Package 'acebayes'

September 13, 2018

Type Package

Title Optimal Bayesian Experimental Design using the ACE Algorithm

Version 1.5.1

Date 2018-09-13

Author Antony M. Overstall, David C. Woods & Maria Adamou

Maintainer Antony M. Overstall <A.M.Overstall@soton.ac.uk>

Description Optimal Bayesian experimental design using the approximate coordinate exchange (ACE) algorithm.

License GPL-2

Depends R (>= 3.0.2), lhs

Imports Rcpp (>= 0.12.2), compare, randtoolbox, parallel

LinkingTo Rcpp, RcppArmadillo (>= 0.6.400.2.2)

Suggests R.rsp

VignetteBuilder R.rsp

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-09-13 10:50:03 UTC

R topics documented:

acebayes-package *Optimal Bayesian Experimental Design using the Approximate Coordinate Exchange (ACE) Algorithm*

Description

Finding an optimal Bayesian experimental design (Chaloner & Verdinelli, 1995) involves maximising an objective function given by the expectation of some appropriately chosen utility function with respect to the joint distribution of unknown quantities (including responses). This objective function is usually not available in closed form and the design space can be continuous and of high dimensionality.

The acebayes package uses Approximate Coordinate Exchange (ACE; Overstall & Woods, 2017) to maximise an approximation to the expectation of the utility function. In Phase I of the algorithm, a continuous version of the coordinate exchange algorithm (Meyer & Nachtsheim, 1995) is used to maximise an approximation to expected utility. The approximation is given by the predictive mean of a Gaussian process (GP) emulator constructing using a 'small' number of approximate evaluations of the expected utility function. In Phase II a point exchange algorithm is used to consolidate clusters of design points into repeated design points.

Details

Package:	acebayes
Type:	Package
Version:	1.5.1
Date:	2018-09-13
License:	GPL-2

The most important functions are as follows.

1. ace
2. pace
3. aceglm
4. paceglm
5. acenlm
6. pacenlm

The function ace implements both phases of the ACE algorithm. It has two mandatory arguments: utility (a function specifying the chosen utility function incorporating the joint distribution of unknown quantities) and start.d (the initial design). The function will return the final design from the algorithm, along with information to assess convergence. The function link{pace} implements repetitions of the ACE algorithm from different starting designs (as specified by the start.d argument).

The computational time of ace (and pace) is highly dependent on the computational time required to evaluate the user-supplied function utility. Therefore it is recommended that users take advantage of R packages such as Rcpp (Eddelbuettel & Francois, 2011), RcppArmadillo (Eddelbuettel & Sanderson, 2014), or RcppEigen (Bates & Eddelbuettel, 2013), that provide convenient interfaces to compiled programming languages.

The functions aceglm and acenlm are user-friendly wrapper functions for ace which use the ACE algorithm to find Bayesian optimal experimental designs for generalised linear models and non-linear models, respectively. As special cases, both of these functions can find pseudo-Bayesian optimal designs. The functions paceglm and pacenlm implement repetitions of the ACE algorithm from different starting designs (as specified by the start.d argument) for generalised linear models and non-linear models, respectively.

For more details on the underpinning methodology, see Overstall & Woods (2017).

Author(s)

Antony M. Overstall, David C. Woods & Maria Adamou

Maintainer: Antony. M.Overstall <A.M.Overstall@soton.ac.uk>

References

Bates, D. & Eddelbuettel, D. (2013). Fast and Elegant Numerical Linear Algebra Using the RcppEigen Package. *Journal of Statistical Software*, **52(5)**, 1-24. http://www.jstatsoft.org/v52/i05/

Chaloner, K. & Verdinelli, I. (1995). Bayesian Experimental Design: A Review. *Statistical Science*, **10**, 273-304.

Eddelbuettel, D. & Francois, R. (2011). Rcpp: Seamless R and C++ Integration. *Journal of Statistical Software*, **40(8)**, 1-18. http://www.jstatsoft.org/v40/i08/

Eddelbuettel, D. & Sanderson, C. (2014). RcppArmadillo: Accelerating R with high-performance C++ linear algebra. *Computational Statistics and Data Analysis*, **71**, 1054-1063.

Meyer, R. & Nachtsheim, C. (1995). The Coordinate Exchange Algorithm for Constructing Exact Optimal Experimental Designs. *Technometrics*, **37**, 60-69.

Overstall, A.M. & Woods, D.C. (2017). Bayesian design of experiments using approximate coordinate exchange. *Technometrics*, **59**, 458-470.

Examples

```
## This example uses aceglm to find a pseudo-Bayesian D-optimal design for a
## first-order logistic regression model with 6 runs 4 factors (i.e. 5 parameters).
## The priors are those used by Overstall & Woods (2017), i.e. a uniform prior
## distribution is assumed for each parameter. The design space for each coordinate
## is [-1, 1].

set.seed(1)
## Set seed for reproducibility.

n<-6
## Specify the sample size (number of runs).

start.d<-matrix(2 * randomLHS(n = n,k = 4) - 1, nrow = n, ncol = 4,
```

```
dimnames = list(as.character(1:n), c("x1", "x2", "x3", "x4")))
## Generate an initial design of appropriate dimension. The initial design is a
## Latin hypercube sample.

low<-c(-3, 4, 5, -6, -2.5)
upp<-c(3, 10, 11, 0, 3.5)
## Lower and upper limits of the uniform prior distributions.

prior<-function(B){
t(t(6*matrix(runif(n = 5*B), ncol = 5)) + low)}
## Create a function which specifies the prior. This function will return a
## B by 5 matrix where each row gives a value generated from the prior
## distribution for the model parameters.

example<-aceglm(formula = ~ x1 + x2 + x3 + x4,  start.d = start.d, family = binomial,
prior = prior , criterion = "D", method= "MC", B = c(1000,1000), N1 = 1, N2 = 0,
upper = 1)
## Call the aceglm function which implements the ACE algorithm requesting
## only one iteration of Phase I and zero iterations of Phase II (chosen for
## illustrative purposes). The Monte Carlo sample size for the comparison
## procedure (B[1]) is set to 1000 (chosen again for illustrative purposes).

example
## Print out a short summary.

#Generalised Linear Model
#Criterion = Bayesian D-optimality
#
#Number of runs = 6
#
#Number of factors = 4
#
#Number of Phase I iterations = 1
#
#Number of Phase II iterations = 0
#
#Computer time = 00:00:02

## The final design found is:

example$phase2.d

#          x1          x2          x3         x4
#1 -0.3571245  0.16069337 -0.61325375  0.9276443
#2 -0.9167309  0.91411512  0.69842151  0.2605092
#3 -0.8843699  0.42863930 -1.00000000 -0.9679402
#4  0.3696224 -0.27126080  0.65284076  0.1850767
#5  0.7172267 -0.34743402 -0.05968457 -0.6588896
#6  0.7469636  0.05854029  1.00000000 -0.1742566
```

---

ace *Approximate Coordinate Exchange (ACE) Algorithm*

Description

These functions implement the approximate coordinate exchange (ACE) algorithm (Overstall & Woods, 2017) for finding optimal Bayesian experimental designs by maximising an approximation to an intractable expected utility function.

Usage

```
ace(utility, start.d, B, Q = 20, N1 = 20, N2 = 100, lower = -1, upper = 1,
limits = NULL, progress = FALSE, binary = FALSE, deterministic = FALSE)

acephase1(utility, start.d, B, Q = 20, N1 = 20, lower, upper, limits = NULL,
progress = FALSE, binary = FALSE, deterministic = FALSE)

acephase2(utility, start.d, B, N2 = 100, progress = FALSE, binary = FALSE,
deterministic = FALSE)

pace(utility, start.d, B, Q = 20, N1 = 20, N2 = 100, lower = -1, upper = 1,
limits = NULL, binary = FALSE, deterministic = FALSE, mc.cores = 1,
n.assess = 20)
```

Arguments

`utility`	A function with two arguments: `d` and `B`. For a Monte Carlo approximation (`deterministic = FALSE`), it should return a ***vector*** of length `B` where each element gives the value of the utility function for design `d`, for a value generated from the joint distribution of all unknown quantities. The mean of the elements of this vector provides a Monte Carlo approximation to the expected utility for design `d`. For a deterministic approximation (`deterministic = TRUE`), it should return a ***scalar*** giving the approximate value of the expected utility for design `d`. In this latter case, the argument `B` can be a list containing tuning parameters for the deterministic approximation. If `B` is not required, the utility function must still accept the argument.
`start.d`	For `ace`, `acephase1` and `acephase2`, an `n` by `k` matrix specifying the initial design for the ACE algorithm. For `pace`, a list with each element, an `n` by `k` matrix specifying the initial design for each repetition of the ACE algorithm.
`B`	An argument for controlling the approximation to the expected utility. For a Monte Carlo approximation (`deterministic = FALSE`), a vector of length two specifying the size of the Monte Carlo samples, generated from the joint distribution of unknown quantities. The first sample size, `B[1]`, gives the sample size to use in the comparison procedures, and the second sample size, `B[2]`, gives the sample size to use for the evaluations of Monte Carlo integration that are used to fit the Gaussian process emulator. If missing when `deterministic = FALSE`, the default value is `c(20000,1000)`. For a deterministic approximation (`deterministic = TRUE`), then `B` may be a list of length two containing any necessary tuning parameters for the expected utility calculations for the comparison and emulation steps.

