode: a numeric vector of length dim(theta) containing the posterior mode.
- hessian: a numeric matrix of dimension dim(theta) x dim(theta) containing the negative Hessian of the log-posterior evaluated at the mode.

For objects of class marginallaplace, a third list item modesandhessians is a data.frame containing the mode and Hessian of the W parameters evaluated at each adapted quadrature point.

### See Also

Other quadrature: aghq(), get_hessian(), get_log_normconst(), get_mode(), get_nodesandweights(), get_numquadpoints(), get_param_dim(), laplace_approximation(), marginal_laplace_tmb(), marginal_laplace(), nested_quadrature(), normalize_logpost(), optimize_theta(), plot.aghq(), print.aghqsummary(), print.aghq(), print.laplacesummary(), print.laplace(), print.marginallaplacesummary summary.aghq(), summary.laplace(), summary.marginallaplace()

---

## get_param_dim *Obtain the parameter dimension from an aghq object*

---

### Description

Quick helper S3 method to retrieve the parameter dimension from an aghq object.

### Usage

```
get_param_dim(obj, ...)

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

### Arguments

| | |
|---|---|
| obj | Object of class aghq returned by aghq::aghq. |
| ... | Not used |

### Value

A numeric vector of length 1 containing p, the parameter dimension.

## See Also

Other quadrature: aghq(), get_hessian(), get_log_normconst(), get_mode(), get_nodesandweights(), get_numquadpoints(), get_opt_results(), laplace_approximation(), marginal_laplace_tmb(), marginal_laplace(), nested_quadrature(), 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_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:<br>• `fn`: function taking argument `theta` and returning a numeric value representing the log-posterior at `theta`<br>• `gr`: function taking argument `theta` and returning a numeric vector representing the gradient of the log-posterior at `theta`<br>• `he`: function taking argument `theta` and returning a numeric matrix representing the hessian of the log-posterior at `theta`<br>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<br>• `method`: optimization method to use:<br>– '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_hessian()`, `get_log_normconst()`, `get_mode()`, `get_nodesandweights()`, `get_numquadpoints()`, `get_opt_results()`, `get_param_dim()`, `marginal_laplace_tmb()`, `marginal_laplace()`, `nested_quadrature()`, `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))
```

---

`make_moment_function` *Moments of Positive Functions*

---

## Description

Given an object quad of class aghq returned by `aghq::aghq()`, `aghq::compute_moment()` will compute the moment of a positive function `g(theta)` of parameter `theta`. The present function, `aghq::make_moment_function()`, assists the user in constructing the appropriate input to `aghq::compute_moment()`.

## Usage

```
make_moment_function(...)

## S3 method for class 'aghqmoment'
make_moment_function(gg, ...)

## S3 method for class 'aghqtrans'
make_moment_function(gg, ...)

## S3 method for class '`function`'
make_moment_function(gg, ...)

## S3 method for class 'character'
make_moment_function(gg, ...)

## S3 method for class 'list'
make_moment_function(gg, ...)

## Default S3 method:
make_moment_function(gg, ...)
```

## Arguments

| | |
|---|---|
| `...` | Used to pass arguments to methods. |
| `gg` | LOGARITHM of function `R^p -> R^+` of which the moment is to be computed along with its two derivatives. So for example providing gg = function(x) x will compute the moment of exp(x). Provided either as a `function`, a `list`, an `aghqtrans` object, or an `aghqmoment` object. See details. |

## Details

The approximation of moments of positive functions implemented in `aghq::compute_moment()` achieves the same asymptotic rate of convergence as the marginal likelihood. This involves computing a new mode and Hessian depending on the original posterior mode and Hessian, and `g`. These computations are handled by `aghq::compute_moment()`, re-using information from the original quadrature when feasible.

