Package ‘ICAOD’

January 20, 2025

Title Optimal Designs for Nonlinear Statistical Models by Imperialist Competitive Algorithm (ICA)

Version 1.0.1

Description Finds optimal designs for nonlinear models using a metaheuristic algorithm called Imperialist Competitive Algorithm (ICA). See, for details, Masoudi et al. (2017) <doi:10.1016/j.csda.2016.06.014> and Masoudi et al. (2019) <doi:10.1080/10618600.2019.1601097>.

Depends R (>= 3.1.3)

License GPL (>= 2)

LazyData true

LinkingTo Rcpp, RcppEigen,

Imports Rcpp, nloptr, stats, utils, graphics, grDevices, cubature, sn, mnormt, methods, mvQuad

Suggests rgl, lattice, R.rsp

RoxygenNote 7.1.1

Encoding UTF-8

Language en-US

NeedsCompilation yes

Author Ehsan Masoudi [aut, cre],
Heinz Holling [aut],
Weng Kee Wong [aut],
Seongho Kim [ctb]

Maintainer Ehsan Masoudi <esn_mud@yahoo.com>

Repository CRAN

Date/Publication 2020-10-11 19:20:02 UTC

Contents

bayes *Bayesian D-Optimal Designs*

Description

Finds (pseudo) Bayesian D-optimal designs for linear and nonlinear models. It should be used when the user assumes a (truncated) prior distribution for the unknown model parameters. If you have a discrete prior, please use the function `robust`.

Usage

```
bayes(
  formula,
  predvars,
  parvars,
  family = gaussian(),
  prior,
  lx,
  ux,
  iter,
  k,
  fimfunc = NULL,
  ICA.control = list(),
  sens.control = list(),
  crt.bayes.control = list(),
  sens.bayes.control = list(),
  initial = NULL,
  npar = NULL,
  plot_3d = c("lattice", "rgl"),
  x = NULL,
  crtfunc = NULL,
  sensfunc = NULL
)
```

Arguments

`formula`	A linear or nonlinear model `formula`. A symbolic description of the model consists of predictors and the unknown model parameters. Will be coerced to a `formula` if necessary.
`predvars`	A vector of characters. Denotes the predictors in the `formula`.
`parvars`	A vector of characters. Denotes the unknown parameters in the `formula`.
`family`	A description of the response distribution and the link function to be used in the model. This can be a family function, a call to a family function or a character string naming the family. Every family function has a link argument allowing to specify the link function to be applied on the response variable. If not specified, default links are used. For details see `family`. By default, a linear gaussian model `gaussian()` is applied.

prior	An object of class cprior. User can also use one of the functions uniform, normal, skewnormal or student to create the prior. See 'Details' of bayes.
lx	Vector of lower bounds for the predictors. Should be in the same order as predvars.
ux	Vector of upper bounds for the predictors. Should be in the same order as predvars.
iter	Maximum number of iterations.
k	Number of design points. Must be at least equal to the number of model parameters to avoid singularity of the FIM.
fimfunc	A function. Returns the FIM as a matrix. Required when formula is missing. See 'Details' of minimax.
ICA.control	ICA control parameters. For details, see ICA.control.
sens.control	Control Parameters for Calculating the ELB. For details, see sens.control.
crt.bayes.control	A list. Control parameters to approximate the integral in the Bayesian criterion at a given design over the parameter space. For details, see crt.bayes.control.
sens.bayes.control	A list. Control parameters to verify the general equivalence theorem. For details, see sens.bayes.control.
initial	A matrix of the initial design points and weights that will be inserted into the initial solutions (countries) of the algorithm. Every row is a design, i.e. a concatenation of x and w. Will be coerced to a matrix if necessary. See 'Details' of minimax.
npar	Number of model parameters. Used when fimfunc is given instead of formula to specify the number of model parameters. If not specified correctly, the sensitivity (derivative) plot may be shifted below the y-axis. When NULL (default), it will be set to length(parvars) or prior$npar when missing(formula).
plot_3d	Which package should be used to plot the sensitivity (derivative) function for two-dimensional design space. Defaults to "lattice".
x	A vector of candidate design (support) points. When is not set to NULL (default), the algorithm only finds the optimal weights for the candidate points in x. Should be set when the user has a finite number of candidate design points and the purpose is to find the optimal weight for each of them (when zero, they will be excluded from the design). For design points with more than one dimension, see 'Details' of sensminimax.
crtfunc	(Optional) a function that specifies an arbitrary criterion. It must have especial arguments and output. See 'Details' of bayes.
sensfunc	(Optional) a function that specifies the sensitivity function for crtfunc. See 'Details' of bayes.

Details

Let Ξ be the space of all approximate designs with k design points (support points) at $x_1, x_2, ..., x_k$ from design space χ with corresponding weights $w_1, ..., w_k$. Let $M(\xi, \theta)$ be the Fisher information matrix (FIM) of a $k-$point design ξ and $\pi(\theta)$ is a user-given prior distribution for the vector of unknown parameters θ. A Bayesian D-optimal design ξ^* minimizes over Ξ

$$\int_{\theta\in\Theta} -\log|M(\xi,\theta)|\pi(\theta)d\theta.$$

An object of class `cprior` is a list with the following components:

- `fn`: Prior distribution as an R `function` with argument `param` that is the vector of the unknown parameters. See below.
- `npar`: Number of unknown parameters and is equal to the length of `param`.
- `lower`: Argument `lower`. It has the same length as `param`.
- `upper`: Argument `upper`. It has the same length as `param`.

A `cprior` object will be passed to the argument `prior` of the function `bayes`. The argument `param` in `fn` has the same order as the argument `parvars` when the model is specified by a `formula`. Otherwise, it is the same as the argument `param` in the function `fimfunc`.
The user can use the implemented priors that are `uniform`, `normal`, `skewnormal` and `student` to create a `cprior` object.

To verify the equivalence theorem of the output design, use `plot` function or change the value of the `checkfreq` in the argument `ICA.control`.

To increase the speed of the algorithm, change the value of the tuning parameters `tol` and `maxEval` via the argument `crt.bayes.control` when `crt.bayes.control$method = "cubature"`. Similarly, this applies when `crt.bayes.control$method = "quadrature"`. In general, if the CPU time matters, the user should find an appropriate speed-accuracy trade-off for her/his own problem. See 'Examples' for more details.

If some of the parameters are known and fixed, they should be set to their values via the argument `paravars` when the model is given by `formula`. In this case, the user must provide the number of parameters via the argument `npar` for verifying the general equivalence theorem. See 'Examples', Section 'Weibull', 'Richards' and 'Exponential' model.

`crtfunc` is a function that is used to specify a new criterion. Its arguments are:

- design points `x` (as a `vector`).
- design weights `w` (as a `vector`).
- model parameters as follows.
  - If `formula` is specified: they should be the same parameter specified by `parvars`. Note that `crtfunc` must be vectorized with respect to the parameters. The parameters enter the body of `crtfunc` as a `vector` with dynamic length.
  - If FIM is specified via the argument `fimfunc`: `param` that is a matrix where its row is a vector of parameters values.
- `fimfunc` is a `function` that takes the other arguments of `crtfunc` and returns the computed Fisher information matrices for each parameter vector. The output is a list of matrices.

The `crtfunc` function must return a vector of criterion values associated with the vector of parameter values. The `sensfunc` is the optional sensitivity function for the criterion `crtfunc`. It has one more argument than `crtfunc`, which is `xi_x`. It denotes the design point of the degenerate design and must be a vector with the same length as the number of predictors. For more details, see 'Examples'.

Value

an object of class `minimax` that is a list including three sub-lists:

`arg` A list of design and algorithm parameters.

`evol` A list of length equal to the number of iterations that stores the information about the best design (design with the minimum criterion value) of each iteration as follows: `evol[[iter]]` contains:

`iter`	Iteration number.
`x`	Design points.
`w`	Design weights.
`min_cost`	Value of the criterion for the best imperialist (design).
`mean_cost`	Mean of the criterion values of all the imperialists.
`sens`	An object of class `'sensminimax'`. See below.

`empires` A list of all the empires of the last iteration.

`alg` A list with following information:

`nfeval`	Number of function evaluations. It does not count the function evaluations from checking the general equiv
`nlocal`	Number of successful local searches.
`nrevol`	Number of successful revolutions.
`nimprove`	Number of successful movements toward the imperialists in the assimilation step.
`convergence`	Stopped by `'maxiter'` or `'equivalence'`?

`method` A type of optimal designs used.

`design` Design points and weights at the final iteration.

`out` A data frame of design points, weights, value of the criterion for the best imperialist (min_cost), and Mean of the criterion values of all the imperialistsat each iteration (mean_cost).

The list `sens` contains information about the design verification by the general equivalence theorem. See `sensbayes` for more Details. It is only given every `ICA.control$checkfreq` iterations and also the last iteration if `ICA.control$checkfreq >= 0`. Otherwise, `NULL`.

References

Atashpaz-Gargari, E, & Lucas, C (2007). Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. In 2007 IEEE congress on evolutionary computation (pp. 4661-4667). IEEE.
Masoudi E, Holling H, Duarte BP, Wong Wk (2019). Metaheuristic Adaptive Cubature Based Algorithm to Find Bayesian Optimal Designs for Nonlinear Models. Journal of Computational and Graphical Statistics. <doi:10.1080/10618600.2019.1601097>

See Also

`sensbayes`

Examples

```
#############################################
# Two parameter logistic model: uniform prior
#############################################
# set the unfirom prior
uni <- uniform(lower =  c(-3, .1), upper = c(3, 2))
# set the logistic model with formula
res1 <- bayes(formula = ~1/(1 + exp(-b *(x - a))),
              predvars = "x", parvars = c("a", "b"),
              family = binomial(), lx = -3, ux = 3,
              k =  5, iter = 1, prior = uni,
              ICA.control = list(rseed = 1366))

## Not run:
  res1 <- update(res1, 500)
  plot(res1)

## End(Not run)
# You can also use your  Fisher information matrix (FIM) if you think it is faster!
## Not run:
  bayes(fimfunc = FIM_logistic, lx = -3, ux = 3, k =  5, iter = 500,
        prior = uni, ICA.control = list(rseed = 1366))

## End(Not run)

# with fixed x
## Not run:
  res1.1 <- bayes(formula = ~1/(1 + exp(-b *(x - a))),
                  predvars = "x", parvars = c("a", "b"),
                  family = binomial(), lx = -3, ux = 3,
                  k =  5, iter = 100, prior = uni,
                  x = c( -3, -1.5, 0,  1.5, 3),
                  ICA.control = list(rseed = 1366))
  plot(res1.1)
  # not optimal

## End(Not run)

# with quadrature formula
## Not run:
  res1.2 <- bayes(formula = ~1/(1 + exp(-b *(x - a))),
                  predvars = "x", parvars = c("a", "b"),
                  family = binomial(), lx = -3, ux = 3,
                  k =  5, iter = 1, prior = uni,
                  crt.bayes.control = list(method = "quadrature"),
                  ICA.control = list(rseed = 1366))
  res1.2 <- update(res1.2, 500)
  plot(res1.2) # not optimal
  # compare it with res1 that was found by automatic integration
  plot(res1)

  # we increase the number of quadrature nodes
```

```
  res1.3 <- bayes(formula = ~1/(1 + exp(-b *(x - a))),
                  predvars = "x", parvars = c("a", "b"),
                  family = binomial(), lx = -3, ux = 3,
                  k =  5, iter = 1, prior = uni,
                  crt.bayes.control = list(method = "quadrature",
                                           quadrature = list(level = 9)),
                  ICA.control = list(rseed = 1366))
  res1.3 <- update(res1.3, 500)
  plot(res1.3)
  # by automatic integration (method = "cubature"),
  #  we did not need to worry about the number of nodes.

## End(Not run)
###############################################
# Two parameter logistic model: normal prior #1
###############################################
# defining the normal prior #1
norm1 <- normal(mu =  c(0, 1),
                sigma = matrix(c(1, -0.17, -0.17, .5), nrow = 2),
                lower =  c(-3, .1), upper = c(3, 2))
## Not run:
  # initializing
 res2 <- bayes(formula = ~1/(1 + exp(-b *(x - a))), predvars = "x", parvars = c("a", "b"),
                family = binomial(), lx = -3, ux = 3, k =  4, iter = 1, prior = norm1,
                ICA.control = list(rseed = 1366))
  res2 <- update(res2, 500)
  plot(res2)

## End(Not run)

###############################################
# Two parameter logistic model: normal prior #2
###############################################
# defining the normal prior #1
norm2 <- normal(mu =  c(0, 1),
                sigma = matrix(c(1, 0, 0, .5), nrow = 2),
                lower =  c(-3, .1), upper = c(3, 2))
## Not run:
  # initializing
 res3 <- bayes(formula = ~1/(1 + exp(-b *(x - a))), predvars = "x", parvars = c("a", "b"),
                family = binomial(), lx = -3, ux = 3, k =  4, iter = 1, prior = norm2,
                ICA.control = list(rseed = 1366))

  res3 <- update(res3, 700)
  plot(res3,
       sens.bayes.control = list(cubature = list(maxEval = 3000, tol = 1e-4)),
       sens.control = list(optslist = list(maxeval = 3000)))

## End(Not run)


######################################################
# Two parameter logistic model: skewed normal prior #1
```

```
######################################################
skew1 <- skewnormal(xi = c(0, 1),
                    Omega = matrix(c(1, -0.17, -0.17, .5), nrow = 2),
                    alpha = c(1, 0), lower =  c(-3, .1), upper = c(3, 2))
## Not run:
  res4 <- bayes(formula = ~1/(1 + exp(-b *(x - a))), predvars = "x", parvars = c("a", "b"),
                family = binomial(), lx = -3, ux = 3, k =  4, iter = 700, prior = skew1,
                ICA.control = list(rseed = 1366, ncount = 60))
  plot(res4,
       sens.bayes.control = list(cubature = list(maxEval = 3000, tol = 1e-4)),
       sens.control = list(optslist = list(maxeval = 3000)))

## End(Not run)


######################################################
# Two parameter logistic model: skewed normal prior #2
######################################################
skew2 <- skewnormal(xi = c(0, 1),
                    Omega = matrix(c(1, -0.17, -0.17, .5), nrow = 2),
                    alpha = c(-1, 0), lower =  c(-3, .1), upper = c(3, 2))
## Not run:
  res5 <- bayes(formula = ~1/(1 + exp(-b *(x - a))), predvars = "x", parvars = c("a", "b"),
                family = binomial(), lx = -3, ux = 3, k =  4, iter = 700, prior = skew2,
                ICA.control = list(rseed = 1366, ncount = 60))
  plot(res5,
       sens.bayes.control = list(cubature = list(maxEval = 3000, tol = 1e-4)),
       sens.control = list(optslist = list(maxeval = 3000)))

## End(Not run)

###############################################
# Two parameter logistic model: t student prior
###############################################
# set the prior
stud <- student(mean =  c(0, 1), S   = matrix(c(1, -0.17, -0.17, .5), nrow = 2),
                df = 3, lower =  c(-3, .1), upper = c(3, 2))
## Not run:
  res6 <- bayes(formula = ~1/(1 + exp(-b *(x - a))), predvars = "x", parvars = c("a", "b"),
                family = binomial(), lx = -3, ux = 3, k =  5, iter = 500, prior = stud,
                ICA.control = list(ncount = 50, rseed = 1366))
  plot(res6)

## End(Not run)
# not bad, but to find a very accurate designs we increase
# the ncount to 200 and repeat the optimization
## Not run:
  res6 <- bayes(formula = ~1/(1 + exp(-b *(x - a))),
                predvars = "x", parvars = c("a", "b"),
                family = binomial(), lx = -3, ux = 3, k =  5, iter = 1000, prior = stud,
                ICA.control = list(ncount = 200,  rseed = 1366))
  plot(res6)
```

```
## End(Not run)


#############################################
# 4-parameter sigmoid Emax model: unform prior
#############################################
lb <- c(4, 11, 100, 5)
ub <- c(8, 15, 130, 9)
## Not run:
 res7 <- bayes(formula = ~ theta1 + (theta2 - theta1)*(x^theta4)/(x^theta4 + theta3^theta4),
               predvars = c("x"), parvars = c("theta1", "theta2", "theta3", "theta4"),
               lx = .001, ux = 500, k = 5, iter = 200, prior = uniform(lb, ub),
               ICA.control = list(rseed = 1366, ncount = 60))
  plot(res7,
       sens.bayes.control = list(cubature = list(maxEval = 500, tol = 1e-3)),
       sens.control = list(optslist = list(maxeval = 500)))

## End(Not run)

#######################################################################
# 2-parameter Cox Proportional-Hazards Model for type one cenosred data
#######################################################################
# The Fisher information matrix is available here with name FIM_2par_exp_censor1
# However, we should reparameterize the function to match the standard of the argument 'fimfunc'
myfim <- function(x, w, param)
  FIM_2par_exp_censor1(x = x, w = w, param = param, tcensor = 30)
## Not run:
  res8 <- bayes(fimfunc = myfim, lx = 0, ux = 1, k = 4,
               iter = 1, prior = uniform(c(-11, -11), c(11, 11)),
               ICA.control = list(rseed = 1366))

  res8 <- update(res8, 200)
  plot(res8,
       sens.bayes.control = list(cubature = list(maxEval = 500, tol = 1e-3)),
       sens.control = list(optslist = list(maxeval = 500)))

## End(Not run)


#######################################################################
# 2-parameter Cox Proportional-Hazards Model for random cenosred data
#######################################################################
# The Fisher information matrix is available here with name FIM_2par_exp_censor2
# However, we should reparameterize the function to match the standard of the argument 'fimfunc'
myfim <- function(x, w, param)
  FIM_2par_exp_censor2(x = x, w = w, param = param, tcensor = 30)
## Not run:
  res9 <- bayes(fimfunc = myfim, lx = 0, ux = 1, k = 2,
               iter = 200, prior = uniform(c(-11, -11), c(11, 11)),
               ICA.control = list(rseed = 1366))
  plot(res9,
       sens.bayes.control = list(cubature = list(maxEval = 100, tol = 1e-3)),
       sens.control = list(optslist = list(maxeval = 100)))
```

```
## End(Not run)

##################################
# Weibull model: Uniform prior
#################################
# see Dette, H., & Pepelyshev, A. (2008).
# Efficient experimental designs for sigmoidal growth models.
# Journal of statistical planning and inference, 138(1), 2-17.

## See how we fixed a some parameters in Bayesian designs
## Not run:
  res10 <- bayes(formula = ~a - b * exp(-lambda * t ^h),
                 predvars = c("t"),
                 parvars = c("a=1", "b=1", "lambda", "h=1"),
                 lx = .00001, ux = 20,
                 prior = uniform(.5, 2.5), k = 5, iter = 400,
                 ICA.control = list(rseed = 1366))
  plot(res10)

## End(Not run)

##################################
# Weibull model: Normal prior
#################################
norm3 <- normal(mu = 1, sigma = .1, lower = .5, upper = 2.5)
res11 <- bayes(formula = ~a - b * exp(-lambda * t ^h),
               predvars = c("t"),
               parvars = c("a=1", "b=1", "lambda", "h=1"),
               lx = .00001, ux = 20, prior = norm3, k = 4, iter = 1,
               ICA.control = list(rseed = 1366))

## Not run:
  res11 <- update(res11, 400)
  plot(res11)

## End(Not run)

##################################
# Richards model: Normal prior
##################################
norm4 <- normal(mu = c(1, 1), sigma = matrix(c(.2, 0.1, 0.1, .4), 2, 2),
                lower = c(.4, .4), upper = c(1.6, 1.6))
## Not run:
  res12 <- bayes(formula = ~a/(1 + b * exp(-lambda*t))^h,
                 predvars = c("t"),
                 parvars = c("a=1", "b", "lambda", "h=1"),
                 lx = .00001, ux = 10,
                 prior = norm4,
                 k = 5, iter = 400,
                 ICA.control = list(rseed = 1366))
  plot(res12,
       sens.bayes.control = list(cubature = list(maxEval = 1000, tol = 1e-3)),
```

```
      sens.control = list(optslist = list(maxeval = 1000)))
  ## or we can use the quadrature formula to plot the derivative function
  plot(res12,
      sens.bayes.control = list(method = "quadrature"),
      sens.control = list(optslist = list(maxeval = 1000)))


## End(Not run)

#################################
# Exponential model: Uniform prior
#################################
## Not run:
res13 <- bayes(formula = ~a + exp(-b*x), predvars = "x",
               parvars = c("a = 1", "b"),
               lx = 0.0001, ux = 1,
               prior = uniform(lower = 1, upper = 20),
               iter = 300, k = 3,
               ICA.control= list(rseed = 100))
  plot(res13)

## End(Not run)

#################################
# Power logistic model
#################################
# See, Duarte, B. P., & Wong, W. K. (2014).
# A Semidefinite Programming based approach for finding
# Bayesian optimal designs for nonlinear models
uni1 <- uniform(lower = c(-.3, 6, .5), upper = c(.3, 8, 1))
## Not run:
  res14 <- bayes(formula = ~1/(1 + exp(-b *(x - a)))^s, predvars = "x",
                 parvars = c("a", "b", "s"),
                 lx = -1, ux = 1, prior = uni1, k = 5, iter = 1)
  res14 <- update(res14, 300)
  plot(res14)

## End(Not run)

############################################################################
# A two-variable generalized linear model with a gamma distributed response
############################################################################
lb <- c(.5, 0, 0, 0, 0, 0)
ub <- c(2, 1, 1, 1, 1, 1)
myformula1 <- ~beta0+beta1*x1+beta2*x2+beta3*x1^2+beta4*x2^2+beta5*x1*x2
## Not run:
  res15 <- bayes(formula = myformula1,
                 predvars = c("x1", "x2"), parvars = paste("beta", 0:5, sep = ""),
                 family = Gamma(),
                 lx = rep(0, 2), ux = rep(1, 2),
                 prior = uniform(lower = lb, upper = ub),
                 k = 7,iter = 1, ICA.control = list(rseed = 1366))
  res14 <- update(res14, 500)
```

```
  plot(res14,
       sens.bayes.control = list(cubature = list(maxEval = 5000, tol = 1e-4)),
       sens.control = list(optslist = list(maxeval = 3000)))

## End(Not run)

#################################
# Three parameter logistic model
#################################
## Not run:
sigma1 <- matrix(-0.1, nrow = 3, ncol = 3)
diag(sigma1) <- c(.5, .4, .1)
norm5 <- normal(mu =  c(0, 1, .2), sigma = sigma1,
                lower =  c(-3, .1, 0), upper = c(3, 2, .7))
  res16 <- bayes(formula = ~ c + (1-c)/(1 + exp(-b *(x - a))), predvars = "x",
                 parvars = c("a", "b", "c"),
                 family = binomial(), lx = -3, ux = 3,
                 k =  4, iter = 500, prior = norm5,
                 ICA.control = list(rseed = 1366, ncount = 50),
                 crt.bayes.control = list(cubature = list(maxEval = 2500, tol = 1e-4)))
  plot(res16,
       sens.bayes.control = list(cubature = list(maxEval = 3000, tol = 1e-4)),
       sens.control = list(optslist = list(maxeval = 3000)))
  # took 925 second on my system

## End(Not run)
```

---

bayes.update *Updating an Object of Class* minimax

Description

Runs the ICA optimization algorithm on an object of class minimax for more number of iterations and updates the results.

Usage

```
bayes.update(object, iter, ...)
```

Arguments

object	An object of class minimax.
iter	Number of iterations.
...	An argument of no further use.

See Also

bayes

bayescomp *Bayesian Compound DP-Optimal Designs*

Description

Finds compound Bayesian DP-optimal designs that meet the dual goal of parameter estimation and increasing the probability of a particular outcome in a binary response model. A compound Bayesian DP-optimal design maximizes the product of the Bayesian efficiencies of a design ξ with respect to D- and average P-optimality, weighted by a pre-defined mixing constant $0 \leq \alpha \leq 1$.

Usage

```
bayescomp(
  formula,
  predvars,
  parvars,
  family = binomial(),
  prior,
  alpha,
  prob,
  lx,
  ux,
  iter,
  k,
  fimfunc = NULL,
  ICA.control = list(),
  sens.control = list(),
  crt.bayes.control = list(),
  sens.bayes.control = list(),
  initial = NULL,
  npar = NULL,
  plot_3d = c("lattice", "rgl")
)
```

Arguments

formula	A linear or nonlinear model `formula`. A symbolic description of the model consists of predictors and the unknown model parameters. Will be coerced to a `formula` if necessary.
predvars	A vector of characters. Denotes the predictors in the `formula`.
parvars	A vector of characters. Denotes the unknown parameters in the `formula`.
family	A description of the response distribution and the link function to be used in the model. This can be a family function, a call to a family function or a character string naming the family. Every family function has a link argument allowing to specify the link function to be applied on the response variable. If not specified, default links are used. For details see `family`. By default, a linear gaussian model `gaussian()` is applied.

prior	An object of class cprior. User can also use one of the functions uniform, normal, skewnormal or student to create the prior. See 'Details' of bayes.
alpha	A value between 0 and 1. Compound or combined DP-criterion is the product of the efficiencies of a design with respect to D- and average P- optimality, weighted by alpha.
prob	Either formula or a function. When function, its argument are x and param, and they are the same as the arguments in fimfunc. prob as a function takes the design points and vector of parameters and returns the probability of success at each design points. See 'Examples'.
lx	Vector of lower bounds for the predictors. Should be in the same order as predvars.
ux	Vector of upper bounds for the predictors. Should be in the same order as predvars.
iter	Maximum number of iterations.
k	Number of design points. Must be at least equal to the number of model parameters to avoid singularity of the FIM.
fimfunc	A function. Returns the FIM as a matrix. Required when formula is missing. See 'Details' of minimax.
ICA.control	ICA control parameters. For details, see ICA.control.
sens.control	Control Parameters for Calculating the ELB. For details, see sens.control.
crt.bayes.control	A list. Control parameters to approximate the integral in the Bayesian criterion at a given design over the parameter space. For details, see crt.bayes.control.
sens.bayes.control	A list. Control parameters to verify the general equivalence theorem. For details, see sens.bayes.control.
initial	A matrix of the initial design points and weights that will be inserted into the initial solutions (countries) of the algorithm. Every row is a design, i.e. a concatenation of x and w. Will be coerced to a matrix if necessary. See 'Details' of minimax.
npar	Number of model parameters. Used when fimfunc is given instead of formula to specify the number of model parameters. If not specified correctly, the sensitivity (derivative) plot may be shifted below the y-axis. When NULL (default), it will be set to length(parvars) or prior$npar when missing(formula).
plot_3d	Which package should be used to plot the sensitivity (derivative) function for two-dimensional design space. Defaults to "lattice".

Details

Let Ξ be the space of all approximate designs with k design points (support points) at $x_1, x_2, ..., x_k$ from design space χ with corresponding weights $w_1, ..., w_k$. Let $M(\xi, \theta)$ be the Fisher information matrix (FIM) of a $k-$point design ξ, $\pi(\theta)$ is a user-given prior distribution for the vector of unknown parameters θ and $p(x_i, \theta)$ is the ith probability of success given by x_i in a binary response model.

