Package ‘ForLion’

February 11, 2025

Type Package

Title 'ForLion' Algorithm to Find D-Optimal Designs for Experiments

Version 0.1.0

Maintainer Siting Lin <slin95@uic.edu>

Description
Designing experimental plans that involve both discrete and continuous factors with general parametric statistical models using the 'ForLion' algorithm and 'EW ForLion' algorithm. The algorithms will search for locally optimal designs and EW optimal designs under the D-criterion. Reference: Huang, Y., Li, K., Mandal, A., & Yang, J., (2024)<doi:10.1007/s11222-024-10465-x>.

License MIT + file LICENSE

Encoding UTF-8

Imports psych, stats, cubature

RoxygenNote 7.3.2

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Yifei Huang [aut],
Siting Lin [aut, cre],
Jie Yang [aut]

Repository CRAN

Date/Publication 2025-02-11 09:10:02 UTC

Contents

`design_initial_self` *function to generate random initial design with design points and the approximate allocation*

Description

function to generate random initial design with design points and the approximate allocation

Usage

```
design_initial_self(
  k.continuous,
  factor.level,
  lvec,
  uvec,
  bvec,
  h.func,
  link = "continuation",
  Fi.func = Fi_MLM_func,
  delta = 1e-06,
  epsilon = 1e-12,
  maxit = 1000
)
```

Arguments

k.continuous	number of continuous variables
factor.level	lower, upper limit of continuous variables, and discrete levels of categorical variables, continuous factors come first
lvec	lower limit of continuous variables
uvec	upper limit of continuous variables
bvec	assumed parameter values of beta
h.func	function, is used to transfer the design point to model matrix (e.g. add interaction term, add intercept)
link	link function, default "continuation", other options "baseline", "adjacent" and "cumulative"
Fi.func	function, is used to calculate Fisher inforamtion for a design point - default to be Fi_MLM_func() in the package
delta	tuning parameter, the distance threshold, || x_i(0) - x_j(0) || >= delta
epsilon	or determining f.det > 0 numerically, f.det <= epsilon will be considered as f.det <= 0
maxit	maximum number of iterations

Value

X matrix of initial design point

p0 initial random approximate allocation

f.det the determinant of Fisher information matrix for the random initial design

Examples

```
k.continuous.temp=5
link.temp = "cumulative"
n.factor.temp = c(0,0,0,0,0,2)  # 1 discrete factor w/ 2 levels + 5 continuous
```

```
## Note: Always put continuous factors ahead of discrete factors,
## pay attention to the order of coefficients paring with predictors
lvec.temp = c(-25,-200,-150,-100,0,-1)
uvec.temp = c(25,200,0,0,16,1)
hfunc.temp = function(y){
if(length(y) != 6){stop("Input should have length 6");}
 model.mat = matrix(NA, nrow=5, ncol=10, byrow=TRUE)
 model.mat[5,]=0
 model.mat[1:4,1:4] = diag(4)
 model.mat[1:4, 5] =((-1)*y[6])
 model.mat[1:4, 6:10] = matrix(((-1)*y[1:5]), nrow=4, ncol=5, byrow=TRUE)
 return(model.mat)
 }
bvec.temp=c(-1.77994301, -0.05287782,  1.86852211, 2.76330779, -0.94437464, 0.18504420,
-0.01638597, -0.03543202, -0.07060306, 0.10347917)

design_initial_self(k.continuous=k.continuous.temp, factor.level=n.factor.temp, lvec=lvec.temp,
uvec=uvec.temp, bvec=bvec.temp, h.func=hfunc.temp, link=link.temp)
```

`discrete_rv_self` *function to generate discrete uniform random variables for initial random design points in ForLion*

Description

function to generate discrete uniform random variables for initial random design points in ForLion

Usage

```
discrete_rv_self(n, xlist)
```

Arguments

`n`	number of discrete random variables
`xlist`	list of levels for variables to be generated

Value

list of discrete uniform random variables

Examples

```
n=3 #three discrete random variables
xlist=list(c(-1,1),c(-1,1),c(-1,0,1)) #two binary and one three-levels
discrete_rv_self(n, xlist)
```

`dprime_func_self` *Function to calculate du/dx in the gradient of d(x, Xi), will be used in ForLion_MLM_func() function, details see Appendix C in Huang, Li, Mandal, Yang (2024)*

Description

Function to calculate du/dx in the gradient of d(x, Xi), will be used in ForLion_MLM_func() function, details see Appendix C in Huang, Li, Mandal, Yang (2024)

Usage

```
dprime_func_self(
  xi,
  bvec,
  h.func,
  h.prime,
  inv.F.mat,
  Ux,
  link = "continuation",
  k.continuous
)
```

Arguments

`xi`	a vector of design point
`bvec`	parameter of the multinomial logistic regression model
`h.func`	function, is used to transfer xi to model matrix (e.g. add interaction term, add intercept)
`h.prime`	function, is used to find dX/dx
`inv.F.mat`	inverse of F_Xi matrix, inverse of fisher information of current design w/o new point
`Ux`	U_x matrix in the algorithm, get from Fi_MLM_func() function
`link`	multinomial link function, default is"continuation", other choices "baseline", "cumulative", and "adjacent"
`k.continuous`	number of continuous factors

Value

dU/dx in the gradient of sensitivity function d(x, Xi)

EW_design_initial_self *function to generate random initial design with design points and the approximate allocation (For EW)*

Description

function to generate random initial design with design points and the approximate allocation (For EW)

Usage

```
EW_design_initial_self(
  k.continuous,
  factor.level,
  lvec,
  uvec,
  bvec_matrix,
  h.func,
  link = "continuation",
  EW_Fi.func = EW_Fi_MLM_func,
  delta = 1e-06,
  epsilon = 1e-12,
  maxit = 1000
)
```