Q	An integer specifying the number of evaluations of the approximate expected utility that are used to fit the Gaussian process emulator. The default value is 20.
N1	An integer specifying the number of iterations of Phase I of the ACE algorithm (the coordinate exchange phase). The default value is 20.
N2	An integer specifying the number of iterations of Phase II of the ACE algorithm (the point exchange phase). The default value is 100.
lower	An argument specifying the bounds on the design space. This argument can either be a scalar or a matrix of the same dimension as the argument start.d which specifies the lower limits of all coordinates of the design space. The default value is -1.
upper	An argument specifying the bounds on the design space. This argument can either be a scalar or a matrix of the same dimension as the argument start.d which specifies the upper limits of all coordinates of the design space. The default value is 1.
limits	An argument specifying the grid over which to maximise the Gaussian process emulator for the expected utility function. It should be a function with three arguments: i, j and d which generates a one-dimensional grid for the ijth coordinate of the design when the current design is d. The default value is NULL which generates values uniformly on the interval (lower[i,j],upper[i,j]) or (lower,upper) depending on whether the arguments lower and upper are matrices or scalars, respectively.
progress	A logical argument indicating whether the iteration number and other information detailing the progress of the algorithm should be printed. The default value is FALSE.
binary	A logical argument indicating whether the utility function has binary or continuous output. In some cases, the utility function is an indicator function of some event giving binary output. The expected utility function will then be the expected posterior probability of the event. Utility functions such as Shannon information gain and negative squared error loss give continuous output. The type of output guides the choice of comparison procedure used in the ACE algorithm. The default value is FALSE, indicating the utility function has continuous output.
deterministic	A logical argument indicating if a Monte Carlo (FALSE, default) or deterministic (TRUE) approximation to the expected utility is being used.
mc.cores	The number of cores to use, i.e. at most how many child processes will be run simultaneously. Must be at least one (the default), and parallelisation requires at least two cores. See mclapply for more information and warnings for mc.cores > 1.
n.assess	If deterministic = FALSE, the approximate expected utility for the design from each repetition of the ACE algorithm will be calculated n.assess times. The terminal design returned will be the design with the largest mean approximate expected utility calculated over the n.assess approximations.

Details

Finding an optimal Bayesian experimental design (Chaloner & Verdinelli, 1995) involves maximising an objective function given by the expectation of some appropriately chosen utility function

with respect to the joint distribution of unknown quantities (including responses). This objective function is usually not available in closed form and the design space can be continuous and of high dimensionality.

Overstall & Woods (2017) proposed the approximate coordinate exchange (ACE) algorithm to approximately maximise the expectation of the utility function. ACE consists of two phases.

Phase I uses a continuous version of the coordinate exchange algorithm (Meyer & Nachtsheim, 1995) to maximise an approximation to the expected utility. Very briefly, the approximate expected utility is sequentially maximised over each one-dimensional element of the design space. The approximate expected utility is given by the predictive mean of a Gaussian process (GP) regression model (also known as an emulator or surrogate) fitted to a 'small' number (argument `Q`) of evaluations of either a Monte Carlo (MC) or deterministic (e.g. quadrature) approximation to the expected utility (the MC sample size or arguments for the deterministic approximation are given by `B`). A GP emulator is a statistical model and, similar to all statistical models, can be an inadequate representation of the underlying process (i.e. the expected utility). Instead of automatically accepting the new design given by the value that maximises the GP emulator, for MC approximations a Bayesian hypothesis test, independent of the GP emulator, is performed to assess whether the expected utility of the new design is larger than the current design. For deterministic approximations, the approximate expected utility is calculated for the new design, and compared to that for the current design.

Phase I tends to produce clusters of design points. This is where, for example, two design points are separated by small Euclidean distance. Phase II allows these points to be consolidated into a single repeated design point by using a point exchange algorithm (e.g Gotwalt et al., 2009) with a candidate set given by the final design from Phase I. In the same way as Phase I, comparisons of the expected loss between two designs is made on the basis of either a Bayesian hypothesis test or a direct comparison of deterministic approximations.

The original Bayesian hypothesis test proposed by Overstall & Woods (2017) is appropriate for utility functions with continuous output. Overstall et al. (2017) extended the idea to utility functions with binary output, e.g. the utility function is an indicator function for some event. The type of test can be specified by the argument `binary`.

Similar to all coordinate exchange algorithms, ACE should be repeated from different initial designs. The function `pace` will implement this where the initial designs are given by a list via the argument `start.d`. On the completion of the repetitions of ACE, `pace` will approximate the expected utility for all final designs and return the design (the terminal design) with the largest approximate expected utility.

Value

The function will return an object of class `"ace"` (for functions `ace`, `acephase1` and `acephase2`) or `"pace"` (for function `"pace"`) which is a list with the following components:

`utility`	The argument `utility`.
`start.d`	The argument `start.d`.
`phase1.d`	The design found from Phase I of the ACE algorithm (only for `ace`, `acephase1` and `acephase2`).
`phase2.d`	The design found from Phase II of the ACE algorithm (only for `ace`, `acephase1` and `acephase2`)..

phase1.trace	A vector containing the approximated expected utility of the current design at each stage of Phase I of the ACE algorithm. This can be used to assess convergence for MC approximations. If deterministic = FALSE, this will be the mean of a call to utility with d being the current design and B being equal to the argument B[1]. If deterministic = TRUE, this will be a call to utility with d being the current design. For pace, this will be phase1.trace for the terminal design.
phase2.trace	A vector containing the approximated expected utility of the current design at each stage of Phase II of the ACE algorithm. This can be used to assess convergence for MC approximations. If deterministic = FALSE, this will be the mean of a call to utility with d being the current design and B being equal to the argument B[1]. If deterministic = TRUE, this will be a call to utility with d being the current design. For pace, this will be phase2.trace for the terminal design.
B	The argument B.
Q	The argument Q.
N1	The argument N1.
N2	The argument N2.
glm	If the object is a result of a direct call to ace then this is FALSE.
nlm	If the object is a result of a direct call to ace then this is FALSE.
criterion	If the object is a result of a direct call to ace then this is "NA".
prior	If the object is a result of a direct call to ace then this is "NA".
time	Computational time (in seconds) to run the ACE algorithm.
binary	The argument binary.
deterministic	The argument deterministic.
d	The terminal design (pace only).
eval	If deterministic = FALSE, a vector containing n.assess approximations to the expected utility for the terminal design (pace only). If deterministic = TRUE, a scalar giving the approximate expected utility for the terminal design (pace only).
final.d	A list of the same length as the argument start.d, where each element is the final design (i.e. phase2.d) for each repetition of the ACE algorithm (pace only).
besti	A scalar indicating which repetition of the ACE algorithm resulted in the terminal design (pace only).

Note

For more detail on the R implementation of the utility function used in the **Examples** section, see utilcomp18bad.

Author(s)

Antony M. Overstall <A.M.Overstall@soton.ac.uk>, David C. Woods & Maria Adamou

References

Chaloner, K. & Verdinelli, I. (1995). Bayesian Experimental Design: A Review. *Statistical Science*, **10**, 273-304.

Gotwalt, C., Jones, B. & Steinberg, D. (2009). Fast Computation of Designs Robust to Parameter Uncertainty for Nonlinear Settings. *Technometrics*, **51**, 88-95.

Meyer, R. & Nachtsheim, C. (1995). The Coordinate Exchange Algorithm for Constructing Exact Optimal Experimental Designs. *Technometrics*, **37**, 60-69.

Overstall, A.M. & Woods, D.C. (2017). Bayesian design of experiments using approximate coordinate exchange. *Technometrics*, **59**, 458-470.

Overstall, A.M., McGree, J.M. & Drovandi, C.C. (2018). An approach for finding fully Bayesian optimal designs using normal-based approximations to loss functions. *Statistics and Computing*, **28**(2), 343-358.

Examples

```
set.seed(1)
## Set seed for reproducibility.

## This example involves finding a pseudo-Bayesian D-optimal design for a
## compartmental model with n = 18 runs. There are three parameters.
## Two parameters have uniform priors and the third has a prior
## point mass. For more details see Overstall & Woods (2017).

start.d<-optimumLHS(n = 18, k = 1)
## Create an initial design.

## Using a MC approximation
example1<-ace(utility = utilcomp18bad, start.d = start.d, N1 = 1, N2 = 2, B = c(100, 20))
## Implement the ACE algorithm with 1 Phase I iterations and 2 Phase II
## iterations. The Monte Carlo sample sizes are 100 (for comparison) and 20 for
## fitting the GP emulator.

example1
## Produce a short summary.

#User-defined model & utility
#
#Number of runs = 18
#
#Number of factors = 1
#
#Number of Phase I iterations = 1
#
#Number of Phase II iterations = 2
#
#Computer time = 00:00:00

mean(utilcomp18bad(d = example1$phase2.d, B = 100))
## Calculate an approximation to the expected utility for the final design.
## Should get:
```

```
#[1] 9.254198

## Not run:
plot(example1)
## Produces a trace plot of the current value of the expected utility. This
## can be used to assess convergence.

## End(Not run)
```

---

aceglm *Approximate Coordinate Exchange (ACE) Algorithm for Generalised Linear Models*

Description

Functions implementing the approximate coordinate exchange (ACE) algorithm (Overstall & Woods, 2017) for finding Bayesian optimal experimental designs for generalised linear models (GLMs).

Usage

```
aceglm(formula, start.d, family, prior, B,
criterion = c("D", "A", "E", "SIG", "NSEL", "SIG-Norm", "NSEL-Norm"),
method = c("quadrature", "MC"), Q = 20, N1 = 20, N2 = 100, lower = -1,
upper = 1, progress = FALSE, limits = NULL)

paceglm(formula, start.d, family, prior, B,
criterion = c("D", "A", "E", "SIG", "NSEL", "SIG-Norm", "NSEL-Norm"),
method = c("quadrature", "MC"), Q = 20, N1 = 20, N2 = 100, lower = -1,
upper = 1, limits = NULL, mc.cores = 1, n.assess = 20)
```

Arguments

formula	An object of class "formula": a symbolic description of the model. The terms should correspond to the column names of the argument start.d.
start.d	For aceglm, an n by k matrix, with column names used by the argument formula, specifying the initial design for the ACE algorithm. For paceglm, a list with each element, an n by k matrix, with column names used by the argument formula, specifying the initial design for each repetition of the ACE algorithm.
family	A description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See family for details of family functions.)
prior	An argument specifying the prior distribution. For method = "MC", a function with one argument: B; a scalar integer. This function should return a B by p matrix, with p the number of model parameters,

containing a random sample from the prior distribution of the parameters. The value of p should correspond to the number of terms specified by the formula argument.

For method = "quadrature", a list specifying a normal or uniform prior for the model parameters. For a normal prior distribution, the list should have named entries mu and sigma2 specifying the prior mean and variance-covariance matrix. The prior mean may be specified as a scalar, which will then be replicated to form an vector of common prior means, or a vector of length p. The prior variance-covariance matrix may be specified as either a scalar common variance or a vector of length p of variances (for independent prior distributions) or as a p by p matrix. For a uniform prior distribution, the list should have a named entry support, a 2 by p matrix with each column giving the lower and upper limits of the support of the independent continuous uniform distribution for the corresponding parameter.