Computation of moments is defined only for scalar-valued functions, with a vector moment just defined as a vector of moments. Consequently, the user may input to `aghq:compute_moment()` a function `g: R^p -> R^q+` for any `q`, and that function will return the corresponding vector of moments. This is handled within `aghq::compute_moment()`. The `aghq::make_moment_function()` interface accepts the logarithm of `gg: R^p -> R^+`, i.e. a multivariable, scalar-valued positive function. This is mostly to keep first and second derivatives as 1d and 2d arrays (i.e. the gradient and the Hessian); I deemed it too confusing for the user and the code-base to work with Jacobians and 2nd derivative tensors (if you're confused just reading this, there you go!). But, see `aghq::compute_moment()` for how to handle the very common case where the *same* transformation is desired of all parameter coordinates; for example when all parameters are on the log-scale and you want to compute `E(exp(theta))` for *vector* `theta`.

If `gg` is a `function` or a `character` (like `'exp'`) it is first passed to `match.fun`, and then the output object is constructed using `numDeriv::grad()` and `numDeriv::hessian()`. If `gg` is a `list` then it is assumed to have elements `fn`, `gr`, and `he` of the correct form, and these elements are extracted and then passed back to `make_moment_function()`. If `gg` is an object of class `aghqtrans` returned by `aghq::make_transformation()`, then `gg$fromtheta` is passed back to `make_moment_function()`. If `gg` is an object of class `aghqtrans` then it is just returned.

## Value

Object of class `aghqmoment`, which is a list with elements `fn`, `gr`, and `he`, exactly like the input to `aghq::aghq()` and related functions. Here `gg$fn` is `log(gg(theta))`, `gg$gr` is its gradient, and `gg$he` its Hessian. Object is suitable for checking with `aghq::validate_moment()` and for inputting into `aghq::compute_moment()`.

## See Also

Other moments: `validate_moment()`

## Examples

```
# E(exp(x))
mom1 <- make_moment_function(force) # force = function(x) x
mom2 <- make_moment_function('force')
mom3 <- make_moment_function(list(fn=function(x) x,gr=function(x) 1,he = function(x) 0))
```

## make_numeric_moment_function *Compute numeric moments*

### Description

Create a function suitable for computation of numeric moments. This function is used internally by `compute_moment` when the user chooses `nn`, and is unlikely to need to be called by a user directly.

### Usage

```
make_numeric_moment_function(nn, j, quad = NULL, centre = 0, shift = NULL, ...)

get_shift(gg)
```

### Arguments

| | |
|---|---|
| `nn` | Order of moment to be computed, see `nn` argument of `compute_moment`. |
| `j` | Numeric, positive integer, index of parameter vector to compute the numeric moment for. |
| `quad` | Optional, object of class `aghq`, only used if `shift` is not `NULL`. |
| `centre` | Numeric scalar, added to `shift` to ensure that central moments remain far from zero. |
| `shift` | Numeric scalar, amount by which to shift `theta`. The function that this outputs is `g(theta) = (theta)^nn + shift`, and `shift` is returned with the object so that it may later be subtracted. Default of `NULL` chooses this value internally. |
| `...` | Not used. |
| `gg` | Object of class `aghqmoment`. Returns the `shift` applied to the moment function. Returns `0` if no shift applied. |

### Value

Object of class `aghqmoment`, see `make_moment_function`

## make_transformation *Marginal Parameter Transformations*

### Description

These functions make it easier for the user to represent marginal parameter transformations for which inferences are to be made. Suppose quadrature is done on the posterior for parameter `theta`, but interest lies in parameter `lambda = g(theta)` for smooth, monotone, univariate `g`. This interface lets the user provide `g`, `g^-1`, and (optionally) the Jacobian `dtheta/dlambda`, and `aghq` will do quadrature on the `theta` scale but report summaries on the `lambda` scale. See a note in the Details below about multidimensional parameters.