Arguments

k.continuous	number of continuous variables
factor.level	lower, upper limit of continuous variables, and discrete levels of categorical variables, continuous factors come first
lvec	lower limit of continuous variables
uvec	upper limit of continuous variables
bvec_matrix	the matrix of the bootstrap parameter values of beta
h.func	function, is used to transfer the design point to model matrix (e.g. add interaction term, add intercept)
link	link function, default "continuation", other options "baseline", "adjacent" and "cumulative"
EW_Fi.func	function, is used to calculate the Expectation of Fisher information for a design point - default to be EW_Fi_MLM_func() in the package
delta	tuning parameter, the distance threshold, || x_i(0) - x_j(0) || >= delta
epsilon	determining f.det > 0 numerically, f.det <= epsilon will be considered as f.det <= 0
maxit	maximum number of iterations

Value

X matrix of initial design point

p0 initial random approximate allocation

f.det the determinant of Fisher information matrix for the random initial design

Examples

```
k.continuous.temp=1
link.temp = "continuation"
n.factor.temp = c(0)
factor.level.temp = list(c(80,200))
hfunc.temp = function(y){
matrix(data=c(1,y,y*y,0,0,0,0,0,1,y,0,0,0,0,0), nrow=3, ncol=5, byrow=TRUE)
}
lvec.temp = 80
uvec.temp = 200
bvec_bootstrap<-matrix(c(-0.2401, -1.9292, -2.7851, -1.614,-1.162,
                         -0.0535, -0.0274, -0.0096,-0.0291, -0.04,
                          0.0004,  0.0003,  0.0002,  0.0003,  0.1,
                         -9.2154, -9.7576, -9.6818, -8.5139, -8.56),nrow=4,byrow=TRUE)
EW_design_initial_self(k.continuous=k.continuous.temp, factor.level=n.factor.temp, lvec=lvec.temp,
uvec=uvec.temp, bvec_matrix=bvec_bootstrap, h.func=hfunc.temp, link=link.temp)
```

`EW_dprime_func_self` *Function to calculate dEu/dx in the gradient of d(x, Xi), will be used in EW_ForLion_MLM_func() function*

Description

Function to calculate dEu/dx in the gradient of d(x, Xi), will be used in EW_ForLion_MLM_func() function

Usage

```
EW_dprime_func_self(
  xi,
  bvec_matrix,
  h.func,
  h.prime,
  inv.F.mat,
  EUx,
  link = "continuation",
  k.continuous
)
```

Arguments

`xi`	a vector of design point
`bvec_matrix`	the matrix of the bootstrap parameter values of beta
`h.func`	function, is used to transfer xi to model matrix (e.g. add interaction term, add intercept)
`h.prime`	function, is used to find dX/dx
`inv.F.mat`	inverse of F_Xi matrix, inverse of the Expectation of fisher information of current design w/o new point
`EUx`	EU_x matrix in the algorithm, get from EW_Fi_MLM_func() function
`link`	link multinomial link function, default is"continuation", other choices "baseline", "cumulative", and "adjacent"
`k.continuous`	number of continuous factors

Value

dEU/dx in the gradient of sensitivity function d(x, Xi)

`EW_Fi_MLM_func` *Function to generate the Expectation of fisher information at one design point xi for multinomial logit models*

Description

Function to generate the Expectation of fisher information at one design point xi for multinomial logit models

Usage

```
EW_Fi_MLM_func(X_x, bvec_matrix, link = "continuation")
```

Arguments

`X_x`	model matrix for a specific design point x_i, X_x=h.func(xi)
`bvec_matrix`	the matrix of the bootstrap parameter values of beta
`link`	multinomial logit model link function name "baseline", "cumulative", "adjacent", or"continuation", default to be "continuation"

Value

F_x Fisher information matrix at x_i

EU_x U matrix for calculation the Expectation of Fisher information matrix at x_i

Examples

```
link.temp = "continuation"
xi.temp=c(80)
hfunc.temp = function(y){
matrix(data=c(1,y,y*y,0,0,0,0,0,1,y,0,0,0,0,0), nrow=3, ncol=5, byrow=TRUE)
}
X_xtemp=hfunc.temp(xi.temp)
bvec_bootstrap<-matrix(c(-0.2401, -1.9292, -2.7851, -1.614,-1.162,
                         -0.0535, -0.0274, -0.0096,-0.0291, -0.04,
                          0.0004,  0.0003,  0.0002,  0.0003,  0.1,
                         -9.2154, -9.7576, -9.6818, -8.5139, -8.56),nrow=4,byrow=TRUE)
EW_Fi_MLM_func(X_x=X_xtemp, bvec_matrix=bvec_bootstrap, link=link.temp)
```

`EW_ForLion_GLM_Optimal` *EW ForLion for generalized linear models*

Description

EW ForLion algorithm to find EW D-optimal design for GLM models with mixed factors, reference: . Factors may include discrete factors with finite number of distinct levels and continuous factors with specified interval range (min, max), continuous factors, if any, must serve as main-effects only, allowing merging points that are close enough.Continuous factors first then discrete factors, model parameters should in the same order of factors.

Usage

```
EW_ForLion_GLM_Optimal(
  n.factor,
  factor.level,
  hfunc,
  joint_Func_b,
  Lowerbounds,
  Upperbounds,
  link,
  reltol = 1e-05,
  rel.diff = 0,
  optim_grad = TRUE,
  maxit = 100,
  random = FALSE,
  nram = 3,
  logscale = FALSE,
  rowmax = NULL,
  Xini = NULL
)
```

Arguments

`n.factor`	vector of numbers of distinct levels, "0" indicates continuous factors, "0"s always come first, "2" or above indicates discrete factor, "1" is not allowed
`factor.level`	list of distinct levels, (min, max) for continuous factor, continuous factors first, should be the same order as n.factor
`hfunc`	function for obtaining model matrix h(y) for given design point y, y has to follow the same order as n.factor
`joint_Func_b`	The prior joint probability distribution of the parameters
`Lowerbounds`	The lower limit of the prior distribution for each parameter
`Upperbounds`	The upper limit of the prior distribution for each parameter
`link`	link function, default "logit", other links: "probit", "cloglog", "loglog", "cauchit", "log", "identity"
`reltol`	the relative convergence tolerance, default value 1e-5
`rel.diff`	points with distance less than that will be merged, default value 0
`optim_grad`	TRUE or FALSE, default is FALSE, whether to use the analytical gradient function or numerical gradient for searching optimal new design point
`maxit`	the maximum number of iterations, default value 100
`random`	TRUE or FALSE, if TRUE then the function will run EW lift-one with additional "nram" number of random approximate allocation, default to be FALSE
`nram`	when random == TRUE, the function will run EW lift-one nram number of initial proportion p00, default is 3
`logscale`	TRUE or FALSE, if TRUE then the EW ForLion will run EW lift-one with logscale, which is EW_liftoneDoptimal_log_GLM_func(); if FALSE then ForLion will run EW lift-one without logscale, which is EW_liftoneDoptimal_GLM_func()
`rowmax`	maximum number of points in the initial design, default NULL indicates no restriction
`Xini`	initial list of design points, default NULL will generate random initial design points