A compound Bayesian DP-optimal design maximizes over Ξ

$$\int_{\theta \in \Theta} \frac{\alpha}{q} \log|M(\xi, \theta)| + (1-\alpha) \log \left(\sum_{i=1}^{k} w_i p(x_i, \theta) \right) \pi(\theta) d\theta.$$

To verify the equivalence theorem of the output design, use `plot` function or change the value of the `checkfreq` in the argument `ICA.control`.

To increase the speed of the algorithm, change the value of the tuning parameters `tol` and `maxEval` via the argument `crt.bayes.control` when its `method` component is equal to `"cubature"`. In general, if the CPU time matters, the user should find an appropriate speed-accuracy trade-off for her/his own problem. See 'Examples' for more details.

Value

an object of class `minimax` that is a list including three sub-lists:

`arg` A list of design and algorithm parameters.

`evol` A list of length equal to the number of iterations that stores the information about the best design (design with the minimum criterion value) of each iteration as follows: `evol[[iter]]` contains:

`iter`	Iteration number.
`x`	Design points.
`w`	Design weights.
`min_cost`	Value of the criterion for the best imperialist (design).
`mean_cost`	Mean of the criterion values of all the imperialists.
`sens`	An object of class `'sensminimax'`. See below.

`empires` A list of all the empires of the last iteration.

`alg` A list with following information:

`nfeval`	Number of function evaluations. It does not count the function evaluations from checking the general equiv
`nlocal`	Number of successful local searches.
`nrevol`	Number of successful revolutions.
`nimprove`	Number of successful movements toward the imperialists in the assimilation step.
`convergence`	Stopped by `'maxiter'` or `'equivalence'`?

`method` A type of optimal designs used.

`design` Design points and weights at the final iteration.

`out` A data frame of design points, weights, value of the criterion for the best imperialist (min_cost), and Mean of the criterion values of all the imperialistsat each iteration (mean_cost).

The list `sens` contains information about the design verification by the general equivalence theorem. See `sensbayes` for more Details. It is only given every `ICA.control$checkfreq` iterations and also the last iteration if `ICA.control$checkfreq >= 0`. Otherwise, `NULL`.

References

McGree, J. M., Eccleston, J. A., and Duffull, S. B. (2008). Compound optimal design criteria for nonlinear models. Journal of Biopharmaceutical Statistics, 18(4), 646-661.

See Also

sensbayescomp

Examples

```
############################################################################
# DP-optimal design for a logitic model with two predictors: with formula
############################################################################
p <- c(1, -2, 1, -1)
myprior <- uniform(p -1.5, p + 1.5)
myformula1 <- ~exp(b0+b1*x1+b2*x2+b3*x1*x2)/(1+exp(b0+b1*x1+b2*x2+b3*x1*x2))
res1 <- bayescomp(formula = myformula1,
                  predvars = c("x1", "x2"),
                  parvars = c("b0", "b1", "b2", "b3"),
                  family = binomial(),
                  lx = c(-1, -1), ux = c(1, 1),
                  prior = myprior, iter = 1, k = 7,
                  prob = ~1-1/(1+exp(b0 + b1 * x1 + b2 * x2 + b3 * x1 * x2)),
                  alpha = .5, ICA.control = list(rseed = 1366),
                  crt.bayes.control = list(cubature = list(tol = 1e-4, maxEval = 1000)))


## Not run:
  res1 <- update(res1, 1000)
  plot(res1, sens.bayes.control = list(cubature = list(tol = 1e-3, maxEval = 1000)))
  # or use quadrature method
  plot(res1, sens.bayes.control= list(method = "quadrature"))

## End(Not run)

############################################################################
# DP-optimal design for a logitic model with two predictors: with fimfunc
############################################################################
# The function of the Fisher information matrix for this model is 'FIM_logistic_2pred'
# We should reparameterize it to match the standard of the argument 'fimfunc'
## Not run:
myfim <- function(x, w, param){
  npoint <- length(x)/2
  x1 <- x[1:npoint]
  x2 <- x[(npoint+1):(npoint*2)]
  FIM_logistic_2pred(x1 = x1,x2 = x2, w = w, param = param)
}

## The following function is equivalent to the function created
# by the formula: ~1-1/(1+exp(b0 + b1 * x1 + b2 * x2 + b3 * x1 * x2))
# It returns probability of success given x and param
# x = c(x1, x2) and param = c()
```

```
myprob <- function(x, param){
  npoint <- length(x)/2
  x1 <- x[1:npoint]
  x2 <- x[(npoint+1):(npoint*2)]
  b0 <- param[1]
  b1 <- param[2]
  b2 <- param[3]
  b3 <- param[4]
  out <- 1-1/(1+exp(b0 + b1 * x1 + b2 * x2 + b3 * x1 * x2))
  return(out)
}

res2 <- bayescomp(fimfunc = myfim,
                  lx = c(-1, -1), ux = c(1, 1),
                  prior = myprior, iter = 1000, k = 7,
                  prob = myprob, alpha = .5,
                  ICA.control = list(rseed = 1366))
  plot(res2, sens.bayes.control = list(cubature = list(maxEval = 1000, tol = 1e-4)))
  # quadrature with 6 nodes (default)
  plot(res2, sens.bayes.control= list(method = "quadrature"))

## End(Not run)
```

---

beff *Calculates Relative Efficiency for Bayesian Optimal Designs*

Description

Given a prior distribution for the parameters, this function calculates the Bayesian D-and PA- efficiency of a design ξ_1 with respect to a design ξ_2. Usually, ξ_2 is an optimal design. This function is especially useful for investigating the robustness of Bayesian optimal designs under different prior distributions (See 'Examples').

Usage

```
beff(
  formula,
  predvars,
  parvars,
  family = gaussian(),
  prior,
  fimfunc = NULL,
  x2,
  w2,
  x1,
  w1,
```

```
  crt.bayes.control = list(),
  npar = NULL,
  type = c("D", "PA"),
  prob = NULL
)
```

Arguments

formula
: A linear or nonlinear model formula. A symbolic description of the model consists of predictors and the unknown model parameters. Will be coerced to a formula if necessary.

predvars
: A vector of characters. Denotes the predictors in the formula.

parvars
: A vector of characters. Denotes the unknown parameters in the formula.

family
: A description of the response distribution and the link function to be used in the model. This can be a family function, a call to a family function or a character string naming the family. Every family function has a link argument allowing to specify the link function to be applied on the response variable. If not specified, default links are used. For details see family. By default, a linear gaussian model `gaussian()` is applied.

prior
: An object of class `cprior`. User can also use one of the functions uniform, normal, skewnormal or student to create the prior. See 'Details' of bayes.

fimfunc
: A function. Returns the FIM as a `matrix`. Required when `formula` is missing. See 'Details' of minimax.

x2
: Vector of design (support) points of the optimal design (ξ_2). Similar to `x1`.

w2
: Vector of corresponding design weights for `x2`.

x1
: Vector of design (support) points of ξ_1. See 'Details' of leff.

w1
: Vector of corresponding design weights for `x1`.

crt.bayes.control
: A list. Control parameters to approximate the integral in the Bayesian criterion at a given design over the parameter space. For details, see crt.bayes.control.

npar
: Number of model parameters. Used when `fimfunc` is given instead of `formula` to specify the number of model parameters. If not given, the sensitivity plot may be shifted below the y-axis. When `NULL`, it will be set here to `length(inipars)`.

type
: A character. `"D"` denotes the D-efficiency and `"PA"` denotes the average P-efficiency.

prob
: Either `formula` or a `function`. When function, its argument are `x` and `param`, and they are the same as the arguments in `fimfunc`. `prob` as a function takes the design points and vector of parameters and returns the probability of success at each design points. See 'Examples'.

Details

See Masoudi et al. (2018) for formula details (the paper is under review and will be updated as soon as accepted).

The argument `x1` is the vector of design points. For design points with more than one dimension (the models with more than one predictors), it is a concatenation of the design points, but **dimension-wise**. For example, let the model has three predictors (I, S, Z). Then, a two-point optimal design has the following points: $\{\text{point1} = (I_1, S_1, Z_1), \text{point2} = (I_2, S_2, Z_2)\}$. Then, the argument `x` is equal to `x = c(I1, I2, S1, S2, Z1, Z2)`.

Examples

```
##############################
#  2PL model
#############################
formula4.1 <- ~ 1/(1 + exp(-b *(x - a)))
predvars4.1 <- "x"
parvars4.1 <- c("a", "b")

# des4.1 is a list of Bayesian optimal designs with corresponding priors.


des4.1 <- vector("list", 6)
des4.1[[1]]$x <- c(-3, -1.20829, 0, 1.20814, 3)
des4.1[[1]]$w <- c(.24701, .18305, .13988, .18309, .24702)
des4.1[[1]]$prior <- uniform(lower =  c(-3, .1), upper = c(3, 2))

des4.1[[2]]$x <- c(-2.41692, -1.16676, .04386, 1.18506, 2.40631)
des4.1[[2]]$w <- c(.26304, .18231, .14205, .16846, .24414)
des4.1[[2]]$prior <- student(mean =  c(0, 1), S  = matrix(c(1, -0.17, -0.17, .5), nrow = 2),
                             df = 3, lower =  c(-3, .1), upper = c(3, 2))

des4.1[[3]]$x <- c(-2.25540, -.76318, .54628, 2.16045)
des4.1[[3]]$w <- c(.31762, .18225, .18159, .31853)
des4.1[[3]]$prior <- normal(mu =  c(0, 1),
                            sigma = matrix(c(1, -0.17, -0.17, .5), nrow = 2),
                            lower =  c(-3, .1), upper = c(3, 2))

des4.1[[4]]$x <- c(-2.23013, -.66995, .67182, 2.23055)
des4.1[[4]]$w <- c(.31420, .18595, .18581, .31404)
des4.1[[4]]$prior <- normal(mu =  c(0, 1),
                            sigma = matrix(c(1, 0, 0, .5), nrow = 2),
                            lower =  c(-3, .1), upper = c(3, 2))

des4.1[[5]]$x <- c(-1.51175, .12043, 1.05272, 2.59691)
des4.1[[5]]$w <- c(.37679, .14078, .12676, .35567)
des4.1[[5]]$prior <- skewnormal(xi = c(0, 1),
                                Omega = matrix(c(1, -0.17, -0.17, .5), nrow = 2),
                                alpha = c(1, 0), lower =  c(-3, .1), upper = c(3, 2))


des4.1[[6]]$x <- c(-2.50914, -1.16780, -.36904, 1.29227)
des4.1[[6]]$w <- c(.35767, .11032, .15621, .37580)
des4.1[[6]]$prior <- skewnormal(xi = c(0, 1),
                                Omega = matrix(c(1, -0.17, -0.17, .5), nrow = 2),
                                alpha = c(-1, 0), lower =  c(-3, .1), upper = c(3, 2))
```

```
## now we want to find the relative efficiency of
## all Bayesian optimal designs assuming different priors (6 * 6)
eff4.1 <- matrix(NA, 6, 6)
colnames(eff4.1) <- c("uni", "t", "norm1", "norm2", "skew1", "skew2")
rownames(eff4.1) <- colnames(eff4.1)
## Not run:
for (i in 1:6)
  for(j in 1:6)
    eff4.1[i, j] <- beff(formula = formula4.1,
                         predvars = predvars4.1,
                         parvars = parvars4.1,
                         family = binomial(),
                         prior = des4.1[[i]]$prior,
                         x2 = des4.1[[i]]$x,
                         w2 = des4.1[[i]]$w,
                         x1 = des4.1[[j]]$x,
                         w1 = des4.1[[j]]$w)
# For example the first row represents Bayesian D-efficiencies of different
# Bayesian optimal design found assuming different priors with respect to
# the Bayesian D-optimal design found under uniform prior distribution.
  eff4.1

## End(Not run)

##############################
# Relative efficiency for the DP-Compund criterion
#############################
p <- c(1, -2, 1, -1)
prior4.4 <- uniform(p -1.5, p + 1.5)
formula4.4 <- ~exp(b0+b1*x1+b2*x2+b3*x1*x2)/(1+exp(b0+b1*x1+b2*x2+b3*x1*x2))
prob4.4 <- ~1-1/(1+exp(b0 + b1 * x1 + b2 * x2 + b3 * x1 * x2))
predvars4.4 <-  c("x1", "x2")
parvars4.4 <- c("b0", "b1", "b2", "b3")
lb <- c(-1, -1)
ub <- c(1, 1)



## des4.4 is a list of DP-optimal designs found using different values for alpha
des4.4 <- vector("list", 5)
des4.4[[1]]$x <- c(-1, 1)
des4.4[[1]]$w <- c(1)
des4.4[[1]]$alpha <- 0


des4.4[[2]]$x <- c(1, -.62534, .11405, -1, 1, .28175, -1, -1, 1, -1, -1, 1, 1, .09359)
des4.4[[2]]$w <- c(.08503, .43128, .01169, .14546, .05945, .08996, .17713)
des4.4[[2]]$alpha <- .25


des4.4[[3]]$x <- c(-1, .30193, 1, 1, .07411, -1, -.31952, -.08251, 1, -1, 1, -1, -1, 1)
des4.4[[3]]$w <- c(.09162, .10288, .15615, .13123, .01993, .22366, .27454)
```

```
des4.4[[3]]$alpha <- .5

des4.4[[4]]$x <- c(1, -1, .28274, 1, -1, -.19674, .03288, 1, -1, 1, -1, -.16751, 1, -1)
des4.4[[4]]$w <- c(.19040, .24015, .10011, .20527, .0388, .20075, .02452)
des4.4[[4]]$alpha <- .75

des4.4[[5]]$x <- c(1, -1, .26606, -.13370, 1, -.00887, -1, 1, -.2052, 1, 1, -1, -1, -1)
des4.4[[5]]$w <- c(.23020, .01612, .09546, .16197, .23675, .02701, .2325)
des4.4[[5]]$alpha <- 1

# D-efficiency of the DP-optimal designs:
# des4.4[[5]]$x and  des4.4[[5]]$w is the D-optimal design

beff(formula = formula4.4,
     predvars = predvars4.4,
     parvars = parvars4.4,
     family = binomial(),
     prior = prior4.4,
     x2 = des4.4[[5]]$x,
     w2 = des4.4[[5]]$w,
     x1 = des4.4[[2]]$x,
     w1 = des4.4[[2]]$w)

beff(formula = formula4.4,
     predvars = predvars4.4,
     parvars = parvars4.4,
     family = binomial(),
     prior = prior4.4,
     x2 = des4.4[[5]]$x,
     w2 = des4.4[[5]]$w,
     x1 = des4.4[[3]]$x,
     w1 = des4.4[[3]]$w)

beff(formula = formula4.4,
     predvars = predvars4.4,
     parvars = parvars4.4,
     family = binomial(),
     prior = prior4.4,
     x2 = des4.4[[5]]$x,
     w2 = des4.4[[5]]$w,
     x1 = des4.4[[4]]$x,
     w1 = des4.4[[4]]$w)

# must be one!
beff(formula = formula4.4,
     predvars = predvars4.4,
     parvars = parvars4.4,
     family = binomial(),
     prior = prior4.4,
     prob = prob4.4,
     type = "PA",
     x2 = des4.4[[5]]$x,
     w2 = des4.4[[5]]$w,
```

```
    x1 = des4.4[[5]]$x,
    w1 = des4.4[[5]]$w)

## P-efficiency
# reported in Table 4 as eff_P
# des4.4[[1]]$x and  des4.4[[1]]$w is the P-optimal design
beff(formula = formula4.4,
     predvars = predvars4.4,
     parvars = parvars4.4,
     family = binomial(),
     prior = prior4.4,
     prob = prob4.4,
     type = "PA",
     x2 = des4.4[[1]]$x,
     w2 = des4.4[[1]]$w,
     x1 = des4.4[[2]]$x,
     w1 = des4.4[[2]]$w)

beff(formula = formula4.4,
     predvars = predvars4.4,
     parvars = parvars4.4,
     family = binomial(),
     prior = prior4.4,
     prob = prob4.4,
     type = "PA",
     x2 = des4.4[[1]]$x,
     w2 = des4.4[[1]]$w,
     x1 = des4.4[[3]]$x,
     w1 = des4.4[[3]]$w)

beff(formula = formula4.4,
     predvars = predvars4.4,
     parvars = parvars4.4,
     family = binomial(),
     prior = prior4.4,
     prob = prob4.4,
     type = "PA",
     x2 = des4.4[[1]]$x,
     w2 = des4.4[[1]]$w,
     x1 = des4.4[[4]]$x,
     w1 = des4.4[[4]]$w)

beff(formula = formula4.4,
     predvars = predvars4.4,
     parvars = parvars4.4,
     family = binomial(),
     prior = prior4.4,
     prob = prob4.4,
     type = "PA",
     x2 = des4.4[[1]]$x,
     w2 = des4.4[[1]]$w,
     x1 = des4.4[[5]]$x,
     w1 = des4.4[[5]]$w)
```

`crt.bayes.control` *Returns Control Parameters for Approximating Bayesian Criteria*

Description

This function returns two lists each corresponds to an implemented integration method for approximating the integrals in Bayesian criteria. The first list is named `cubature` and contains the `hcubature` control parameters to approximate the integrals with an adaptive multivariate integration method over hypercubes. The second list is named `quadrature` and contains the `createNIGrid` tuning parameters to approximate the integrals with the quadrature methods.

Usage

```
crt.bayes.control(
  method = c("cubature", "quadrature"),
  cubature = list(tol = 1e-05, maxEval = 50000, absError = 0),
 quadrature = list(type = c("GLe", "GHe"), level = 6, ndConstruction = "product",
    level.trans = FALSE)
)
```

Arguments

`method` A character denotes which method to be used to approximate the integrals in Bayesian criteria. `"cubature"` corresponds to the adaptive multivariate integration method using the `hcubature` algorithm (default). `"quadrature"` corresponds the traditional quadrature formulas and calls the function `createNIGrid`.

`cubature` A list that will be passed to the arguments of the function `hcubature` for the adaptive multivariate integration. It is required and used when `crt.bayes.control$method = "cubature"` in the parent function, e.g. `bayes`. See 'Details'.

`quadrature` A list that will be passed to the arguments of the function `createNIGrid` for the quadrature-based integration. It is required and used when `crt.bayes.control$method = "quadrature"` in the parent function, e.g. `bayes`. See 'Details'.

Details

`cubature` is a list that its components will be passed to the function `hcubature` and they are:

`tol` The maximum tolerance. Defaults to `1e-5`.

`maxEval` The maximum number of function evaluations needed. Note that the actual number of function evaluations performed is only approximately guaranteed not to exceed this number. Defaults to `5000`.

absError The maximum absolute error tolerated. Defaults to 0.

One can specify a maximum number of function evaluations. Otherwise, the integration stops when the estimated error is less than the absolute error requested, or when the estimated error is less than tol times the absolute value of the integral, or when the maximum number of iterations is reached, whichever is earlier. cubature is activated when crt.bayes.control$method = "cubature" in any of the parent functions (for example, bayes).

quadrature is a list that its components will be passed to the function createNIGrid and they are:

type Quadrature rule (see Details of createNIGrid) Defaults to "GLe".

level Accuracy level (typically number of grid points for the underlying 1D quadrature rule). Defaults to 6.

ndConstruction Character vector which denotes the construction rule for multidimensional grids. "product" for product rule, returns a full grid (default). "sparse" for combination technique, leads to a regular sparse grid.

level.trans Logical variable denotes either to take the levels as number of grid points (FALSE = default) or to transform in that manner that number of grid points = 2^(levels-1) (TRUE). See, codecreateNIGrid, for details.

quadrature is activated when crt.bayes.control$method = "quadrature" in any of the parent functions (for example, bayes).

Value

A list of two lists each contains the control parameters for hcubature and createNIGrid, respectively.

Examples

```
crt.bayes.control()
crt.bayes.control(cubature = list(tol = 1e-4))
crt.bayes.control(quadrature = list(level = 4))
```

---

crt.minimax.control *Returns Control Parameters for Optimizing Minimax Criteria Over The Parameter Space*

Description

The function crt.minimax.control returns a list of nloptr control parameters for optimizing the minimax criterion over the parameter space.
The key tuning parameter for our application is maxeval. Its value should be increased when either the dimension or the size of the parameter space becomes larger to avoid pre-mature convergence in the inner optimization problem over the parameter space. If the CPU time matters, the user should find an appropriate speed-accuracy trade-off for her/his own design problem.

Usage

```
crt.minimax.control(
  x0 = NULL,
 optslist = list(stopval = -Inf, algorithm = "NLOPT_GN_DIRECT_L", xtol_rel = 1e-06,
    ftol_rel = 0, maxeval = 1000),
  ...
)
```

Arguments

`x0`	Vector of the starting values for the optimization problem (must be from the parameter space).
`optslist`	A list. It will be passed to the argument `opts` of the function `nloptr`. See 'Details'.
`...`	Further arguments will be passed to `nl.opts` from package `nloptr`.

Details

Argument `optslist` will be passed to the argument `opts` of the function `nloptr`:

`stopval` Stop minimization when an objective value <= `stopval` is found. Setting `stopval` to `-Inf` disables this stopping criterion (default).

`algorithm` Defaults to `NLOPT_GN_DIRECT_L`. DIRECT-L is a deterministic-search algorithm based on systematic division of the search domain into smaller and smaller hyperrectangles.

`xtol_rel` Stop when an optimization step (or an estimate of the optimum) changes every parameter by less than `xtol_rel` multiplied by the absolute value of the parameter. Criterion is disabled if `xtol_rel` is non-positive. Defaults to `1e-5`.

`ftol_rel` Stop when an optimization step (or an estimate of the optimum) changes the objective function value by less than `ftol_rel` multiplied by the absolute value of the function value. Criterion is disabled if `ftol_rel` is non-positive. Defaults to `1e-8`.

`maxeval` Stop when the number of function evaluations exceeds `maxeval`. Criterion is disabled if `maxeval` is non-positive. Defaults to `1000`. See below.

A detailed explanation of all the options is shown by `nloptr.print.options()` in package `nloptr`.

Value

A list of control parameters for the function `nloptr`.

Examples

```
crt.minimax.control(optslist = list(maxeval = 2000))
```

FIM_2par_exp_censor1 *Fisher Information Matrix for a 2-Parameter Cox Proportional-Hazards Model for Type One Censored Data*

Description

It provides the cpp function for the FIM introduced in Eq. (3.1) of Schmidt and Schwabe (2015) for type one censored data.

Usage

```
FIM_2par_exp_censor1(x, w, param, tcensor)
```

Arguments

x	Vector of design points.
w	Vector of design weight. Its length must be equal to the length of x and sum(w) = 1.
param	Vector of values for the model parameters $c(\beta_0, \beta_1)$.
tcensor	The experiment is terminated at the fixed time point tcensor.

Value

Fisher information matrix.

References

Schmidt, D., & Schwabe, R. (2015). On optimal designs for censored data. Metrika, 78(3), 237-257.

FIM_2par_exp_censor2 *Fisher Information Matrix for a 2-Parameter Cox Proportional-Hazards Model for Random Censored Data*

Description

It provides the cpp function for the FIM introduced in Eq. (3.1) of Schmidt and Schwabe (2015) for random censored data (type two censored data).

Usage

```
FIM_2par_exp_censor2(x, w, param, tcensor)
```

Arguments

x	Vector of design points.
w	Vector of design weight. Its length must be equal to the length of x and sum(w) = 1.
param	Vector of values for the model parameters $c(\beta_0, \beta_1)$.
tcensor	The experiment is terminated at the fixed time point tcensor.

Value

Fisher information matrix.

References

Schmidt, D., & Schwabe, R. (2015). On optimal designs for censored data. Metrika, 78(3), 237-257.

FIM_3par_exp_censor1 *Fisher Information Matrix for a 3-Parameter Cox Proportional-Hazards Model for Type One Censored Data*

Description

It provides the cpp function for the FIM introduced in Page 247 of Schmidt and Schwabe (2015) for type one censored data.

Usage

```
FIM_3par_exp_censor1(x, w, param, tcensor)
```

Arguments

x	Vector of design points.
w	Vector of design weight. Its length must be equal to the length of x and sum(w) = 1.
param	Vector of values for the model parameters $c(\beta_0, \beta_1, \beta_2)$.
tcensor	The experiment is terminated at the fixed time point tcensor.

Value

Fisher information matrix.

References

Schmidt, D., & Schwabe, R. (2015). On optimal designs for censored data. Metrika, 78(3), 237-257.

FIM_3par_exp_censor2 — *Fisher Information Matrix for a 3-Parameter Cox Proportional-Hazards Model for Random Censored Data*

Description

It provides the cpp function for the FIM introduced in Page 247 of Schmidt and Schwabe (2015) for random censored data (type two censored data).

Usage

```
FIM_3par_exp_censor2(x, w, param, tcensor)
```

Arguments

`x`	Vector of design points.
`w`	Vector of design weight. Its length must be equal to the length of `x` and `sum(w) = 1`.
`param`	Vector of values for the model parameters $(\beta_0, \beta_1, \beta_2)$.
`tcensor`	The experiment is terminated at the fixed time point `tcensor`.

Value

Fisher information matrix.

References

Schmidt, D., & Schwabe, R. (2015). On optimal designs for censored data. Metrika, 78(3), 237-257.

FIM_exp_2par — *Fisher Information Matrix for the 2-Parameter Exponential Model*

Description

It provides the cpp function for FIM for the model `~a + exp(-b*x)`.

Usage

```
FIM_exp_2par(x, w, param)
```

Arguments

- `x` Vector of design points.
- `w` Vector of design weight. Its length must be equal to the length of `x` and `sum(w) = 1`.
- `param` Vector of values for the model parameters `c(a, b)`.

Details

The FIM does not depend on the value of `a`.

Value

Fisher information matrix.

References

Dette, H., & Neugebauer, H. M. (1997). Bayesian D-optimal designs for exponential regression models. Journal of Statistical Planning and Inference, 60(2), 331-349.

Examples

```
FIM_exp_2par(x = c(1, 2), w = c(.5, .5), param = c(3, 4))
```

FIM_kinetics_alcohol *Fisher Information Matrix for the Alcohol-Kinetics Model*

Description

It provides the cpp function for FIM for the model `~(b3 * x1)/(1 + b1 * x1 + b2 * x2)`

Usage

```
FIM_kinetics_alcohol(x1, x2, w, param)
```

Arguments

- `x1` Vector of design points (first dimension).
- `x2` Vector of design points (second dimension).
- `w` Vector of design weight. Its length must be equal to the length of `x` and `sum(w) = 1`.
- `param` Vector of values for the model parameters `c(b1, b2, b3)`.

Value

Fisher information matrix.

`FIM_logistic` *Fisher Information Matrix for the 2-Parameter Logistic (2PL) Model*

Description

It provides the cpp function for FIM for the model `~1/(1 + exp(-b *(x - a)))`. In item response theory (IRT), a is the item difficulty parameter, b is the item discrimination parameter and x is the person ability parameter.

Usage

```
FIM_logistic(x, w, param)
```

Arguments

x	Vector of design points.
w	Vector of design weight. Its length must be equal to the length of `x` and `sum(w) = 1`.
param	Vector of values for the model parameters `c(a, b)`.

Details

It can be shown that minimax and standardized D-optimal designs for the 2PL model is symmetric around point $a_M = (a^L + a^U)/2$ where a^L and a^U are the lower bound and upper bound for parameter a, respectively. In `ICA.control`, arguments `sym` and `sym_point` can be used to specify a_M and find accurate symmetric optimal designs.