B

An optional argument for controlling the approximation to the expected utility. It should be a vector of length two.

For method = "MC", it specifies the size of the Monte Carlo samples, generated from the joint distribution of unknown quantities. The first sample size, B[1], gives the sample size to use in the comparison procedures, and the second sample size, B[2], gives the sample size to use for the evaluations of Monte Carlo integration that are used to fit the Gaussian process emulator. If left unspecified, the default value is c(20000,1000).

For method = "quadrature", it specifies the tuning parameters (numbers of radial abscissas and random rotations) for the implemented quadrature method; see **Details** for more information. If left unspecified, the default value is c(2, 8).

criterion

An optional character argument specifying the utility function. There are currently seven utility functions implemented as follows:

1. **pseudo-Bayesian D-optimality** (criterion = "D");
2. **pseudo-Bayesian A-optimality** (criterion = "A");
3. **pseudo-Bayesian E-optimality** (criterion = "E").
4. **Shannon information gain** with Monte Carlo (MC) approximation to marginal likelihood (criterion = "SIG");
5. **Shannon information gain** with normal-based Laplace approximation to marginal likelihood (criterion = "SIG-Norm");
6. **negative squared error loss** with importance sampling approximation to posterior mean (criterion = "NSEL");
7. **negative squared error loss** with normal-based approximation to posterior mean (criterion = "NSEL-Norm");

If left unspecified, the default is "D" denoting pseudo-Bayesian D-optimality. See **Details** for more information.

method

An optional character argument specifying the method of approximating the expected utility function. Current choices are method = "quadrature" for a deterministic quadrature approximation and method = "MC" for a stochastic Monte Carlo approximation. The first of these choices is **only** available when the argument criterion = "A", "D" or "E". The second choice is available

for all possible values of the argument `criterion`. If left unspecified, the argument defaults to `"quadrature"` for `criterion = "A"`, `"D"` or `"E"` and to `"MC"` otherwise. See **Details** for more information.

`Q` An integer specifying the number of evaluations of the approximate expected utility that are used to fit the Gaussian process emulator. The default value is `20`.

`N1` An integer specifying the number of iterations of Phase I of the ACE algorithm (the coordinate exchange phase). The default value is `20`.

`N2` An integer specifying the number of iterations of Phase II of the ACE algorithm (the point exchange phase). The default value is `100`.

`lower` An argument specifying the design space. This argument can either be a scalar or a matrix of the same dimension as the argument `start.d` which specifies the lower limits of all coordinates of the design space. The default value is -1.

`upper` An argument specifying the design space. This argument can either be a scalar or a matrix of the same dimension as the argument `start.d` which specifies the upper limits of all coordinates of the design space. The default value is 1.

`progress` A logical argument indicating whether the iteration number and other information detailing the progress of the algorithm should be printed. The default value is `FALSE`.

`limits` An argument specifying the grid over which to maximise the Gaussian process emulator for the expected utility function. It should be a function with three arguments: `i`, `j` and `d` which generates a one-dimensional grid for the `ij`th coordinate of the design when the current design is `d`. The default value is NULL which generates values uniformly on the interval `(lower[i,j],upper[i,j])` or `(lower,upper)` depending on whether the arguments `lower` and `upper` are matrices or scalars, respectively.

`mc.cores` The number of cores to use, i.e. at most how many child processes will be run simultaneously. Must be at least one (the default), and parallelisation requires at least two cores. See `mclapply` for more information and warnings for `mc.cores` > 1.

`n.assess` If `method = "MC"`, the approximate expected utility for the design from each repetition of the ACE algorithm will be calculated `n.assess` times. The terminal design returned will be the design with the largest mean approximate expected utility calculated over the `n.assess` approximations.

Details

The `aceglm` function implements the ACE algorithm to find designs for the class of generalised linear models (GLMs) for certain cases of utility function meaning the user does not have to write their own utility function.

Two utility functions are implemented.

1. **Shannon information gain (SIG)**
   The utility function is
   $$u^{SIG}(d) = \pi(\theta|y,d) - \pi(\theta),$$
   where $\pi(\theta|y,d)$ and $\pi(\theta)$ denote the posterior and prior densities of the parameters θ, respectively.

2. **Negative squared error loss (NSEL)**
   The utility function is

$$u^{NSEL}(d) = -\left(\theta - E(\theta|y,d)\right)^T \left(\theta - E(\theta|y,d)\right),$$

   where $E(\theta|y,d)$ denotes the posterior mean of θ.

In both cases the utility function is not available in closed form due to the analytical intractability of either the posterior distribution (for SIG) or the posterior mean (for NSEL). The `acebayes` package implements two approximations to both utility functions. If `criterion = "SIG"` or `criterion = "NSEL"` then sampling-based Monte Carlo or importance sampling approximations will be employed. This was the original approach used by Overstall & Woods (2017). If `criterion = "SIG-Norm"` or `criterion = "NSEL-Norm"` then approximations based on approximate normality of the posterior (Overstall et al., 2017) will be used.

The normal approximation to the posterior can be taken further leading to the approximation by some scalar function of the Fisher information matrix, $\mathcal{I}(\theta;d)$, which only depends on θ (Chaloner & Verdinelli, 1995). In the case of SIG, the approximate utility is given by

$$u^D(d) = \log|\mathcal{I}(\theta;d)|,$$

and the resulting design is typically called pseudo-Bayesian D-optimal. For NSEL, the approximate utility is given by

$$u^A(d) = -\mathrm{tr}\left\{\mathcal{I}(\theta;d)^{-1}\right\}$$

with the resulting design termed pseudo-Bayesian A-optimal. These designs are often used under the frequentist approach to optimal experimental design and so to complete the usual set, the following utility for finding a pseudo-Bayesian E-optimal design is also implemented:

$$u^E(d) = \min e\left(\mathcal{I}(\theta;d)\right),$$

where $e()$ denotes the function that calculates the eigenvalues of its argument.

The expected utilities can be approximated using Monte Carlo methods (`method = "MC"` for all criteria) or using a deterministic quadrature method (`method = "quadrature"`, implemented for the D, A and E criteria). The former approach approximates the expected utility via sampling from the prior. The latter approach uses a radial-spherical integration rule (Monahan and Genz, 1997) and `B[1]` specifies the number, n_r, of radial abscissas and `B[2]` specifies the number, n_q, of random rotations. Larger values of n_r will produce more accurate, but also more computationally expensive, approximations. See Gotwalt et al. (2009) for further details.

Note that the utility functions for SIG and NSEL are currently only implemented for logistic regression, i.e. `family = binomial`, or Poisson regression, i.e. `family = poisson(link="log")`, whereas the utility functions for pseudo-Bayesian designs are implemented for generic GLM families.

For more details on the ACE algorithm, see Overstall & Woods (2017).

Similar to all coordinate exchange algorithms, ACE should be repeated from different initial designs. The function `paceglm` will implement this where the initial designs are given by a list via the argument `start.d`. On the completion of the repetitions of ACE, `paceglm` will approximate the expected utility for all final designs and return the design (the terminal design) with the largest approximate expected utility.

Value

The function will return an object of class "ace" (for aceglm) or "pace" (for paceglm) which is a list with the following components:

utility	The utility function resulting from the choice of arguments.
start.d	The argument start.d.
phase1.d	The design found from Phase I of the ACE algorithm.
phase2.d	The design found from Phase II of the ACE algorithm.
phase1.trace	A vector containing the evaluations of the approximate expected utility of the current design at each stage of Phase I of the ACE algorithm. This can be used to assess convergence.
phase2.trace	A vector containing the evaluations of the approximate expected utility of the current design at each stage of Phase II of the ACE algorithm. This can be used to assess convergence.
B	The argument B.
Q	The argument Q.
N1	The argument N1.
N2	The argument N2.
glm	If the object is a result of a direct call to aceglm then this is TRUE.
nlm	This will be FALSE.
criterion	If the object is a result of a direct call to aceglm then this is the argument criterion.
method	If the object is a result of a direct call to aceglm then this is the argument method.
prior	If the object is a result of a direct call to aceglm then this is the argument prior.
family	If the object is a result of a direct call to aceglm then this is the argument family.
formula	If the object is a result of a direct call to acenlm then this is the argument formula.
time	Computational time (in seconds) to run the ACE algorithm.
binary	The argument binary. Will be FALSE for the utility functions currently implemented.
d	The terminal design (paceglm only).
eval	If deterministic = "MC", a vector containing n.assess approximations to the expected utility for the terminal design (paceglm only). If deterministic = "quadrature", a scalar giving the approximate expected utility for the terminal design (paceglm only).
final.d	A list of the same length as the argument start.d, where each element is the final design (i.e. phase2.d) for each repetition of the ACE algorithm (paceglm only).
besti	A scalar indicating which repetition of the ACE algorithm resulted in the terminal design (paceglm only).

Note

This is a wrapper function for ace.

Author(s)

Antony M. Overstall <A.M.Overstall@soton.ac.uk>, David C. Woods & Maria Adamou

References

Chaloner, K. & Verdinelli, I. (1995). Bayesian experimental design: a review. *Statistical Science*, **10**, 273-304.

Gotwalt, C. M., Jones, B. A. & Steinberg, D. M. (2009). Fast computation of designs robust to parameter uncertainty for nonlinear settings. *Technometrics*, **51**, 88-95.

Meyer, R. & Nachtsheim, C. (1995). The coordinate exchange algorithm for constructing exact optimal experimental designs. *Technometrics*, **37**, 60-69.

Monahan, J. and Genz, A. (1997). Spherical-radial integration rules for Bayesian computation," Journal of the American Statistical Association, 92, 664–674.

Overstall, A.M. & Woods, D.C. (2017). Bayesian design of experiments using approximate coordinate exchange. *Technometrics*, **59**, 458-470.

Overstall, A.M., McGree, J.M. & Drovandi, C.C. (2018). An approach for finding fully Bayesian optimal designs using normal-based approximations to loss functions. *Statistics and Computing*, **28**(2), 343-358.

See Also

ace, acenlm, pace, pacenlm.