Value

m number of design points

x.factor matrix with rows indicating design point

p EW D-optimal approximate allocation

det Optimal determinant of Fisher information matrix

x.model model matrix X

E_w vector of E_w such that E_w=diag(p*E_w)

convergence TRUE or FALSE

min.diff the minimum Euclidean distance between design points

x.close a pair of design points with minimum distance

Examples

```
#Example  Crystallography Experiment
hfunc.temp = function(y) {c(y,1)}   # y -> h(y)=(y1,1)
n.factor.temp = c(0)  # 1 continuous factors
factor.level.temp = list(c(-1,1))
link.temp="logit"
paras_lowerbound<-c(4,-3)
paras_upperbound<-c(10,3)
 gjoint_b<- function(x) {
 Func_b<-1/(prod(paras_upperbound-paras_lowerbound))
 ##the prior distributions are follow uniform distribution
return(Func_b)
}
EW_ForLion_GLM_Optimal(n.factor=n.factor.temp, factor.level=factor.level.temp,
hfunc=hfunc.temp,joint_Func_b=gjoint_b, Lowerbounds=paras_lowerbound,
Upperbounds=paras_upperbound, link=link.temp, reltol=1e-2, rel.diff=0.01,
optim_grad=FALSE, maxit=500, random=FALSE, nram=3, logscale=FALSE,Xini=NULL)
```

EW_ForLion_MLM_Optimal

EW ForLion function for multinomial logit models

Description

EW ForLion function for multinomial logit models

Usage

```
EW_ForLion_MLM_Optimal(
  J,
  n.factor,
  factor.level,
  hfunc,
  h.prime,
  bvec_matrix,
  link = "continuation",
  EW_Fi.func = EW_Fi_MLM_func,
  delta = 1e-05,
  epsilon = 1e-12,
  reltol = 1e-05,
  rel.diff = 0,
  maxit = 100,
  random = FALSE,
  nram = 3,
  rowmax = NULL,
  Xini = NULL,
  random.initial = FALSE,
  nram.initial = 3,
```

```
  optim_grad = FALSE
)
```

Arguments

`J`	number of response levels in the multinomial logit model
`n.factor`	vector of numbers of distinct levels, "0" indicates continuous factors, "0"s always come first, "2" or above indicates discrete factor, "1" is not allowed
`factor.level`	list of distinct levels, (min, max) for continuous factor, continuous factors first, should be the same order as n.factor
`hfunc`	function for obtaining model matrix h(y) for given design point y, y has to follow the same order as n.factor
`h.prime`	function to obtain dX/dx
`bvec_matrix`	the matrix of the bootstrap parameter values of beta
`link`	link function, default "continuation", other choices "baseline", "cumulative", and "adjacent"
`EW_Fi.func`	function to calculate row-wise Expectation of Fisher information Fi, default is EW_Fi_MLM_func
`delta`	tuning parameter, the generated design pints distance threshold, || x_i(0) - x_j(0) || >= delta, default 1e-5
`epsilon`	determining f.det > 0 numerically, f.det <= epsilon will be considered as f.det <= 0, default 1e-12
`reltol`	the relative convergence tolerance, default value 1e-5
`rel.diff`	points with distance less than that will be merged, default value 0
`maxit`	the maximum number of iterations, default value 100
`random`	TRUE or FALSE, if TRUE then the function will run EW lift-one with additional "nram" number of random approximate allocation, default to be FALSE
`nram`	when random == TRUE, the function will run EW lift-one nram number of initial proportion p00, default is 3
`rowmax`	maximum number of points in the initial design, default NULL indicates no restriction
`Xini`	initial list of design points, default NULL will generate random initial design points
`random.initial`	TRUE or FALSE, if TRUE then the function will run EW ForLion with additional "nram.initial" number of random initial design points, default FALSE
`nram.initial`	when random.initial == TRUE, the function will run EW ForLion algorithm with nram.initial number of initial design points Xini, default is 3
`optim_grad`	TRUE or FALSE, default is FALSE, whether to use the analytical gradient function or numerical gradient for searching optimal new design point

Value

m the number of design points

x.factor matrix of experimental factors with rows indicating design point

p the reported EW D-optimal approximate allocation

det the determinant of the maximum Expectation of Fisher information

convergence TRUE or FALSE, whether converge

min.diff the minimum Euclidean distance between design points

x.close pair of design points with minimum distance

itmax iteration of the algorithm

Examples

```
J=3
p=5
hfunc.temp = function(y){
matrix(data=c(1,y,y*y,0,0,0,0,0,1,y,0,0,0,0,0), nrow=3, ncol=5, byrow=TRUE)
} #hfunc is a 3*5 matrix, transfer x design matrix to model matrix for emergence of flies example

hprime.temp = function(y){
matrix(data=c(0, 1, 2*y, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0), nrow=3, ncol=5, byrow=TRUE)
}

link.temp = "continuation"
n.factor.temp = c(0)  # 1 continuous factor no discrete factor in EW ForLion
factor.level.temp = list(c(80,200)) #boundary for continuous parameter in Forlion
bvec_bootstrap<-matrix(c(-0.2401, -1.9292, -2.7851, -1.614,-1.162,
                         -0.0535, -0.0274, -0.0096,-0.0291, -0.04,
                          0.0004,  0.0003,  0.0002,  0.0003,  0.1,
                         -9.2154, -9.7576, -9.6818, -8.5139, -8.56),nrow=4,byrow=TRUE)
EW_ForLion_MLM_Optimal(J=J, n.factor=n.factor.temp, factor.level=factor.level.temp,
         hfunc=hfunc.temp,h.prime=h.prime.temp, bvec_matrix=bvec_bootstrap,rel.diff=1,
         link=link.temp, optim_grad=FALSE)
```