Value

Fisher information matrix.

Examples

```
FIM_logistic(x = c(1, 2), w = c(.5, .5), param = c(2, 1))
```

`FIM_logistic_2pred` *Fisher Information Matrix for the Logistic Model with Two Predictors*

Description

It provides the cpp function for FIM for the following model:
`~exp(b0+ b1 * x1 + b2 * x2 + b3 * x1 * x2)/(1 + exp(b0 + b1 * x1 + b2 * x2 + b3 * x1 * x2))`.

Usage

```
FIM_logistic_2pred(x1, x2, w, param)
```

Arguments

- `x1` Vector of design points (for first predictor).
- `x2` Vector of design points (for second predictor).
- `w` Vector of design weight. Its length must be equal to the length of `x` and `sum(w) = 1`.
- `param` Vector of values for the model parameters `c(b0, b1, b2, b3)`.

Value

Fisher information matrix.

`FIM_logistic_4par` *Fisher Information Matrix for the 4-Parameter Logistic Model*

Description

It provides the cpp function for the FIM for the model `~theta1/(1+exp(theta2*x+theta3))+theta4`. This model is another re-parameterization of the 4-parameter Hill model. For more details, see Eq. (1) and (2) in Hyun and Wong (2015).

Usage

```
FIM_logistic_4par(x, w, param)
```

Arguments

- `x` Vector of design points.
- `w` Vector of design weight. Its length must be equal to the length of `x` and `sum(w) = 1`.
- `param` Vector of values for the model parameters `c(theta1, theta2, theta3, theta4)`.

Details

The fisher information matrix does not depend on `theta4`.

Value

Fisher information matrix.

References

Hyun, S. W., & Wong, W. K. (2015). Multiple-Objective Optimal Designs for Studying the Dose Response Function and Interesting Dose Levels. The international journal of biostatistics, 11(2), 253-271.

See Also

multiple

Examples

```
FIM_logistic_4par(x = c(-6.9, -4.6, -3.9, 6.7 ),
                  w = c(0.489, 0.40, 0.061, 0.050),
                  param = c(1.563, 1.790, 8.442, 0.137))
```

FIM_loglin *Fisher Information Matrix for the Mixed Inhibition Model*

Description

It provides the cpp function for the FIM for the model `~theta0 + theta1* log(x + theta2)`.

Usage

```
FIM_loglin(x, w, param)
```

Arguments

x	Vector of design points.
w	Vector of design weight. Its length must be equal to the length of `x` and `sum(w) = 1`.
param	Vector of values for the model parameters `c(theta0, theta1, theta2)`.

Details

The FIM of this model does not depend on the parameter `theta0`.

Value

Fisher information matrix.

References

Dette, H., Kiss, C., Bevanda, M., & Bretz, F. (2010). Optimal designs for the EMAX, log-linear and exponential models. Biometrika, 97(2), 513-518.

`FIM_mixed_inhibition` *Fisher Information Matrix for the Mixed Inhibition Model.*

Description

It provides the cpp function for FIM for the model `~ V*S/(Km * (1 + I/Kic)+ S * (1 + I/Kiu))`

Usage

```
FIM_mixed_inhibition(S, I, w, param)
```

Arguments

`S`	Vector of `S` component of design points. `S` is the substrate concentration.
`I`	Vector of `I` component of design points. `I` is the inhibitor concentration.
`w`	Vector of design weight. Its length must be equal to the length of `S` and `I`, besides `sum(w) = 1`.
`param`	Vector of values for the model parameters `c(V, Km, Kic, Kiu)`.

Details

The optimal design does not depend on parameter V.

Value

Fisher information matrix of design.

References

Bogacka, B., Patan, M., Johnson, P. J., Youdim, K., & Atkinson, A. C. (2011). Optimum design of experiments for enzyme inhibition kinetic models. Journal of biopharmaceutical statistics, 21(3), 555-572.

Examples

```
FIM_mixed_inhibition(S = c(30, 3.86, 30, 4.60),
                     I = c(0, 0, 5.11, 4.16), w = rep(.25, 4),
                     param = c(1.5, 5.2, 3.4, 5.6))
```

FIM_power_logistic *Fisher Information Matrix for the Power Logistic Model*

Description

It provides the cpp function for FIM for the model `~1/(1 + exp(-b *(x - a)))^s`, but when `s` is fixed (a two by two matrix).

Usage

```
FIM_power_logistic(x, w, param, s)
```

Arguments

`x`	Vector of design points.
`w`	Vector of design weight. Its length must be equal to the length of `x` and `sum(w) = 1`.
`param`	Vector of values for the model parameters `c(a, b)`.
`s`	parameter `s`.

Value

Fisher information matrix.

Note

This matrix is a two by two matrix and not equal to the Fisher information matrix for the power logistic model when the derivative is taken with respect to all the three parameters. This matrix is only given to be used in some illustrative examples.

FIM_sig_emax *Fisher Information Matrix for the Sigmoid Emax Model*

Description

It provides the cpp function for FIM for the model `~b1+(b2-b1)*(x^b4)/(x^b4+b3^b4)`

Usage

```
FIM_sig_emax(x, w, param)
```

Arguments

`x`	Vector of design points.
`w`	Vector of design weight. Its length must be equal to the length of `x` and `sum(w) = 1`.
`param`	Vector of values for the model parameters `c(b1, b2, b3, b4)`. The mean of response variable is .

Value

Fisher information matrix.

`ICA.control` *Returns ICA Control Optimization Parameters*

Description

The function `ICA.control` returns a list of ICA control parameters.

Usage

```
ICA.control(
  ncount = 40,
  nimp = ncount/10,
  assim_coeff = 4,
  revol_rate = 0.3,
  damp = 0.99,
  uniting_threshold = 0.02,
  equal_weight = FALSE,
  sym = FALSE,
  sym_point = NULL,
  stop_rule = c("maxiter", "equivalence"),
  stoptol = 0.99,
  checkfreq = 0,
  plot_cost = TRUE,
  plot_sens = TRUE,
  plot_3d = c("lattice", "rgl"),
  trace = TRUE,
  rseed = NULL
)
```

Arguments

`ncount`	Number of countries. Defaults to `40`.
`nimp`	Number of imperialists. Defaults to 10 percent of `ncount`.
`assim_coeff`	Assimilation coefficient. Defaults to `4`.

revol_rate Revolution rate. Defaults to 0.3.

damp Damp ratio for revolution rate. revol_rate is decreased in every iteration by a factor of damp (revol_rate * damp). Defaults to 0.99.

uniting_threshold
If the distance between two imperialists is less than the product of the uniting threshold by the largest distance in the search space, ICA unites the empires. Defaults to 0.02.

equal_weight Should the weights of design points assumed to be equal? Defaults to FALSE. If TRUE, it reduces the dimension of the search space and produces a design that gives equal weight to all of its support points.

sym Should the design points be symmetric around sym_point? Defaults to FALSE. When TRUE, sym_point must be given.

sym_point If sym = TRUE, the design points will be symmetric around sym_point. See 'Details'.

stop_rule Either 'maxiter' or 'equivalence'. Denotes the type of stopping rule. See 'Details'. Defaults to 'maxiter'.

stoptol If stop_rule = 'equivalence', algorithm stops when ELB is larger than stoptol. Defaults to 0.99.

checkfreq The algorithm verifies the general equivalence theorem in every checkfreq iterations. When checkfreq = 0, no verification will be done. When checkfreq = Inf, only the output design will be verified. Defaults to 0.

plot_cost Plot the iterations (evolution) of algorithm? Defaults to TRUE.

plot_sens Plot the sensitivity (derivative) function at every checkfreq. Defaults to TRUE.

plot_3d Character. Which package should be used to plot the sensitivity plot for models with two explanatory variables?

trace Print the information in every iteration? Defaults to TRUE.

rseed Random seed. Defaults to NULL.

Details

If stop_rule = 'maxiter', the algorithm iterates until maximum number of iterations.
If stope_rule = 'equivalence', the algorithm stops when either ELB is greater than stoptol or it reaches maxiter. In this case, you must specify the check frequency by checkfreq. Note that checking equivalence theorem is a very time consuming process, especially for Bayesian and minimax design problems. We advise using this option only for locally, multiple objective and robust optimal designs.

What to follows shows how sym_point and sym may be useful?
Assume the 2PL model of the form $P(Y = 1) = \frac{1}{1+exp(-b(x-a))}$ and let the parameters a and b belong to $[a_L, a_U]$ and $[b_L, b_U]$, respectively. It can be shown that the optimal design for this model is symmetric around $a_M = \frac{a_L+a_U}{2}$. For this model, to find accurate symmetric designs, one can set sym = TRUE and provide the value of the a_M via sym_point. In this case, the output design will be symmetric around the point sym_point. The length of sym_point must be equal to the number of model predictors, here, is equal to 1.

Value

A list of ICA control parameters.

Examples

```
ICA.control(ncount = 100)
```

ICAOD *ICAOD: Finding Optimal Designs for Nonlinear Models Using Imperialist Competitive Algorithm*

Description

Different functions are available to find optimal designs for linear and nonlinear models using the imperialist competitive algorithm (ICA). Because the optimality criteria for linear and nonlinear models depend on the unknown parameters, one should choose on of the following method to deal with the parameter-dependency based on the available information for the unknown parameters:

- `locally`: finds locally optimal designs. A vector of initial estimates or guess is available for the vector of model parameters from a pilot or similar study.
- `bayes`: finds Bayesian optimal designs. A continuous prior is available for the vector of unknown model parameters.
- `robust`: finds robust or optimum-in-average designs. It is similar to `bayes`, but uses a discrete prior.
- `minimax`: finds minimax and standardized maximin optimal designs. Each of the unknown parameters belongs to a user-specified interval. The purpose is to find a design that protects the user against the worst scenario over the parameter space. Standardized designs should be used when locally optimal design of the model of interest has an analytical solution.

Some functions are also available to find optimal designs for special applications:

- `multiple`: finds locally multiple objective optimal designs for the 4-parameter Hill model with application in dose-response stuides. It uses the same strategy as `locally` to deal with the unknown model parameters.
- `bayescomp`: finds a design that meets the dual goal of the parameter estimation and increasing the probability of a particular outcome in a binary response model. It uses the same strategy as the function `bayes` to deal with the unknown mode parameters and applicable in medicine studies.

Details

The functions `locally` and `robust` are very easy to be applied and they are usually fast. The speed of the functions `bayes` and `minimax` considerably depends on the value of the tuning parameters.

The following functions may also be used to verify the optimality of an output design for each of the above criterion:

- `senslocally`

- sensrobust
- sensbayes
- sensminimax
- sensmultiple
- sensbayescomp

For more details see Masoudi et al. (2017, 2019).

References

Masoudi E, Holling H, Wong WK (2017). Application of Imperialist Competitive Algorithm to Find Minimax and Standardized Maximin Optimal Designs. Computational Statistics and Data Analysis, 113, 330-345. <doi:10.1016/j.csda.2016.06.014>

Masoudi E, Holling H, Duarte BP, Wong Wk (2019). Metaheuristic Adaptive Cubature Based Algorithm to Find Bayesian Optimal Designs for Nonlinear Models. Journal of Computational and Graphical Statistics. <doi:10.1080/10618600.2019.1601097>

---

leff *Calculates Relative Efficiency for Locally Optimal Designs*

Description

Given a vector of initial estimates for the parameters, this function calculates the D-and PA- efficiency of a design ξ_1 with respect to a design ξ_2. Usually, ξ_2 is an optimal design.

Usage

```
leff(
  formula,
  predvars,
  parvars,
  family = gaussian(),
  inipars,
  type = c("D", "PA"),
  fimfunc = NULL,
  x2,
  w2,
  x1,
  w1,
  npar = length(inipars),
  prob = NULL
)
```

Arguments

formula — A linear or nonlinear model formula. A symbolic description of the model consists of predictors and the unknown model parameters. Will be coerced to a formula if necessary.

predvars — A vector of characters. Denotes the predictors in the formula.

parvars — A vector of characters. Denotes the unknown parameters in the formula.

family — A description of the response distribution and the link function to be used in the model. This can be a family function, a call to a family function or a character string naming the family. Every family function has a link argument allowing to specify the link function to be applied on the response variable. If not specified, default links are used. For details see family. By default, a linear gaussian model `gaussian()` is applied.

inipars — Vector. Initial values for the unknown parameters. It will be passed to the information matrix and also probability function.

type — A character. `"D"` denotes the D-efficiency and `"PA"` denotes the average P-efficiency.

fimfunc — A function. Returns the FIM as a `matrix`. Required when `formula` is missing. See 'Details' of minimax.

x2 — Vector of design (support) points of the optimal design (ξ_2). Similar to `x1`.

w2 — Vector of corresponding design weights for `x2`.

x1 — Vector of design (support) points of ξ_1. See 'Details' of leff.

w1 — Vector of corresponding design weights for `x1`.

npar — Number of model parameters. Used when `fimfunc` is given instead of `formula` to specify the number of model parameters. If not given, the sensitivity plot may be shifted below the y-axis. When `NULL`, it will be set here to `length(inipars)`.

prob — Either `formula` or a `function`. When function, its argument are `x` and `param`, and they are the same as the arguments in `fimfunc`. `prob` as a function takes the design points and vector of parameters and returns the probability of success at each design points. See 'Examples'.

Details

For a known θ_0, relative D-efficiency is

$$exp(\frac{log|M(\xi_1, \theta_0)| - log|M(\xi_2, \theta_0)|}{npar})$$

The relative P-efficiency is

$$\exp(\log(\sum_{i=1}^{k} w_{1i} p(x_{1i}, \theta_0) - \log(\sum_{i=1}^{k} w_{2i} p(x2_i, \theta_0))$$

where x_2 and w_2 are usually the support points and the corresponding weights of the optimal design, respectively.

The argument `x1` is the vector of design points. For design points with more than one dimension (the models with more than one predictors), it is a concatenation of the design points, but **dimension-wise**. For example, let the model has three predictors (I, S, Z). Then, a two-point optimal design has the following points: $\{\text{point1} = (I_1, S_1, Z_1), \text{point2} = (I_2, S_2, Z_2)\}$. Then, the argument `x1` is equal to `x = c(I1, I2, S1, S2, Z1, Z2)`.

Value

A value between 0 and 1.

References

McGree, J. M., Eccleston, J. A., and Duffull, S. B. (2008). Compound optimal design criteria for nonlinear models. Journal of Biopharmaceutical Statistics, 18(4), 646-661.

Examples

```
p <- c(1, -2, 1, -1)
prior4.4 <- uniform(p -1.5, p + 1.5)
formula4.4 <- ~exp(b0+b1*x1+b2*x2+b3*x1*x2)/(1+exp(b0+b1*x1+b2*x2+b3*x1*x2))
prob4.4 <- ~1-1/(1+exp(b0 + b1 * x1 + b2 * x2 + b3 * x1 * x2))
predvars4.4 <-  c("x1", "x2")
parvars4.4 <- c("b0", "b1", "b2", "b3")


# Locally D-optimal design is as follows:
## weight and point of D-optimal design
# Point1     Point2     Point3     Point4
# /1.00000 \ /-1.00000\ /0.06801 \ /1.00000 \
# \-1.00000/ \-1.00000/ \1.00000 / \1.00000 /
#   Weight1    Weight2    Weight3    Weight4
# 0.250      0.250      0.250      0.250

xopt_D <- c(1, -1, .0680, 1, -1, -1, 1, 1)
wopt_D <- rep(.25, 4)

# Let see if we use only three of the design points, what is the relative efficiency.
leff(formula = formula4.4, predvars = predvars4.4, parvars = parvars4.4, family = binomial(),
     x1 = c(1, -1, .0680,  -1, -1, 1), w1 = c(.33, .33, .33),
     inipars = p,
     x2 = xopt_D, w2 = wopt_D)
# Wow, it heavily drops!


# Locally P-optimal design has only one support point and is -1 and 1
xopt_P <- c(-1, 1)
wopt_P <- 1

# What is the relative P-efficiency of the D-optimal design with respect to P-optimal design?
leff(formula = formula4.4, predvars = predvars4.4, parvars = parvars4.4, family = binomial(),
     x1 = xopt_D, w1 = wopt_D,
     inipars = p,
```

```
    type = "PA",
    prob = prob4.4,
    x2 = xopt_P, w2 = wopt_P)
# .535
```

---

locally *Locally D-Optimal Designs*

Description

Finds locally D-optimal designs for linear and nonlinear models. It should be used when a vector of initial estimates is available for the unknown model parameters. Locally optimal designs may not be efficient when the initial estimates are far away from the true values of the parameters.

Usage

```
locally(
  formula,
  predvars,
  parvars,
  family = gaussian(),
  lx,
  ux,
  iter,
  k,
  inipars,
  fimfunc = NULL,
  ICA.control = list(),
  sens.control = list(),
  initial = NULL,
  npar = length(inipars),
  plot_3d = c("lattice", "rgl"),
  x = NULL,
  crtfunc = NULL,
  sensfunc = NULL
)
```

Arguments

formula	A linear or nonlinear model formula. A symbolic description of the model consists of predictors and the unknown model parameters. Will be coerced to a formula if necessary.
predvars	A vector of characters. Denotes the predictors in the formula.
parvars	A vector of characters. Denotes the unknown parameters in the formula.

- `family` A description of the response distribution and the link function to be used in the model. This can be a family function, a call to a family function or a character string naming the family. Every family function has a link argument allowing to specify the link function to be applied on the response variable. If not specified, default links are used. For details see `family`. By default, a linear gaussian model `gaussian()` is applied.
- `lx` Vector of lower bounds for the predictors. Should be in the same order as `predvars`.
- `ux` Vector of upper bounds for the predictors. Should be in the same order as `predvars`.
- `iter` Maximum number of iterations.
- `k` Number of design points. Must be at least equal to the number of model parameters to avoid singularity of the FIM.
- `inipars` A vector of initial estimates for the unknown parameters. It must match `parvars` or the argument `param` of the function `fimfunc`, when provided.
- `fimfunc` A function. Returns the FIM as a `matrix`. Required when `formula` is missing. See 'Details' of `minimax`.
- `ICA.control` ICA control parameters. For details, see `ICA.control`.
- `sens.control` Control Parameters for Calculating the ELB. For details, see `sens.control`.
- `initial` A matrix of the initial design points and weights that will be inserted into the initial solutions (countries) of the algorithm. Every row is a design, i.e. a concatenation of `x` and `w`. Will be coerced to a `matrix` if necessary. See 'Details' of `minimax`.
- `npar` Number of model parameters. Used when `fimfunc` is given instead of `formula` to specify the number of model parameters. If not given, the sensitivity plot may be shifted below the y-axis. When `NULL`, it is set to `length(inipars)`.
- `plot_3d` Which package should be used to plot the sensitivity (derivative) function for two-dimensional design space. Defaults to `"lattice"`.
- `x` A vector of candidate design (support) points. When is not set to `NULL` (default), the algorithm only finds the optimal weights for the candidate points in `x`. Should be set when the user has a finite number of candidate design points and the purpose is to find the optimal weight for each of them (when zero, they will be excluded from the design). For design points with more than one dimension, see 'Details' of `sensminimax`.
- `crtfunc` (Optional) a function that specifies an arbitrary criterion. It must have especial arguments and output. See 'Details' of `minimax`.
- `sensfunc` (Optional) a function that specifies the sensitivity function for `crtfunc`. See 'Details' of `minimax`.

Details

Let $M(\xi, \theta_0)$ be the Fisher information matrix (FIM) of a $k-$point design ξ and θ_0 be the vector of the initial estimates for the unknown parameters. A locally D-optimal design ξ^* minimizes over Ξ

$$-\log|M(\xi, \theta_0)|.$$

One can adjust the tuning parameters in `ICA.control` to set a stopping rule based on the general equivalence theorem. See "Examples" below.

Value

an object of class `minimax` that is a list including three sub-lists:

`arg` A list of design and algorithm parameters.

`evol` A list of length equal to the number of iterations that stores the information about the best design (design with least criterion value) of each iteration. `evol[[iter]]` contains:

`iter`	Iteration number.
`x`	Design points.
`w`	Design weights.
`min_cost`	Value of the criterion for the best imperialist (design).
`mean_cost`	Mean of the criterion values of all the imperialists.
`sens`	An object of class `'sensminimax'`. See below.
`param`	Vector of parameters.

`empires` A list of all the empires of the last iteration.

`alg` A list with following information:

`nfeval`	Number of function evaluations. It does not count the function evaluations from checking the general equiv
`nlocal`	Number of successful local searches.
`nrevol`	Number of successful revolutions.
`nimprove`	Number of successful movements toward the imperialists in the assimilation step.
`convergence`	Stopped by `'maxiter'` or `'equivalence'`?

`method` A type of optimal designs used.

`design` Design points and weights at the final iteration.

`out` A data frame of design points, weights, value of the criterion for the best imperialist (min_cost), and Mean of the criterion values of all the imperialistsat each iteration (mean_cost).

The list `sens` contains information about the design verification by the general equivalence theorem. See `sensminimax` for more details. It is given every `ICA.control$checkfreq` iterations and also the last iteration if `ICA.control$checkfreq >= 0`. Otherwise, `NULL`.

`param` is a vector of parameters that is the global minimum of the minimax criterion or the global maximum of the standardized maximin criterion over the parameter space, given the current `x`, `w`.

References

Masoudi E, Holling H, Wong W.K. (2017). Application of Imperialist Competitive Algorithm to Find Minimax and Standardized Maximin Optimal Designs. Computational Statistics and Data Analysis, 113, 330-345.

See Also

senslocally

Examples

```
##################################
# Exponential growth model
#################################
# See how we set stopping rule by adjusting 'stop_rule', 'checkfreq' and 'stoptol'
# It calls the 'senslocally' function every checkfreq = 50 iterations to
# calculate the ELB. if ELB is greater than stoptol = .95, then the algoithm stops.

# initializing by one iteration
res1 <- locally(formula = ~a + exp(-b*x), predvars = "x", parvars = c("a", "b"),
                lx = 0, ux = 1, inipars = c(1, 10),
                iter = 1, k = 2,
                ICA.control= ICA.control(rseed = 100,
                                         stop_rule = "equivalence",
                                         checkfreq = 20, stoptol = .95))
## Not run:
  # update the algorithm
  res1 <- update(res1, 150)
  #stops at iteration 21 because ELB is greater than .95

## End(Not run)

### fixed x, lx and ux are only required for equivalence theorem
## Not run:
  res1.1 <- locally(formula = ~a + exp(-b*x), predvars = "x", parvars = c("a", "b"),
                    lx = 0, ux = 1, inipars = c(1, 10),
                    iter = 100,
                    x = c(.25, .5, .75),
                    ICA.control= ICA.control(rseed = 100))
  plot(res1.1)
  # we can not have an optimal design using this x

## End(Not run)

################################
## two parameter logistic model
################################
res2 <- locally(formula = ~1/(1 + exp(-b *(x - a))),
                predvars = "x", parvars = c("a", "b"),
                family = binomial(), lx = -3, ux = 3,
                inipars = c(1, 3), iter = 1, k = 2,
                ICA.control= list(rseed = 100, stop_rule = "equivalence",
                                  checkfreq = 50, stoptol = .95))
## Not run:
  res2 <- update(res2, 100)
  # stops at iteration 51

## End(Not run)
```

```
################################
# A model with two predictors
################################
# mixed inhibition model
## Not run:
  res3 <- locally(formula =  ~ V*S/(Km * (1 + I/Kic)+ S * (1 + I/Kiu)),
                  predvars = c("S", "I"),
                  parvars = c("V", "Km", "Kic", "Kiu"),
                  family = gaussian(),
                  lx = c(0, 0), ux = c(30, 60),
                  k = 4,
                  iter = 300,
                  inipars = c(1.5, 5.2, 3.4, 5.6),
                  ICA.control= list(rseed = 100, stop_rule = "equivalence",
                                    checkfreq = 50, stoptol = .95))
  # stops at iteration 100

## End(Not run)


## Not run:
  # fixed x
  res3.1 <- locally(formula =  ~ V*S/(Km * (1 + I/Kic)+ S * (1 + I/Kiu)),
                    predvars = c("S", "I"),
                    parvars = c("V", "Km", "Kic", "Kiu"),
                    family = gaussian(),
                    lx = c(0, 0), ux = c(30, 60),
                    iter = 100,
                    x = c(20, 4, 20, 4, 10,  0, 0, 30, 3, 2),
                    inipars = c(1.5, 5.2, 3.4, 5.6),
                    ICA.control= list(rseed = 100))

## End(Not run)


###################################
# user-defined optimality criterion
##################################
# When the model is defined by the formula interface
# A-optimal design for the 2PL model.
# the criterion function must have argument x, w fimfunc and the parameters defined in 'parvars'.
# use 'fimfunc' as a function of the design points x,  design weights w and
#  the 'parvars' parameters whenever needed.
Aopt <-function(x, w, a, b, fimfunc){
  sum(diag(solve(fimfunc(x = x, w = w, a = a, b = b))))
}
## the sensitivtiy function
# xi_x is a design that put all its mass on x in the definition of the sensitivity function
# x is a vector of design points
```

```
Aopt_sens <- function(xi_x, x, w, a, b, fimfunc){
  fim <- fimfunc(x = x, w = w, a = a, b = b)
  M_inv <- solve(fim)
  M_x <- fimfunc(x = xi_x, w = 1, a  = a, b = b)
  sum(diag(M_inv %*% M_x %*%  M_inv)) - sum(diag(M_inv))
}

res4 <- locally(formula = ~1/(1 + exp(-b * (x-a))), predvars = "x",
                parvars = c("a", "b"), family = "binomial",
                lx = -3, ux = 3, inipars = c(1, 1.25),
                iter = 1, k = 2,
                crtfunc = Aopt,
                sensfunc = Aopt_sens,
                ICA.control = list(checkfreq = Inf))
## Not run:
  res4 <- update(res4, 50)

## End(Not run)

# When the FIM of the model is defined directly via the argument 'fimfunc'
# the criterion function must have argument x, w fimfunc and param.
# use 'fimfunc' as a function of the design points x,  design weights w
# and param whenever needed.
Aopt2 <-function(x, w, param, fimfunc){
  sum(diag(solve(fimfunc(x = x, w = w, param = param))))
}
## the sensitivtiy function
# xi_x is a design that put all its mass on x in the definition of the sensitivity function
# x is a vector of design points
Aopt_sens2 <- function(xi_x, x, w, param, fimfunc){
  fim <- fimfunc(x = x, w = w, param = param)
  M_inv <- solve(fim)
  M_x <- fimfunc(x = xi_x, w = 1, param = param)
  sum(diag(M_inv %*% M_x %*%  M_inv)) - sum(diag(M_inv))
}

res4.1 <- locally(fimfunc = FIM_logistic,
                  lx = -3, ux = 3, inipars = c(1, 1.25),
                  iter = 1, k = 2,
                  crtfunc = Aopt2,
                  sensfunc = Aopt_sens2,
                  ICA.control = list(checkfreq = Inf))
## Not run:
  res4.1 <- update(res4.1, 50)
  plot(res4.1)

## End(Not run)


# locally c-optimal design
# example from Chaloner and Larntz (1989) Figure 3
c_opt <-function(x, w, a, b, fimfunc){
  gam <- log(.95/(1-.95))
```

```
  M <- fimfunc(x = x, w = w, a = a, b = b)
  c <- matrix(c(1, -gam * b^(-2)), nrow = 1)
  B <- t(c) %*% c
  sum(diag(B %*% solve(M)))
}

c_sens <- function(xi_x, x, w, a, b, fimfunc){
  gam <- log(.95/(1-.95))
  M <- fimfunc(x = x, w = w, a = a, b = b)
  M_inv <- solve(M)
  M_x <- fimfunc(x = xi_x, w = 1, a = a, b = b)
  c <- matrix(c(1, -gam * b^(-2)), nrow = 1)
  B <- t(c) %*% c
  sum(diag(B %*% M_inv %*% M_x %*%  M_inv)) - sum(diag(B %*% M_inv))
}


res4.2 <- locally(formula = ~1/(1 + exp(-b * (x-a))), predvars = "x",
                  parvars = c("a", "b"), family = "binomial",
                  lx = -1, ux = 1, inipars = c(0, 7),
                  iter = 1, k = 2,
                  crtfunc = c_opt, sensfunc = c_sens,
                  ICA.control = list(rseed = 1, checkfreq = Inf))
## Not run:
res4.2 <- update(res4.2, 100)

## End(Not run)
```

`locallycomp` *Locally DP-Optimal Designs*

Description

Finds compound locally DP-optimal designs that meet the dual goal of parameter estimation and increasing the probability of a particular outcome in a binary response model. A compound locally DP-optimal design maximizes the product of the efficiencies of a design ξ with respect to D- and average P-optimality, weighted by a pre-defined mixing constant $0 \leq \alpha \leq 1$.