Examples

```
## This example uses aceglm to find a Bayesian D-optimal design for a
## first order logistic regression model with 6 runs 4 factors. The priors are
## those used by Overstall & Woods (2017), with each of the five
## parameters having a uniform prior. The design space for each coordinate is [-1, 1].

set.seed(1)
## Set seed for reproducibility.

n<-6
## Specify the sample size (number of runs).

start.d<-matrix(2 * randomLHS(n = n,k = 4) - 1,nrow = n,ncol = 4,
dimnames = list(as.character(1:n), c("x1", "x2", "x3", "x4")))
## Generate an initial design of appropriate dimension. The initial design is a
## Latin hypercube sample.

low<-c(-3, 4, 5, -6, -2.5)
upp<-c(3, 10, 11, 0, 3.5)
## Lower and upper limits of the uniform prior distributions.
```

```
prior<-function(B){
t(t(6*matrix(runif(n = 5 * B),ncol = 5)) + low)}
## Create a function which specifies the prior. This function will return a
## B by 5 matrix where each row gives a value generated from the prior
## distribution for the model parameters.

example1<-aceglm(formula=~x1+x2+x3+x4, start.d = start.d, family = binomial,
prior = prior, method = "MC", N1 = 1, N2 = 0, B = c(1000, 1000))
## Call the aceglm function which implements the ACE algorithm requesting
## only one iteration of Phase I and zero iterations of Phase II. The Monte
## Carlo sample size for the comparison procedure (B[1]) is set to 100.

example1
## Print out a short summary.

#Generalised Linear Model
#Criterion = Bayesian D-optimality
#Formula: ~x1 + x2 + x3 + x4
#
#Family: binomial
#Link function: logit
#
#Method:  MC
#
#B:  1000 1000
#
#Number of runs = 6
#
#Number of factors = 4
#
#Number of Phase I iterations = 1
#
#Number of Phase II iterations = 0
#
#Computer time = 00:00:01

example1$phase2.d
## Look at the final design.

#          x1          x2          x3         x4
#1 -0.3571245  0.16069337 -0.61325375  0.9276443
#2 -0.9167309  0.91411512  0.69842151  0.2605092
#3 -0.8843699  0.42863930 -1.00000000 -0.9679402
#4  0.3696224 -0.27126080  0.65284076  0.1850767
#5  0.7172267 -0.34743402 -0.05968457 -0.6588896
#6  0.7469636  0.05854029  1.00000000 -0.1742566


prior2 <- list(support = rbind(low, upp))
## A list specifying the parameters of the uniform prior distribution

example2<-aceglm(formula = ~ x1 +x2 + x3 + x4, start.d = start.d, family = binomial,
prior = prior2, N1 = 1, N2 = 0)
```

```
## Call the aceglm function with the default method of "quadrature"

example2$phase2.d
## Final design

#          x1          x2         x3          x4
#1 -0.3269814  0.08697755 -0.7583228  1.00000000
#2 -0.8322237  0.86652194  0.5747066  0.51442169
#3 -0.8987852  0.48881387 -0.8554894 -1.00000000
#4  0.3441093 -0.29050147  0.4704248  0.07628932
#5  0.8371670 -0.42361888  0.1429862 -0.95080251
#6  0.6802119  0.10853163  1.0000000  0.75421678

mean(example1$utility(d = example1$phase2.d, B = 20000))
#[1] -11.55139
mean(example2$utility(d = example2$phase2.d, B = 20000))
#[1] -11.19838
## Compare the two designs using the Monte Carlo approximation
```

---

acenlm *Approximate Coordinate Exchange (ACE) Algorithm for Non-Linear Models*

Description

Function implementing the approximate coordinate exchange algorithm (Overstall & Woods, 2017) for finding optimal Bayesian designs for non-linear regression models.

Usage

```
acenlm(formula, start.d, prior, B, criterion = c("D", "A", "E", "SIG", "NSEL"),
method = c("quadrature", "MC"),  Q = 20, N1 = 20, N2 = 100, lower = -1, upper = 1,
progress = FALSE, limits = NULL)

pacenlm(formula, start.d, prior, B, criterion = c("D", "A", "E", "SIG", "NSEL"),
method = c("quadrature", "MC"),  Q = 20, N1 = 20, N2 = 100, lower = -1, upper = 1,
limits = NULL, mc.cores = 1, n.assess = 20)
```

Arguments

formula	An object of class "formula": a symbolic description of the model. The terms should correspond to the column names of the argument start.d and the argument prior.
start.d	For aceglm, an n by k matrix, with column names used by the argument formula, specifying the initial design for the ACE algorithm. For paceglm, a list with each element, an n by k matrix, with column names used by the argument formula, specifying the initial design for each repetition of the ACE algorithm.

`prior` An argument specifying the prior distribution.

For `method = "MC"`, a function with one argument: `B`; a scalar integer. This function should return a `B` by `p` matrix (or `p+1` for `criterion = "SIG"` or `criterion = "NSEL"`), with `p` the number of model parameters, containing a random sample from the prior distribution of the parameters. The value of `p` should correspond to the number of terms specified by the `formula` argument. The column names must match the names of parameters in the `formula` argument. For `criterion="SIG"` or `criterion="NSEL"`, an extra column (named `sig2`) should contain a sample from the prior distribution of the error variance.

For `method = "quadrature"`, a list specifying a normal or uniform prior for the model parameters. For a normal prior distribution, the list should have named entries `mu` and `sigma2` specifying the prior mean and variance-covariance matrix. The prior mean may be specified as a scalar, which will then be replicated to form an vector of common prior means, or a vector of length `p`. The prior variance-covariance matrix may be specified as either a scalar common variance or a vector of length `p` of variances (for independent prior distributions) or as a `p` by `p` matrix. The names attribute of `mu` must match the names of the parameters in the `formula` argument. For a uniform prior distribution, the list should have a named entry `support`, a `2` by `p` matrix with each column giving the lower and upper limits of the support of the independent continuous uniform distribution for the corresponding parameter. The column names of `support` must match the names of parameters in the `formula` argument.

`B` An optional argument for controlling the approximation to the expected utility. It should be a vector of length two.

For `criterion = "MC"`, it specifies the size of the Monte Carlo samples, generated from the joint distribution of unknown quantities. The first sample size, `B[1]`, gives the sample size to use in the comparison procedures, and the second sample size, `B[2]`, gives the sample size to use for the evaluations of Monte Carlo integration that are used to fit the Gaussian process emulator. If left unspecified, the default value is `c(20000,1000)`.

For `criterion = "quadrature"`, it specifies the tuning parameters (numbers of radial abscissas and random rotations) for the implemented quadrature method; see **Details** for more information. If left unspecified, the default value is `c(2, 8)`.

`criterion` An optional character argument specifying the utility function. There are currently five utility functions implemented consisting of

1. **pseudo-Bayesian D-optimality** (`criterion = "D"`);
2. **pseudo-Bayesian A-optimality** (`criterion = "A"`);
3. **pseudo-Bayesian E-optimality** (`criterion = "E"`);
4. **Shannon information gain** (`criterion = "SIG"`);
5. **negative squared error loss** (`criterion = "NSEL"`).

The default value is `"D"` denoting pseudo-Bayesian D-optimality. See **Details** for more information.

`method` An optional character argument specifying the method of approximating the expected utility function. Current choices are `method = "quadrature"` for a deterministic quadrature approximation and `method = "MC"` for a stochastic

Monte Carlo approximation. The first of these choices is **only** available when the argument `criterion = "A"`, `"D"` or `"E"`. The second choice is available for all possible values of the argument `criterion`. If left unspecified, the argument defaults to `"quadrature"` for `criterion = "A"`, `"D"` or `"E"` and to `"MC"` otherwise. See **Details** for more information.

`Q` An integer specifying the number of evaluations of the approximate expected utility that are used to fit the Gaussian process emulator. The default value is `20`.

`N1` An integer specifying the number of iterations of Phase I of the ACE algorithm (the coordinate exchange phase). The default value is `20`.

`N2` An integer specifying the number of iterations of Phase II of the ACE algorithm (the point exchange phase). The default value is `100`.

`lower` An argument specifying the design space. This argument can either be a scalar or a matrix of the same dimension as the argument `start.d` which specifies the lower limits of all coordinates of the design space. The default value is -1.

`upper` An argument specifying the design space. This argument can either be a scalar or a matrix of the same dimension as the argument `start.d` which specifies the upper limits of all coordinates of the design space. The default value is 1.

`progress` A logical argument indicating whether the iteration number should be printed. The default value is `FALSE`.

`limits` An argument specifying the grid over which to maximise the Gaussian process emulator for the expected utility function. It should be a function with three arguments: `i`, `j` and `d` which generates a one-dimensional grid for the `ij`th coordinate of the design when the current design is `d`. The default value is NULL which generates values uniformly on the interval `(lower[i,j],upper[i,j])` or `(lower,upper)` depending on whether the arguments `lower` and `upper` are matrices or scalars, respectively.

`mc.cores` The number of cores to use, i.e. at most how many child processes will be run simultaneously. Must be at least one (the default), and parallelisation requires at least two cores. See `mclapply` for more information and warnings for `mc.cores` > 1.

`n.assess` If `method = "MC"`, the approximate expected utility for the design from each repetition of the ACE algorithm will be calculated `n.assess` times. The terminal design returned will be the design with the largest mean approximate expected utility calculated over the `n.assess` approximations.

Details

The `acenlm` function implements the ACE algorithm to find designs for general classes of nonlinear regression models with identically and independently normally distributed errors meaning the user does not have to write their own utility function.

Two utility functions are implemented.

1. **Shannon information gain (SIG)**
   The utility function is
   $$u^{SIG}(d) = \pi(\theta|y,d) - \pi(\theta),$$
   where $\pi(\theta|y,d)$ and $\pi(\theta)$ denote the posterior and prior densities of the parameters θ, respectively.

2. **Negative squared error loss (NSEL)**
   The utility function is

$$u^{NSEL}(d) = -\left(\theta - E(\theta|y,d)\right)^T \left(\theta - E(\theta|y,d)\right),$$

   where $E(\theta|y,d)$ denotes the posterior mean of θ.

In both cases the utility function is not available in closed form due to the analytical intractability of either the posterior distribution (for SIG) or the posterior mean (for NSEL). Sampling-based Monte Carlo or importance sampling approximations will be employed. This was the original approach used by Overstall & Woods (2017).