EW_liftoneDoptimal_GLM_func
EW Lift-one algorithm for D-optimal approximate design

Description

EW Lift-one algorithm for D-optimal approximate design

Usage

```
EW_liftoneDoptimal_GLM_func(
  X,
  E_w,
  reltol = 1e-05,
  maxit = 100,
  random = FALSE,
  nram = 3,
  p00 = NULL
)
```

Arguments

X	Model matrix, with nrow = num of design points and ncol = num of parameters
E_w	Diagonal of E_W matrix in Fisher information matrix, can be calculated EW_Xw_maineffects_self() function in the ForLion package
reltol	reltol The relative convergence tolerance, default value 1e-5
maxit	The maximum number of iterations, default value 100
random	TRUE or FALSE, if TRUE then the function will run with additional "nram" number of initial allocation p00, default to be TRUE
nram	When random == TRUE, the function will generate nram number of initial points, default is 3
p00	Specified initial design approximate allocation; default to be NULL, this will generate a random initial design

Value

p EW D-optimal approximate allocation

p0 Initial approximate allocation that derived the reported EW D-optimal approximate allocation

Maximum The maximum of the determinant of the Fisher information matrix of the reported EW D-optimal design

convergence Convergence TRUE or FALSE

itmax number of the iteration

Examples

```
hfunc.temp = function(y) {c(y,1);};   # y -> h(y)=(y1,y2,y3,1)
link.temp="logit"
paras_lowerbound<-rep(-Inf, 4)
paras_upperbound<-rep(Inf, 4)
gjoint_b<- function(x) {
mu1 <- -0.5; sigma1 <- 1
mu2 <- 0.5; sigma2 <- 1
mu3 <- 1; sigma3 <- 1
mu0 <- 1; sigma0 <- 1
d1 <- stats::dnorm(x[1], mean = mu1, sd = sigma1)
```

```
d2 <- stats::dnorm(x[2], mean = mu2, sd = sigma2)
d3 <- stats::dnorm(x[3], mean = mu3, sd = sigma3)
d4 <- stats::dnorm(x[4], mean = mu0, sd = sigma0)
return(d1 * d2 * d3 * d4)
}
x.temp=matrix(data=c(-2,-1,-3,2,-1,-3,-2,1,-3,2,1,-3,-2,-1,3,2,-1,3,-2,1,3,2,1,3),ncol=3,byrow=TRUE)
m.temp=dim(x.temp)[1]     # number of design points
p.temp=length(paras_upperbound)    # number of predictors
Xmat.temp=matrix(0, m.temp, p.temp)
EW_wvec.temp=rep(0, m.temp)
for(i in 1:m.temp) {
htemp=EW_Xw_maineffects_self(x=x.temp[i,],joint_Func_b=gjoint_b, Lowerbounds=paras_lowerbound,
                       Upperbounds=paras_upperbound, link=link.temp, h.func=hfunc.temp);
Xmat.temp[i,]=htemp$X;
EW_wvec.temp[i]=htemp$E_w;
}
EW_liftoneDoptimal_GLM_func(X=Xmat.temp, E_w=EW_wvec.temp, reltol=1e-8, maxit=1000,
                            random=TRUE, nram=3, p00=NULL)
```

EW_liftoneDoptimal_log_GLM_func
EW Lift-one algorithm for D-optimal approximate design in log scale

---

Description

EW Lift-one algorithm for D-optimal approximate design in log scale

Usage

```
EW_liftoneDoptimal_log_GLM_func(
  X,
  E_w,
  reltol = 1e-05,
  maxit = 100,
  random = FALSE,
  nram = 3,
  p00 = NULL
)
```

Arguments

X	Model matrix, with nrow = num of design points and ncol = num of parameters
E_w	Diagonal of E_W matrix in Fisher information matrix, can be calculated EW_Xw_maineffects_self() function in the ForLion package
reltol	reltol The relative convergence tolerance, default value 1e-5
maxit	The maximum number of iterations, default value 100
random	TRUE or FALSE, if TRUE then the function will run with additional "nram" number of initial allocation p00, default to be TRUE

nram	When random == TRUE, the function will generate nram number of initial points, default is 3
p00	Specified initial design approximate allocation; default to be NULL, this will generate a random initial design

Value

p EW D-optimal approximate allocation

p0 Initial approximate allocation that derived the reported EW D-optimal approximate allocation

Maximum The maximum of the determinant of the Fisher information matrix of the reported EW D-optimal design

convergence Convergence TRUE or FALSE

itmax number of the iteration

Examples

```
hfunc.temp = function(y) {c(y,1);};   # y -> h(y)=(y1,y2,y3,1)
link.temp="logit"
paras_lowerbound<-rep(-Inf, 4)
paras_upperbound<-rep(Inf, 4)
gjoint_b<- function(x) {
mu1 <- -0.5; sigma1 <- 1
mu2 <- 0.5; sigma2 <- 1
mu3 <- 1; sigma3 <- 1
mu0 <- 1; sigma0 <- 1
d1 <- stats::dnorm(x[1], mean = mu1, sd = sigma1)
d2 <- stats::dnorm(x[2], mean = mu2, sd = sigma2)
d3 <- stats::dnorm(x[3], mean = mu3, sd = sigma3)
d4 <- stats::dnorm(x[4], mean = mu0, sd = sigma0)
return(d1 * d2 * d3 * d4)
}
x.temp=matrix(data=c(-2,-1,-3,2,-1,-3,-2,1,-3,2,1,-3,-2,-1,3,2,-1,3,-2,1,3,2,1,3),
              ncol=3,byrow=TRUE)
m.temp=dim(x.temp)[1]     # number of design points
p.temp=length(paras_upperbound)    # number of predictors
Xmat.temp=matrix(0, m.temp, p.temp)
EW_wvec.temp=rep(0, m.temp)
for(i in 1:m.temp) {
htemp=EW_Xw_maineffects_self(x=x.temp[i,],joint_Func_b=gjoint_b, Lowerbounds=paras_lowerbound,
                       Upperbounds=paras_upperbound, link=link.temp, h.func=hfunc.temp);
Xmat.temp[i,]=htemp$X;
EW_wvec.temp[i]=htemp$E_w;
}
EW_liftoneDoptimal_GLM_func(X=Xmat.temp, E_w=EW_wvec.temp, reltol=1e-8, maxit=1000, random=TRUE,
                            nram=3, p00=NULL)
```