Usage

```
locallycomp(
  formula,
  predvars,
  parvars,
  family = gaussian(),
  lx,
  ux,
  alpha,
```

```
  prob,
  iter,
  k,
  inipars,
  fimfunc = NULL,
  ICA.control = list(),
  sens.control = list(),
  initial = NULL,
  npar = length(inipars),
  plot_3d = c("lattice", "rgl")
)
```

Arguments

`formula`	A linear or nonlinear model `formula`. A symbolic description of the model consists of predictors and the unknown model parameters. Will be coerced to a `formula` if necessary.
`predvars`	A vector of characters. Denotes the predictors in the `formula`.
`parvars`	A vector of characters. Denotes the unknown parameters in the `formula`.
`family`	A description of the response distribution and the link function to be used in the model. This can be a family function, a call to a family function or a character string naming the family. Every family function has a link argument allowing to specify the link function to be applied on the response variable. If not specified, default links are used. For details see `family`. By default, a linear gaussian model `gaussian()` is applied.
`lx`	Vector of lower bounds for the predictors. Should be in the same order as `predvars`.
`ux`	Vector of upper bounds for the predictors. Should be in the same order as `predvars`.
`alpha`	A value between 0 and 1. Compound or combined DP-criterion is the product of the efficiencies of a design with respect to D- and average P- optimality, weighted by `alpha`.
`prob`	Either `formula` or a `function`. When function, its argument are `x` and `param`, and they are the same as the arguments in `fimfunc`. `prob` as a function takes the design points and vector of parameters and returns the probability of success at each design points. See 'Examples'.
`iter`	Maximum number of iterations.
`k`	Number of design points. When `alpha = 0`, then `k` can be less than the number of parameters.
`inipars`	Vector. Initial values for the unknown parameters. It will be passed to the information matrix and also probability function.
`fimfunc`	A function. Returns the FIM as a `matrix`. Required when `formula` is missing. See 'Details' of `minimax`.
`ICA.control`	ICA control parameters. For details, see `ICA.control`.
`sens.control`	Control Parameters for Calculating the ELB. For details, see `sens.control`.

`initial` A matrix of the initial design points and weights that will be inserted into the initial solutions (countries) of the algorithm. Every row is a design, i.e. a concatenation of `x` and `w`. Will be coerced to a `matrix` if necessary. See 'Details' of `minimax`.

`npar` Number of model parameters. Used when `fimfunc` is given instead of `formula` to specify the number of model parameters. If not given, the sensitivity plot may be shifted below the y-axis. When `NULL`, it will be set here to `length(inipars)`.

`plot_3d` Which package should be used to plot the sensitivity (derivative) function for two-dimensional design space. Defaults to `"lattice"`.

Details

Let Ξ be the space of all approximate designs with k design points (support points) at $x_1, x_2, ..., x_k$ from design space χ with corresponding weights $w_1, ..., w_k$. Let $M(\xi, \theta)$ be the Fisher information matrix (FIM) of a $k-$point design ξ, θ_0 is a user-given vector of initial estimates for the unknown parameters θ and $p(x_i, \theta)$ is the ith probability of success given by x_i in a binary response model. A compound locally DP-optimal design maximizes over Ξ

$$\frac{\alpha}{q}\log|M(\xi,\theta_0)| + (1-\alpha)\log\left(\sum_{i=1}^{k} w_i p(x_i,\theta_0)\right).$$

Use `plot` function to verify the general equivalence theorem for the output design or change `checkfreq` in `ICA.control`.

One can adjust the tuning parameters in `ICA.control` to set a stopping rule based on the general equivalence theorem. See "Examples" in `locally`.

Value

an object of class `minimax` that is a list including three sub-lists:

`arg` A list of design and algorithm parameters.

`evol` A list of length equal to the number of iterations that stores the information about the best design (design with least criterion value) of each iteration. `evol[[iter]]` contains:

`iter`	Iteration number.
`x`	Design points.
`w`	Design weights.
`min_cost`	Value of the criterion for the best imperialist (design).
`mean_cost`	Mean of the criterion values of all the imperialists.
`sens`	An object of class `'sensminimax'`. See below.
`param`	Vector of parameters.

`empires` A list of all the empires of the last iteration.

`alg` A list with following information:

`nfeval` Number of function evaluations. It does not count the function evaluations from checking the general equi

`nlocal`	Number of successful local searches.
`nrevol`	Number of successful revolutions.
`nimprove`	Number of successful movements toward the imperialists in the assimilation step.
`convergence`	Stopped by `'maxiter'` or `'equivalence'`?

`method` A type of optimal designs used.

`design` Design points and weights at the final iteration.

`out` A data frame of design points, weights, value of the criterion for the best imperialist (min_cost), and Mean of the criterion values of all the imperialistsat each iteration (mean_cost).

The list `sens` contains information about the design verification by the general equivalence theorem. See `sensminimax` for more details. It is given every `ICA.control$checkfreq` iterations and also the last iteration if `ICA.control$checkfreq >= 0`. Otherwise, `NULL`.

`param` is a vector of parameters that is the global minimum of the minimax criterion or the global maximum of the standardized maximin criterion over the parameter space, given the current `x`, `w`.

References

McGree, J. M., Eccleston, J. A., and Duffull, S. B. (2008). Compound optimal design criteria for nonlinear models. Journal of Biopharmaceutical Statistics, 18(4), 646-661.

Examples

```
## Here we produce the results of Table 2 in in McGree and Eccleston (2008)
# For D- and P-efficiency see, ?leff and ?peff

p <- c(1, -2, 1, -1)
prior4.4 <- uniform(p -1.5, p + 1.5)
formula4.4 <- ~exp(b0+b1*x1+b2*x2+b3*x1*x2)/(1+exp(b0+b1*x1+b2*x2+b3*x1*x2))
prob4.4 <- ~1-1/(1+exp(b0 + b1 * x1 + b2 * x2 + b3 * x1 * x2))
predvars4.4 <-  c("x1", "x2")
parvars4.4 <- c("b0", "b1", "b2", "b3")
lb <- c(-1, -1)
ub <- c(1, 1)


# set checkfreq = Inf to ask for equivalence theorem at final step.
res.0 <- locallycomp(formula = formula4.4, predvars = predvars4.4, parvars = parvars4.4,
                     family = binomial(), prob = prob4.4, lx = lb, ux = ub,
                     alpha = 0, k = 1, inipars = p, iter = 10,
                     ICA.control = ICA.control(checkfreq = Inf))

## Not run:
res.25 <- locallycomp(formula = formula4.4, predvars = predvars4.4, parvars = parvars4.4,
                      family = binomial(), prob = prob4.4, lx = lb, ux = ub,
                      alpha = .25, k = 4, inipars = p, iter = 350,
                      ICA.control = ICA.control(checkfreq = Inf))

res.5 <- locallycomp(formula = formula4.4, predvars = predvars4.4, parvars = parvars4.4,
```

```
                     family = binomial(), prob = prob4.4, lx = lb, ux = ub,
                     alpha = .5, k = 4, inipars = p, iter = 350,
                     ICA.control = ICA.control(checkfreq = Inf))
res.75 <- locallycomp(formula = formula4.4, predvars = predvars4.4, parvars = parvars4.4,
                      family = binomial(), prob = prob4.4, lx = lb, ux = ub,
                      alpha = .75, k = 4, inipars = p, iter = 350,
                      ICA.control = ICA.control(checkfreq = Inf))

res.1 <- locallycomp(formula = formula4.4, predvars = predvars4.4, parvars = parvars4.4,
                     family = binomial(), prob = prob4.4, lx = lb, ux = ub,
                     alpha = 1, k = 4, inipars = p, iter = 350,
                     ICA.control = ICA.control(checkfreq = Inf))

#### computing the D-efficiency
# locally D-optimal design is locally DP-optimal design when alpha = 1.

leff(formula = formula4.4, predvars = predvars4.4, parvars = parvars4.4, family = binomial(),
     x1 = res.0$evol[[10]]$x, w1 = res.0$evol[[10]]$w,
     inipars = p,
     x2 = res.1$evol[[350]]$x, w2 = res.1$evol[[350]]$w)

leff(formula = formula4.4, predvars = predvars4.4, parvars = parvars4.4, family = binomial(),
     x1 = res.25$evol[[350]]$x, w1 = res.25$evol[[350]]$w,
     inipars = p,
     x2 = res.1$evol[[350]]$x, w2 = res.1$evol[[350]]$w)

leff(formula = formula4.4, predvars = predvars4.4, parvars = parvars4.4, family = binomial(),
     x1 = res.5$evol[[350]]$x, w1 = res.5$evol[[350]]$w,
     inipars = p,
     x2 = res.1$evol[[350]]$x, w2 = res.1$evol[[350]]$w)


leff(formula = formula4.4, predvars = predvars4.4, parvars = parvars4.4, family = binomial(),
     x1 = res.75$evol[[350]]$x, w1 = res.75$evol[[350]]$w,
     inipars = p,
     x2 = res.1$evol[[350]]$x, w2 = res.1$evol[[350]]$w)



#### computing the P-efficiency
# locally p-optimal design is locally DP-optimal design when alpha = 0.

leff(formula = formula4.4, predvars = predvars4.4, parvars = parvars4.4, family = binomial(),
     x2 = res.0$evol[[10]]$x, w2 = res.0$evol[[10]]$w,
     prob = prob4.4,
     type = "PA",
     inipars = p,
     x1 = res.25$evol[[350]]$x, w1 = res.25$evol[[350]]$w)

leff(formula = formula4.4, predvars = predvars4.4, parvars = parvars4.4, family = binomial(),
     x2 = res.0$evol[[10]]$x, w2 = res.0$evol[[10]]$w,
     prob = prob4.4,
     inipars = p,
```

```
    type = "PA",
    x1 = res.5$evol[[350]]$x, w1 = res.5$evol[[350]]$w)

leff(formula = formula4.4, predvars = predvars4.4, parvars = parvars4.4, family = binomial(),
    x2 = res.0$evol[[10]]$x, w2 = res.0$evol[[10]]$w,
    prob = prob4.4,
    inipars = p,
    type = "PA",
    x1 = res.75$evol[[350]]$x, w1 = res.75$evol[[350]]$w)


leff(formula = formula4.4, predvars = predvars4.4, parvars = parvars4.4, family = binomial(),
    x2 = res.0$evol[[10]]$x, w2 = res.1$evol[[10]]$w,
    prob = prob4.4,
    type = "PA",
    inipars = p,
    x1 = res.1$evol[[350]]$x, w1 = res.1$evol[[350]]$w)



## End(Not run)
```

`meff` *Calculates Relative Efficiency for Minimax Optimal Designs*

Description

Given a parameter space for the unknown parameters, this function calculates the D-efficiency of a design ξ_1 with respect to a design ξ_2. Usually, ξ_2 is an optimal design.

Usage

```
meff(
  formula,
  predvars,
  parvars,
  family = gaussian(),
  lp,
  up,
  fimfunc = NULL,
  x2,
  w2,
  x1,
  w1,
  standardized = FALSE,
  localdes = NULL,
  crt.minimax.control = list(),
  npar = length(lp)
)
```

Arguments

- `formula` — A linear or nonlinear model `formula`. A symbolic description of the model consists of predictors and the unknown model parameters. Will be coerced to a `formula` if necessary.
- `predvars` — A vector of characters. Denotes the predictors in the `formula`.
- `parvars` — A vector of characters. Denotes the unknown parameters in the `formula`.
- `family` — A description of the response distribution and the link function to be used in the model. This can be a family function, a call to a family function or a character string naming the family. Every family function has a link argument allowing to specify the link function to be applied on the response variable. If not specified, default links are used. For details see `family`. By default, a linear gaussian model `gaussian()` is applied.
- `lp` — Vector of lower bounds for the model parameters. Should be in the same order as `parvars` or `param` in the argument `fimfunc`.
- `up` — Vector of upper bounds for the model parameters. Should be in the same order as `parvars` or `param` in the argument `fimfunc`. When a parameter is known (has a fixed value), its associated lower and upper bound values in `lp` and `up` must be set equal.
- `fimfunc` — A function. Returns the FIM as a `matrix`. Required when `formula` is missing. See 'Details' of `minimax`.
- `x2` — Vector of design (support) points of the optimal design (ξ_2). Similar to `x`.
- `w2` — Vector of corresponding design weights for `x2`.
- `x1` — Vector of design (support) points of ξ_1. See 'Details' of `leff`.
- `w1` — Vector of corresponding design weights for `x`.
- `standardized` — Maximin standardized design? When `standardized = TRUE`, the argument `localdes` must be given. Defaults to `FALSE`. See 'Details' of `minimax`.
- `localdes` — A function that takes the parameter values as inputs and returns the design points and weights of the locally optimal design. Required when `standardized = "TRUE"`. See 'Details' of `minimax`.
- `crt.minimax.control` — Control parameters to optimize the minimax or standardized maximin criterion at a given design over a **continuous** parameter space (when `n.grid = 0`). For details, see the function `crt.minimax.control`.
- `npar` — Number of model parameters. Used when `fimfunc` is given instead of `formula` to specify the number of model parameters. If not specified truly, the sensitivity (derivative) plot may be shifted below the y-axis. When `NULL` (default), it is set to `length(lp)`.

Details

See Masoudi et al. (2017) for formula details.

The argument `x1` is the vector of design points. For design points with more than one dimension (the models with more than one predictors), it is a concatenation of the design points, but **dimension-wise**. For example, let the model has three predictors (I, S, Z). Then, a two-point optimal design has the following points: $\{\text{point1} = (I_1, S_1, Z_1), \text{point2} = (I_2, S_2, Z_2)\}$. Then, the argument `x` is equal to `x = c(I1, I2, S1, S2, Z1, Z2)`.

Value

A value between 0 and 1.

Examples

```
# Relative D-efficiency with respect to the minimax criterion
meff(formula = ~1/(1 + exp(-b * (x-a))), predvars = "x",
     parvars = c("a", "b"), family = "binomial",
     lp = c(-3, .5), up = c(3, 2),
     x2 = c(-3, -1.608782, 0, 1.608782, 3),
     w2 = c(0.22291601, 0.26438449, 0.02539899, 0.26438449, 0.22291601),
     x1 = c(-1, 1), w1 = c(.5, .5))



# A function to calculate the locally D-optimal design for the 2PL model
Dopt_2pl <- function(a, b){
  x <- c(a + (1/b) * 1.5434046, a - (1/b) * 1.5434046)
  return(list(x = x, w = c(.5, .5)))
}
# Relative D-efficiency with respect to the standardized maximin criterion
meff (formula = ~1/(1 + exp(-b * (x-a))), predvars = "x",
      parvars = c("a", "b"), family = "binomial",
      lp = c(-3, .5), up = c(3, 2),
      x2 = c(-3, -1.611255, 0, 1.611255, 3),
      w2 = c(0.22167034, 0.26592974, 0.02479984, 0.26592974, 0.22167034),
      x1 = c(0, -1), w1 = c(.5, .5),
      standardized = TRUE,
      localdes = Dopt_2pl)
```

`minimax` *Minimax and Standardized Maximin D-Optimal Designs*

Description

Finds minimax and standardized maximin D-optimal designs for linear and nonlinear models. It should be used when the user assumes the unknown parameters belong to a parameter region Θ, which is called “region of uncertainty”, and the purpose is to protect the experiment from the worst case scenario over Θ.

Usage

```
minimax(
  formula,
  predvars,
  parvars,
  family = gaussian(),
```

```
  lx,
  ux,
  lp,
  up,
  iter,
  k,
  n.grid = 0,
  fimfunc = NULL,
  ICA.control = list(),
  sens.control = list(),
  sens.minimax.control = list(),
  crt.minimax.control = list(),
  standardized = FALSE,
  initial = NULL,
  localdes = NULL,
  npar = length(lp),
  plot_3d = c("lattice", "rgl"),
  x = NULL,
  crtfunc = NULL,
  sensfunc = NULL
)
```

Arguments

formula	A linear or nonlinear model formula. A symbolic description of the model consists of predictors and the unknown model parameters. Will be coerced to a formula if necessary.
predvars	A vector of characters. Denotes the predictors in the formula.
parvars	A vector of characters. Denotes the unknown parameters in the formula.
family	A description of the response distribution and the link function to be used in the model. This can be a family function, a call to a family function or a character string naming the family. Every family function has a link argument allowing to specify the link function to be applied on the response variable. If not specified, default links are used. For details see family. By default, a linear gaussian model gaussian() is applied.
lx	Vector of lower bounds for the predictors. Should be in the same order as predvars.
ux	Vector of upper bounds for the predictors. Should be in the same order as predvars.
lp	Vector of lower bounds for the model parameters. Should be in the same order as parvars or param in the argument fimfunc.
up	Vector of upper bounds for the model parameters. Should be in the same order as parvars or param in the argument fimfunc. When a parameter is known (has a fixed value), its associated lower and upper bound values in lp and up must be set equal.
iter	Maximum number of iterations.

k
: Number of design points. Must be at least equal to the number of model parameters to avoid singularity of the FIM.

n.grid
: Only required when the parameter space is going to be discretized. The total number of grid points from the parameter space is `n.grid^p`. When `n.grid > 0`, optimal design protects the experimenter against the worst case scenario only over the grid points, and not over the continuous parameter space. The resulting designs may not be globally optimal. In some literature, this type of designs has been used as a compromise to the minimax type designs to avoid continuous optimization problem over the parameter space and simplify the minimax design problems. Especially when the design criterion is convex with respect to the given parameter space at every given design from the design space, the obtained design may also be globally optimal (because the maximum of a convex function is attained on the bounds, and here, are included in the grid points). See 'Details' of `minimax`.

fimfunc
: A function. Returns the FIM as a `matrix`. Required when `formula` is missing. See 'Details' of `minimax`.

ICA.control
: ICA control parameters. For details, see `ICA.control`.

sens.control
: Control Parameters for Calculating the ELB. For details, see `sens.control`.

sens.minimax.control
: Control parameters to construct the answering set required for verify the general equivalence theorem and calculating the ELB. For details, see the function `sens.minimax.control`.

crt.minimax.control
: Control parameters to optimize the minimax or standardized maximin criterion at a given design over a **continuous** parameter space (when `n.grid = 0`). For details, see the function `crt.minimax.control`.

standardized
: Maximin standardized design? When `standardized = TRUE`, the argument `localdes` must be given. Defaults to `FALSE`. See 'Details' of `minimax`.

initial
: A matrix of the initial design points and weights that will be inserted into the initial solutions (countries) of the algorithm. Every row is a design, i.e. a concatenation of `x` and `w`. Will be coerced to a `matrix` if necessary. See 'Details' of `minimax`.

localdes
: A function that takes the parameter values as inputs and returns the design points and weights of the locally optimal design. Required when `standardized = "TRUE"`. See 'Details' of `minimax`.

npar
: Number of model parameters. Used when `fimfunc` is given instead of `formula` to specify the number of model parameters. If not specified truly, the sensitivity (derivative) plot may be shifted below the y-axis. When `NULL` (default), it is set to `length(lp)`.

plot_3d
: Which package should be used to plot the sensitivity (derivative) function for two-dimensional design space. Defaults to `"lattice"`.

x
: A vector of candidate design (support) points. When is not set to `NULL` (default), the algorithm only finds the optimal weights for the candidate points in `x`. Should be set when the user has a finite number of candidate design points and the purpose is to find the optimal weight for each of them (when zero, they will

be excluded from the design). For design points with more than one dimension, see 'Details' of sensminimax.

crtfunc (Optional) a function that specifies an arbitrary criterion. It must have especial arguments and output. See 'Details' of minimax.

sensfunc (Optional) a function that specifies the sensitivity function for crtfunc. See 'Details' of minimax.

Details

Let Ξ be the space of all approximate designs with k design points (support points) at $x_1, x_2, ..., x_k$ from the design space χ with corresponding weights $w_1, ..., w_k$. Let $M(\xi, \theta)$ be the Fisher information matrix (FIM) of a $k-$point design ξ and θ be the vector of unknown parameters. A minimax D-optimal design ξ^* minimizes over Ξ

$$\max_{\theta \in \Theta} - \log |M(\xi, \theta)|.$$

A standardized maximin D-optimal design ξ^* maximizes over Ξ

$$\inf_{\theta \in \Theta} \left[\left(\frac{|M(\xi, \theta)|}{|M(\xi_{\theta}, \theta)|} \right)^{\frac{1}{p}} \right],$$

where p is the number of model parameters and ξ_θ is the locally D-optimal design with respect to θ.

A minimax criterion (cost function or objective function) is evaluated at each design (decision variables) by maximizing the criterion over the parameter space. We call the optimization problem over the parameter space as *inner optimization problem*. Two different strategies may be applied to solve the inner problem at a given design (design points and weights):

1. **Continuous inner problem**: we optimize the criterion over a continuous parameter space using the function nloptr. In this case, the tuning parameters can be regulated via the argument crt.minimax.control, when the most influential one is maxeval.
2. **Discrete inner problem**: we map the parameter space to the grid points and optimize the criterion over a discrete parameter space. In this case, the number of grid points can be regulated via n.grid. This strategy is quite efficient (ans fast) when the maxima most likely attain the vertices of the continuous parameter space at any given design. The output design here protects the experiment from the worst scenario over the grid points.

The formula is used to automatically create the Fisher information matrix (FIM) for a linear or nonlinear model provided that the distribution of the response variable belongs to the natural exponential family. Function minimax also provides an option to assign a user-defined FIM directly via the argument fimfunc. In this case, the argument fimfunc takes a function that has three arguments as follows:

1. x a vector of design points. For design points with more than one dimension, it is a concatenation of the design points, but dimension-wise. For example, let the model has three predictors (I, S, Z). Then, a two-point design is of the format $\{\text{point1} = (I_1, S_1, Z_1), \text{point2} = (I_2, S_2, Z_2)\}$. and the argument x is equivalent to x = c(I1, I2, S1, S2, Z1, Z2).

2. `w` a vector that includes the design weights associated with `x`.
3. `param` a vector of parameter values associated with `lp` and `up`.

The output must be the Fisher information matrix with number of rows equal to `length(param)`. See 'Examples'.

Minimax optimal designs can have very different criterion values depending on the nominal set of parameter values. Accordingly, it is desirable to standardize the criterion and control for the potentially widely varying magnitude of the criterion (Dette, 1997). Evaluating a standardized maximin criterion requires knowing locally optimal designs. We strongly advise setting `standardized = 'TRUE'` only when analytical solutions for the locally D-optimal designs is available. In this case, for any initial estimate of the unknown parameters, an analytical solution for the locally optimal design, i.e, the support points `x` and the corresponding weights `w`, must be provided via the argument `localdes`. Therefore, depending on how the model is specified, `localdes` is a `function` with the following arguments (input).

- If `formula` is given (`!missing(formula)`):
  - The parameter names given by `parvars` in the same order.
- If FIM is given via the argument `fimfunc` (`missing(formula)`):
  - `param`: A vector of the parameters equal to the argument `param` in `fimfunc`.

This function must return a list with the components `x` and `w` (they match the same arguments in the function `fimfunc`). See 'Examples'.

The standardized D-criterion is equal to the D-efficiency and it must be between 0 and 1. However, in practice, when running the algorithm, it may be the case that the criterion takes a value larger than one. This may happen because the user-function that is given via `localdes` does not return the true (accurate) locally optimal designs for some requested initial estimates of the parameters from Θ. In this case, the function `minimax` throw an error where the error message helps the user to debug her/his function.

Each row of `initial` is one design, i.e. a concatenation of values for design (support) points and the associated design weights. Let `x0` and `w0` be the vector of initial values with exactly the same length and order as `x` and `w` (the arguments of `fimfunc`). As an example, the first row of the matrix `initial` is equal to `initial[1, ] = c(x0, w0)`. For models with more than one predictors, `x0` is a concatenation of the initial values for design points, but **dimension-wise**. See the details of the argument `fimfunc`, above.

To verify the optimality of the output design by the general equivalence theorem, the user can either `plot` the results or set `checkfreq` in `ICA.control` to `Inf`. In either way, the function `sensminimax` is called for verification. Note that the function `sensminimax` always verifies the optimality of a design assuming a continues parameter space. See 'Examples'.

`crtfunc` is a function that is used to specify a new criterion. Its arguments are:

- design points `x` (as a `vector`).
- design weights `w` (as a `vector`).
- model parameters as follows.
  - If `formula` is specified: they should be the same parameter specified by `parvars`.

- If FIM is specified via the argument `fimfunc`: `param` that is a vector of the parameters in `fimfunc`.

- `fimfunc` is a `function` that takes the other arguments of `crtfunc` and returns the computed Fisher information matrix as a `matrix`.

The `crtfunc` function must return the criterion value. `crtfunc`. It has one more argument than `crtfunc`, which is `xi_x`. It denotes the design point of the degenerate design and must be a vector with the same length as the number of predictors. For more details, see 'Examples'.

Value

an object of class `minimax` that is a list including three sub-lists:

`arg` A list of design and algorithm parameters.

`evol` A list of length equal to the number of iterations that stores the information about the best design (design with least criterion value) of each iteration. `evol[[iter]]` contains:

`iter`	Iteration number.
`x`	Design points.
`w`	Design weights.
`min_cost`	Value of the criterion for the best imperialist (design).
`mean_cost`	Mean of the criterion values of all the imperialists.
`sens`	An object of class `'sensminimax'`. See below.
`param`	Vector of parameters.

`empires` A list of all the empires of the last iteration.

`alg` A list with following information:

`nfeval`	Number of function evaluations. It does not count the function evaluations from checking the general equi
`nlocal`	Number of successful local searches.
`nrevol`	Number of successful revolutions.
`nimprove`	Number of successful movements toward the imperialists in the assimilation step.
`convergence`	Stopped by `'maxiter'` or `'equivalence'`?

`method` A type of optimal designs used.

`design` Design points and weights at the final iteration.