A normal approximation to the posterior can be taken leading to the approximation by some scalar function of the Fisher information matrix, $\mathcal{I}(\theta;d)$, which only depends on θ (Chaloner & Verdinelli, 1995). In the case of SIG, the approximate utility is given by

$$u^D(d) = \log|\mathcal{I}(\theta;d)|,$$

and the resulting design is typically called pseudo-Bayesian D-optimal. For NSEL, the approximate utility is given by

$$u^A(d) = -\mathrm{tr}\left\{\mathcal{I}(\theta;d)^{-1}\right\}$$

with the resulting design termed pseudo-Bayesian A-optimal. These designs are often used under the frequentist approach to optimal experimental design and so to complete the usual set, the following utility for finding a pseudo-Bayesian E-optimal design is also implemented:

$$u^E(d) = \min e\left(\mathcal{I}(\theta;d)\right),$$

where $e()$ denotes the function that calculates the eigenvalues of its argument.

The expected utilities can be approximated using Monte Carlo methods (`method = "MC"` for all criteria) or using a deterministic quadrature method (`method = "quadrature"`, implemented for the D, A and E criteria). The former approach approximates the expected utility via sampling from the prior. The latter approach uses a radial-spherical integration rule (Monahan and Genz, 1997) and `B[1]` specifies the number, n_r, of radial abscissas and `B[2]` specifies the number, n_q, of random rotations. Larger values of n_r will produce more accurate, but also more computationally expensive, approximations. See Gotwalt et al. (2009) for further details.

For more details on the ACE algorithm, see Overstall & Woods (2017).

Similar to all coordinate exchange algorithms, ACE should be repeated from different initial designs. The function `pacenlm` will implement this where the initial designs are given by a list via the argument `start.d`. On the completion of the repetitions of ACE, `pacenlm` will approximate the expected utility for all final designs and return the design (the terminal design) with the largest approximate expected utility.

Value

The function will return an object of class `"ace"` (for `acenlm`) or `"pace"` (for `pacenlm`) which is a list with the following components:

`utility`	The utility function resulting from the choice of arguments.
`start.d`	The argument `start.d`.

phase1.d	The design found from Phase I of the ACE algorithm.
phase2.d	The design found from Phase II of the ACE algorithm.
phase1.trace	A vector containing the evaluations of the approximate expected utility of the current design at each stage of Phase I of the ACE algorithm. This can be used to assess convergence.
phase2.trace	A vector containing the evaluations of the approximate expected utility of the current design at each stage of Phase II of the ACE algorithm. This can be used to assess convergence.
B	The argument B.
Q	The argument Q.
N1	The argument N1.
N2	The argument N2.
glm	This will be FALSE.
nlm	If the object is a result of a direct call to acenlm then this is TRUE.
criterion	If the object is a result of a direct call to acenlm then this is the argument criterion.
method	If the object is a result of a direct call to acenlm then this is the argument method.
prior	If the object is a result of a direct call to aceglm then this is the argument prior.
formula	If the object is a result of a direct call to acenlm then this is the argument formula.
time	Computational time (in seconds) to run the ACE algorithm.
binary	The argument binary. Will be FALSE for the utility functions currently implemented.
d	The terminal design (pacenlm only).
eval	If deterministic = "MC", a vector containing n.assess approximations to the expected utility for the terminal design (pacenlm only). If deterministic = "quadrature", a scalar giving the approximate expected utility for the terminal design (pacenlm only).
final.d	A list of the same length as the argument start.d, where each element is the final design (i.e. phase2.d) for each repetition of the ACE algorithm (pacenlm only).
besti	A scalar indicating which repetition of the ACE algorithm resulted in the terminal design (pacenlm only).

Note

This is a wrapper function for ace.

Author(s)

Antony M. Overstall <A.M.Overstall@soton.ac.uk>, David C. Woods & Maria Adamou

References

Chaloner, K. & Verdinelli, I. (1995). Bayesian experimental design: a review. *Statistical Science*, **10**, 273-304.

Gotwalt, C. M., Jones, B. A. & Steinberg, D. M. (2009). Fast computation of designs robust to parameter uncertainty for nonlinear settings. *Technometrics*, **51**, 88-95.

Meyer, R. & Nachtsheim, C. (1995). The coordinate exchange algorithm for constructing exact optimal experimental designs. *Technometrics*, **37**, 60-69.

Monahan, J. and Genz, A. (1997). Spherical-radial integration rules for Bayesian computation," Journal of the American Statistical Association, 92, 664–674.

Overstall, A.M. & Woods, D.C. (2017). Bayesian design of experiments using approximate coordinate exchange. *Technometrics*, **59**, 458-470.

See Also

ace, aceglm, pace, paceglm.

Examples

```
## This example uses aceglm to find a Bayesian D-optimal design for a
## compartmental model with 6 runs 1 factor. The priors are
## those used by Overstall & Woods (2017). The design space for each
## coordinate is [0, 24] hours.

set.seed(1)
## Set seed for reproducibility.

n<-6
## Specify the sample size (number of runs).

start.d<-matrix(24 * randomLHS(n = n, k = 1), nrow = n, ncol = 1,
dimnames = list(as.character(1:n), c("t")))
## Generate an initial design of appropriate dimension. The initial design is a
## Latin hypercube sample.

low<-c(0.01884, 0.298, 21.8)
upp<-c(0.09884, 8.298, 21.8)
## Lower and upper limits of the support of the uniform prior distributions. Note that the prior
## for the third element is a point mass.

prior<-function(B){
out<-cbind(runif(n = B, min = low[1], max = upp[1]), runif(n = B, min = low[2],max = upp[2]),
runif(n = B, min = low[3], max = upp[3]))
colnames(out)<-c("a", "b", "c")
out}

## Create a function which specifies the prior. This function will return a
## B by 3 matrix where each row gives a value generated from the prior
## distribution for the model parameters.

example1<-acenlm(formula = ~ c*(exp( - a * t) - exp( - b * t)), start.d = start.d, prior = prior,
```

```
N1 = 1, N2 = 0, B = c(1000, 1000), lower = 0, upper = 24, method = "MC")
## Call the acenlm function which implements the ACE algorithm requesting
## only one iteration of Phase I and zero iterations of Phase II. The Monte
## Carlo sample size for the comparison procedure (B[1]) is set to 1000.

example1
## Print out a short summary.

#Non Linear Model
#Criterion = Bayesian D-optimality
#Formula: ~c * (exp(-a * t) - exp(-b * t))
#Method:  MC
#
#B:  1000 1000
#
#Number of runs = 6
#
#Number of factors = 1
#
#Number of Phase I iterations = 1
#
#Number of Phase II iterations = 0
#
#Computer time = 00:00:00

example1$phase2.d
## Look at the final design.

#           t
#1 19.7787011
#2  2.6431912
#3  0.2356938
#4  8.2471451
#5  1.4742319
#6 12.7062270

prior2 <- list(support = cbind(rbind(low, upp)))
colnames(prior2$support) <- c("a", "b", "c")
example2 <- acenlm(formula = ~ c * (exp( - a * t) - exp( - b *t)), start.d = start.d,
prior = prior2, lower = 0, upper = 24, N1 = 1, N2 = 0 )
## Call the acenlm function with the default method of "quadrature"

example2$phase2.d
## Final design

#           t
#1  0.3090187
#2  2.0710214
#3  6.5164562
#4  0.5964350
#5 23.3100644
#6 20.1339899
```

```
utility <- utilitynlm(formula = ~c * (exp( - a * t) - exp( - b *t)), prior = prior,
                            desvars = "t", method = "MC" )$utility
## create a utility function to compare designs

mean(utility(example1$phase1.d, B = 20000))
#[1] 12.13773
mean(utility(example2$phase1.d, B = 20000))
#[1] 11.19745
## Compare the two designs using the Monte Carlo approximation
```

---

aceobjects *Print and Summary of* ace *and* pace *Objects*

Description

These functions print and summarise objects of class "ace" or "pace".

Usage

```
## S3 method for class 'ace'
print(x, ...)
## S3 method for class 'ace'
summary(object, ...)

## S3 method for class 'pace'
print(x, ...)
## S3 method for class 'pace'
summary(object, ...)
```

Arguments

x	An object of class "ace" or "pace".
object	An object of class "ace" or "pace".
...	Arguments to be passed to and from other methods.

Value

These functions both provide a print out with the following information.

If the object is a result of a direct call to aceglm, acenlm, paceglm, or pacenlm, then the argument criterion will be printed, otherwise the statement User-defined utility will be printed.

Also printed are the number of repetitions ("pace" objects only), runs, factors, Phase I and II iterations of the ACE algorithm and the computational time required.

For more details on the ACE algorithm, see Overstall & Woods (2017).

Note

For examples see ace, aceglm, and acenlm.

Author(s)

Antony M. Overstall <A.M.Overstall@soton.ac.uk>, David C. Woods & Maria Adamou

References

Overstall, A.M. & Woods, D.C. (2017). Bayesian design of experiments using approximate coordinate exchange. *Technometrics*, **59**, 458-470.

See Also

ace, aceglm, acenlm, pace, paceglm, pacenlm.

---

assess *Compares two designs under the approximate expected utility*

Description

Calculates approximations to the expected utility for two designs.

Usage

```
assess(d1, d2, ...)

## S3 method for class 'ace'
assess(d1, d2, B = NULL, n.assess = 20, relative = TRUE, ...)

## S3 method for class 'pace'
assess(d1, d2, B = NULL, n.assess = 20, relative = TRUE, ...)
```

Arguments

d1, d2	d1 should be an object of class "ace" or "pace" and d2 should be an object of class "ace", "pace" or "matrix".
B	An optional argument for controlling the approximation to the expected utility (see ace, aceglm and acenlm). If left unspecified, the value is inherited from the argument d1.
n.assess	If d1 was generated from a call to (p)ace with argument deterministic = FALSE or from a call to (p)aceglm or (p)acenlm with argument method being "MC", then n.assess is an optional argument giving the number of evaluations of the stochastic approximation to the expected utility.
relative	An optional argument, for when d1 was generated as a call to (p)aceglm or (p)acenlm with argument criterion being "A", "D" or "E", controlling whether the measure of relative efficiency is calculated for d1 relative to d2 (TRUE; the default) or for d2 relative to d1 (FALSE).
...	Arguments to be passed to and from other methods.