## Usage

```
make_transformation(...)

## S3 method for class 'aghqtrans'
make_transformation(transobj, ...)

## S3 method for class 'list'
make_transformation(translist, ...)

## Default S3 method:
make_transformation(totheta, fromtheta, jacobian = NULL, ...)
```

## Arguments

| | |
|---|---|
| ... | Used to pass arguments to methods. |
| transobj | An object of class aghqtrans. Just returns this object. This is for internal compatibility. |
| translist | A list with elements totheta, fromtheta, and, optionally, jacobian. |
| totheta | Inverse function g^-1(theta). Specifically, takes vector g_1(theta_1)...g_p(theta_p) and returns vector theta_1...theta_p. |
| fromtheta | Function g: R^p -> R^p, where p = dim(theta). Must take vector theta_1...theta_p and return vector g_1(theta_1)...g_p(theta_p), i.e. only independent/marginal transformations are allowed (but these are the only ones of interest, see below). For j=1...p, the parameter of inferential interest is lambda_j = g_j(theta_j) and the parameter whose posterior is being normalized via aghq is theta_j. Passed to match.fun. |
| jacobian | (optional) Function taking theta and returning the absolute value of the determinant of the Jacobian dtheta/dg(theta). If not provided, a numerically differentiated Jacobian is used as follows: numDeriv::jacobian(totheta,fromtheta(theta)). Passed to match.fun. |

## Details

Often, the scale on which quadrature is done is not the scale on which the user wishes to make inferences. For example, when a parameter lambda>0 is of interest, the posterior for theta = log(lambda) may be better approximated by a log-quadratic than that for lambda, so running aghq on the likelihood and prior for theta may lead to faster and more stable optimization as well as more accurate estimates. But, interest is still in the original parameter lambda = exp(theta).

These considerations are by no means unique to the use of quadrature-based approximate Bayesian inferences. However, when using (say) MCMC, inferences for summaries of transformations of the parameter are just as easy as for the un-transformed parameter. When using quadrature, a little bit more work is needed.

The aghq package provides an interface for computing posterior summaries of smooth, monotonic parameter transformations. If quadrature is done on parameter theta and g(theta) is a univariate, smooth, monotone function, then inferences are made for lambda = g(theta). In the case that theta is p-dimensional, p > 1, the supplied function g is understood to take in theta_1...theta_p and return g_1(theta_1)...g_p(theta_p). The Jacobian is diagonal.

To reiterate, all of this discussion applies only to *marginal* parameter transformations. For the full joint parameter, the only summary statistics you can even calculate at all (at present?) are moments, and you can already calculate the moment of any function `h(theta)` using `aghq::compute_moment`, so no additional interface is needed here.

## Value

Object of class `aghqtrans`, which is simply a list with elements `totheta`, `fromtheta`, and `jacobian`. Object is suitable for checking with `aghq::validate_transformation` and for inputting into any function in `aghq` which takes a `transformation` argument.

## See Also

Other transformations: `default_transformation()`, `validate_transformation()`

## Examples

```
make_transformation('log','exp')
make_transformation(log,exp)
```

---

`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,
  transformation = default_transformation(),
  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:<br>• fn: function taking arguments W and theta and returning a numeric value representing the log-joint posterior at W, theta<br>• 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<br>• 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. |
| transformation | Optional. Do the quadrature for parameter theta, but return summaries and plots for parameter g(theta). This applies to the theta parameters only, not the W parameters. transformation is either: a) an aghqtrans object returned by aghq::make_transformation, or b) a list that will be passed to that function internally. See ?aghq::make_transformation for details. |
| 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<br>• method: optimization method to use for the theta optimization:<br>– 'sparse_trust' (default): trustOptim::trust.optim<br>– 'sparse': trust::trust<br>– 'BFGS': optim(...,method = "BFGS")<br>• inner_method: optimization method to use for the W optimization; same options as for method<br>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_hessian(), get_log_normconst(), get_mode(), get_nodesandweights(), get_numquadpoints(), get_opt_results(), get_param_dim(), laplace_approximation(), marginal_laplace_tmb(), nested_quadrature(), 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,
  transformation = default_transformation(),
  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. |
| transformation | Optional. Do the quadrature for parameter theta, but return summaries and plots for parameter g(theta). This applies to the theta parameters only, not the W parameters. transformation is either: a) an aghqtrans object returned by aghq::make_transformation, or b) a list that will be passed to that function internally. See ?aghq::make_transformation for details. |
| 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),'GH`
**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_hessian()`, `get_log_normconst()`, `get_mode()`, `get_nodesandweights()`, `get_numquadpoints()`, `get_opt_results()`, `get_param_dim()`, `laplace_approximation()`, `marginal_laplace()`, `nested_quadrature()`, `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(...)