`out` A data frame of design points, weights, value of the criterion for the best imperialist (min_cost), and Mean of the criterion values of all the imperialistsat each iteration (mean_cost).

The list `sens` contains information about the design verification by the general equivalence theorem. See `sensminimax` for more details. It is given every `ICA.control$checkfreq` iterations and also the last iteration if `ICA.control$checkfreq >= 0`. Otherwise, `NULL`.

`param` is a vector of parameters that is the global minimum of the minimax criterion or the global maximum of the standardized maximin criterion over the parameter space, given the current `x`, `w`.

Note

For larger parameter space or model with more number of unknown parameters, it is always important to increase the value of `ncount` in `ICA.control` and `optslist$maxeval` in `crt.minimax.control` to produce very accurate designs.

Although standardized criteria have been preferred theoretically, in practice, they should be applied only when an analytical solution for the locally D-optimal designs is available for the model of interest. Otherwise, we encounter a three-level nested-optimization algorithm, which is very slow.

References

Atashpaz-Gargari, E, & Lucas, C (2007). Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. In 2007 IEEE congress on evolutionary computation (pp. 4661-4667). IEEE.
Dette, H. (1997). Designing experiments with respect to 'standardized' optimality criteria. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 59(1), 97-110.
Masoudi E, Holling H, Wong WK (2017). Application of Imperialist Competitive Algorithm to Find Minimax and Standardized Maximin Optimal Designs. Computational Statistics and Data Analysis, 113, 330-345. <doi:10.1016/j.csda.2016.06.014>

See Also

sensminimax

Examples

```
########################################
# Two-parameter exponential growth model
########################################
res1 <- minimax (formula = ~a + exp(-b*x), predvars = "x", parvars = c("a", "b"),
                 lx = 0, ux = 1, lp = c(1, 1), up = c(1, 10),
                 iter = 1, k = 4,
                 ICA.control= ICA.control(rseed = 100),
                 crt.minimax.control = list(optslist = list(maxeval = 100)))
# The optimal design has 3 points, but we set k = 4 for illustration purpose to
#   show how the algorithm modifies the design by adjusting the weights
# The value of maxeval is changed to reduce the CPU time
## Not run:
  res1 <- update(res1, 150)
  # iterating the algorithm up to 150 more iterations

## End(Not run)

res1 # print method
plot(res1) # Veryfying the general equivalence theorem

## Not run:
  ## fixed x
  res1.1 <- minimax (formula = ~a + exp(-b*x), predvars = "x", parvars = c("a", "b"),
                     lx = 0, ux = 1, lp = c(1, 1), up = c(1, 10),
```

```
                    x = c(0, .5, 1),
                    iter = 150, k = 3, ICA.control= ICA.control(rseed = 100))
  # not optimal

## End(Not run)

########################################
# Two-parameter logistic model.
########################################
# A little playing with the tuning parameters
# The value of maxeval is reduced to 200 to increase the speed
cont1 <- crt.minimax.control(optslist = list(maxeval = 200))
cont2 <- ICA.control(rseed = 100, checkfreq = Inf, ncount = 60)

## Not run:
  res2 <- minimax (formula = ~1/(1 + exp(-b *(x - a))), predvars = "x",
                   parvars = c("a", "b"),
                   family = binomial(), lx = -3, ux = 3,
                   lp = c(0, 1), up = c(1, 2.5), iter = 200, k = 3,
                   ICA.control= cont2, crt.minimax.control = cont1)
  print(res2)
  plot(res2)

## End(Not run)

############################################
# An example of a model with two predictors
############################################
# Mixed inhibition model
lower <- c(1, 4, 2, 4)
upper <- c(1, 5, 3, 5)
cont <- crt.minimax.control(optslist = list(maxeval = 100)) # to be faster
## Not run:
  res3 <- minimax(formula =  ~ V*S/(Km * (1 + I/Kic)+ S * (1 + I/Kiu)),
                  predvars = c("S", "I"),
                  parvars = c("V", "Km", "Kic", "Kiu"),
                  lx = c(0, 0), ux = c(30, 60), k = 4,
                  iter = 100, lp = lower, up = upper,
                  ICA.control= list(rseed = 100),
                  crt.minimax.control = cont)

  res3 <- update(res3, 100)
  print(res3)
  plot(res3) # sensitivity plot
  res3$arg$time

## End(Not run)

# Now consider grid points instead of assuming continuous parameter space
# set n.grid to 5
## Not run:
  res4 <- minimax(formula =  ~ V*S/(Km * (1 + I/Kic)+ S * (1 + I/Kiu)),
                  predvars = c("S", "I"),
```

```
                  parvars = c("V", "Km", "Kic", "Kiu"),
                  lx = c(0, 0), ux = c(30, 60),
                  k = 4, iter = 130, n.grid = 5, lp = lower, up = upper,
                  ICA.control= list(rseed = 100, checkfreq = Inf),
                  crt.minimax.control = cont)
  print(res4)
  plot(res4) # sensitivity plot

## End(Not run)

############################################
# Standardized maximin D-optimal designs
############################################
# Assume the purpose is finding STANDARDIZED designs
# We know from literature that the locally D-optimal design (LDOD)
# for this model has an analytical solution.
# The follwoing function takes the parameter as input and returns
# the design points and weights of LDOD.
# x and w are exactly similar to the arguments of 'fimfunc'.
# x is a vector and returns the design points 'dimension-wise'.
# see explanation of the arguments of 'fimfunc' in 'Details'.

LDOD <- function(V, Km, Kic, Kiu){
  #first dimention is for S and the second one is for I.
  S_min <- 0
  S_max <- 30
  I_min <- 0
  I_max <- 60
  s2 <- max(S_min, S_max*Km*Kiu*(Kic+I_min)/
              (S_max*Kic*I_min+S_max*Kic*Kiu+2*Km*Kiu*I_min+2*Km*Kiu*Kic))
  i3 <- min((2*S_max*Kic*I_min + S_max*Kic*Kiu+2*Km*Kiu*I_min+Km*Kiu*Kic)/
              (Km*Kiu+S_max*Kic), I_max)
  i4 <- min(I_min + (sqrt((Kic+I_min)*(Km*Kic*Kiu+Km*Kiu*I_min+
                                          S_max*Kic*Kiu+S_max*Kic*I_min)/
                            (Km*Kiu+S_max*Kic))), I_max )
  s4 <- max(-Km*Kiu*(Kic+2*I_min-i4)/(Kic*(Kiu+2*I_min-i4)), S_min)
  x <- c(S_max, s2, S_max, s4, I_min, I_min, i3, i4)
  return(list(x = x, w =rep(1/4, 4)))

}
formalArgs(LDOD)
## Not run:
  minimax(formula =  ~ V*S/(Km * (1 + I/Kic)+ S * (1 + I/Kiu)),
          predvars = c("S", "I"),
          parvars = c("V", "Km", "Kic", "Kiu"),
          lx = c(0, 0), ux = c(30, 60),
          k = 4, iter = 300,
          lp = lower, up = upper,
          ICA.control= list(rseed = 100, checkfreq = Inf),
          crt.minimax.control = cont,
          standardized = TRUE,
          localdes = LDOD)
```

```
## End(Not run)


################################################################
# Not necessary!
# The rest of the examples here are only for professional uses.
################################################################
# Imagine you have written your own FIM, say in Rcpp that is faster than
# the FIM created by the formula interface above.

###########################################
# An example of a model with two predictors
###########################################
# For example, th cpp FIM function for the mixed inhibition model is named:
formalArgs(FIM_mixed_inhibition)

# We should reparamterize the arguments to match the standard of the
# argument 'fimfunc' (see 'Details').
myfim <- function(x, w, param){
  npoint <- length(x)/2
  S <- x[1:npoint]
  I <- x[(npoint+1):(npoint*2)]
  out <- FIM_mixed_inhibition(S = S, I = I, w = w, param = param)
  return(out)
}
formalArgs(myfim)

# Finds minimax optimal design, exactly as before, but NOT using the
# formula interface.
## Not run:
  res5 <- minimax(fimfunc = myfim,
                  lx = c(0, 0), ux = c(30, 60), k = 4,
                  iter = 100, lp = lower, up = upper,
                  ICA.control= list(rseed = 100),
                  crt.minimax.control = cont)
  print(res5)
  plot(res5) # sensitivity plot

## End(Not run)
#########################################
# Standardized maximin D-optimal designs
#########################################
# To match the argument 'localdes' when no formula inteface is used,
# we should reparameterize LDOD.
# The input must be 'param' same as the argument of 'fimfunc'
LDOD2 <- function(param)
  LDOD(V = param[1], Km = param[2], Kic = param[3], Kiu = param[4])

# compare these two:
formalArgs(LDOD)
formalArgs(LDOD2)
## Not run:
  res6 <- minimax(fimfunc = myfim,
```

```
                 lx = c(0, 0), ux = c(30, 60), k = 4,
                 iter = 300, lp = lower, up = upper,
                 ICA.control= list(rseed = 100, checkfreq = Inf),
                 crt.minimax.control = cont,
                 standardized = TRUE,
                 localdes = LDOD2)
  res6
  plot(res6)

## End(Not run)

###################################
# user-defined optimality criterion
##################################
# When the model is defined by the formula interface
# A-optimal design for the 2PL model.
# the criterion function must have argument x, w fimfunc and the parameters defined in 'parvars'.
# use 'fimfunc' as a function of the design points x,  design weights w and
#  the 'parvars' parameters whenever needed.
Aopt <-function(x, w, a, b, fimfunc){
  sum(diag(solve(fimfunc(x = x, w = w, a = a, b = b))))
}
## the sensitivtiy function
# xi_x is a design that put all its mass on x in the definition of the sensitivity function
# x is a vector of design points
Aopt_sens <- function(xi_x, x, w, a, b, fimfunc){
  fim <- fimfunc(x = x, w = w, a = a, b = b)
  M_inv <- solve(fim)
  M_x <- fimfunc(x = xi_x, w = 1, a  = a, b = b)
  sum(diag(M_inv %*% M_x %*%  M_inv)) - sum(diag(M_inv))
}
## Not run:
res7 <- minimax(formula = ~1/(1 + exp(-b * (x-a))), predvars = "x",
                parvars = c("a", "b"), family = "binomial",
                lx = -2, ux = 2,
                lp = c(-2, 1), up = c(2, 1.5),
                iter = 400, k = 3,
                crtfunc = Aopt,
                sensfunc = Aopt_sens,
                crt.minimax.control = list(optslist = list(maxeval = 200)),
                ICA.control = list(rseed = 1))
  plot(res7)

## End(Not run)
# with grid points
res7.1 <- minimax(formula = ~1/(1 + exp(-b * (x-a))), predvars = "x",
                  parvars = c("a", "b"), family = "binomial",
                  lx = -2, ux = 2,
                  lp = c(-2, 1), up = c(2, 1.5),
                  iter = 1, k = 3,
                  crtfunc = Aopt,
                  sensfunc = Aopt_sens,
                  n.grid = 9,
```

```
                  ICA.control = list(rseed = 1))
## Not run:
  res7.1 <- update(res7.1, 400)
  plot(res7.1)

## End(Not run)

# When the FIM of the model is defined directly via the argument 'fimfunc'
# the criterion function must have argument x, w fimfunc and param.
# use 'fimfunc' as a function of the design points x,  design weights w and
#  the 'parvars' parameters whenever needed.
Aopt2 <-function(x, w, param, fimfunc){
  sum(diag(solve(fimfunc(x = x, w = w, param = param))))
}
## the sensitivtiy function
# xi_x is a design that put all its mass on x in the definition of the sensitivity function
# x is a vector of design points
Aopt_sens2 <- function(xi_x, x, w, param, fimfunc){
  fim <- fimfunc(x = x, w = w, param = param)
  M_inv <- solve(fim)
  M_x <- fimfunc(x = xi_x, w = 1, param = param)
  sum(diag(M_inv %*% M_x %*%  M_inv)) - sum(diag(M_inv))
}
## Not run:
res7.2 <- minimax(fimfunc = FIM_logistic,
                  lx = -2, ux = 2,
                  lp = c(-2, 1), up = c(2, 1.5),
                  iter = 1, k = 3,
                  crtfunc = Aopt2,
                  sensfunc = Aopt_sens2,
                  crt.minimax.control = list(optslist = list(maxeval = 200)),
                  ICA.control = list(rseed = 1))
  res7.2 <- update(res7.2, 200)
  plot(res7.2)

## End(Not run)
# with grid points
res7.3 <- minimax(fimfunc = FIM_logistic,
                  lx = -2, ux = 2,
                  lp = c(-2, 1), up = c(2, 1.5),
                  iter = 1, k = 3,
                  crtfunc = Aopt2,
                  sensfunc = Aopt_sens2,
                  n.grid = 9,
                  ICA.control = list(rseed = 1))
## Not run:
  res7.3 <- update(res7.2, 200)
  plot(res7.3)

## End(Not run)


# robust c-optimal design
```

```
# example from Chaloner and Larntz (1989), Figure 3, but robust design
c_opt <-function(x, w, a, b, fimfunc){
  gam <- log(.95/(1-.95))
  M <- fimfunc(x = x, w = w, a = a, b = b)
  c <- matrix(c(1, -gam * b^(-2)), nrow = 1)
  B <- t(c) %*% c
  sum(diag(B %*% solve(M)))
}

c_sens <- function(xi_x, x, w, a, b, fimfunc){
  gam <- log(.95/(1-.95))
  M <- fimfunc(x = x, w = w, a = a, b = b)
  M_inv <- solve(M)
  M_x <- fimfunc(x = xi_x, w = 1, a = a, b = b)
  c <- matrix(c(1, -gam * b^(-2)), nrow = 1)
  B <- t(c) %*% c
  sum(diag(B %*% M_inv %*% M_x %*%  M_inv)) - sum(diag(B %*% M_inv))
}

## Not run:
res8 <- minimax(formula = ~1/(1 + exp(-b * (x-a))), predvars = "x",
                parvars = c("a", "b"), family = "binomial",
                lx = -1, ux = 1,
                lp = c(-.3, 6), up = c(.3, 8),
                iter = 500, k = 3,
                crtfunc = c_opt, sensfunc = c_sens,
                ICA.control = list(rseed = 1, ncount = 100),
                n.grid = 12)
  plot(res8)

## End(Not run)
```

---

multiple — *Locally Multiple Objective Optimal Designs for the 4-Parameter Hill Model*

Description

The 4-parameter Hill model is of the form

$$f(D) = c + \frac{(d-c)(\frac{D}{a})^b}{1 + (\frac{D}{a})^b} + \epsilon,$$

where $\epsilon \sim N(0, \sigma^2)$, D is the dose level and the predictor, a is the ED50, d is the upper limit of response, c is the lower limit of response and b denotes the Hill constant that control the flexibility

in the slope of the response curve.
Sometimes, the Hill model is re-parameterized and written as

$$f(x) = \frac{\theta_1}{1 + exp(\theta_2 x + \theta_3)} + \theta_4,$$

where $\theta_1 = d - c$, $\theta_2 = -b$, $\theta_3 = b\log(a)$, $\theta_4 = c$, $\theta_1 > 0$, $\theta_2 \neq 0$, and $-\infty < ED50 < \infty$, where $x = log(D) \in [-M, M]$ for some sufficiently large value of M. The new form is sometimes called 4-parameter logistic model.
The function `multiple` finds locally multiple-objective optimal designs for estimating the model parameters, the ED50, and the MED, simultaneously. For more details, see Hyun and Wong (2015).

Usage

```
multiple(
  minDose,
  maxDose,
  iter,
  k,
  inipars,
  Hill_par = TRUE,
  delta,
  lambda,
  fimfunc = NULL,
  ICA.control = list(),
  sens.control = list(),
  initial = NULL,
  tol = sqrt(.Machine$double.xmin),
  x = NULL
)
```

Arguments

`minDose`	Minimum dose D. For the 4-parameter logistic model, i.e. when `Hill_par = FALSE`, it is the minimum of $log(D)$.
`maxDose`	Maximum dose D. For the 4-parameter logistic model, i.e. when `Hill_par = FALSE`, it is the maximum of $log(D)$.
`iter`	Maximum number of iterations.
`k`	Number of design points. Must be at least equal to the number of model parameters to avoid singularity of the FIM.
`inipars`	A vector of initial estimates for the vector of parameters (a, b, c, d). For the 4-parameter logistic model, i.e. when `Hill_par = FALSE`, it is a vector of initial estimates for $(\theta_1, \theta_2, \theta_3, \theta_4)$.
`Hill_par`	Hill model parameterization? Defaults to `TRUE`.
`delta`	Predetermined meaningful value of the minimum effective dose MED. When $\delta < 0$, then $\theta_2 > 0$ or when $\delta > 0$, then $\theta_2 < 0$.
`lambda`	A vector of relative importance of each of the three criteria, i.e. $\lambda = (\lambda_1, \lambda_2, \lambda_3)$. Here $0 < \lambda_i < 1$ and s $\sum \lambda_i = 1$.

`fimfunc` A function. Returns the FIM as a `matrix`. Required when `formula` is missing. See 'Details' of `minimax`.

`ICA.control` ICA control parameters. For details, see `ICA.control`.

`sens.control` Control Parameters for Calculating the ELB. For details, see `sens.control`.

`initial` A matrix of the initial design points and weights that will be inserted into the initial solutions (countries) of the algorithm. Every row is a design, i.e. a concatenation of `x` and `w`. Will be coerced to a `matrix` if necessary. See 'Details' of `minimax`.

`tol` Tolerance for finding the general inverse of the Fisher information matrix. Defaults to `.Machine$double.xmin`.

`x` A vector of candidate design (support) points. When is not set to `NULL` (default), the algorithm only finds the optimal weights for the candidate points in `x`. Should be set when the user has a finite number of candidate design points and the purpose is to find the optimal weight for each of them (when zero, they will be excluded from the design). For design points with more than one dimension, see 'Details' of `sensminimax`.

Details

When $\lambda_1 > 0$, then the number of support points `k` must at least be four to avoid singularity of the Fisher information matrix.

One can adjust the tuning parameters in `ICA.control` to set a stopping rule based on the general equivalence theorem. See 'Examples' below.

Value

an object of class `minimax` that is a list including three sub-lists:

`arg` A list of design and algorithm parameters.

`evol` A list of length equal to the number of iterations that stores the information about the best design (design with least criterion value) of each iteration. `evol[[iter]]` contains:

`iter`	Iteration number.
`x`	Design points.
`w`	Design weights.
`min_cost`	Value of the criterion for the best imperialist (design).
`mean_cost`	Mean of the criterion values of all the imperialists.
`sens`	An object of class `'sensminimax'`. See below.
`param`	Vector of parameters.

`empires` A list of all the empires of the last iteration.

`alg` A list with following information:

`nfeval`	Number of function evaluations. It does not count the function evaluations from checking the general equiv
`nlocal`	Number of successful local searches.

nrevol	Number of successful revolutions.
nimprove	Number of successful movements toward the imperialists in the assimilation step.
convergence	Stopped by 'maxiter' or 'equivalence'?

method A type of optimal designs used.

design Design points and weights at the final iteration.

out A data frame of design points, weights, value of the criterion for the best imperialist (min_cost), and Mean of the criterion values of all the imperialistsat each iteration (mean_cost).

The list sens contains information about the design verification by the general equivalence theorem. See sensminimax for more details. It is given every ICA.control$checkfreq iterations and also the last iteration if ICA.control$checkfreq >= 0. Otherwise, NULL.

param is a vector of parameters that is the global minimum of the minimax criterion or the global maximum of the standardized maximin criterion over the parameter space, given the current x, w.

Note

This function is NOT appropriate for finding c-optimal designs for estimating 'MED' or 'ED50' (single objective optimal designs) and the results may not be stable. The reason is that for the c-optimal criterion the generalized inverse of the Fisher information matrix is not stable and depends on the tolerance value (tol).

References

Hyun, S. W., and Wong, W. K. (2015). Multiple-Objective Optimal Designs for Studying the Dose Response Function and Interesting Dose Levels. The international journal of biostatistics, 11(2), 253-271.

See Also

sensmultiple

Examples

```
# All the examples are available in Hyun and Wong (2015)

#################################
#  4-parameter logistic model
# Example 1, Table 3
#################################
lam <- c(0.05, 0.05, .90)
# Initial estimates are derived from Table 1
# See how the stopping rules are set via 'stop_rul', checkfreq' and 'stoptol'
Theta1 <- c(1.563, 1.790, 8.442, 0.137)
res1 <- multiple(minDose = log(.001), maxDose = log(1000),
                 inipars = Theta1, k = 4, lambda = lam, delta = -1,
                 Hill_par = FALSE,
                 iter = 1,
                 ICA.control = list(rseed = 1366, ncount = 100,
```

```
                                   stop_rule = "equivalence",
                                   checkfreq = 100, stoptol = .95))
## Not run:
res1 <- update(res1, 1000)
# stops at iteration 101

## End(Not run)

#################################
#  4-parameter Hill model
#################################
## initial estimates for the parameters of Hill model:
a <- 0.008949  # ED50
b <- -1.79 # Hill constant
c <- 0.137 # lower limit
d <- 1.7 # upper limit
# D belongs to c(.001, 1000) ## dose in mg
## the vector of Hill parameters are now c(a, b, c, d)
## Not run:
res2 <- multiple(minDose = .001, maxDose = 1000,
                 inipars =  c(a, b, c, d),
                 Hill_par = TRUE, k = 4, lambda = lam,
                 delta = -1, iter = 1000,
                 ICA.control = list(rseed = 1366, ncount = 100,
                                    stop_rule = "equivalence",
                                    checkfreq = 100, stoptol = .95))
# stops at iteration 100

## End(Not run)



# use x argument to provide fix number of  dose levels.
# In this case, the optimization is only over weights
## Not run:
res3 <- multiple(minDose = log(.001), maxDose = log(1000),
                 inipars = Theta1, k = 4, lambda = lam, delta = -1,
                 iter = 300,
                 Hill_par = FALSE,
                 x = c(-6.90, -4.66, -3.93, 3.61),
                 ICA.control = list(rseed = 1366))
res3$evol[[300]]$w
# if the user provide the desugn points via x, there is no guarantee
#   that the resulted design is optimal. It only provides the optimal weights given
#   the x points of the design.
plot(res3)

## End(Not run)
```

`normal` *Assumes A Multivariate Normal Prior Distribution for The Model Parameters*

Description

Creates a multivariate normal prior distribution for the unknown parameters as an object of class `cprior`.

Usage

```
normal(mu, sigma, lower, upper)
```

Arguments

`mu`	A vector representing the mean values.
`sigma`	A symmetric positive-definite matrix representing the variance-covariance matrix of the distribution.
`lower`	A vector of lower bounds for the model parameters.
`upper`	A vector of upper bounds for the model parameters.

Value

An object of class `cprior` that is a list with the following components:

- `fn`: prior distribution as an R `function` with argument `param` that is the vector of the unknown parameters. See below.
- `npar`: Number of unknown parameters and is equal to the length of `param`.
- `lower`: Argument `lower`. It has the same length as `param`.
- `upper`: Argument `lower`. It has the same length as `param`.

The list will be passed to the argument `prior` of the function `bayes`. The order of the argument `param` in `fn` has the same order as the argument `parvars` when the model is specified by a formula. Otherwise, it is equal to the argument `param` in the function `fimfunc`.

See Also

`bayes` `sensbayes`

Examples

```
normal(mu =  c(0, 1), sigma = matrix(c(1, -0.17, -0.17, .5), nrow = 2),
  lower =  c(-3, .1), upper = c(3, 2))
```

plot.minimax *Plotting* minimax *Objects*

Description

This function plots the evolution of the ICA algorithm (iteration vs the best (minimum) criterion value at each iteration) and also verifies the optimality of the last obtained design using the general equivalence theorem. It plots the sensitivity function and calculates the ELB for the best design generated at iteration number `iter`.

Usage

```
## S3 method for class 'minimax'
plot(
  x,
  iter = NULL,
  sensitivity = TRUE,
  calculate_criterion = FALSE,
  sens.minimax.control = list(),
  crt.minimax.control = list(),
  sens.bayes.control = list(),
  crt.bayes.control = list(),
  sens.control = list(),
  silent = TRUE,
  plot_3d = c("lattice", "rgl"),
  evolution = FALSE,
  ...
)
```

Arguments

`x` An object of class `minimax`.

`iter` Iteration number. if `NULL` (default), it will be set to the last iteration.

`sensitivity` Logical. If `TRUE` (default), the general equivalence theorem is used to check the optimality if the best design in iteration number `iter` and the sensitivity function will be plotted.

`calculate_criterion`
: Logical. Re-calculate the criterion value (maybe with a set of new tuning parameters to be sure of the globality of the maximum over the parameter space given the design)? It only assumes a continuous parameter space for the minimax and standardized maximin designs. Defaults to `FALSE`. See 'Details'.

`sens.minimax.control`
: Control parameters to verify general equivalence theorem. For details, see `sens.minimax.control`. If `NULL` (default), it will be set to the tuning parameters used to create object `x`.

crt.minimax.control
: Control parameters to optimize the minimax or standardized maximin criterion at a given design over a **continuous** parameter space. For details, see crt.minimax.control.

sens.bayes.control
: Control parameters to verify general equivalence theorem for the Bayesian optimal designs. For details, see sens.bayes.control. If NULL (default), it will be set to the tuning parameters used to create object x.

crt.bayes.control
: Control parameters to optimize the integration in the Bayesian criterion at a given design over a **continuous** parameter space. For details, see crt.bayes.control. If NULL (default), it will be set to the tuning parameters used to create object x. If NULL (default), it will be set to the tuning parameters used to create object x.

sens.control
: Control Parameters for Calculating the ELB. For details, see the function sens.control.

silent
: Do not print anything? Defaults to TRUE.

plot_3d
: Which package should be used to plot the sensitivity function for two-dimensional design space. Defaults to plot_3d = "lattice". Only applicable when sensitivity = TRUE.

evolution
: Plot Evolution? Defaults to FALSE.

...
: Argument with no further use.

Details

In addition to verifying the general equivalence theorem, this function makes it possible to re-calculated the criterion value for the output designs using a new set of tuning parameters, especially, a large value for maxeval in the function crt.minimax.control. This is useful for minimax and standardized maximin optimal designs to assess the robustness of the criterion value with respect to different values of maxeval. To put it simple, for these designs, the user can re-calculate the criterion value (finds the global maximum over the parameter space given an output design in a minimax problem) with larger values for maxeval in crt.minimax.control to be sure that the function nloptr finds global optima of the inner optimization problem over the parameter space using the default value (or the user-given value) of maxeval. If increasing the value of maxeval returns different criterion values, then the results can not be trusted and the algorithm should be repeated with a higher value for maxeval.

See Also

minimax, locally, robust

---

print.minimax *Printing* minimax *Objects*

Description

Print method for an object of class minimax.

Usage

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

Arguments

`x`	An object of class `minimax`.
`...`	Argument with no further use.

See Also

`minimax`, `locally`, `robust`, `bayes`

`print.sensminimax` *Printing* `sensminimax` *Objects*

Description

Print method for an object of class `sensminimax`.

Usage

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

Arguments

`x`	An object of class `sensminimax`.
`...`	Argument with no further use.