`EW_liftoneDoptimal_MLM_func` *function of EW liftone for multinomial logit model*

Description

function of EW liftone for multinomial logit model

Usage

```
EW_liftoneDoptimal_MLM_func(
  m,
  p,
  Xi,
  J,
  thetavec_matrix,
  link = "continuation",
  reltol = 1e-05,
  maxit = 500,
  p00 = NULL,
  random = FALSE,
  nram = 3
)
```

Arguments

`m`	number of design points
`p`	number of parameters in the multinomial logit model
`Xi`	model matrix
`J`	number of response levels in the multinomial logit model
`thetavec_matrix`	the matrix of the bootstrap parameter values of beta
`link`	multinomial logit model link function name "baseline", "cumulative", "adjacent", or"continuation", default to be "continuation"
`reltol`	relative tolerance for convergence, default to 1e-5
`maxit`	the number of maximum iteration, default to 500
`p00`	specified initial approximate allocation, default to NULL, if NULL, will generate a random initial approximate allocation
`random`	TRUE or FALSE, if TRUE then the function will run with additional "nram" number of initial allocation p00, default to be TRUE
`nram`	when random == TRUE, the function will generate nram number of initial points, default is 3

Value

p reported EW D-optimal approximate allocation

p0 the initial approximate allocation that derived the reported EW D-optimal design

Maximum the maximum of the determinant of the Expectation of Fisher information matrix

Convergence TRUE or FALSE, whether the algorithm converges

itmax, maximum iterations

Examples

```
m=7
p=5
J=3
link.temp = "continuation"
factor_x=c(80,100,120,140,160,180,200)
hfunc.temp = function(y){
matrix(data=c(1,y,y*y,0,0,0,0,0,1,y,0,0,0,0,0), nrow=3, ncol=5, byrow=TRUE)
}
Xi=rep(0,J*p*m); dim(Xi)=c(J,p,m)
for(i in 1:m) {
Xi[,,i]=hfunc.temp(factor_x[i])
}
bvec_bootstrap<-matrix(c(-0.2401, -1.9292, -2.7851, -1.614,-1.162,
                         -0.0535, -0.0274, -0.0096,-0.0291, -0.04,
                          0.0004,  0.0003,  0.0002,  0.0003,  0.1,
                         -9.2154, -9.7576, -9.6818, -8.5139, -8.56),nrow=4,byrow=TRUE)
EW_liftoneDoptimal_MLM_func(m=m, p=p, Xi=Xi, J=J, thetavec_matrix=bvec_bootstrap,
link = "continuation",reltol=1e-5, maxit=500, p00=rep(1/7,7), random=FALSE, nram=3)
```

EW_Xw_maineffects_self

function for calculating X=h(x) and E_w=E(nu(beta^T h(x))) give a design point x=(1,x1,...,xd)^T

Description

function for calculating X=h(x) and E_w=E(nu(beta^T h(x))) give a design point x=(1,x1,...,xd)^T

Usage

```
EW_Xw_maineffects_self(
  x,
  joint_Func_b,
  Lowerbounds,
  Upperbounds,
  link = "logit",
```

```
  h.func = NULL
)
```

Arguments

x	x=(x1,...,xd) – design point/experimental setting
joint_Func_b	The prior joint probability distribution of the parameters
Lowerbounds	The lower limit of the prior distribution for each parameter
Upperbounds	The upper limit of the prior distribution for each parameter
link	link = "logit" – link function, default: "logit", other links: "probit", "cloglog", "loglog", "cauchit", "log"
h.func	function h(x)=(h1(x),...,hp(x)), default (1,x1,...,xd)

Value

X=h(x)=(h1(x),...,hp(x)) – a row for design matrix

E_w – E(nu(b^t h(x)))

link – link function applied

Examples

```
hfunc.temp = function(y) {c(y,1);};   # y -> h(y)=(y1,y2,y3,1)
link.temp="logit"
paras_lowerbound<-rep(-Inf, 4)
paras_upperbound<-rep(Inf, 4)
gjoint_b<- function(x) {
mu1 <- -0.5; sigma1 <- 1
mu2 <- 0.5; sigma2 <- 1
mu3 <- 1; sigma3 <- 1
mu0 <- 1; sigma0 <- 1
d1 <- stats::dnorm(x[1], mean = mu1, sd = sigma1)
d2 <- stats::dnorm(x[2], mean = mu2, sd = sigma2)
d3 <- stats::dnorm(x[3], mean = mu3, sd = sigma3)
d4 <- stats::dnorm(x[4], mean = mu0, sd = sigma0)
return(d1 * d2 * d3 * d4)
}
x.temp = c(2,1,3)
EW_Xw_maineffects_self(x=x.temp,joint_Func_b=gjoint_b, Lowerbounds=paras_lowerbound,
 Upperbounds=paras_upperbound, link=link.temp, h.func=hfunc.temp)
```

Fi_MLM_func *Function to generate fisher information at one design point xi for multinomial logit models*

Description

Function to generate fisher information at one design point xi for multinomial logit models

Usage

```
Fi_MLM_func(X_x, bvec, link = "continuation")
```

Arguments

X_x	model matrix for a specific design point x_i, X_x=h.func(xi)
bvec	beta coefficients in the model
link	multinomial logit model link function name "baseline", "cumulative", "adjacent", or"continuation", default to be "continuation"

Value

F_x Fisher information matrix at x_i

U_x U matrix for calculation of Fisher information matrix at x_i (see Corollary 3.1 in Bu, Majumdar, Yang(2020))