## S3 method for class 'aghq'
marginal_posterior(
  quad,
  j,
  qq = NULL,
  method = c("auto", "reuse", "correct"),
  ...
)

## S3 method for class 'list'
marginal_posterior(
  optresults,
  k,
  j,
  basegrid = NULL,
  ndConstruction = "product",
  ...
)
```

## Arguments

| | |
|---|---|
| `...` | Additional arguments to be passed to `optresults$ff`, see `?optimize_theta`. |
| `quad` | Object returned by `aghq::aghq`. |
| `j` | Integer between 1 and the dimension of the parameter space. Which index of the parameter vector to compute the marginal posterior for. |
| `qq` | Numeric vector of length `>=1` giving the points at which to evaluate the marginal posterior. The default, `NULL`, chooses these points in a 'clever' way, see Details. |
| `method` | Method for computing the quadrature points used to approximate moment. One of 'reuse' (default) or 'correct'. See details. The default SHOULD be 'correct'; it is currently set to 'reuse' to maintain compatibility of results with previous versions. This will be switched in a future major release. |
| `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. |
| 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),'GH |
| 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. |

## Details

If qq=NULL, then it is set to the unique values in an adapted GHQ grid computed assuming that j=1 (there is nothing special about this procedure, it's just a way to provide an apparently sensible default).

If method='reuse', then the parameter vector is reordered so j=1, and the approximate marginal is computed by first computing the whole AGHQ grid, then summing over the other indices. This is an outdated method that does not have any theory pertaining to it, and is included for backwards compatibility. It does not use qq if supplied.

If method='correct' then the theoretically-justified approximation from Section 2.4 of the 'Stochastic Convergence Rates...' paper is returned.

method='auto' currently chooses 'reuse' for backwards compatibility, but this will be changed in a future release.

## 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_pdf_and_cdf(), compute_quantiles(), interpolate_marginal_posterior()

## 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)
```

---

`nested_quadrature` *Nested, sparse Gaussian quadrature in AGHQ*

---

## Description

Compute a whole sequence of log normalizing constants for `1,3,5,...,k` points, using only the function evaluations from the `k`-point nested rule.

## Usage

```
nested_quadrature(optresults, k, ndConstruction = "product", ...)

adaptive_nested_quadrature(optresults, k, ndConstruction = "product", ...)

get_quadtable(p, k, ndConstruction = "product", ...)
```

## Arguments

| | |
|---|---|
| `optresults` | The results of calling `aghq::optimize_theta()`: see return value of that function. The dimension of the parameter `p` will be taken from `optresults$mode`. |
| `k` | Integer, the number of quadrature points to use. |
| `ndConstruction` | Create a multivariate grid using a product or sparse construction? Passed directly to `mvQuad::createNIGrid()`, see that function for further details. |
| `...` | Additional arguments to be passed to `optresults$ff`, see `?optimize_theta`. |
| `p` | Dimension of the variable of integration. |

## Details

get_quadtable currently uses mvQuad to compute the nodes and weights. This will be replaced by a manual reading of the pre-tabulated nodes and weights.

nested_quadrature and adaptive_nested_quadrature take the **log** function values, just like optimize_theta(), and return the **log** of the base/adapted quadrature rule.

## Value

For get_quadtable, a pre-computed table of nodes for the k-point rule, with weights for the points from each of the 1, 3, . . . , k-point rules, for passing to nested_quadrature. For nested_quadrature and adaptive_nested_quadrature, a named numeric vector of optresults$fn values for each k.