See Also

`sensminimax`, `senslocally`, `sensrobust`

robust *Robust D-Optimal Designs*

Description

Finds Robust designs or optimal in-average designs for linear and nonlinear models. It is useful when a set of different vectors of initial estimates along with a discrete probability measure are available for the unknown model parameters. It is a discrete version of `bayes`.

Usage

```
robust(
  formula,
  predvars,
  parvars,
  family = gaussian(),
  lx,
  ux,
  iter,
  k,
  prob,
  parset,
  fimfunc = NULL,
  ICA.control = list(),
  sens.control = list(),
  initial = NULL,
  npar = dim(parset)[2],
  plot_3d = c("lattice", "rgl"),
  x = NULL,
  crtfunc = NULL,
  sensfunc = NULL
)
```

Arguments

`formula`	A linear or nonlinear model `formula`. A symbolic description of the model consists of predictors and the unknown model parameters. Will be coerced to a `formula` if necessary.
`predvars`	A vector of characters. Denotes the predictors in the `formula`.
`parvars`	A vector of characters. Denotes the unknown parameters in the `formula`.
`family`	A description of the response distribution and the link function to be used in the model. This can be a family function, a call to a family function or a character string naming the family. Every family function has a link argument allowing to specify the link function to be applied on the response variable. If not specified, default links are used. For details see `family`. By default, a linear gaussian model `gaussian()` is applied.

lx	Vector of lower bounds for the predictors. Should be in the same order as predvars.
ux	Vector of upper bounds for the predictors. Should be in the same order as predvars.
iter	Maximum number of iterations.
k	Number of design points. Must be at least equal to the number of model parameters to avoid singularity of the FIM.
prob	A vector of the probability measure π associated with each row of parset.
parset	A matrix that provides the vector of initial estimates for the model parameters, i.e. support of π. Every row is one vector (nrow(parset) == length(prob)). See 'Details'.
fimfunc	A function. Returns the FIM as a matrix. Required when formula is missing. See 'Details' of minimax.
ICA.control	ICA control parameters. For details, see ICA.control.
sens.control	Control Parameters for Calculating the ELB. For details, see sens.control.
initial	A matrix of the initial design points and weights that will be inserted into the initial solutions (countries) of the algorithm. Every row is a design, i.e. a concatenation of x and w. Will be coerced to a matrix if necessary. See 'Details' of minimax.
npar	Number of model parameters. Used when fimfunc is given instead of formula to specify the number of model parameters. If not given, the sensitivity plot may be shifted below the y-axis. When NULL, it is set to dim(parset)[2].
plot_3d	Which package should be used to plot the sensitivity (derivative) function for models with two predictors. Either "rgl" or "lattice" (default).
x	Vector of the design (support) points. See 'Details' of sensminimax for models with more than one predictors.
crtfunc	(Optional) a function that specifies an arbitrary criterion. It must have especial arguments and output. See 'Details' of minimax.
sensfunc	(Optional) a function that specifies the sensitivity function for crtfunc. See 'Details' of minimax.

Details

Let Θ be a set of initial estimates for the unknown parameters. A robust criterion is evaluated at the elements of Θ weighted by a probability measure π as follows:

$$B(\xi, \pi) = \int_{\Theta} |M(\xi, \theta)|\pi(\theta) d\theta.$$

A robust design ξ^* maximizes $B(\xi, \pi)$ over the space of all designs.

When the model is given via formula, columns of parset must match the parameters introduced in parvars. Otherwise, when the model is introduced via fimfunc, columns of parset must match the argument param in fimfunc.

To verify the optimality of the output design by the general equivalence theorem, the user can either plot the results or set checkfreq in ICA.control to Inf. In either way, the function sensrobust is called for verification. One can also adjust the tuning parameters in ICA.control to set a stopping rule based on the general equivalence theorem. See 'Examples' below.

Value

an object of class `minimax` that is a list including three sub-lists:

`arg` A list of design and algorithm parameters.

`evol` A list of length equal to the number of iterations that stores the information about the best design (design with least criterion value) of each iteration. `evol[[iter]]` contains:

`iter`	Iteration number.
`x`	Design points.
`w`	Design weights.
`min_cost`	Value of the criterion for the best imperialist (design).
`mean_cost`	Mean of the criterion values of all the imperialists.
`sens`	An object of class `'sensminimax'`. See below.
`param`	Vector of parameters.

`empires` A list of all the empires of the last iteration.

`alg` A list with following information:

`nfeval`	Number of function evaluations. It does not count the function evaluations from checking the general equiv
`nlocal`	Number of successful local searches.
`nrevol`	Number of successful revolutions.
`nimprove`	Number of successful movements toward the imperialists in the assimilation step.
`convergence`	Stopped by `'maxiter'` or `'equivalence'`?

`method` A type of optimal designs used.

`design` Design points and weights at the final iteration.

`out` A data frame of design points, weights, value of the criterion for the best imperialist (min_cost), and Mean of the criterion values of all the imperialistsat each iteration (mean_cost).

The list `sens` contains information about the design verification by the general equivalence theorem. See `sensminimax` for more details. It is given every `ICA.control$checkfreq` iterations and also the last iteration if `ICA.control$checkfreq >= 0`. Otherwise, `NULL`.

`param` is a vector of parameters that is the global minimum of the minimax criterion or the global maximum of the standardized maximin criterion over the parameter space, given the current `x`, `w`.

Note

When a continuous prior distribution for the unknown model parameters is available, use `bayes`. When only one initial estimates of the unknown model parameters is available (Θ has only one element), use `locally`.

See Also

`bayes` `sensrobust`

Examples

```
# Finding a robust design for the two-parameter logistic model
# See how we set a stopping rule.
# The ELB is computed every checkfreq = 30 iterations
# The optimization stops when the ELB is larger than stoptol = .95
res1 <- robust(formula = ~1/(1 + exp(-b *(x - a))),
               predvars = c("x"), parvars = c("a", "b"),
               family = binomial(),
               lx = -5, ux = 5, prob = rep(1/4, 4),
               parset = matrix(c(0.5, 1.5, 0.5, 1.5, 4.0, 4.0, 5.0, 5.0), 4, 2),
               iter = 1, k =3,
               ICA.control = list(stop_rule = "equivalence",
                                  stoptol = .95, checkfreq = 30))

## Not run:
  res1 <- update(res1, 100)
  # stops at iteration 51

## End(Not run)


## Not run:
  res1.1 <- robust(formula = ~1/(1 + exp(-b *(x - a))),
                   predvars = c("x"), parvars = c("a", "b"),
                   family = binomial(),
                   lx = -5, ux = 5, prob = rep(1/4, 4),
                   parset = matrix(c(0.5, 1.5, 0.5, 1.5, 4.0, 4.0, 5.0, 5.0), 4, 2),
                   x = c(-3, 0, 3),
                   iter = 150, k =3)
  plot(res1.1)
  # not optimal

## End(Not run)


###################################
# user-defined optimality criterion
##################################
# When the model is defined by the formula interface
# A-optimal design for the 2PL model.
# the criterion function must have argument x, w fimfunc and the parameters defined in 'parvars'.
# use 'fimfunc' as a function of the design points x,  design weights w and
#  the 'parvars' parameters whenever needed.
Aopt <-function(x, w, a, b, fimfunc){
  sum(diag(solve(fimfunc(x = x, w = w, a = a, b = b))))
}
## the sensitivtiy function
# xi_x is a design that put all its mass on x in the definition of the sensitivity function
# x is a vector of design points
Aopt_sens <- function(xi_x, x, w, a, b, fimfunc){
  fim <- fimfunc(x = x, w = w, a = a, b = b)
  M_inv <- solve(fim)
```

```
  M_x <- fimfunc(x = xi_x, w = 1, a  = a, b = b)
  sum(diag(M_inv %*% M_x %*%  M_inv)) - sum(diag(M_inv))
}

res2 <- robust(formula = ~1/(1 + exp(-b * (x-a))), predvars = "x",
               parvars = c("a", "b"), family = "binomial",
               lx = -3, ux = 3,
               iter = 1, k = 4,
               crtfunc = Aopt,
               sensfunc = Aopt_sens,
               prob = c(.25, .5, .25),
               parset = matrix(c(-2, 0, 2, 1.25, 1.25, 1.25), 3, 2),
               ICA.control = list(checkfreq = 50, stoptol = .999,
                                  stop_rule = "equivalence",
                                  rseed = 1))
## Not run:
  res2 <- update(res2, 500)

## End(Not run)





# robust c-optimal design
# example from Chaloner and Larntz (1989), Figure 3, but robust design
c_opt <-function(x, w, a, b, fimfunc){
  gam <- log(.95/(1-.95))
  M <- fimfunc(x = x, w = w, a = a, b = b)
  c <- matrix(c(1, -gam * b^(-2)), nrow = 1)
  B <- t(c) %*% c
  sum(diag(B %*% solve(M)))
}

c_sens <- function(xi_x, x, w, a, b, fimfunc){
  gam <- log(.95/(1-.95))
  M <- fimfunc(x = x, w = w, a = a, b = b)
  M_inv <- solve(M)
  M_x <- fimfunc(x = xi_x, w = 1, a = a, b = b)
  c <- matrix(c(1, -gam * b^(-2)), nrow = 1)
  B <- t(c) %*% c
  sum(diag(B %*% M_inv %*% M_x %*%  M_inv)) - sum(diag(B %*% M_inv))
}


res3 <- robust(formula = ~1/(1 + exp(-b * (x-a))), predvars = "x",
               parvars = c("a", "b"), family = "binomial",
               lx = -1, ux = 1,
               parset = matrix(c(0, 7, .2, 6.5), 2, 2, byrow = TRUE),
               prob = c(.5, .5),
               iter = 1, k = 3,
               crtfunc = c_opt, sensfunc = c_sens,
               ICA.control = list(rseed = 1, checkfreq = Inf))
```

```
## Not run:
  res3 <- update(res3, 300)

## End(Not run)
```

---

sens.bayes.control *Returns Control Parameters for Approximating The Integrals In The Bayesian Sensitivity Functions*

Description

This function returns two lists each corresponds to an implemented integration method for approximating the integrals in the sensitivity (derivative) functions for the Bayesian optimality criteria.

Usage

```
sens.bayes.control(
  method = c("cubature", "quadrature"),
  cubature = list(tol = 1e-05, maxEval = 50000, absError = 0),
 quadrature = list(type = c("GLe", "GHe"), level = 6, ndConstruction = "product",
    level.trans = FALSE)
)
```

Arguments

`method`	A character denotes which method to be used to approximate the integrals in Bayesian criteria. `"cubature"` corresponds to the adaptive multivariate integration method using the `hcubature` algorithm (default). `"quadrature"` corresponds the traditional quadrature formulas and calls the function `createNIGrid`. The tuning parameters are adjusted by `crt.bayes.control`. Default is set to `"cubature"`.
`cubature`	A list that will be passed to the arguments of the `hcubature` function. See 'Details' of `crt.bayes.control`.
`quadrature`	A list that will be passed to the arguments of the `createNIGrid` function. See 'Details' of `crt.bayes.control`.

Value

A list of control parameters for approximating the integrals.

Examples

```
sens.bayes.control()
sens.bayes.control(cubature = list(maxEval = 50000))
sens.bayes.control(quadrature =  list(level = 4))
```

sens.control — *Returns Control Parameters To Find Maximum of The Sensitivity (Derivative) Function Over The Design Space*

Description

It returns some arguments of the `nloptr` function including the list of control parameters. This function is used to find the maximum of the sensitivity (derivative) function over the design space in order to calculate the efficiency lower bound (ELB).

Usage

```
sens.control(
  x0 = NULL,
 optslist = list(stopval = -Inf, algorithm = "NLOPT_GN_DIRECT_L", xtol_rel = 1e-08,
    ftol_rel = 1e-08, maxeval = 1000),
  ...
)
```

Arguments

`x0`	Vector of starting values for maximizing the sensitivity (derivative) function over the design space x. It will be passed to the optimization function `nloptr`.
`optslist`	A list. It will be passed to the argument `opts` of the function `nloptr` to find the maximum of the sensitivity function over the design space. See 'Details'.
`...`	Further arguments will be passed to `nl.opts` from package `nloptr`.

Details

ELB is a measure of proximity of a design to the optimal design without knowing the latter. Given a design, let ϵ be the global maximum of the sensitivity (derivative) function with respect the vector of the model predictors x over the design space. ELB is given by

$$ELB = p/(p + \epsilon),$$

where p is the number of model parameters. Obviously, calculating ELB requires finding ϵ and therefore, a maximization problem to be solved. The function `nloptr` is used here to solve this maximization problem. The arguments `x0` and `optslist` will be passed to this function as follows:

Argument `x0` provides the user initial values for this maximization problem and will be passed to the argument with the same name in the function `nloptr`.

Argument `optslist` will be passed to the argument `opts` of the function `nloptr`. `optslist` is a `list` and the most important components are listed as follows:

`stopval` Stop minimization when an objective value <= `stopval` is found. Setting `stopval` to `-Inf` disables this stopping criterion (default).

`algorithm` Defaults to `NLOPT_GN_DIRECT_L`. DIRECT-L is a deterministic-search algorithm based on systematic division of the search domain into smaller and smaller hyperrectangles.

`xtol_rel` Stop when an optimization step (or an estimate of the optimum) changes every parameter by less than `xtol_rel` multiplied by the absolute value of the parameter. Criterion is disabled if `xtol_rel` is non-positive.

`ftol_rel` Stop when an optimization step (or an estimate of the optimum) changes the objective function value by less than `ftol_rel` multiplied by the absolute value of the function value. Criterion is disabled if `ftol_rel` is non-positive.

`maxeval` Stop when the number of function evaluations exceeds `maxeval`. Criterion is disabled if `maxeval` is non-positive.

For more details, see `?nloptr::nloptr.print.options`.

Note

ELB must be `0 <=ELB <= 1`. When the computed ELB is larger than one (equivalently ϵ is negative), it may be a signal that the obtained ϵ is not the global maximum. To overcome this issue, please increase the value of the parameter `maxeval` to allow the optimization algorithm to find the global maximum of the sensitivity (derivative) function over the design space.

Examples

```
sens.control()
sens.control(optslist = list(maxeval = 1000))
```

`sens.minimax.control` *Returns Control Parameters for Verifying General Equivalence Theorem For Minimax Optimal Designs*

Description

This function returns a list of control parameters that are used to find the "answering set" for minimax and standardized maximin designs. The answering set is required to obtain the sensitivity (derivative) function in order to verify the optimality of a given design.

Usage

```
sens.minimax.control(n_seg = 6, merge_tol = 0.005)
```

Arguments

`n_seg`	For a given design, the number of starting points in the local search to find all the local maxima of the minimax criterion over the parameter space is equal to `(n_seg + 1)^p`. Defaults to `6`. Please increase its value when the parameter space is large. It is also applicable for standardized maximin designs. See 'Details' of sens.minimax.control.
`merge_tol`	Merging tolerance. It is used to specify the elements of the answering set by choosing only the local maxima (found by the local search) that are nearer to the global maximum. See 'Details' of sens.minimax.control. Defaults to `0.005`. We advise to not change its default value because it has been successfully tested on many optimal design problems.

Details

Given a design, an "answering set" is a subset of all the local optima of the optimality criterion over the parameter space. Answering set is used to obtain the sensitivity function of a minimax or standardized maximin criterion. Therefore, an invalid answering set may result in a false sensitivity plot and ELB. Unfortunately, there is no theoretical rule on how to choose the number of elements of the answering set; and they have to be found by trial and error. Given a design, the answering set for a minimax criterion is obtained as follows:

- Step 1: Find all the local maxima of the optimality criterion (minimax) over the parameter space. For this purpose, the parameter space is divided into `(n_seg + 1)^p` segments, where `p` is the number of unknown model parameters. Then, each boundary point of the resulted segments (intervals) is assigned to the argument `par` of the function `optim` in order to start a local search using the `"L-BFGS-B"` method.
- Step 2: Pick the ones nearest to the global minimum subject to a merging tolerance `merge_tol` (default `0.005`).

Obviously, the answering set is a subset of all the local maxima over the parameter space (or local minima in case of standardized maximin criteria) Therefore, it is very important to be able to find all the local maxima to create the true answering set with no missing elements. Otherwise, even when the design is optimal, the sensitivity (derivative) plot may not reveal its optimality.

Note that the minimax criterion (or standardized maximin criterion) is a multimodel function especially near the optimal design and this makes the job of finding all the locall maxima (minima) over the parameter space very complicated.

Value

A list of control parameters for verifying the general equivalence theorem for minimax and standardized maximin optimal designs.

Examples

```
sens.minimax.control()
sens.minimax.control(n_seg = 4)
```

`sensbayes` *Verifying Optimality of Bayesian D-optimal Designs*

Description

Plots the sensitivity (derivative) function and calculates the efficiency lower bound (ELB) for a given Bayesian design. Let $\boldsymbol{x}$ belongs to χ that denotes the design space. Based on the general equivalence theorem, a design ξ^* is optimal if and only if the value of the sensitivity (derivative) function is non-positive for all $\boldsymbol{x}$ in χ and zero when $\boldsymbol{x}$ belongs to the support of ξ^* (be equal to the one of the design points).

For an approximate (continuous) design, when the design space is one or two-dimensional, the user can visually verify the optimality of the design by observing the sensitivity plot. Furthermore, the proximity of the design to the optimal design can be measured by the ELB without knowing the latter.

Usage

```
sensbayes(
  formula,
  predvars,
  parvars,
  family = gaussian(),
  x,
  w,
  lx,
  ux,
  fimfunc = NULL,
  prior = list(),
  sens.control = list(),
  sens.bayes.control = list(),
  crt.bayes.control = list(),
  plot_3d = c("lattice", "rgl"),
  plot_sens = TRUE,
  npar = NULL,
  calculate_criterion = TRUE,
  silent = FALSE,
  crtfunc = NULL,
  sensfunc = NULL
)
```

Arguments

`formula`	A linear or nonlinear model `formula`. A symbolic description of the model consists of predictors and the unknown model parameters. Will be coerced to a `formula` if necessary.
`predvars`	A vector of characters. Denotes the predictors in the `formula`.
`parvars`	A vector of characters. Denotes the unknown parameters in the `formula`.
`family`	A description of the response distribution and the link function to be used in the model. This can be a family function, a call to a family function or a character string naming the family. Every family function has a link argument allowing to specify the link function to be applied on the response variable. If not specified, default links are used. For details see `family`. By default, a linear gaussian model `gaussian()` is applied.
`x`	A vector of candidate design (support) points. When is not set to `NULL` (default), the algorithm only finds the optimal weights for the candidate points in `x`. Should be set when the user has a finite number of candidate design points and the purpose is to find the optimal weight for each of them (when zero, they will be excluded from the design). For design points with more than one dimension, see 'Details' of `sensminimax`.
`w`	Vector of the corresponding design weights for `x`.
`lx`	Vector of lower bounds for the predictors. Should be in the same order as `predvars`.

ux	Vector of upper bounds for the predictors. Should be in the same order as `predvars`.
fimfunc	A function. Returns the FIM as a `matrix`. Required when `formula` is missing. See 'Details' of `minimax`.
prior	An object of class `cprior`. User can also use one of the functions `uniform`, `normal`, `skewnormal` or `student` to create the prior. See 'Details' of `bayes`.
sens.control	Control Parameters for Calculating the ELB. For details, see `sens.control`.
sens.bayes.control	A list. Control parameters to verify the general equivalence theorem. For details, see `sens.bayes.control`.
crt.bayes.control	A list. Control parameters to approximate the integral in the Bayesian criterion at a given design over the parameter space. For details, see `crt.bayes.control`.
plot_3d	Which package should be used to plot the sensitivity (derivative) function for two-dimensional design space. Defaults to `"lattice"`.
plot_sens	Plot the sensitivity (derivative) function? Defaults to `TRUE`.
npar	Number of model parameters. Used when `fimfunc` is given instead of `formula` to specify the number of model parameters. If not specified correctly, the sensitivity (derivative) plot may be shifted below the y-axis. When `NULL` (default), it will be set to `length(parvars)` or `prior$npar` when `missing(formula)`.
calculate_criterion	Calculate the optimality criterion? See 'Details' of `sensminimax`.
silent	Do not print anything? Defaults to `FALSE`.
crtfunc	(Optional) a function that specifies an arbitrary criterion. It must have especial arguments and output. See 'Details' of `bayes`.
sensfunc	(Optional) a function that specifies the sensitivity function for `crtfunc`. See 'Details' of `bayes`.

Details

Let Ξ be the space of all approximate designs with k design points (support points) at $x_1, x_2, ..., x_k$ from design space χ with corresponding weights $w_1, ..., w_k$. Let $M(\xi, \theta)$ be the Fisher information matrix (FIM) of a $k-$point design ξ and $\pi(\theta)$ is a user-given prior distribution for the vector of unknown parameters θ. A design ξ^* is Bayesian D-optimal among all designs on χ if and only if the following inequality holds for all $\boldsymbol{x} \in \chi$

$$c(\boldsymbol{x}, \xi^*) = \int_{\theta \in Theta} trM^{-1}(\xi^*, \theta)I(\boldsymbol{x}, \theta) - p\pi(\theta)d\theta \leq 0,$$

with equality at all support points of ξ^*. Here, p is the number of model parameters. $c(\boldsymbol{x}, \xi^*)$ is called **sensitivity** or **derivative** function.

Depending on the complexity of the problem at hand, sometimes, the CPU time can be considerably reduced by choosing a set of less conservative values for the tuning parameters `tol` and `maxEval` in the function `sens.bayes.control` when `sens.bayes.control$method = "cubature"`. Similarly, this applies when `sens.bayes.control$method = "quadrature"`. In general, if the CPU time matters, the user should find an appropriate speed-accuracy trade-off for her/his own problem. See 'Examples' for more details.

Note

The default values of the tuning parameters in `sens.bayes.control` are set in a way that having accurate plots for the sensitivity (derivative) function and calculating the ELB to a high precision to be the primary goals, although the process may take too long. The user should choose a set of less conservative values via the argument `sens.bayes.control` when the CPU-time is too long or matters.

Examples

```
###################################################################
# Checking the Bayesian D-optimality of a design for the 2Pl model
###################################################################
skew2 <- skewnormal(xi = c(0, 1), Omega = matrix(c(1, -0.17, -0.17, .5), nrow = 2),
                    alpha = c(-1, 0), lower =  c(-3, .1), upper = c(3, 2))
## Not run:
  sensbayes(formula = ~1/(1 + exp(-b *(x - a))),
            predvars = "x", parvars = c("a", "b"),
            family = binomial(),
            x= c(-2.50914, -1.16780, -0.36904, 1.29227),
            w =c(0.35767, 0.11032, 0.15621, 0.37580),
            lx = -3, ux = 3,
            prior = skew2)
  # took 29 seconds on my system!

## End(Not run)

# It took very long.
# We re-adjust the tuning parameters in sens.bayes.control to be faster
# See how we drastically reduce the maxEval and increase the tolerance
## Not run:
  sensbayes(formula = ~1/(1 + exp(-b *(x - a))),
            predvars = "x", parvars = c("a", "b"),
            family = binomial(),
            x= c(-2.50914, -1.16780, -0.36904, 1.29227),
            w =c(0.35767, 0.11032, 0.15621, 0.37580),
            lx = -3, ux = 3,prior = skew2,
            sens.bayes.control = list(cubature = list(tol = 1e-4, maxEval = 300)))
  # took 5 Seconds on my system!

## End(Not run)



# Compare it with the following:
sensbayes(formula = ~1/(1 + exp(-b *(x - a))),
          predvars = "x", parvars = c("a", "b"),
          family = binomial(),
          x= c(-2.50914, -1.16780, -0.36904, 1.29227),
          w =c(0.35767, 0.11032, 0.15621, 0.37580),
          lx = -3, ux = 3,prior = skew2,
          sens.bayes.control = list(cubature = list(tol = 1e-4, maxEval = 200)))
# Look at the plot!
```

```
# took 3 seconds on my system


##########################################################################################
# Checking the Bayesian D-optimality of a design for the 4-parameter sigmoid emax model
##########################################################################################
lb <- c(4, 11, 100, 5)
ub <- c(9, 17, 140, 10)
## Not run:
  sensbayes(formula = ~ theta1 + (theta2 - theta1)*(x^theta4)/(x^theta4 + theta3^theta4),
            predvars = c("x"), parvars = c("theta1", "theta2", "theta3", "theta4"),
            x = c(0.78990, 95.66297, 118.42964,147.55809, 500),
            w = c(0.23426, 0.17071, 0.17684, 0.1827, 0.23549),
            lx = .001, ux = 500,  prior = uniform(lb, ub))
  # took 200 seconds on my system

## End(Not run)

# Re-adjust the tuning parameters to have it faster
## Not run:
  sensbayes(formula = ~ theta1 + (theta2 - theta1)*(x^theta4)/(x^theta4 + theta3^theta4),
            predvars = c("x"), parvars = c("theta1", "theta2", "theta3", "theta4"),
            x = c(0.78990, 95.66297, 118.42964,147.55809, 500),
            w = c(0.23426, 0.17071, 0.17684, 0.1827, 0.23549),
            lx = .001, ux = 500,  prior = uniform(lb, ub),
            sens.bayes.control = list(cubature = list(tol = 1e-3, maxEval = 300)))
  # took 4 seconds on my system. See how much it makes difference

## End(Not run)

## Not run:
  # Now we try it with quadrature. Default is 6 nodes
  sensbayes(formula = ~ theta1 + (theta2 - theta1)*(x^theta4)/(x^theta4 + theta3^theta4),
            predvars = c("x"), parvars = c("theta1", "theta2", "theta3", "theta4"),
            x = c(0.78990, 95.66297, 118.42964,147.55809, 500),
            w = c(0.23426, 0.17071, 0.17684, 0.1827, 0.23549),
            sens.bayes.control = list(method = "quadrature"),
            lx = .001, ux = 500,  prior = uniform(lb, ub))
  # 166.519 s

  # use less number of nodes  to see if we can reduce the CPU time
  sensbayes(formula = ~ theta1 + (theta2 - theta1)*(x^theta4)/(x^theta4 + theta3^theta4),
            predvars = c("x"), parvars = c("theta1", "theta2", "theta3", "theta4"),
            x = c(0.78990, 95.66297, 118.42964,147.55809, 500),
            w = c(0.23426, 0.17071, 0.17684, 0.1827, 0.23549),
            sens.bayes.control = list(method = "quadrature",
                                      quadrature = list(level = 3)),
            lx = .001, ux = 500,  prior = uniform(lb, ub))
  # we don't have an accurate plot

  # use less number of levels: use 4 nodes
  sensbayes(formula = ~ theta1 + (theta2 - theta1)*(x^theta4)/(x^theta4 + theta3^theta4),
            predvars = c("x"), parvars = c("theta1", "theta2", "theta3", "theta4"),
```

```
            x = c(0.78990, 95.66297, 118.42964,147.55809, 500),
            w = c(0.23426, 0.17071, 0.17684, 0.1827, 0.23549),
            sens.bayes.control = list(method = "quadrature",
                                      quadrature = list(level = 4)),
            lx = .001, ux = 500,  prior = uniform(lb, ub))


## End(Not run)
```

`sensbayescomp` *Verifying Optimality of Bayesian Compound DP-optimal Designs*

Description

This function plot the sensitivity (derivative) function given an approximate (continuous) design and calculate the efficiency lower bound (ELB) for Bayesian DP-optimal designs. Let $\boldsymbol{x}$ belongs to χ that denotes the design space. Based on the general equivalence theorem, generally, a design ξ^* is optimal if and only if the value of its sensitivity (derivative) function be non-positive for all $\boldsymbol{x}$ in χ and it only reaches zero when $\boldsymbol{x}$ belong to the support of ξ^* (be equal to one of the design point). Therefore, the user can look at the sensitivity plot and the ELB and decide whether the design is optimal or close enough to the true optimal design (ELB tells us that without knowing the latter).