Details

In the case of when d1 was generated from a call to (p)ace with argument deterministic = FALSE or from a call to (p)aceglm or (p)acenlm with argument method being "MC", n.assess evaluations of the stochastic approximation to the expected utility will be calculated for each of the designs from d1 and d2. Otherwise, one evaluation of the deterministic approximation to the expected utility will be calculated for each of the designs from d1 and d2.

In the case when d1 was generated as a call to (p)aceglm or (p)acenlm with argument criterion being "A", "D" or "E", the relative D-, E-, or A-efficiency of the two designs will be calculated. The direction of the relative efficiency can be controlled by the relative argument.

Value

The function will an object of class "assess" which is a list with the following components:

U1	In the case of when d1 was generated from a call to (p)ace with argument deterministic = FALSE or from a call to (p)aceglm or (p)acenlm with argument method being "MC", U1 will be a vector of n.assess evaluations of the stochastic approximation to the expected utility for design d1. Otherwise, U1 will be a scalar of one evaluation of the deterministic approximation to the expected utility for design d1.
U2	In the case of when d1 was generated from a call to (p)ace with argument deterministic = FALSE or from a call to (p)aceglm or (p)acenlm with argument method being "MC", U2 will be a vector of n.assess evaluations of the stochastic approximation to the expected utility for design d2. Otherwise, U2 will be a scalar of one evaluation of the deterministic approximation to the expected utility for design d2.
eff	In the case when d1 was generated as a call to (p)aceglm or (p)acenlm with argument criterion being "A", "D" or "E", eff is a scalar of the relative D-, E-, or A-efficiency of the two designs. Otherwise it will be NULL.
d1	The argument d1.
d2	The argument d2.

Author(s)

Antony M. Overstall <A.M.Overstall@soton.ac.uk>, David C. Woods & Maria Adamou

See Also

ace, pace,aceglm, acenlm, paceglm, pacenlm.

Examples

```
## This example involves finding a Bayesian D-optimal design for a
## compartmental model with n = 18 runs. There are three parameters.
## Two parameters have uniform priors and the third has a prior
## point mass.

n <- 18
```

```
k <- 1
p <- 3
set.seed(1)
start.d <- randomLHS(n = n, k = k) * 24
colnames(start.d) <- c("t")

a1<-c(0.01884, 0.298)
a2<-c(0.09884, 8.298)

prior <- list(support = cbind(rbind(a1, a2), c(21.8, 21.8)))
colnames(prior[[1]]) <- c("theta1", "theta2", "theta3")

example <- acenlm(formula = ~ theta3 * (exp( - theta1 * t) - exp( - theta2 * t)),
start.d = start.d, prior = prior, lower = 0, upper = 24, N1 = 2, N2 = 0)

## Compute efficiency of final design compared to starting design.
assess(d1 = example, d2 = start.d)

## Should get

# Approximate expected utility of d1 = 15.40583
# Approximate expected utility of d2 = 11.26968
# Approximate relative D-efficiency = 396.9804%
```

`assessobjects` *Print and Summary of* `assess` *Objects*

Description

These functions print and summarise objects of class `"assess"`.

Usage

```
## S3 method for class 'assess'
print(x, ...)
## S3 method for class 'assess'
summary(object, ...)
```

Arguments

`x`	An object of class `"assess"`.
`object`	An object of class `"assess"`.
`...`	Arguments to be passed to and from other methods.

Value

These functions both provide a print out with the following information.

In the case of when d1 was generated from a call to (p)ace with argument deterministic = FALSE or from a call to (p)aceglm or (p)acenlm with argument method being "MC", then the mean and standard deviation of the n.assess evaluations of the approximate expected utility for each of the designs d1 and d2 will be printed.

Otherwise, one evaluation of the deterministic approximation to the expected utility will be printed for each of the designs from d1 and d2. In the case when d1 was generated as a call to (p)aceglm or (p)acenlm with argument criterion being "A", "D" or "E", the relative D-, E-, or A-efficiency of the two designs will be also be printed.

Author(s)

Antony M. Overstall <A.M.Overstall@soton.ac.uk>, David C. Woods & Maria Adamou

See Also

assess

---

overstallwoods *Functions implementing the examples of Overstall & Woods (2017).*

Description

This suite of functions implement the examples in Overstall & Woods (2017).

Usage

```
######## Compartmental model ###################################

utilcomp18bad(d, B)
optdescomp18bad(type = "ACE")
utilcomp15bad(d, B)
optdescomp15bad()
utilcomp15sig(d, B)
optdescomp15sig()
utilcomp15sigDRS(d, B)
optdescomp15sigDRS()

######## Logistic regression model ############################

utillrbad(d, B)
optdeslrbad(n, type = "ACE")
utillrsig(d, B)
inideslrsig(n, rep)
```

```
optdeslrsig(n)

utilhlrbad(d, B)
optdeshlrbad(n)
utilhlrsig(d, B)
inideshlrsig(n, rep)
optdeshlrsig(n)

utillrbaa(d, B)
optdeslrbaa(n)
utillrnsel(d, B)
inideslrnsel(n, rep)
optdeslrnsel(n)

optdeshlrbaa(n)
utilhlrbaa(d, B)
utilhlrnsel(d, B)
inideshlrnsel(n, rep)
optdeshlrnsel(n)

######## Beetle mortality experiment ########################

utilbeetle(d, B)
optdesbeetle(n)

######## Linear model #######################################

utillinmod(d, B)
optdeslinmod(n, type = "ACE")

#############################################################
```

Arguments

d	An n by k matrix specifying the design matrix, where n and k denote the number of runs and factors, respectively. See **Details** and **Value** for the values that n and k can take for each of the examples. Each element of d is scaled to the interval [-1,1].
n	The number of runs ins the experiment.
rep	A scalar integer in the set $\{1, \ldots, 20\}$ specifying the initial design.
B	A scalar integer specifying the Monte Carlo sample size.
type	An optional character argument specifying which design to return. For `optdeslrbad`, possible values are `c("ACE","Gotwalt","Atkinson")`. If `"ACE"` (the default) then the design found by the ACE algorithm will be returned. If `"Gotwalt"` then the design published in Gotwalt et al (2009) is returned. If `"Atkinson"` then the design found by Atkinson et al (1993) is returned.

For optdeslinmod, possible values are c("ACE","BoxDraper"). If "ACE" (the default) then the design found by the ACE algorithm will be returned. If "BoxDraper" then the true optimal design, as found by Box & Draper (1971), will be returned.

Details

This section provides details on the examples considered and the functions implemented in acebayes.

Compartmental model

Compartmental models are used in Pharmokinetics to study how materials flow through an organism. A drug is administered to an individual or animal and then the amount present at a certain body location is measured at a set of n pre-determined sampling times (in hours). There is one design variable: sampling time. Therefore the design matrix d is an n by 1 matrix with elements controlling the n sampling times, i.e. the number of factors is k=1.

In Overstall & Woods (2017), two different compartmental model examples are considered. The first (in the Supplementary Material) comes from Atkinson et al (1993) and Gotwalt et al (2009) where there are n = 18 sampling times and interest lies in finding a Bayesian D-optimal design. The functions whose name includes "comp18" refer to this example.

The second example (in Section 3.2) comes from Ryan et al (2014), where there are n = 15 sampling times and the ultimate interest lies in finding an optimal design under the Shannon information gain utility. Also considered is the Bayesian D-optimal design. The functions whose name includes "comp15" refer to this example. Note that Ryan et al (2014) used a dimension reduction scheme (DRS) to find optimal designs. The function whose name is suffixed by "DRS" refer to this situation.

Logistic regression model

In Section 3.3 of Overstall & Woods (2017), a first-order logistic regression model in k = 4 factors and n runs is considered. Woods et al (2006) and Gotwalt et al (2009) considered generating Bayesian D-optimal designs for n = 16 and n = 48. Overstall & Woods (2017) extended this example by considering Bayesian A-optimal, Shannon information gain (SIG) and negative squared error loss (NSEL) utility functions, a range of number of runs from 6 to 48, and "random effects" to form a hierarchical logistic regression model.

Beetle mortality experiment

Overstall & Woods (2017, Section 3.4) considers generating an optimal design for a follow-up experiment. The original design and data (Bliss, 1935) involves administering different doses of poison to $N = 8$ groups of beetles. The number of beetles that die in each group are recorded. Six different models are considered formed from the combination of three link functions and two linear predictors (following the analysis of O'Hagan & Forster, 2004). Interest lies in the quantity known as lethal dose 50 (LD50) which is the dose required to kill 50% of the beetles and is a function of the model parameters for a given model. Consider finding an optimal design for estimating LD50 under the negative squared error loss (NSEL) function for n new doses of poison (i.e. k = 1 factor). The prior distribution is equivalent to the posterior distribution arising from the original data and includes model uncertainty.

Linear model

In the supplementary material, Overstall & Woods (2017) considers finding D-optimal designs for a second-order (i.e. k = 2) response surface in n=6,7,8,9 runs. Note that the D-optimal design is equivalent to the optimal design under Shannon information gain and a non-informative prior distribution.

The expected utility function in this case is available in closed form, i.e. it does not require approximation. Box & Draper (1971) found optimal designs analytically for the number of runs considered here. Overstall & Woods (2017) use this example to demonstrate the efficacy of the ACE algorithm.

Value

Compartmental model

For the example in the Supplementary Material;

- The function `utilcomp18bad` will return a vector of length `B` where each element is the value of the Bayesian D-optimal utility function (i.e. the log determinant of the Fisher information matrix) evaluated at a sample of size `B` generated from the prior distribution of the model parameters.
- The function `optdescomp18bad` will return an 18 by 1 matrix giving the optimal design (specified by the argument `type`). The elements will be scaled to be in the interval [-1, 1], i.e. a -1 corresponds to a sampling times of 0 hours, and 1 corresponds to a time of 24 hours.