Examples

```
# Reference minimizing surface example in supplementary material
# Section S.3 in Huang, Li, Mandal, Yang (2024)
xi.temp = c(-1, -25, 199.96, -150, -100, 16)
hfunc.temp = function(y){
if(length(y) != 6){stop("Input should have length 6");}
model.mat = matrix(NA, nrow=5, ncol=10, byrow=TRUE)
model.mat[5,]=0
model.mat[1:4,1:4] = diag(4)
model.mat[1:4, 5] =((-1)*y[6])
model.mat[1:4, 6:10] = matrix(((-1)*y[1:5]), nrow=4, ncol=5, byrow=TRUE)
return(model.mat)
}
X_x.temp = hfunc.temp(xi.temp)
bvec.temp = c(-1.77994301, -0.05287782,  1.86852211, 2.76330779, -0.94437464,
0.18504420,  -0.01638597, -0.03543202, -0.07060306, 0.10347917)
link.temp = "cumulative"
Fi_MLM_func(X_x=X_x.temp, bvec=bvec.temp, link=link.temp)
```

ForLion_GLM_Optimal *ForLion for generalized linear models*

Description

ForLion algorithm to find D-optimal design for GLM models with mixed factors, reference: Section 4 in Huang, Li, Mandal, Yang (2024). Factors may include discrete factors with finite number of distinct levels and continuous factors with specified interval range (min, max), continuous factors, if any, must serve as main-effects only, allowing merging points that are close enough. Continuous factors first then discrete factors, model parameters should in the same order of factors.

Usage

```
ForLion_GLM_Optimal(
  n.factor,
  factor.level,
  hfunc,
  bvec,
  link,
  reltol = 1e-05,
  rel.diff = 0,
  maxit = 100,
  random = FALSE,
  nram = 3,
  logscale = FALSE,
  rowmax = NULL,
  Xini = NULL
)
```

Arguments

n.factor	vector of numbers of distinct levels, "0" indicates continuous factors, "0"s always come first, "2" or above indicates discrete factor, "1" is not allowed
factor.level	list of distinct levels, (min, max) for continuous factor, continuous factors first, should be the same order as n.factor
hfunc	function for obtaining model matrix h(y) for given design point y, y has to follow the same order as n.factor
bvec	assumed parameter values of model parameters beta, same length of h(y)
link	link function, default "logit", other links: "probit", "cloglog", "loglog", "cauchit", "log", "identity"
reltol	the relative convergence tolerance, default value 1e-5
rel.diff	points with distance less than that will be merged, default value 0
maxit	the maximum number of iterations, default value 100
random	TRUE or FALSE, if TRUE then the function will run lift-one with additional "nram" number of random approximate allocation, default to be FALSE
nram	when random == TRUE, the function will run lift-one nram number of initial proportion p00, default is 3
logscale	TRUE or FALSE, if TRUE then the ForLion will run lift-one with logscale, which is liftoneDoptimal_log_GLM_func(); if FALSE then ForLion will run lift-one without logscale, which is liftoneDoptimal_GLM_func()
rowmax	maximum number of points in the initial design, default NULL indicates no restriction
Xini	initial list of design points, default NULL will generate random initial design points

Value

m number of design points

x.factor matrix with rows indicating design point

p D-optimal approximate allocation

det Optimal determinant of Fisher information matrix

convergence TRUE or FALSE

min.diff the minimum Euclidean distance between design points

x.close a pair of design points with minimum distance

itmax iteration of the algorithm

Examples

```
#Example 3 in Huang, Li, Mandal, Yang (2024), electrostatic discharge experiment
hfunc.temp = function(y) {c(y,y[4]*y[5],1);};   # y -> h(y)=(y1,y2,y3,y4,y5,y4*y5,1)
n.factor.temp = c(0, 2, 2, 2, 2)  # 1 continuous factor with 4 discrete factors
factor.level.temp = list(c(25,45),c(-1,1),c(-1,1),c(-1,1),c(-1,1))
link.temp="logit"
b.temp = c(0.3197169, 1.9740922, -0.1191797, -0.2518067, 0.1970956, 0.3981632, -7.6648090)
ForLion_GLM_Optimal(n.factor=n.factor.temp, factor.level=factor.level.temp, hfunc=hfunc.temp,
bvec=b.temp, link=link.temp, reltol=1e-2, rel.diff=0.03, maxit=500, random=FALSE,
nram=3, logscale=TRUE)
```

`ForLion_MLM_Optimal` *ForLion function for multinomial logit models*

Description

Function for ForLion algorithm to find D-optimal design under multinomial logit models with mixed factors. Reference Section 3 of Huang, Li, Mandal, Yang (2024). Factors may include discrete factors with finite number of distinct levels and continuous factors with specified interval range (min, max), continuous factors, if any, must serve as main-effects only, allowing merging points that are close enough. Continuous factors first then discrete factors, model parameters should in the same order of factors.

Usage

```
ForLion_MLM_Optimal(
  J,
  n.factor,
  factor.level,
  hfunc,
  h.prime,
  bvec,
  link = "continuation",
```

```
  Fi.func = Fi_MLM_func,
  delta = 1e-05,
  epsilon = 1e-12,
  reltol = 1e-05,
  rel.diff = 0,
  maxit = 100,
  random = FALSE,
  nram = 3,
  rowmax = NULL,
  Xini = NULL,
  random.initial = FALSE,
  nram.initial = 3,
  optim_grad = FALSE
)
```

Arguments

`J`	number of response levels in the multinomial logit model
`n.factor`	vector of numbers of distinct levels, "0" indicates continuous factors, "0"s always come first, "2" or above indicates discrete factor, "1" is not allowed
`factor.level`	list of distinct levels, (min, max) for continuous factor, continuous factors first, should be the same order as n.factor
`hfunc`	function for obtaining model matrix h(y) for given design point y, y has to follow the same order as n.factor
`h.prime`	function to obtain dX/dx
`bvec`	assumed parameter values of model parameters beta, same length of h(y)
`link`	link function, default "continuation", other choices "baseline", "cumulative", and "adjacent"
`Fi.func`	function to calculate row-wise Fisher information Fi, default is Fi_MLM_func
`delta`	tuning parameter, the generated design pints distance threshold, || x_i(0) - x_j(0) || >= delta, default 1e-5
`epsilon`	for determining f.det > 0 numerically, f.det <= epsilon will be considered as f.det <= 0, default 1e-12
`reltol`	the relative convergence tolerance, default value 1e-5
`rel.diff`	points with distance less than that will be merged, default value 0
`maxit`	the maximum number of iterations, default value 100
`random`	TRUE or FALSE, if TRUE then the function will run lift-one with additional "nram" number of random approximate allocation, default to be FALSE
`nram`	when random == TRUE, the function will run lift-one nram number of initial proportion p00, default is 3
`rowmax`	maximum number of points in the initial design, default NULL indicates no restriction
`Xini`	initial list of design points, default NULL will generate random initial design points