Usage

```
sensbayescomp(
  formula,
  predvars,
  parvars,
  family = gaussian(),
  x,
  w,
  lx,
  ux,
  fimfunc = NULL,
  prior = list(),
  prob,
  alpha,
  sens.control = list(),
  sens.bayes.control = list(),
  crt.bayes.control = list(),
  plot_3d = c("lattice", "rgl"),
  plot_sens = TRUE,
  npar = NULL,
  calculate_criterion = TRUE,
  silent = FALSE
)
```

Arguments

`formula` A linear or nonlinear model `formula`. A symbolic description of the model consists of predictors and the unknown model parameters. Will be coerced to a `formula` if necessary.

`predvars` A vector of characters. Denotes the predictors in the `formula`.

`parvars` A vector of characters. Denotes the unknown parameters in the `formula`.

`family` A description of the response distribution and the link function to be used in the model. This can be a family function, a call to a family function or a character string naming the family. Every family function has a link argument allowing to specify the link function to be applied on the response variable. If not specified, default links are used. For details see `family`. By default, a linear gaussian model `gaussian()` is applied.

`x` A vector of candidate design (support) points. When is not set to `NULL` (default), the algorithm only finds the optimal weights for the candidate points in `x`. Should be set when the user has a finite number of candidate design points and the purpose is to find the optimal weight for each of them (when zero, they will be excluded from the design). For design points with more than one dimension, see 'Details' of `sensminimax`.

`w` Vector of the corresponding design weights for `x`.

`lx` Vector of lower bounds for the predictors. Should be in the same order as `predvars`.

`ux` Vector of upper bounds for the predictors. Should be in the same order as `predvars`.

`fimfunc` A function. Returns the FIM as a `matrix`. Required when `formula` is missing. See 'Details' of `minimax`.

`prior` An object of class `cprior`. User can also use one of the functions `uniform`, `normal`, `skewnormal` or `student` to create the prior. See 'Details' of `bayes`.

`prob` Either `formula` or a `function`. When function, its argument are `x` and `param`, and they are the same as the arguments in `fimfunc`. `prob` as a function takes the design points and vector of parameters and returns the probability of success at each design points. See 'Examples'.

`alpha` A value between 0 and 1. Compound or combined DP-criterion is the product of the efficiencies of a design with respect to D- and average P- optimality, weighted by `alpha`.

`sens.control` Control Parameters for Calculating the ELB. For details, see `sens.control`.

`sens.bayes.control` A list. Control parameters to verify the general equivalence theorem. For details, see `sens.bayes.control`.

`crt.bayes.control` A list. Control parameters to approximate the integral in the Bayesian criterion at a given design over the parameter space. For details, see `crt.bayes.control`.

`plot_3d` Which package should be used to plot the sensitivity (derivative) function for two-dimensional design space. Defaults to `"lattice"`.

`plot_sens`	Plot the sensitivity (derivative) function? Defaults to `TRUE`.
`npar`	Number of model parameters. Used when `fimfunc` is given instead of `formula` to specify the number of model parameters. If not specified correctly, the sensitivity (derivative) plot may be shifted below the y-axis. When `NULL` (default), it will be set to `length(parvars)` or `prior$npar` when `missing(formula)`.
`calculate_criterion`	Calculate the optimality criterion? See 'Details' of `sensminimax`.
`silent`	Do not print anything? Defaults to `FALSE`.

Details

Depending on the complexity of the problem at hand, sometimes, the CPU time can be considerably reduced by choosing a set of less conservative values for the tuning parameters `tol` and `maxEval` in the function `sens.bayes.control` when its `method` component is equal to `"cubature"`. Similarly, this applies when `sens.bayes.control$method = "quadrature"`. In general, if the CPU time matters, the user should find an appropriate speed-accuracy trade-off for her/his own problem. See 'Examples' for more details.

Note

The default values of the tuning parameters in `sens.bayes.control` are set in a way that having accurate plots for the sensitivity (derivative) function and calculating the ELB to a high precision to be the primary goals, although the process may take too long. The user should choose a set of less conservative values via the argument `sens.bayes.control` when the CPU-time is too long or matters.

See Also

`bayescomp`

Examples

```
###########################################
# Verifing the DP-optimality of a design
# The logistic model with two predictors
###########################################

# The design points and corresponding weights are as follows:
# Point1     Point2     Point3     Point4     Point5     Point6     Point7
# 0.07410   -0.31953   -1.00000    1.00000   -1.00000    1.00000    0.30193
# -1.00000   1.00000   -1.00000    1.00000   -0.08251   -1.00000    1.00000
# Weight1    Weight2    Weight3    Weight4    Weight5    Weight6    Weight7
# 0.020      0.275      0.224      0.131      0.092      0.156      0.103

# It should be given to the function as two seperate vectors:
x1 <- c(0.07409639, -0.3195265, -1, 1, -1, 1, 0.3019317, -1, 1, -1, 1, -0.08251169, -1, 1)
w1 <- c(0.01992863, 0.2745394, 0.2236575, 0.1312331, 0.09161503, 0.1561454, 0.1028811)


p <- c(1, -2, 1, -1)
```

```
## Not run:
sensbayescomp(formula = ~exp(b0+b1*x1+b2*x2+b3*x1*x2)/(1+exp(b0+b1*x1+b2*x2+b3*x1*x2)),
              predvars = c("x1", "x2"),
              parvars = c("b0", "b1", "b2", "b3"),
              family = binomial(),
              x = x1, w = w1,
              lx = c(-1, -1), ux = c(1, 1),
              prior = uniform(p -1.5, p + 1.5),
              prob = ~1-1/(1+exp(b0 + b1 * x1 + b2 * x2 + b3 * x1 * x2)),
              alpha = .5, plot_3d = "rgl",
              sens.bayes.control = list(cubature = list(tol = 1e-3, maxEval = 1000)))

## End(Not run)
```

`senslocally` *Verifying Optimality of The Locally D-optimal Designs*

Description

It plots the sensitivity (derivative) function of the locally D-optimal criterion at a given approximate (continuous) design and also calculates its efficiency lower bound (ELB) with respect to the optimality criterion. For an approximate (continuous) design, when the design space is one or two-dimensional, the user can visually verify the optimality of the design by observing the sensitivity plot. Furthermore, the proximity of the design to the optimal design can be measured by the ELB without knowing the latter. See, for more details, Masoudi et al. (2017).

Usage

```
senslocally(
  formula,
  predvars,
  parvars,
  family = gaussian(),
  x,
  w,
  lx,
  ux,
  inipars,
  fimfunc = NULL,
  sens.control = list(),
  calculate_criterion = TRUE,
  plot_3d = c("lattice", "rgl"),
```

```
  plot_sens = TRUE,
  npar = length(inipars),
  silent = FALSE,
  crtfunc = NULL,
  sensfunc = NULL
)
```

Arguments

formula
: A linear or nonlinear model formula. A symbolic description of the model consists of predictors and the unknown model parameters. Will be coerced to a formula if necessary.

predvars
: A vector of characters. Denotes the predictors in the formula.

parvars
: A vector of characters. Denotes the unknown parameters in the formula.

family
: A description of the response distribution and the link function to be used in the model. This can be a family function, a call to a family function or a character string naming the family. Every family function has a link argument allowing to specify the link function to be applied on the response variable. If not specified, default links are used. For details see family. By default, a linear gaussian model `gaussian()` is applied.

x
: Vector of the design (support) points. See 'Details' of sensminimax for models with more than one predictors.

w
: Vector of the corresponding design weights for `x`.

lx
: Vector of lower bounds for the predictors. Should be in the same order as `predvars`.

ux
: Vector of upper bounds for the predictors. Should be in the same order as `predvars`.

inipars
: A vector of initial estimates for the unknown parameters. It must match `parvars` or the argument `param` of the function `fimfunc`, when provided.

fimfunc
: A function. Returns the FIM as a `matrix`. Required when `formula` is missing. See 'Details' of minimax.

sens.control
: Control Parameters for Calculating the ELB. For details, see sens.control.

calculate_criterion
: Calculate the optimality criterion? See 'Details' of sensminimax.

plot_3d
: Which package should be used to plot the sensitivity (derivative) function for models with two predictors. Either `"rgl"` or `"lattice"` (default).

plot_sens
: Plot the sensitivity (derivative) function? Defaults to `TRUE`.

npar
: Number of model parameters. Used when `fimfunc` is given instead of `formula` to specify the number of model parameters. If not given, the sensitivity plot may be shifted below the y-axis. When `NULL`, it is set to `length(inipars)`.

silent
: Do not print anything? Defaults to `FALSE`.

crtfunc
: (Optional) a function that specifies an arbitrary criterion. It must have especial arguments and output. See 'Details' of minimax.

sensfunc
: (Optional) a function that specifies the sensitivity function for `crtfunc`. See 'Details' of minimax.

Details

Let θ_0 denotes the vector of initial estimates for the unknown parameters. A design ξ^* is locally D-optimal among all designs on χ if and only if the following inequality holds for all $\boldsymbol{x} \in \chi$

$$c(\boldsymbol{x}, \xi^*, \theta_0) = trM^{-1}(\xi^*, \theta_0)I(\boldsymbol{x}, \theta_0) - p \leq 0,$$

with equality at all support points of ξ^*. Here, p is the number of model parameters. $c(\boldsymbol{x}, \xi^*, \theta_0)$ is called **sensitivity** or **derivative** function.

ELB is a measure of proximity of a design to the optimal design without knowing the latter. Given a design, let ϵ be the global maximum of the sensitivity (derivative) function over $x \in \chi$. ELB is given by

$$ELB = p/(p + \epsilon),$$

where p is the number of model parameters. Obviously, calculating ELB requires finding ϵ and another optimization problem to be solved. The tuning parameters of this optimization can be regulated via the argument `sens.minimax.control`. See, for more details, Masoudi et al. (2017).

Value

an object of class `sensminimax` that is a list with the following elements:

`type` Argument `type` that is required for print methods.

`optima` A `matrix` that stores all the local optima over the parameter space. The cost (criterion) values are stored in a column named `Criterion_Value`. The last column (`Answering_Set`) shows if the optimum belongs to the answering set (1) or not (0). See 'Details' of `sens.minimax.control`. Only applicable for minimax or standardized maximin designs.

`mu` Probability measure on the answering set. Corresponds to the rows of `optima` for which the associated row in column `Answering_Set` is equal to 1. Only applicable for minimax or standardized maximin designs.

`max_deriv` Global maximum of the sensitivity (derivative) function (ϵ in 'Details').

`ELB` D-efficiency lower bound. Can not be larger than 1. If negative, see 'Note' in `sensminimax` or `sens.minimax.control`.

`merge_tol` Merging tolerance to create the answering set from the set of all local optima. See 'Details' in `sens.minimax.control`. Only applicable for minimax or standardized maximin designs.

`crtval` Criterion value. Compare it with the column `Crtiterion_Value` in `optima` for minimax and standardized maximin designs.

`time` Used CPU time (rough approximation).

Note

Theoretically, ELB can not be larger than 1. But if so, it may have one of the following reasons:

- `max_deriv` is not a GLOBAL maximum. Please increase the value of the parameter `maxeval` in `sens.minimax.control` to find the global maximum.
- The sensitivity function is shifted below the y-axis because the number of model parameters has not been specified correctly (less value given). Please specify the correct number of model parameters via the argument `npar`.

References

Masoudi E, Holling H, Wong W.K. (2017). Application of Imperialist Competitive Algorithm to Find Minimax and Standardized Maximin Optimal Designs. Computational Statistics and Data Analysis, 113, 330-345.

Examples

```
############################
# Exponential growth model
############################
# Verifying optimailty of a locally D-optimal design
senslocally(formula = ~a + exp(-b*x),
            predvars = "x", parvars = c("a", "b"),
            x = c(.1, 1), w = c(.5, .5),
            lx = 0, ux = 1, inipars = c(1, 10))


##############################
# A model with two predictors
##############################
x0 <- c(30, 3.861406, 30, 4.600633, 0, 0, 5.111376, 4.168798)
w0 <- rep(.25, 4)
senslocally(formula =  ~ V*S/(Km * (1 + I/Kic)+ S * (1 + I/Kiu)),
            predvars = c("S", "I"),
            parvars = c("V", "Km", "Kic", "Kiu"),
            x = x0, w = w0,
            lx = c(0, 0), ux = c(30, 60),
            inipars = c(1.5, 5.2, 3.4, 5.6))
## Not run:
  # using package rgl for 3d plot:
  res<- senslocally(formula =  ~ V*S/(Km * (1 + I/Kic)+ S * (1 + I/Kiu)),
                    predvars = c("S", "I"),
                    parvars = c("V", "Km", "Kic", "Kiu"),
                    x = x0, w = w0,
                    lx = c(0, 0), ux = c(30, 60),
                    inipars = c(1.5, 5.2, 3.4, 5.6),
                    plot_3d = "rgl")


## End(Not run)

###################################
# user-defined optimality criterion
##################################
# When the model is defined by the formula interface
# Checking the A-optimality  for the 2PL model.
# the criterion function must have argument x, w fimfunc and the parameters defined in 'parvars'.
# use 'fimfunc' as a function of the design points x,  design weights w and
#  the 'parvars' parameters whenever needed.
Aopt <-function(x, w, a, b, fimfunc){
  sum(diag(solve(fimfunc(x = x, w = w, a = a, b = b))))
```

```
}
## the sensitivtiy function
# xi_x is a design that put all its mass on x in the definition of the sensitivity function
# x is a vector of design points
Aopt_sens <- function(xi_x, x, w, a, b, fimfunc){
  fim <- fimfunc(x = x, w = w, a = a, b = b)
  M_inv <- solve(fim)
  M_x <- fimfunc(x = xi_x, w = 1, a  = a, b = b)
  sum(diag(M_inv %*% M_x %*%  M_inv)) - sum(diag(M_inv))
}

senslocally(formula = ~1/(1 + exp(-b * (x-a))), predvars = "x",
            parvars = c("a", "b"), family = "binomial",
            inipars = c(0, 1.5),
            crtfunc = Aopt,
            lx = -2, ux = 2,
            sensfunc = Aopt_sens,
            x = c(-1,  1), w = c(.5, .5))
# not optimal
```

senslocallycomp *Verifying Optimality of The Locally DP-optimal Designs*

Description

This function plot the sensitivity (derivative) function given an approximate (continuous) design and calculate the efficiency lower bound (ELB) for locally DP-optimal designs. Let $\boldsymbol{x}$ belongs to χ that denotes the design space. Based on the general equivalence theorem, generally, a design ξ^* is optimal if and only if the value of its sensitivity (derivative) function be non-positive for all $\boldsymbol{x}$ in χ and it only reaches zero when $\boldsymbol{x}$ belong to the support of ξ^* (be equal to one of the design point). Therefore, the user can look at the sensitivity plot and the ELB to decide whether the design is optimal or close enough to the true optimal design.

Usage

```
senslocallycomp(
  formula,
  predvars,
  parvars,
  alpha,
  prob,
  family = gaussian(),
  x,
  w,
  lx,
  ux,
  inipars,
  fimfunc = NULL,
```

```
  sens.control = list(),
  calculate_criterion = TRUE,
  plot_3d = c("lattice", "rgl"),
  plot_sens = TRUE,
  npar = length(inipars),
  silent = FALSE
)
```

Arguments

- `formula` A linear or nonlinear model `formula`. A symbolic description of the model consists of predictors and the unknown model parameters. Will be coerced to a `formula` if necessary.
- `predvars` A vector of characters. Denotes the predictors in the `formula`.
- `parvars` A vector of characters. Denotes the unknown parameters in the `formula`.
- `alpha` A value between 0 and 1. Compound or combined DP-criterion is the product of the efficiencies of a design with respect to D- and average P- optimality, weighted by `alpha`.
- `prob` Either `formula` or a `function`. When function, its argument are `x` and `param`, and they are the same as the arguments in `fimfunc`. `prob` as a function takes the design points and vector of parameters and returns the probability of success at each design points. See 'Examples'.
- `family` A description of the response distribution and the link function to be used in the model. This can be a family function, a call to a family function or a character string naming the family. Every family function has a link argument allowing to specify the link function to be applied on the response variable. If not specified, default links are used. For details see `family`. By default, a linear gaussian model `gaussian()` is applied.
- `x` Vector of the design (support) points. See 'Details' of `sensminimax` for models with more than one predictors.
- `w` Vector of the corresponding design weights for `x`.
- `lx` Vector of lower bounds for the predictors. Should be in the same order as `predvars`.
- `ux` Vector of upper bounds for the predictors. Should be in the same order as `predvars`.
- `inipars` Vector of initial estimates for the unknown parameters. It must match `parvars` or argument `param` of the function provided in `fimfunc`.
- `fimfunc` A function. Returns the FIM as a `matrix`. Required when `formula` is missing. See 'Details' of `minimax`.
- `sens.control` Control Parameters for Calculating the ELB. For details, see `sens.control`.
- `calculate_criterion` Calculate the optimality criterion? See 'Details' of `sensminimax`.
- `plot_3d` Which package should be used to plot the sensitivity (derivative) function for models with two predictors. Either `"rgl"` or `"lattice"` (default).

plot_sens
: Plot the sensitivity (derivative) function? Defaults to TRUE.

npar
: Number of model parameters. Used when fimfunc is given instead of formula to specify the number of model parameters. If not given, the sensitivity plot may be shifted below the y-axis. When NULL, it is set to length(inipars).

silent
: Do not print anything? Defaults to FALSE.

Value

an object of class sensminimax that is a list with the following elements:

type Argument type that is required for print methods.

optima A matrix that stores all the local optima over the parameter space. The cost (criterion) values are stored in a column named Criterion_Value. The last column (Answering_Set) shows if the optimum belongs to the answering set (1) or not (0). See 'Details' of sens.minimax.control. Only applicable for minimax or standardized maximin designs.

mu Probability measure on the answering set. Corresponds to the rows of optima for which the associated row in column Answering_Set is equal to 1. Only applicable for minimax or standardized maximin designs.

max_deriv Global maximum of the sensitivity (derivative) function (ϵ in 'Details').

ELB D-efficiency lower bound. Can not be larger than 1. If negative, see 'Note' in sensminimax or sens.minimax.control.

merge_tol Merging tolerance to create the answering set from the set of all local optima. See 'Details' in sens.minimax.control. Only applicable for minimax or standardized maximin designs.

crtval Criterion value. Compare it with the column Crtiterion_Value in optima for minimax and standardized maximin designs.

time Used CPU time (rough approximation).

References

McGree, J. M., Eccleston, J. A., and Duffull, S. B. (2008). Compound optimal design criteria for nonlinear models. Journal of Biopharmaceutical Statistics, 18(4), 646-661.

Examples

```
p <- c(1, -2, 1, -1)
prior4.4 <- uniform(p -1.5, p + 1.5)
formula4.4 <- ~exp(b0+b1*x1+b2*x2+b3*x1*x2)/(1+exp(b0+b1*x1+b2*x2+b3*x1*x2))
prob4.4 <- ~1-1/(1+exp(b0 + b1 * x1 + b2 * x2 + b3 * x1 * x2))
predvars4.4 <-  c("x1", "x2")
parvars4.4 <- c("b0", "b1", "b2", "b3")
lb <- c(-1, -1)
ub <- c(1, 1)

## That is the optimal design when alpha = .25, see ?locallycomp on how to find it
xopt <- c(-1, -0.389, 1, 0.802, -1, 1, -1, 1)
wopt <- c(0.198, 0.618, 0.084, 0.1)
```

```
# We want to verfiy the optimality of the optimal design by the general equivalence theorem.

senslocallycomp(formula = formula4.4, predvars = predvars4.4, parvars = parvars4.4,
                family = binomial(), prob = prob4.4, lx = lb, ux = ub,
                alpha = .25, inipars = p, x = xopt, w = wopt)

## Not run:
# is this design also optimal when alpha = .3

senslocallycomp(formula = formula4.4, predvars = predvars4.4, parvars = parvars4.4,
                family = binomial(), prob = prob4.4, lx = lb, ux = ub,
                alpha = .3, inipars = p, x = xopt, w = wopt)

# when alpha = .3
senslocallycomp(formula = formula4.4, predvars = predvars4.4, parvars = parvars4.4,
                family = binomial(), prob = prob4.4, lx = lb, ux = ub,
                alpha = .5, inipars = p, x = xopt, w = wopt)
# when alpha = .8
senslocallycomp(formula = formula4.4, predvars = predvars4.4, parvars = parvars4.4,
                family = binomial(), prob = prob4.4, lx = lb, ux = ub,
                alpha = .8, inipars = p, x = xopt, w = wopt)


# when alpha = .9
senslocallycomp(formula = formula4.4, predvars = predvars4.4, parvars = parvars4.4,
                family = binomial(), prob = prob4.4, lx = lb, ux = ub,
                alpha = .9, inipars = p, x = xopt, w = wopt)

## As can be seen, the design looses efficiency as alpha increases.

## End(Not run)
```

---

sensminimax *Verifying Optimality of The Minimax and Standardized maximin D-optimal Designs*

Description

It plots the sensitivity (derivative) function of the minimax or standardized maximin D-optimal criterion at a given approximate (continuous) design and also calculates its efficiency lower bound (ELB) with respect to the optimality criterion. For an approximate (continuous) design, when the design space is one or two-dimensional, the user can visually verify the optimality of the design by observing the sensitivity plot. Furthermore, the proximity of the design to the optimal design can be measured by the ELB without knowing the latter. See, for more details, Masoudi et al. (2017).

Usage

```
sensminimax(
  formula,
```

```
  predvars,
  parvars,
  family = gaussian(),
  x,
  w,
  lx,
  ux,
  lp,
  up,
  fimfunc = NULL,
  standardized = FALSE,
  localdes = NULL,
  sens.control = list(),
  sens.minimax.control = list(),
  calculate_criterion = TRUE,
  crt.minimax.control = list(),
  plot_3d = c("lattice", "rgl"),
  plot_sens = TRUE,
  npar = length(lp),
  silent = FALSE,
  crtfunc = NULL,
  sensfunc = NULL
)
```

Arguments

`formula`	A linear or nonlinear model `formula`. A symbolic description of the model consists of predictors and the unknown model parameters. Will be coerced to a `formula` if necessary.
`predvars`	A vector of characters. Denotes the predictors in the `formula`.
`parvars`	A vector of characters. Denotes the unknown parameters in the `formula`.
`family`	A description of the response distribution and the link function to be used in the model. This can be a family function, a call to a family function or a character string naming the family. Every family function has a link argument allowing to specify the link function to be applied on the response variable. If not specified, default links are used. For details see `family`. By default, a linear gaussian model `gaussian()` is applied.
`x`	Vector of the design (support) points. See 'Details' of `sensminimax` for models with more than one predictors.
`w`	Vector of the corresponding design weights for `x`.
`lx`	Vector of lower bounds for the predictors. Should be in the same order as `predvars`.
`ux`	Vector of upper bounds for the predictors. Should be in the same order as `predvars`.
`lp`	Vector of lower bounds for the model parameters. Should be in the same order as `parvars` or `param` in the argument `fimfunc`.

up
: Vector of upper bounds for the model parameters. Should be in the same order as parvars or param in the argument fimfunc. When a parameter is known (has a fixed value), its associated lower and upper bound values in lp and up must be set equal.

fimfunc
: A function. Returns the FIM as a matrix. Required when formula is missing. See 'Details' of minimax.

standardized
: Maximin standardized design? When standardized = TRUE, the argument localdes must be given. Defaults to FALSE. See 'Details' of minimax.

localdes
: A function that takes the parameter values as inputs and returns the design points and weights of the locally optimal design. Required when standardized = "TRUE". See 'Details' of minimax.

sens.control
: Control Parameters for Calculating the ELB. For details, see sens.control.

sens.minimax.control
: Control parameters to construct the answering set required for verify the general equivalence theorem and calculating the ELB. For details, see the function sens.minimax.control.

calculate_criterion
: Calculate the optimality criterion? See 'Details' of sensminimax.

crt.minimax.control
: Control parameters to calculate the value of the minimax or standardized maximin optimality criterion over the continuous parameter space. Only applicable when calculate_criterion = TRUE. For more details, see crt.minimax.control.

plot_3d
: Which package should be used to plot the sensitivity (derivative) function for models with two predictors. Either "rgl" or "lattice" (default).

plot_sens
: Plot the sensitivity (derivative) function? Defaults to TRUE.

npar
: Number of model parameters. Used when fimfunc is given instead of formula to specify the number of model parameters. If not specified truly, the sensitivity (derivative) plot may be shifted below the y-axis. When NULL (default), it is set to length(lp).

silent
: Do not print anything? Defaults to FALSE.

crtfunc
: (Optional) a function that specifies an arbitrary criterion. It must have especial arguments and output. See 'Details' of minimax.

sensfunc
: (Optional) a function that specifies the sensitivity function for crtfunc. See 'Details' of minimax.

Details

Let the unknown parameters belong to Θ. A design ξ^* is minimax D-optimal among all designs on χ if and only if there exists a probability measure μ^* on

$$A(\xi^*) = \left\{ \nu \in \Theta \mid -log|M(\xi^*, \nu)| = \max_{\theta \in \Theta} -log|M(\xi^*, \theta)| \right\},$$

such that the following inequality holds for all $\boldsymbol{x} \in \chi$

$$c(\boldsymbol{x}, \mu^*, \xi^*) = \int_{A(\xi^*)} trM^{-1}(\xi^*, \nu)I(\boldsymbol{x}, \nu)\mu^* d(\nu) - p \leq 0,$$

with equality at all support points of ξ^*. Here, p is the number of model parameters. $c(\boldsymbol{x}, \mu^*, \xi^*)$ is called **sensitivity** or **derivative** function. The set $A(\xi^*)$ is sometimes called **answering set** of ξ^* and the measure μ^* is a sub-gradient of the non-differentiable criterion evaluated at $M(\xi^*, \nu)$.
For the standardized maximin D-optimal designs, the answering set $N(\xi^*)$ is

$$N(\xi^*) = \left\{ \boldsymbol{\nu} \in \Theta \mid \text{eff}_D(\xi^*, \boldsymbol{\nu}) = \min_{\boldsymbol{\theta} \in \Theta} \text{eff}_D(\xi^*, \boldsymbol{\theta}) \right\}.$$

where $\text{eff}_D(\xi, \boldsymbol{\theta}) = (\frac{|M(\xi,\boldsymbol{\theta})|}{|M(\xi_{\boldsymbol{\theta}},\boldsymbol{\theta})|})^{\frac{1}{p}}$ and ξ_θ is the locally D-optimal design with respect to θ. See 'Details' of `sens.minimax.control` on how we find the answering set.