## See Also

Other quadrature: aghq(), get_hessian(), get_log_normconst(), get_mode(), get_nodesandweights(), get_numquadpoints(), get_opt_results(), get_param_dim(), 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

```
# 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)
```

---

# 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),'GH |
| 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.

## See Also

Other quadrature: aghq(), get_hessian(), get_log_normconst(), get_mode(), get_nodesandweights(), get_numquadpoints(), get_opt_results(), get_param_dim(), laplace_approximation(), marginal_laplace_tmb(), marginal_laplace(), nested_quadrature(), 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:<br>• fn: function taking argument theta and returning a numeric value representing the log-posterior at theta<br>• gr: function taking argument theta and returning a numeric vector representing the gradient of the log-posterior at theta<br>• he: function taking argument theta and returning a numeric matrix representing the hessian of the log-posterior at theta<br>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$he(startingvalue) must all return appropriate values without error. |
| control | A list with elements<br>• method: optimization method to use:<br>– 'sparse_trust' (default): trustOptim::trust.optim with method = 'sparse'<br>– 'SR1' (default): trustOptim::trust.optim with method = 'SR1'<br>– 'trust': trust::trust<br>– 'BFGS': optim(...,method = "BFGS")<br>Default is 'sparse_trust'.<br>• optcontrol: optional: a list of control parameters to pass to the internal optimizer you chose. The aghq package uses sensible defaults. |
| ... | Additional arguments to be passed to ff$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_hessian(), get_log_normconst(), get_mode(), get_nodesandweights(), get_numquadpoints(), get_opt_results(), get_param_dim(), laplace_approximation(), marginal_laplace_tmb(), marginal_laplace(), nested_quadrature(), 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 of the transformed parameter from an aghq object.

## Usage

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

## Arguments

| | |
|---|---|
| `x` | The return value of `aghq::aghq`. Plots are created for the marginal pdf and cdf of `x$transformation$fromtheta(theta)`. |
| `...` | not used. |

## Value

Silently plots.

## See Also

Other quadrature: `aghq()`, `get_hessian()`, `get_log_normconst()`, `get_mode()`, `get_nodesandweights()`, `get_numquadpoints()`, `get_opt_results()`, `get_param_dim()`, `laplace_approximation()`, `marginal_laplace_tmb()`, `marginal_laplace()`, `nested_quadrature()`, `normalize_logpost()`, `optimize_theta()`, `print.aghqsummary()`,

print.aghq(), print.laplacesummary(), print.laplace(), print.marginallaplacesummary(), summary.aghq(), summary.laplace(), summary.marginallaplace()

Other quadrature: aghq(), get_hessian(), get_log_normconst(), get_mode(), get_nodesandweights(), get_numquadpoints(), get_opt_results(), get_param_dim(), laplace_approximation(), marginal_laplace_tmb(), marginal_laplace(), nested_quadrature(), 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_hessian(), get_log_normconst(), get_mode(), get_nodesandweights(), get_numquadpoints(), get_opt_results(), get_param_dim(), laplace_approximation(), marginal_laplace_tmb(), marginal_laplace(), nested_quadrature(), 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_hessian()`, `get_log_normconst()`, `get_mode()`, `get_nodesandweights()`, `get_numquadpoints()`, `get_opt_results()`, `get_param_dim()`, `laplace_approximation()`, `marginal_laplace_tmb()`, `marginal_laplace()`, `nested_quadrature()`, `normalize_logpost()`, `optimize_theta()`, `plot.aghq()`, `print.aghq()`, `print.laplacesummary()`, `print.laplace()`, `print.marginallaplacesummary()`, `summary.aghq()`, `summary.laplace()`, `summary.marginallaplace()`

Other quadrature: `aghq()`, `get_hessian()`, `get_log_normconst()`, `get_mode()`, `get_nodesandweights()`, `get_numquadpoints()`, `get_opt_results()`, `get_param_dim()`, `laplace_approximation()`, `marginal_laplace_tmb()`, `marginal_laplace()`, `nested_quadrature()`, `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_hessian()`, `get_log_normconst()`, `get_mode()`, `get_nodesandweights()`, `get_numquadpoints()`, `get_opt_results()`, `get_param_dim()`, `laplace_approximation()`, `marginal_laplace_tmb()`, `marginal_laplace()`, `nested_quadrature()`, `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_hessian(), get_log_normconst(), get_mode(), get_nodesandweights(), get_numquadpoints(), get_opt_results(), get_param_dim(), laplace_approximation(), marginal_laplace_tmb(), marginal_laplace(), nested_quadrature(), normalize_logpost(), optimize_theta(), plot.aghq(), print.aghqsummary(), print.aghq(), print.laplace(), print.marginallaplacesummary(), summary.aghq(), summary.laplace(), summary.marginallaplace()