`random.initial` TRUE or FALSE, if TRUE then the function will run ForLion with additional "nram.initial" number of random initial design points, default FALSE

`nram.initial` when random.initial == TRUE, the function will run ForLion algorithm with nram.initial number of initial design points Xini, default is 3

`optim_grad` TRUE or FALSE, default is FALSE, whether to use the analytical gradient function or numerical gradient for searching optimal new design point

Value

m the number of design points

x.factor matrix of experimental factors with rows indicating design point

p the reported D-optimal approximate allocation

det the determinant of the maximum Fisher information

convergence TRUE or FALSE, whether converge

min.diff the minimum Euclidean distance between design points

x.close pair of design points with minimum distance

itmax iteration of the algorithm

Examples

```
m=5
p=10
J=5
link.temp = "cumulative"
n.factor.temp = c(0,0,0,0,0,2)  # 1 discrete factor w/ 2 levels + 5 continuous
## Note: Always put continuous factors ahead of discrete factors,
## pay attention to the order of coefficients paring with predictors
factor.level.temp = list(c(-25,25), c(-200,200),c(-150,0),c(-100,0),c(0,16),c(-1,1))
hfunc.temp = function(y){
if(length(y) != 6){stop("Input should have length 6");}
 model.mat = matrix(NA, nrow=5, ncol=10, byrow=TRUE)
 model.mat[5,]=0
 model.mat[1:4,1:4] = diag(4)
 model.mat[1:4, 5] =((-1)*y[6])
 model.mat[1:4, 6:10] = matrix(((-1)*y[1:5]), nrow=4, ncol=5, byrow=TRUE)
 return(model.mat)
 }
bvec.temp=c(-1.77994301, -0.05287782,  1.86852211, 2.76330779, -0.94437464, 0.18504420,
-0.01638597, -0.03543202, -0.07060306, 0.10347917)

h.prime.temp = NULL #use numerical gradient (optim_grad=FALSE)
ForLion_MLM_Optimal(J=J, n.factor=n.factor.temp, factor.level=factor.level.temp, hfunc=hfunc.temp,
h.prime=h.prime.temp, bvec=bvec.temp, link=link.temp, optim_grad=FALSE)
```

GLM_Exact_Design *Approximation to exact design algorithm for generalized linear model*

Description

Approximation to exact design algorithm for generalized linear model

Usage

```
GLM_Exact_Design(
  k.continuous,
  design_x,
  design_p,
  det.design,
  p,
  ForLion,
  bvec,
  joint_Func_b,
  Lowerbounds,
  Upperbounds,
  rel.diff,
  L,
  N,
  hfunc,
  link
)
```

Arguments

`k.continuous`	number of continuous factors
`design_x`	the matrix with rows indicating design point which we got from the approximate design
`design_p`	D-optimal approximate allocation
`det.design`	the determinant of D-optimal approximate allocation
`p`	number of parameters
`ForLion`	TRUE or FALSE, TRUE: this approximate design was generated by ForLion algorithm, FALSE: this approximate was generated by EW ForLion algorithm
`bvec`	assumed parameter values of model parameters beta, same length of h(y)
`joint_Func_b`	The prior joint probability distribution of the parameters
`Lowerbounds`	The lower limit of the prior distribution for each parameter
`Upperbounds`	The upper limit of the prior distribution for each parameter
`rel.diff`	points with distance less than that will be merged
`L`	rounding factor

N	total number of observations
hfunc	function for obtaining model matrix h(y) for given design point y, y has to follow the same order as n.factor
link	link function, default "logit", other links: "probit", "cloglog", "loglog", "cauchit", "log", "identity"

Value

x.design matrix with rows indicating design point

ni.design EW D-optimal or D-optimal exact allocation

rel.efficiency relative efficiency of the Exact and Approximate Designs

Examples

```
k.continuous=1
design_x=matrix(c(25, -1, -1,-1, -1 ,
                 25, -1, -1, -1, 1,
                 25, -1, -1, 1, -1,
                 25, -1, -1, 1, 1,
                 25, -1, 1, -1, -1,
                 25, -1, 1, -1, 1,
                 25, -1, 1, 1, -1,
                 25, -1, 1, 1, 1,
                 25, 1, -1, 1, -1,
                 25, 1, 1, -1, -1,
                 25, 1, 1, -1, 1,
                 25, 1, 1, 1, -1,
                 25, 1, 1, 1, 1,
                 38.9479, -1, 1, 1, -1,
                 34.0229, -1, 1, -1, -1,
                 35.4049, -1, 1, -1, 1,
                 37.1960, -1, -1, 1, -1,
                 33.0884, -1, 1, 1, 1),nrow=18,ncol=5,byrow = TRUE)
hfunc.temp = function(y) {c(y,y[4]*y[5],1);};   # y -> h(y)=(y1,y2,y3,y4,y5,y4*y5,1)
link.temp="logit"
design_p=c(0.0848, 0.0875, 0.0410, 0.0856, 0.0690, 0.0515,
          0.0901, 0.0845, 0.0743, 0.0356, 0.0621, 0.0443,
          0.0090, 0.0794, 0.0157, 0.0380, 0.0455, 0.0022)
det.design=4.552715e-06
paras_lowerbound<-c(0.25,1,-0.3,-0.3,0.1,0.35,-8.0)
paras_upperbound<-c(0.45,2,-0.1,0.0,0.4,0.45,-7.0)
 gjoint_b<- function(x) {
 Func_b<-1/(prod(paras_upperbound-paras_lowerbound))
 ##the prior distributions are follow uniform distribution
return(Func_b)
}
 GLM_Exact_Design(k.continuous=k.continuous,design_x=design_x,
 design_p=design_p,det.design=det.design,p=7,ForLion=FALSE,joint_Func_b=gjoint_b,
 Lowerbounds=paras_lowerbound, Upperbounds=paras_upperbound,rel.diff=0,L=1,
 N=100,hfunc=hfunc.temp,link=link.temp)
```