The argument `x` is the vector of design points. For design points with more than one dimension (the models with more than one predictors), it is a concatenation of the design points, but **dimension-wise**. For example, let the model has three predictors (I, S, Z). Then, a two-point optimal design has the following points: $\{\text{point1} = (I_1, S_1, Z_1), \text{point2} = (I_2, S_2, Z_2)\}$. Then, the argument `x` is equal to `x = c(I1, I2, S1, S2, Z1, Z2)`.

ELB is a measure of proximity of a design to the optimal design without knowing the latter. Given a design, let ϵ be the global maximum of the sensitivity (derivative) function with respect x where $x \in \chi$. ELB is given by

$$ELB = p/(p + \epsilon),$$

where p is the number of model parameters. Obviously, calculating ELB requires finding ϵ and another optimization problem to be solved. The tuning parameters of this optimization can be regulated via the argument `sens.minimax.control`. See, for more details, Masoudi et al. (2017).

The criterion value for the minimax D-optimal design is (global maximum over Θ)

$$\max_{\theta \in \Theta} -\log |M(\xi, \theta)|;$$

for the standardized maximin D-optimal design is (global minimum over Θ)

$$\inf_{\theta \in \Theta} \left[\left(\frac{|M(\xi, \theta)|}{|M(\xi_\theta, \theta)|} \right)^{\frac{1}{p}} \right].$$

This function confirms the optimality assuming only a continuous parameter space Θ.

Value

an object of class `sensminimax` that is a list with the following elements:

`type` Argument `type` that is required for print methods.

`optima` A `matrix` that stores all the local optima over the parameter space. The cost (criterion) values are stored in a column named `Criterion_Value`. The last column (`Answering_Set`) shows if the optimum belongs to the answering set (1) or not (0). See 'Details' of `sens.minimax.control`. Only applicable for minimax or standardized maximin designs.

`mu` Probability measure on the answering set. Corresponds to the rows of `optima` for which the associated row in column `Answering_Set` is equal to 1. Only applicable for minimax or standardized maximin designs.

`max_deriv` Global maximum of the sensitivity (derivative) function (ϵ in 'Details').

`ELB` D-efficiency lower bound. Can not be larger than 1. If negative, see 'Note' in `sensminimax` or `sens.minimax.control`.

`merge_tol` Merging tolerance to create the answering set from the set of all local optima. See 'Details' in `sens.minimax.control`. Only applicable for minimax or standardized maximin designs.

`crtval` Criterion value. Compare it with the column `Crtiterion_Value` in `optima` for minimax and standardized maximin designs.

`time` Used CPU time (rough approximation).

Note

Theoretically, ELB can not be larger than 1. But if so, it may have one of the following reasons:

- `max_deriv` is not a GLOBAL maximum. Please increase the value of the parameter `maxeval` in `sens.minimax.control` to find the global maximum.
- The sensitivity function is shifted below the y-axis because the number of model parameters has not been specified correctly (less value given). Please specify the correct number of model parameters via argument `npar`.

Please increase the value of the parameter `n_seg` in `sens.minimax.control` for models with larger number of parameters or large parameter space to find the true answering set for minimax and standardized maximin designs. See `sens.minimax.control` for more details.

References

Masoudi E, Holling H, Wong W.K. (2017). Application of Imperialist Competitive Algorithm to Find Minimax and Standardized Maximin Optimal Designs. Computational Statistics and Data Analysis, 113, 330-345.

Examples

```
##########################
# Power logistic model
##########################
# verifying the minimax D-optimality of a design with points x0 and weights w0
x0 <- c(-4.5515, 0.2130, 2.8075)
w0 <- c(0.4100, 0.3723, 0.2177)
# Power logistic model when s = .2
sensminimax(formula =  ~ (1/(1 + exp(-b * (x-a))))^.2,
            predvars = "x",
            parvars = c("a", "b"),
            family = binomial(),
            x = x0, w = w0,
            lx = -5, ux = 5,
            lp = c(0, 1), up = c(3, 1.5))

##############################
# A model with two predictors
##############################
```

```
# Verifying the minimax D-optimality of a design for a model with two predictors
# The model is the mixed inhibition model.
# X0 is the vector of four design points that are:
# (3.4614, 0) (4.2801, 3.1426) (30, 0) (30, 4.0373)
x0 <- c(3.4614, 4.2801, 30, 30, 0, 3.1426, 0, 4.0373)
w0 <- rep(1/4, 4)
sensminimax(formula =  ~ V*S/(Km * (1 + I/Kic)+ S * (1 + I/Kiu)),
            predvars = c("S", "I"),
            parvars = c("V", "Km", "Kic", "Kiu"),
            family = "gaussian",
            x = x0, w = w0,
            lx = c(0, 0), ux = c(30, 60),
            lp = c(1, 4, 2, 4), up = c(1, 5, 3, 5))

##########################################
# Standardized maximin D-optimal designs
##########################################
# Verifying the standardized maximin D-optimality of a design for
# the loglinear model
# First we should define the function for 'localdes' argument
# The function LDOD takes the parameters and returns the points and
# weights of the locally D-optimal design
LDOD <- function(theta0, theta1, theta2){
  ## param is the vector of theta = (theta0, theta1, theta2)
  lx <- 0 # lower bound of the design space
  ux <- 150 # upper bound of the design space
  param <- c()
  param[1] <- theta0
  param[2] <- theta1
  param[3] <- theta2
  xstar <- (ux+param[3]) * (lx + param[3]) *
    (log(ux + param[3]) - log(lx + param[3]))/(ux - lx) - param[3]
  return(list(x = c(lx, xstar, ux) , w = rep(1/3, 3)))
}
x0 <- c(0, 4.2494, 17.0324, 149.9090)
w0 <- c(0.3204, 0.1207, 0.2293, 0.3296)
## Not run:
  sensminimax(formula = ~theta0 + theta1* log(x + theta2),
              predvars = c("x"),
              parvars = c("theta0", "theta1", "theta2"),
              x = x0, w = w0,
              lx = 0, ux = 150,
              lp = c(2, 2, 1), up = c(2, 2, 15),
              localdes = LDOD,
              standardized = TRUE,
              sens.minimax.control = list(n_seg = 10))

## End(Not run)
################################################################
# Not necessary!
# The rest of the examples here are only for professional uses.
################################################################
# Imagine you have written your own FIM, say in Rcpp that is faster than
```

```
# the FIM created by the formula interface here.

##########################
# Power logistic model
##########################
# For example, th cpp FIM function for the power logistic model is named:
FIM_power_logistic
args(FIM_power_logistic)
# The arguments do not match the standard of the argument 'fimfunc'
# in 'sensminimax'
# So we reparameterize it:
myfim1 <- function(x, w, param)
  FIM_power_logistic(x = x, w = w, param =param, s = .2)

args(myfim1)
## Not run:
  # Verify minimax D-optimality of a design
  sensminimax(fimfunc = myfim1,
              x = c(-4.5515, 0.2130, 2.8075),
              w = c(0.4100, 0.3723, 0.2177),
              lx = -5, ux = 5,
              lp = c(0, 1), up = c(3, 1.5))

## End(Not run)
##############################
# A model with two predictors
##############################
# An example of a  model with two-predictors: mixed inhibition model
# Fisher information matrix:
FIM_mixed_inhibition
args(FIM_mixed_inhibition)

# We should first reparameterize the FIM to match the standard of the
# argument 'fimfunc'
myfim2 <- function(x, w, param){
  npoint <- length(x)/2
  S <- x[1:npoint]
  I <- x[(npoint+1):(npoint*2)]
  out <- FIM_mixed_inhibition(S = S, I = I, w = w, param = param)
  return(out)
}
args(myfim2)
## Not run:
  # Verifyng minimax D-optimality of a design
  sensminimax(fimfunc = myfim2,
              x = c(3.4614, 4.2801, 30, 30, 0, 3.1426, 0, 4.0373),
              w = rep(1/4, 4),
              lx = c(0, 0), ux = c(30, 60),
              lp = c(1, 4, 2, 4), up = c(1, 5, 3, 5))

## End(Not run)

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

```
# Standardized maximin D-optimal designs
#########################################
# An example of a user-written FIM function:
help(FIM_loglin)
# An example of verfying standardaized maximin D-optimality for a design
# Look how we re-define the function LDOD above
LDOD2 <- function(param){
  ## param is the vector of theta = (theta0, theta1, theta2)
  lx <- 0 # lower bound of the design space
  ux <- 150 # upper bound of the design space
  xstar <- (ux + param[3]) * (lx + param[3]) *
    (log(ux + param[3]) - log(lx + param[3]))/(ux - lx) - param[3]
  return(list(x = c(lx, xstar, ux) , w = rep(1/3, 3)))
}

args(LDOD2)

sensminimax(fimfunc = FIM_loglin,
            x = x0,
            w = w0,
            lx = 0, ux = 150,
            lp = c(2, 2, 1), up = c(2, 2, 15),
            localdes = LDOD2,
            standardized = TRUE)



###################################
# user-defined optimality criterion
##################################
# When the model is defined by the formula interface
# Checking the A-optimality  for the 2PL model.
# the criterion function must have argument x, w fimfunc and the parameters defined in 'parvars'.
# use 'fimfunc' as a function of the design points x,  design weights w and
#  the 'parvars' parameters whenever needed.
Aopt <-function(x, w, a, b, fimfunc){
  sum(diag(solve(fimfunc(x = x, w = w, a = a, b = b))))
}
## the sensitivtiy function
# xi_x is a design that put all its mass on x in the definition of the sensitivity function
# x is a vector of design points
Aopt_sens <- function(xi_x, x, w, a, b, fimfunc){
  fim <- fimfunc(x = x, w = w, a = a, b = b)
  M_inv <- solve(fim)
  M_x <- fimfunc(x = xi_x, w = 1, a  = a, b = b)
  sum(diag(M_inv %*% M_x %*%  M_inv)) - sum(diag(M_inv))
}

sensminimax(formula = ~1/(1 + exp(-b * (x-a))), predvars = "x",
            parvars = c("a", "b"), family = "binomial",
            lp = c(-2, 1), up = c(2, 1.5),
            crtfunc = Aopt,
            lx = -2, ux = 2,
```

```
sensfunc = Aopt_sens,
x = c(-2, .0033, 2), w = c(.274, .452, .274))
```

---

sensmultiple — *Verifying Optimality of The Multiple Objective Designs for The 4-Parameter Hill Model*

Description

This function uses general equivalence theorem to verify the optimality of a multiple objective optimal design found for the 4-Parameter Hill model and the 4-parameter logistic model. For more details, See Hyun and Wong (2015).

Usage

```
sensmultiple(
  dose,
  w,
  minDose,
  maxDose,
  inipars,
  lambda,
  delta,
  Hill_par = TRUE,
  sens.control = list(),
  calculate_criterion = TRUE,
  plot_sens = TRUE,
  tol = sqrt(.Machine$double.xmin),
  silent = FALSE
)
```

Arguments

`dose`	A vector of design points. It is either dose values or logarithm of dose values when `Hill_par = TRUE`.
`w`	A vector of design weights.
`minDose`	Minimum dose D. For the 4-parameter logistic model, i.e. when `Hill_par = FALSE`, it is the minimum of $log(D)$.
`maxDose`	Maximum dose D. For the 4-parameter logistic model, i.e. when `Hill_par = FALSE`, it is the maximum of $log(D)$.
`inipars`	A vector of initial estimates for the vector of parameters (a, b, c, d). For the 4-parameter logistic model, i.e. when `Hill_par = FALSE`, it is a vector of initial estimates for $(\theta_1, \theta_2, \theta_3, \theta_4)$.
`lambda`	A vector of relative importance of each of the three criteria, i.e. $\lambda = (\lambda_1, \lambda_2, \lambda_3)$. Here $0 < \lambda_i < 1$ and s $\sum \lambda_i = 1$.

`delta` Predetermined meaningful value of the minimum effective dose MED. When $\delta < 0$, then $\theta_2 > 0$ or when $\delta > 0$, then $\theta_2 < 0$.

`Hill_par` Hill model parameterization? Defaults to `TRUE`.

`sens.control` Control Parameters for Calculating the ELB. For details, see `sens.control`.

`calculate_criterion` Calculate the criterion? Defaults to `TRUE`.

`plot_sens` Plot the sensitivity (derivative) function? Defaults to `TRUE`.

`tol` Tolerance for finding the general inverse of the Fisher information matrix. Defaults to `.Machine$double.xmin`.

`silent` Do not print anything? Defaults to `FALSE`.

Details

ELB is a measure of proximity of a design to the optimal design without knowing the latter. Given a design, let ϵ be the global maximum of the sensitivity (derivative) function over $x \in \chi$. ELB is given by

$$ELB = p/(p + \epsilon),$$

where p is the number of model parameters. Obviously, calculating ELB requires finding ϵ and another optimization problem to be solved. The tuning parameters of this optimization can be regulated via the argument `sens.minimax.control`. See, for more details, Masoudi et al. (2017).

Value

an object of class `sensminimax` that is a list with the following elements:

`type` Argument `type` that is required for print methods.

`optima` A `matrix` that stores all the local optima over the parameter space. The cost (criterion) values are stored in a column named `Criterion_Value`. The last column (`Answering_Set`) shows if the optimum belongs to the answering set (1) or not (0). See 'Details' of `sens.minimax.control`. Only applicable for minimax or standardized maximin designs.

`mu` Probability measure on the answering set. Corresponds to the rows of `optima` for which the associated row in column `Answering_Set` is equal to 1. Only applicable for minimax or standardized maximin designs.

`max_deriv` Global maximum of the sensitivity (derivative) function (ϵ in 'Details').

`ELB` D-efficiency lower bound. Can not be larger than 1. If negative, see 'Note' in `sensminimax` or `sens.minimax.control`.

`merge_tol` Merging tolerance to create the answering set from the set of all local optima. See 'Details' in `sens.minimax.control`. Only applicable for minimax or standardized maximin designs.

`crtval` Criterion value. Compare it with the column `Crtiterion_Value` in `optima` for minimax and standardized maximin designs.

`time` Used CPU time (rough approximation).

Note

DO NOT use this function to verify c-optimal designs for estimating 'MED' or 'ED50' (verifying single objective optimal designs) because the results may be unstable. The reason is that for the c-optimal criterion the generalized inverse of the Fisher information matrix is not stable and depends on the tolerance value (`tol`).

Theoretically, ELB can not be larger than 1. But if so, it may have one of the following reasons:

- `max_deriv` is not a GLOBAL maximum. Please increase the value of the parameter `maxeval` in `sens.minimax.control` to find the global maximum.
- The sensitivity function is shifted below the y-axis because the number of model parameters has not been specified correctly (less value given). Please specify the correct number of model parameters via argument `npar`.

References

Hyun, S. W., and Wong, W. K. (2015). Multiple-Objective Optimal Designs for Studying the Dose Response Function and Interesting Dose Levels. The international journal of biostatistics, 11(2), 253-271.

See Also

`multiple`

Examples

```
##################################################################
# Verifying optimality of a design for the 4-parameter Hill model
##################################################################

## initial estiamtes for the parameters of the Hill model
a <- 0.008949  # ED50
b <- -1.79 # Hill constant
c <- 0.137 # lower limit
d <- 1.7 # upper limit
# D belongs to c(.001, 1000) ## dose in mg
## Hill parameters are c(a, b, c, d)
# dose, minDose and maxDose vector in mg scale

sensmultiple (dose = c(0.001, 0.009426562, 0.01973041, 999.9974),
              w =   c(0.4806477, 0.40815, 0.06114173, 0.05006055),
              minDose = .001, maxDose = 1000,
              Hill_par = TRUE,
              inipars = c(a, b, c, d),
              lambda = c(0.05, 0.05, .90),
              delta = -1)
```

sensrobust *Verifying Optimality of The Robust Designs*

Description

It plots the sensitivity (derivative) function of the robust criterion at a given approximate (continuous) design and also calculates its efficiency lower bound (ELB) with respect to the optimality criterion. For an approximate (continuous) design, when the design space is one or two-dimensional, the user can visually verify the optimality of the design by observing the sensitivity plot. Furthermore, the proximity of the design to the optimal design can be measured by the ELB without knowing the latter. See, for more details, Masoudi et al. (2017).

Usage

```
sensrobust(
  formula,
  predvars,
  parvars,
  family = gaussian(),
  x,
  w,
  lx,
  ux,
  prob,
  parset,
  fimfunc = NULL,
  sens.control = list(),
  calculate_criterion = TRUE,
  plot_3d = c("lattice", "rgl"),
  plot_sens = TRUE,
  npar = dim(parset)[2],
  silent = FALSE,
  crtfunc = NULL,
  sensfunc = NULL
)
```

Arguments

`formula`	A linear or nonlinear model `formula`. A symbolic description of the model consists of predictors and the unknown model parameters. Will be coerced to a `formula` if necessary.
`predvars`	A vector of characters. Denotes the predictors in the `formula`.
`parvars`	A vector of characters. Denotes the unknown parameters in the `formula`.
`family`	A description of the response distribution and the link function to be used in the model. This can be a family function, a call to a family function or a character string naming the family. Every family function has a link argument allowing to

specify the link function to be applied on the response variable. If not specified, default links are used. For details see `family`. By default, a linear gaussian model `gaussian()` is applied.

`x` Vector of the design (support) points. See 'Details' of `sensminimax` for models with more than one predictors.

`w` Vector of the corresponding design weights for `x`.

`lx` Vector of lower bounds for the predictors. Should be in the same order as `predvars`.

`ux` Vector of upper bounds for the predictors. Should be in the same order as `predvars`.

`prob` A vector of the probability measure π associated with each row of `parset`.

`parset` A matrix that provides the vector of initial estimates for the model parameters, i.e. support of π. Every row is one vector (`nrow(parset) == length(prob)`). See 'Details'.

`fimfunc` A function. Returns the FIM as a `matrix`. Required when `formula` is missing. See 'Details' of `minimax`.

`sens.control` Control Parameters for Calculating the ELB. For details, see `sens.control`.

`calculate_criterion`
Calculate the optimality criterion? See 'Details' of `sensminimax`.

`plot_3d` Which package should be used to plot the sensitivity (derivative) function for models with two predictors. Either `"rgl"` or `"lattice"` (default).

`plot_sens` Plot the sensitivity (derivative) function? Defaults to `TRUE`.

`npar` Number of model parameters. Used when `fimfunc` is given instead of `formula` to specify the number of model parameters. If not given, the sensitivity plot may be shifted below the y-axis. When `NULL`, it is set to `dim(parset)[2]`.

`silent` Do not print anything? Defaults to `FALSE`.

`crtfunc` (Optional) a function that specifies an arbitrary criterion. It must have especial arguments and output. See 'Details' of `minimax`.

`sensfunc` (Optional) a function that specifies the sensitivity function for `crtfunc`. See 'Details' of `minimax`.

Details

Let Θ be the set initial estimates for the model parameters and π be a probability measure having support in Θ. A design ξ^* is robust with respect to π if the following inequality holds for all $\boldsymbol{x} \in \chi$:

$$c(\boldsymbol{x}, \pi, \xi^*) = \int_\pi trM^{-1}(\xi^*, \theta)I(\boldsymbol{x}, \theta)\pi(\theta)d(\theta) - p \leq 0,$$

with equality at all support points of ξ^*. Here, p is the number of model parameters.

ELB is a measure of proximity of a design to the optimal design without knowing the latter. Given a design, let ϵ be the global maximum of the sensitivity (derivative) function over $x \in \chi$. ELB is given by

$$ELB = p/(p + \epsilon),$$

where p is the number of model parameters. Obviously, calculating ELB requires finding ϵ and another optimization problem to be solved. The tuning parameters of this optimization can be regulated via the argument `sens.minimax.control`.

Value

an object of class sensminimax that is a list with the following elements:

- type Argument type that is required for print methods.
- optima A matrix that stores all the local optima over the parameter space. The cost (criterion) values are stored in a column named Criterion_Value. The last column (Answering_Set) shows if the optimum belongs to the answering set (1) or not (0). See 'Details' of sens.minimax.control. Only applicable for minimax or standardized maximin designs.
- mu Probability measure on the answering set. Corresponds to the rows of optima for which the associated row in column Answering_Set is equal to 1. Only applicable for minimax or standardized maximin designs.
- max_deriv Global maximum of the sensitivity (derivative) function (ϵ in 'Details').
- ELB D-efficiency lower bound. Can not be larger than 1. If negative, see 'Note' in sensminimax or sens.minimax.control.
- merge_tol Merging tolerance to create the answering set from the set of all local optima. See 'Details' in sens.minimax.control. Only applicable for minimax or standardized maximin designs.
- crtval Criterion value. Compare it with the column Crtiterion_Value in optima for minimax and standardized maximin designs.
- time Used CPU time (rough approximation).

Note

Theoretically, ELB can not be larger than 1. But if so, it may have one of the following reasons:

- max_deriv is not a GLOBAL maximum. Please increase the value of the parameter maxeval in sens.minimax.control to find the global maximum.
- The sensitivity function is shifted below the y-axis because the number of model parameters has not been specified correctly (less value given). Please specify the correct number of model parameters via the argument npar.

See Also

bayes sensbayes robust

Examples

```
# Verifying a robust design for the two-parameter logistic model
sensrobust(formula = ~1/(1 + exp(-b *(x - a))),
           predvars = c("x"),
           parvars = c("a", "b"),
           family = binomial(),
           prob = rep(1/4, 4),
           parset = matrix(c(0.5, 1.5, 0.5, 1.5, 4.0, 4.0, 5.0, 5.0), 4, 2),
           x = c(0.260, 1, 1.739), w = c(0.275, 0.449, 0.275),
           lx = -5, ux = 5)
```

```
####################################
# user-defined optimality criterion
###################################
# When the model is defined by the formula interface
# Checking the A-optimality  for the 2PL model.
# the criterion function must have argument x, w fimfunc and the parameters defined in 'parvars'.
# use 'fimfunc' as a function of the design points x,  design weights w and
#  the 'parvars' parameters whenever needed.
Aopt <-function(x, w, a, b, fimfunc){
  sum(diag(solve(fimfunc(x = x, w = w, a = a, b = b))))
}
## the sensitivtiy function
# xi_x is a design that put all its mass on x in the definition of the sensitivity function
# x is a vector of design points
Aopt_sens <- function(xi_x, x, w, a, b, fimfunc){
  fim <- fimfunc(x = x, w = w, a = a, b = b)
  M_inv <- solve(fim)
  M_x <- fimfunc(x = xi_x, w = 1, a  = a, b = b)
  sum(diag(M_inv %*% M_x %*%  M_inv)) - sum(diag(M_inv))
}

sensrobust(formula = ~1/(1 + exp(-b * (x-a))), predvars = "x",
           parvars = c("a", "b"), family = "binomial",
           crtfunc = Aopt,
           sensfunc = Aopt_sens,
           lx = -3, ux = 3,
           prob = c(.25, .5, .25),
           parset = matrix(c(-2, 0, 2, 1.25, 1.25, 1.25), 3, 2),
           x = c(-2.469, 0, 2.469), w = c(.317, .365, .317))
# not optimal. the optimal design has four points. see the last example in ?robust
```

---

skewnormal — *Assumes A Multivariate Skewed Normal Prior Distribution for The Model Parameters*

Description

Creates a multivariate skewed normal prior distribution for the unknown parameters as an object of class `cprior`.

Usage

```
skewnormal(xi, Omega, alpha, lower, upper)
```

Arguments

`xi`	A numeric vector of length `d=length(alpha)` representing the location parameter of the distribution. For more details, see 'Background' in `dmsn`.
`Omega`	A symmetric positive-definite matrix of dimension `(d,d)`. For more details, see 'Background' in `dmsn`.

alpha — A numeric vector which regulates the slant of the density. For more details, see 'Background' in dmsn.

lower — A vector of lower bounds for the model parameters.

upper — A vector of upper bounds for the model parameters.

Value

An object of class cprior that is a list with the following components:

- fn: prior distribution as an R function with argument param that is the vector of the unknown parameters. See below.
- npar: Number of unknown parameters and is equal to the length of param.
- lower: Argument lower. It has the same length as param.
- upper: Argument lower. It has the same length as param.

The list will be passed to the argument prior of the function bayes. The order of the argument param in fn has the same order as the argument parvars when the model is specified by a formula. Otherwise, it is equal to the argument param in the function fimfunc.

See Also

bayes sensbayes

Examples

```
skewnormal(xi = c(0, 1),
 Omega = matrix(c(1, -0.17, -0.17, .5), nrow = 2),
  alpha = c(1, 0), lower =  c(-3, .1), upper = c(3, 2))
```

---

student — *Multivariate Student's t Prior Distribution for Model Parameters*

Description

Creates the prior distribution for the parameters as an object of class cprior.

Usage

```
student(mean, S, df, lower, upper)
```

Arguments

mean	A vector of length d=ncol(S), representing the location parameter (equal to the mean vector when df>1). For more details, see 'Arguments' in dmt.
S	A symmetric positive-definite matrix representing the scale matrix of the distribution, such that S*df/(df-2) is the variance-covariance matrix when df>2. For more details, see 'Arguments' in dmt.
df	Degrees of freedom; it must be a positive integer. For more details, see 'Arguments' in dmt.
lower	A vector of lower bounds for the model parameters.
upper	A vector of upper bounds for the model parameters.

Value

An object of class cprior that is a list with the following components:

- fn: prior distribution as an R function with argument param that is the vector of the unknown parameters. See below.
- npar: Number of unknown parameters and is equal to the length of param.
- lower: Argument lower. It has the same length as param.
- upper: Argument lower. It has the same length as param.

The list will be passed to the argument prior of the function bayes. The order of the argument param in fn has the same order as the argument parvars when the model is specified by a formula. Otherwise, it is equal to the argument param in the function fimfunc.

See Also

bayes sensbayes

Examples

```
skewnormal(xi = c(0, 1),
 Omega = matrix(c(1, -0.17, -0.17, .5), nrow = 2),
  alpha = c(1, 0), lower =  c(-3, .1), upper = c(3, 2))
```

---

uniform — *Assume A Multivariate Uniform Prior Distribution for The Model Parameters*

Description

Creates independent uniform prior distributions for the unknown model parameters as an object of class cprior.

Usage

```
uniform(lower, upper)
```

Arguments

`lower`	A vector of lower bounds for the model parameters.
`upper`	A vector of upper bounds for the model parameters.

Value

An object of class `cprior` that is a list with the following components:

- `fn`: prior distribution as an R `function` with argument `param` that is the vector of the unknown parameters. See below.
- `npar`: Number of unknown parameters and is equal to the length of `param`.
- `lower`: Argument `lower`. It has the same length as `param`.
- `upper`: Argument `lower`. It has the same length as `param`.

The list will be passed to the argument `prior` of the function `bayes`. The order of the argument `param` in `fn` has the same order as the argument `parvars` when the model is specified by a formula. Otherwise, it is equal to the argument `param` in the function `fimfunc`.

Note

The order of the argument `param` in `fn` has the same order as the argument `parvars` when the model is specified by a formula. Otherwise, it is the same as the argument `param` in the function `fimfunc`.

See Also

`bayes` `sensbayes`

Examples

```
uniform(lower =  c(-3, .1), upper = c(3, 2))
```

`update.minimax` *Updating an Object of Class* `minimax`

Description

Runs the ICA optimization algorithm on an object of class `minimax` for more number of iterations and updates the results.

Usage

```
## S3 method for class 'minimax'
update(object, iter, ...)
```

Arguments

object	An object of class minimax.
iter	Number of iterations.
...	An argument of no further use.

See Also

minimax

Index