Other quadrature: aghq(), get_hessian(), get_log_normconst(), get_mode(), get_nodesandweights(), get_numquadpoints(), get_opt_results(), get_param_dim(), laplace_approximation(), marginal_laplace_tmb(), marginal_laplace(), nested_quadrature(), 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_hessian()`, `get_log_normconst()`, `get_mode()`, `get_nodesandweights()`, `get_numquadpoints()`, `get_opt_results()`, `get_param_dim()`, `laplace_approximation()`, `marginal_laplace_tmb()`, `marginal_laplace()`, `nested_quadrature()`, `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,
  M,
  transformation = default_transformation(),
  interpolation = "auto",
  ...
)

## S3 method for class 'aghq'
sample_marginal(
  quad,
  M,
```

```
  transformation = quad$transformation,
  interpolation = "auto",
  ...
)

## S3 method for class 'marginallaplace'
sample_marginal(
  quad,
  M,
  transformation = quad$transformation,
  interpolation = "auto",
  ...
)
```

## 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. |
| M | Numeric, integer saying how many samples to draw |
| transformation | Optional. Draw samples for a transformation of the parameter whose posterior was normalized using adaptive quadrature. transformation is either: a) an aghqtrans object returned by aghq::make_transformation, or b) a list that will be passed to that function internally. See ?aghq::make_transformation for details. |
| 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(). |
| ... | Used to pass additional arguments. |

## 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 ff$env$last.par, for example (where ff is your template obtained from a call to TMB::MakeADFun.

If getOption('mc.cores',1L) > 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.

## 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 of the transformed parameter. 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`. Summaries are computed for `object$transformation$fromtheta(` |
| `...` | 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 triangle 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 transformed parameter, computed according to the posterior normalized using AGHQ

## See Also

Other quadrature: aghq(), get_hessian(), get_log_normconst(), get_mode(), get_nodesandweights(), get_numquadpoints(), get_opt_results(), get_param_dim(), laplace_approximation(), marginal_laplace_tmb(), marginal_laplace(), nested_quadrature(), 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_hessian(), get_log_normconst(), get_mode(), get_nodesandweights(), get_numquadpoints(), get_opt_results(), get_param_dim(), laplace_approximation(), marginal_laplace_tmb(), marginal_laplace(), nested_quadrature(), normalize_logpost(), optimize_theta(), plot.aghq(), print.aghqsummary(), print.aghq(), print.laplacesummary(), print.laplace(), print.marginallaplacesummary, summary.aghq(), summary.marginallaplace()

Other quadrature: aghq(), get_hessian(), get_log_normconst(), get_mode(), get_nodesandweights(), get_numquadpoints(), get_opt_results(), get_param_dim(), laplace_approximation(), marginal_laplace_tmb(), marginal_laplace(), nested_quadrature(), 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`).

## See Also

Other quadrature: aghq(), get_hessian(), get_log_normconst(), get_mode(), get_nodesandweights(), get_numquadpoints(), get_opt_results(), get_param_dim(), laplace_approximation(), marginal_laplace_tmb(), marginal_laplace(), nested_quadrature(), 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")
```

validate_moment *Validate a moment function object*

## Description

Routine for checking whether a given moment function object is valid.