For the example in Section 3.2;

- The function `utilcomp15bad` will return a vector of length `B` where each element is the value of the Bayesian D-optimal utility function (i.e. the log determinant of the Fisher information matrix) evaluated at a sample of size `B` generated from the prior distribution of the model parameters.
- The function `optdescomp15bad` will return an 15 by 1 matrix giving the optimal design (found using ACE) under Bayesian D-optimality. The elements will be scaled to be in the interval [-1, 1], i.e. a -1 corresponds to a sampling times of 0 hours, and 1 corresponds to a time of 24 hours.
- The function `utilcomp15sig` will return a vector of length `B` where each element is the value of the SIG utility function evaluated at a sample of size `B` generated from the joint distribution of model parameters and unobserved responses.
- The function `optdescomp15sig` will return an 18 by 1 matrix giving the optimal design (found using ACE) under the SIG utility. The elements will be scaled to be in the interval [-1, 1], i.e. a -1 corresponds to a sampling times of 0 hours, and 1 corresponds to a time of 24 hours.
- The function `utilcomp15sigDRS` will return a vector of length `B` where each element is the value of the SIG utility function (where a DRS has been used) evaluated at a sample of size `B` generated from the joint distribution of model parameters and unobserved responses. Here the Beta DRS (see Overstall & Woods, 2017) has been used so `d` should be a 2 by 1 matrix containing the positive beta parameters.
- The function `optdescomp15sigDRS` will return a 2 by 1 matrix giving the optimal design (found using ACE) under the SIG utility, where a DRS has been used. The elements correspond to the parameters of a beta distribution.

Logistic regression model

A function whose name includes `"lr"` refers to standard logistic regression, whereas `"hlr"` refers to hierarchical logistic regression. Under standard logistic regression the possible values for the argument `n` can be any even integer between 6 and 48. For hierarchical logistic regression, `n` can be any integer divisible by 6 between 12 and 48. The function name also indicates the utility function:

- "bad" Bayesian D-optimal
- "baa" Bayesian A-optimal
- "sig" Shannon information gain
- "nsel" Negative squared error loss

The functions prefixed by "util" will return a vector of length B where each element is the utility function evaluated at a sample generated from the prior distribution of model parameters (for Bayesian D- and A-optimality) or the joint distribution of model parameters and unobserved responses (for SIG and NSEL).

The functions prefixed by "optdes" will return an n by k = 4 matrix giving the optimal design found by ACE. The designs given by this function are those reported on in Overstall & Woods (2017). The function optdeslrbad will also return designs (for n = 16, 48) found by Woods et al (2006) and Gotwalt et al (2009) by specifying the type argument appropriately.

The functions prefixed by "inides" will return an n by k = 4 matrix giving an initial design for ACE to find the optimal designs under the SIG and NSEL utility functions. These are 20 designs found using ACE under approximations to the Bayesian A- and D-optimal utility functions, respectively. The argument rep specifies which of these 20 designs to use.

Beetle mortality experiment

The function utilbeetle will return a vector of length B where each element is the value of the utility function for a sample generated from the joint distribution of the model parameters, model and unobserved responses.

The function optdesbeetle will return an n by 1 matrix giving the optimal design under the NSEL utility function (found using ACE) for estimating the LD50. The elements will be scaled to be in the interval [-1, 1], where -1 corresponds to a dose of 1.6907, 0 to a dose of 1.7873 and 1 to a dose of 1.8839. The designs given by this function are those reported in Overstall & Woods (2017) for n = 1, . . . , 10.

Linear model

The function utillinmod will return a vector of length B where each element is a realisation of a stochastic approximation to the expected utility.

The function optdeslinmod will return an n by 2 matrix giving the D-optimal design (specified by the argument type). If type = "ACE", the designs returned by this function are those found using the ACE algorithm and are reported in the Supplementary Material of Overstall & Woods (2017), and if type = "BoxDraper", the designs returned are the exact D-optimal designs.

Author(s)

Antony M. Overstall <A.M.Overstall@soton.ac.uk>, David C. Woods & Maria Adamou

References

Atkinson, A., Chaloner, K., Herzberg, A., & Juritz, J. (1993). Experimental Designs for Properties of a Compartmental Model. *Biometrics*, **49**, 325-337.

Bliss, C. (1935). The calculation of the dosage-mortality curve. *Annals of Applied Biology*, **22**, 134-167.

Box, M. & Draper, N. (1971). Factorial designs, the $|F^T F|$ criterion and some related matters. *Techometrics*, **13**, 731-742.

Gotwalt, C., Jones, B. & Steinberg, D. (2009). Fast Computation of Designs Robust to Parameter Uncertainty for Nonlinear Settings. *Technometrics*, **51**, 88-95.

O'Hagan, A, & Forster, J.J. (2004). *Kendall's Advanced Theory of Statistics, Volume 2B: Bayesian Inference. 2nd edition*. John Wiley & Sons.

Overstall, A.M. & Woods, D.C. (2017). Bayesian design of experiments using approximate coordinate exchange. *Technometrics*, **59**, 458-470.

Ryan, E., Drovandi, C., Thompson, M., Pettitt, A. (2014). Towards Bayesian experimental design for nonlinear models that require a large number of sampling times. *Computational Statistics and Data Analysis*, **70**, 45-60.

Woods, D.C., Lewis, S., Eccleston, J., Russell, K. (2006). Designs for Generalized Linear Models With Several Variables and Model Uncertainty. *Technometrics*, **48**, 284-292.

See Also

ace, pace.

Examples

```
######## Compartmental model ##################################

set.seed(1)
## Set seed for reproducibility

d<-optimumLHS(n = 18, k = 1) * 2 - 1
## Generate an 18-run design.

u<-utilcomp18bad(d = d, B = 20000)
## Calculate the D-optimal utility function for a
## sample of size 20000.

u[1:5]
## Look at first 5 elements.
#[1] 14.283473 10.525390  4.126233  7.061498 12.793245

d0<-optdescomp18bad( )
u0<-utilcomp18bad(d = d0, B = 20000)
## Optimal design found by ACE and calculate the D-optimal
## utility function for a sample of size 20000.

u0[1:5]
## Look at first 5 elements.
#[1] 15.04721 15.37141 16.84287 14.06750 14.01523

mean(u)
mean(u0)
## Caluclate expected Bayesian D-optimal utility.

d<-matrix(runif(2), ncol = 1)
## Generate two beta parameters.
```

```
u<-utilcomp15sigDRS(d = d, B = 5)
u
## Calculate the SIG utility function for a
## sample of size 5.
#[1] 17.652044  4.878998 19.919559 22.017760  5.600473

######## Logistic regression model ############################

set.seed(1)
## Set seed for reproducibility

d<-optimumLHS(n = 16,k = 4) * 2 - 1
## Generate an 16-run design.

u<-utillrbad(d = d, B = 20000)
## Calculate the D-optimal utility function for a
## sample of size 20000.

u[1:5]
## Look at first 5 elements.
#[1] -11.426457  -5.925467  -9.238526  -9.392765  -7.689553

d0<-optdeslrbad(16)
u0<-utillrbad(d = d0, B = 20000)
## Optimal design found by ACE and calculate the D-optimal
## utility function for a sample of size 20000.

u0[1:5]
## Look at first 5 elements.
#[1] -4.644116 -2.411431 -4.999891 -2.906558 -2.282687

mean(u)
mean(u0)
## Calculate expected Bayesian D-optimal utility.
#[1] -9.349215
#[1] -2.992012

######## Beetle mortality experiment ##########################

set.seed(1)
## Set seed for repoducibility

d<-optimumLHS(n = 10, k = 1)*2-1
## Generate a design of 10 doses with elements in [-1, 1]

utilbeetle(d = d, B = 5)
## Calculate the utility function for a sample of size 5.

#-4.085626e-06 -1.209499e-06 -1.214244e-05 -6.243750e-06 -3.083291e-05

d0<-optdesbeetle(10)
d0
```

```
## Print out optimal design from Overstall & Woods (2017) for 10 doses

0.5*( d0 + 1)*( 1.8839 - 1.6907 ) + 1.6907
## On original scale.
#          [,1]
# [1,] 1.769957
# [2,] 1.769520
# [3,] 1.768641
# [4,] 1.777851
# [5,] 1.768641
# [6,] 1.769520
# [7,] 1.777851
# [8,] 1.765997
# [9,] 1.768641
#[10,] 1.768641

######## Linear model ########################################

set.seed(1)
## Set seed for reproducibility

d<-cbind(rep(c( -1, 0, 1), each = 3), rep(c( -1, 0, 1), 3))
## Create a 9-run design which is the true D-optimal design

utillinmod(d = d, B = 5)
## Calculate the approximation to the true expected D-optimal utility
## function for a sample of size 5.

#[1]  7.926878  8.736976  7.717704 10.148613  8.882840

d0<-optdeslinmod(9)
## Optimal D-optimal design found using ACE

X<-cbind(1, d, d^2, d[,1] * d[,2])
X0<-cbind(1, d0, d0^2, d0[,1] * d0[,2])
## Calculate model matrices under both designs

detX<-determinant( t(X) %*% X)$modulus[1]
detX0<-determinant( t(X0) %*% X0)$modulus[1]
## Calculate true expected D-optimal utility function for both designs

100 * exp( 0.2 * ( detX0 - detX ))
## Calculate D-efficiency of ACE design.

# 99.93107
```

---

plot.ace *Plot* ace *Objects*

Description

This function plots objects of class "ace" or "pace" .

Usage

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

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

Arguments

x	An object of class "ace" or "pace".
...	Arguments to be passed to and from other methods.

Value

A trace plot of the current evaluations of the approximate expected utility function. Separate lines are produced for the traces from Phases I and II of the ACE algorithm.

For objects of class "pace", the evaluations of the approximate expected utility function are from the repetition which resulted in the terminal design (see pace, paceglm, and pacenlm for more details).

These trace plots can be used to assess convergence. See Overstall & Woods (2017) for more details.

Note

For an example see ace.

Author(s)

Antony M. Overstall <A.M.Overstall@soton.ac.uk>, David C. Woods & Maria Adamou

References

Overstall, A.M. & Woods, D.C. (2017). Bayesian design of experiments using approximate coordinate exchange. *Technometrics*, **59**, 458-470.

See Also

ace, pace

plot.assess *Plot* assess *Objects*

Description

This function plots objects of class "assess".

Usage

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

Arguments

x	An object of class "assess".
...	Arguments to be passed to and from other methods.

Value

In the case of when d1 was generated from a call to (p)ace with argument deterministic = FALSE or from a call to (p)aceglm or (p)acenlm with argument method being "MC", then boxplots of the n.assess evaluations of the approximate expected utility for each of the designs d1 and d2 will be produced. Otherwise, a plot is not meaningful and a warning will be produced.

Author(s)

Antony M. Overstall <A.M.Overstall@soton.ac.uk>, David C. Woods & Maria Adamou

See Also

assess

utilities *Approximate expected utility function for generalised linear models and non-linear regression models*

Description

Generates an approximate utility function for generalised linear models and non-linear regression models.

Usage

```
utilityglm(formula, family, prior,
criterion = c("D", "A", "E", "SIG", "NSEL", "SIG-Norm", "NSEL-Norm"),
method = c("quadrature", "MC"), nrq)

utilitynlm(formula, prior, desvars, criterion = c("D", "A", "E", "SIG", "NSEL"),
method = c("quadrature", "MC"), nrq)
```

Arguments

formula	An argument providing a symbolic description of the model. For utilityglm, an object of class "formula": a symbolic description of the model. For utilitynlm, an object of class "formula": a symbolic description of the model. The terms should correspond to the argument prior
family	For utilityglm, a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See family for details of family functions.)
prior	An argument specifying the prior distribution. For method = "MC", a function with one argument: B; a scalar integer. This function should return a B by p matrix (p+1 for criterion = "SIG" or criterion = "NSEL"), with p the number of model parameters, containing a random sample from the prior distribution of the parameters. The value of p should correspond to the number of terms specified by the formula argument. For utilitynlm, the column names must match the names of parameters in the formula argument. For utilitynlm, if criterion="SIG", criterion="NSEL", criterion="SIG-Norm" or criterion="NSEL-Norm" then an extra column called sig2 should be included with a sample from the error variance. For method = "quadrature", a list specifying a normal or uniform prior for the model parameters. For a normal prior distribution, the list should have named entries mu and sigma2 specifying the prior mean and variance-covariance matrix. The prior mean may be specified as a scalar, which will then be replicated to form an vector of common prior means, or a vector of length p. The prior variance-covariance matrix may be specified as either a scalar common variance or a vector of length p of variances (for independent prior distributions) or as a p by p matrix. For utilitynlm, the names attribute of mu must match the names of the parameters in the formula argument. For a uniform prior distribution, the list should have a named entry support, a 2 by p matrix with each column giving the lower and upper limits of the support of the independent continuous uniform distribution for the corresponding parameter. For utilitynlm, the column names of support must match the names of parameters in the formula argument.
desvars	For utilitynlm, a character vector listing the design variables that appear in the argument formula.
criterion	An optional character argument specifying the utility function. There are currently seven utility functions implemented as follows:

1. **pseudo-Bayesian D-optimality** (`criterion = "D"`);
2. **pseudo-Bayesian A-optimality** (`criterion = "A"`);
3. **pseudo-Bayesian E-optimality** (`criterion = "E"`).
4. **Shannon information gain** with Monte Carlo (MC) approximation to marginal likelihood (`criterion = "SIG"`);
5. **Shannon information gain** with normal-based Laplace approximation to marginal likelihood (`criterion = "SIG-Norm"`, only for `utilityglm`));
6. **negative squared error loss** with importance sampling approximation to posterior mean (`criterion = "NSEL"`);
7. **negative squared error loss** with normal-based approximation to posterior mean (`criterion = "NSEL-Norm"`, only for `utilityglm`)) ;

The default value is `"D"` denoting pseudo-Bayesian D-optimality. See **Details** for more information.

`method` An optional character argument specifying the method of approximating the expected utility function. Current choices are `method = "quadrature"` for a deterministic quadrature approximation and `method = "MC"` for a stochastic Monte Carlo approximation. The first of these choices is **only** available when the argument `criterion = "A"`, `"D"` or `"E"`. The second choice is available for all possible values of the argument `criterion`. If left unspecified, the argument defaults to `"quadrature"` for `criterion = "A"`, `"D"` or `"E"` and to `"MC"` otherwise. See **Details** for more information.

`nrq` For `method = "quadrature"`, a vector of length two specifying the number of radial abscissas (`nrq[1]`) and quasi-random rotations (`nrq[2]`) required for the implemented quadrature scheme; see **Details** for more information. If left unspecified, the default value is `c(2, 8)`.

Details

Two utility functions are implemented.

1. **Shannon information gain (SIG)**
   The utility function is
   $$u^{SIG}(d) = \pi(\theta|y,d) - \pi(\theta),$$
   where $\pi(\theta|y,d)$ and $\pi(\theta)$ denote the posterior and prior densities of the parameters θ, respectively.
2. **Negative squared error loss (NSEL)**
   The utility function is
   $$u^{NSEL}(d) = -\left(\theta - E(\theta|y,d)\right)^T \left(\theta - E(\theta|y,d)\right),$$
   where $E(\theta|y,d)$ denotes the posterior mean of θ.

In both cases the utility function is not available in closed form due to the analytical intractability of either the posterior distribution (for SIG) or the posterior mean (for NSEL). The `acebayes` package implements two approximations to both utility functions. If `criterion = "SIG"` or `criterion = "NSEL"` then sampling-based Monte Carlo or importance sampling approximations will be employed. This was the original approach used by Overstall & Woods (2017). If

`criterion = "SIG-Norm"` or `criterion = "NSEL-Norm"` then approximations based on approximate normality of the posterior (Overstall et al., 2017) will be used.

The normal approximation to the posterior can be taken further leading to the approximation by some scalar function of the Fisher information matrix, $\mathcal{I}(\theta; d)$, which only depends on θ (Chaloner & Verdinelli, 1995). In the case of SIG, the approximate utility is given by

$$u^D(d) = \log |\mathcal{I}(\theta; d)|,$$

and the resulting design is typically called pseudo-Bayesian D-optimal. For NSEL, the approximate utility is given by

$$u^A(d) = -\text{tr}\left\{\mathcal{I}(\theta; d)^{-1}\right\}$$

with the resulting design termed pseudo-Bayesian A-optimal. These designs are often used under the frequentist approach to optimal experimental design and so to complete the usual set, the following utility for finding a pseudo-Bayesian E-optimal design is also implemented:

$$u^E(d) = \min e\left(\mathcal{I}(\theta; d)\right),$$

where $e()$ denotes the function that calculates the eigenvalues of its argument.

The expected utilities can be approximated using Monte Carlo methods (`method = "MC"` for all criteria) or using a deterministic quadrature method (`method = "quadrature"`, implemented for the D, A and E criteria). The former approach approximates the expected utility via sampling from the prior. The latter approach uses a radial-spherical integration rule (Monahan and Genz, 1997) and `B[1]` specifies the number, n_r, of radial abscissas and `B[2]` specifies the number, n_q, of random rotations. Larger values of n_r will produce more accurate, but also more computationally expensive, approximations. See Gotwalt et al. (2009) for further details.

For `utilityglm`, note that the utility functions for SIG and NSEL are currently only implemented for logistic regression, i.e. `family = binomial`, or Poisson regression, i.e. `family = poisson(link = "log")`, whereas the utility functions for pseudo-Bayesian designs are implemented for generic GLM families.

For more details on the ACE algorithm, see Overstall & Woods (2017).

Value

The function will return a list with the following components:

`utility` The utility function resulting from the choice of arguments.

Author(s)

Antony M. Overstall <A.M.Overstall@soton.ac.uk>, David C. Woods & Maria Adamou

References

Chaloner, K. & Verdinelli, I. (1995). Bayesian experimental design: a review. *Statistical Science*, **10**, 273-304.

Gotwalt, C. M., Jones, B. A. & Steinberg, D. M. (2009). Fast computation of designs robust to parameter uncertainty for nonlinear settings. *Technometrics*, **51**, 88-95.

Monahan, J. and Genz, A. (1997). Spherical-radial integration rules for Bayesian computation," Journal of the American Statistical Association, 92, 664-674.

Overstall, A.M. & Woods, D.C. (2017). Bayesian design of experiments using approximate coordinate exchange. *Technometrics*, **59**, 458-470.

See Also

aceglm, acenlm, paceglm, pacenlm.

Examples

```
## 1. This example uses utilityglm to generate the pseudo-Bayesian D-optimality
## approximate expected utility function using a Monte Carlo approximation.

low<-c(-3, 4, 5, -6, -2.5)
upp<-c(3, 10, 11, 0, 3.5)
## Lower and upper limits of the uniform prior distributions.

prior<-function(B){
t(t(6*matrix(runif(n=5*B),ncol=5))+low)}
## Create a function which specifies the prior. This function will return a
## B by 5 matrix where each row gives a value generated from the prior
## distribution for the model parameters.

ex <- utilityglm(formula = ~x1+x2+x3+x4, family = binomial, prior = prior, method = "MC")

set.seed(1)
## Set seed for reproducibility.

n<-6
## Specify the sample size (number of runs).

start.d<-matrix( 2 * randomLHS(n = n,k = 4) - 1,nrow = n,ncol = 4,
dimnames = list(as.character(1:n),c("x1", "x2", "x3", "x4")))
## Generate an initial design of appropriate dimension. The initial design is a
## Latin hypercube sample.

ex$utility(d = start.d, B = 10)
## Evaluate resulting approximate utility. Should get:

## [1] -14.01218 -18.53715 -20.78817 -24.02731 -14.89368 -14.25400 -17.38152
## [8] -17.70984 -14.15254 -21.40505

## 2. This example uses utilitynlm to generate the psuedo-Bayesian A-optimality expected utility
## function using a quadrature approximation

low<-c(0.01884, 0.298, 21.8)
upp<-c(0.09884, 8.298, 21.8)
## Lower and upper limits of the uniform prior distributions. Note that the prior
## for the third element is a point mass.

prior2 <- list(support = cbind(rbind(low, upp)))
```

```
colnames(prior2$support) <- c("a", "b", "c")
## Specify a uniform prior with ranges given by low and upp

ex2 <- utilitynlm(formula = ~ c * (exp( - a * t) - exp( - b *t)), prior = prior2,
                            desvars = "t")

n <- 6
start.d <- matrix(24 * randomLHS(n = n, k = 1), nrow = n)
colnames(start.d) <- "t"
ex2$utility(d = start.d)
## -13.17817
```

Index