## Usage

```
validate_moment(...)

## S3 method for class 'aghqmoment'
validate_moment(moment, checkpositive = FALSE, ...)

## S3 method for class 'list'
validate_moment(moment, checkpositive = FALSE, ...)

## S3 method for class '`function`'
validate_moment(moment, checkpositive = FALSE, ...)

## S3 method for class 'character'
validate_moment(moment, checkpositive = FALSE, ...)

## Default S3 method:
validate_moment(moment, ...)
```

## Arguments

| | |
|---|---|
| `...` | Used to pass arguments to methods. |
| `moment` | An object to check if it is a valid moment function or not. Can be an object of class `aghqmoment` returned by `aghq::make_moment_function()`, or any object that can be passed to `aghq::make_moment_function()`. |
| `checkpositive` | Default `FALSE`, do not check that `gg$fn(theta) > 0`. Otherwise, a vector of values for which to perform that check. No default values are provided, since `validate_moment` has no way of determining the domain and range of `gg$fn`. This argument is used internally in `aghq` package functions, with cleverly chosen check values. |

## Details

This function checks that:

- The supplied object contains elements `fn`, `gr`, and `he`, and that they are all functions,
- If `checkpositive` is a vector of numbers, then it checks that `gg$fn(checkpositive)` is not `-Inf`, `NA`, or `NaN`. (It actually uses `is.infinite` for the first.)

In addition, if a `list` is provided, the function first checks that it contains the right elements, then passes it to `make_moment_function`, then checks that. If a `function` or a `character` is provided, it checks that `match.fun` works, and returns any errors or warnings from doing so in a clear way.

This function throws an informative error messages when checks don't pass or themselves throw errors.

## Value

`TRUE` if the function runs to completion without throwing an error.

## See Also

Other moments: `make_moment_function()`

## Examples

```
mom1 <- make_moment_function(exp)
mom2 <- make_moment_function('exp')
mom3 <- make_moment_function(list(fn=function(x) x,gr=function(x) 1,he = function(x) 0))
validate_moment(mom1)
validate_moment(mom2)
validate_moment(mom3)
## Not run:
mombad1 <- list(exp,exp,exp) # No names
mombad2 <- list('exp','exp','exp') # List of not functions
mombad3 <- make_moment_function(function(x) -exp(x)) # Not positive
validate_moment(mombad1)
validate_moment(mombad2)
validate_moment(mombad3)

## End(Not run)
```

---

# validate_transformation

*Validate a transformation object*

---

## Description

Routine for checking whether a given transformation is valid.

## Usage

```
validate_transformation(...)

## S3 method for class 'aghqtrans'
validate_transformation(trans, checkinverse = FALSE, ...)

## S3 method for class 'list'
```

```
validate_transformation(translist, checkinverse = FALSE, ...)

## Default S3 method:
validate_transformation(...)
```

## Arguments

| | |
|---|---|
| ... | Used to pass arguments to methods. |
| trans | A transformation object of class aghqtrans returned by make_transformation. |
| checkinverse | Default FALSE, do not check that totheta(fromtheta(theta)) = theta. Otherwise, a vector of values for which to perform that check. No default values are provided, since validate_transformation has no way of determining the domain and range of totheta and fromtheta. This argument is used internally in aghq package functions, with cleverly chosen check values. |
| translist | A list. Will be checked, passed to aghqtrans, and then checked again. |

## Details

This function checks that:

- The supplied object contains elements totheta, fromtheta, and jacobian, and that they are all functions,
- If checkinverse is a vector of numbers, then it checks that totheta(fromtheta(checkinverse)) == checkinverse.

In addition, if a list is provided, the function first checks that it contains the right elements, then passes it to make_transformation, then checks that.

This function throws an informative error messages when checks don't pass or themselves throw errors.

## Value

TRUE if the function runs to completion without throwing an error.

## See Also

Other transformations: default_transformation(), make_transformation()

## Examples

```
t <- make_transformation(log,exp)
validate_transformation(t)
t2 <- list(totheta = log,fromtheta = exp)
validate_transformation(t2)
## Not run:
t3 <- make_transformation(log,log)
checkvals <- exp(exp(rnorm(10)))
# Should throw an informative error because log isn't the inverse of log.
validate_transformation(t3,checkinverse = checkvals)

## End(Not run)
```

# Index