`liftoneDoptimal_GLM_func` *Lift-one algorithm for D-optimal approximate design*

Description

Lift-one algorithm for D-optimal approximate design

Usage

```
liftoneDoptimal_GLM_func(
  X,
  w,
  reltol = 1e-05,
  maxit = 100,
  random = FALSE,
  nram = 3,
  p00 = NULL
)
```

Arguments

`X`	Model matrix, with nrow = num of design points and ncol = num of parameters
`w`	Diagonal of W matrix in Fisher information matrix, can be calculated Xw_maineffects_self() function in the ForLion package
`reltol`	The relative convergence tolerance, default value 1e-5
`maxit`	The maximum number of iterations, default value 100
`random`	TRUE or FALSE, if TRUE then the function will run with additional "nram" number of initial allocation p00, default to be TRUE
`nram`	When random == TRUE, the function will generate nram number of initial points, default is 3
`p00`	Specified initial design approximate allocation; default to be NULL, this will generate a random initial design

Value

p D-optimal approximate allocation

p0 Initial approximate allocation that derived the reported D-optimal approximate allocation

Maximum The maximum of the determinant of the Fisher information matrix of the reported D-optimal design

convergence Convergence TRUE or FALSE

itmax number of the iteration

Examples

```
hfunc.temp = function(y) {c(y,y[4]*y[5],1);};   # y -> h(y)=(y1,y2,y3,y4,y5,y4*y5,1)
link.temp="logit"
x.temp = matrix(data=c(25.00000,1,-1,1,-1,25.00000,1,1,1,-1,32.06741,-1,1,-1,1,40.85698,
-1,1,1,-1,28.86602,-1,1,-1,-1,29.21486,-1,-1,1,1,25.00000,1,1,1,1, 25.00000,1,1,-1,-1),
ncol=5, byrow=TRUE)
b.temp = c(0.3197169, 1.9740922, -0.1191797, -0.2518067, 0.1970956, 0.3981632, -7.6648090)
X.mat = matrix(,nrow=8, ncol=7)
w.vec = rep(NA,8)
for(i in 1:8) {
htemp=Xw_maineffects_self(x=x.temp[i,], b=b.temp, link=link.temp, h.func=hfunc.temp);
X.mat[i,]=htemp$X;
w.vec[i]=htemp$w;
};
liftoneDoptimal_GLM_func(X=X.mat, w=w.vec, reltol=1e-5, maxit=500, random=TRUE, nram=3, p00=NULL)
```

`liftoneDoptimal_log_GLM_func`
Lift-one algorithm for D-optimal approximate design in log scale

---

Description

Lift-one algorithm for D-optimal approximate design in log scale

Usage

```
liftoneDoptimal_log_GLM_func(
  X,
  w,
  reltol = 1e-05,
  maxit = 100,
  random = FALSE,
  nram = 3,
  p00 = NULL
)
```

Arguments

`X`	Model matrix, with nrow = num of design points and ncol = num of parameters
`w`	Diagonal of W matrix in Fisher information matrix, can be calculated Xw_maineffects_self() function in the ForLion package
`reltol`	The relative convergence tolerance, default value 1e-5
`maxit`	The maximum number of iterations, default value 100
`random`	TRUE or FALSE, if TRUE then the function will run with additional "nram" number of initial allocation p00, default to be TRUE

nram	When random == TRUE, the function will generate nram number of initial points, default is 3
p00	Specified initial design approximate allocation; default to be NULL, this will generate a random initial design

Value

p D-optimal approximate allocation

p0 Initial approximate allocation that derived the reported D-optimal approximate allocation

Maximum The maximum of the determinant of the Fisher information matrix of the reported D-optimla design

convergence Convergence TRUE or FALSE

itmax number of the iteration

Examples

```
hfunc.temp = function(y) {c(y,y[4]*y[5],1);};   # y -> h(y)=(y1,y2,y3,y4,y5,y4*y5,1)
link.temp="logit"
x.temp = matrix(data=c(25.00000,1,-1,1,-1,25.00000,1,1,1,-1,32.06741,-1,1,-1,1,40.85698,
-1,1,1,-1,28.86602,-1,1,-1,-1,29.21486,-1,-1,1,1,25.00000,1,1,1,1, 25.00000,1,1,-1,-1),
ncol=5, byrow=TRUE)
b.temp = c(0.3197169, 1.9740922, -0.1191797, -0.2518067, 0.1970956, 0.3981632, -7.6648090)
X.mat = matrix(,nrow=8, ncol=7)
w.vec = rep(NA,8)
for(i in 1:8) {
htemp=Xw_maineffects_self(x=x.temp[i,], b=b.temp, link=link.temp, h.func=hfunc.temp);
X.mat[i,]=htemp$X;
w.vec[i]=htemp$w;
};
liftoneDoptimal_log_GLM_func(X=X.mat, w=w.vec, reltol=1e-5, maxit=500,
random=TRUE, nram=3, p00=NULL)
```

---

liftoneDoptimal_MLM_func

function of liftone for multinomial logit model

Description

function of liftone for multinomial logit model

Usage

```
liftoneDoptimal_MLM_func(
  m,
  p,
  Xi,
  J,
```

```
  thetavec,
  link = "continuation",
  reltol = 1e-05,
  maxit = 500,
  p00 = NULL,
  random = FALSE,
  nram = 3
)
```

Arguments

m	number of design points
p	number of parameters in the multinomial logit model
Xi	model matrix
J	number of response levels in the multinomial logit model
thetavec	model parameter
link	multinomial logit model link function name "baseline", "cumulative", "adjacent", or"continuation", default to be "continuation"
reltol	relative tolerance for convergence, default to 1e-5
maxit	the number of maximum iteration, default to 500
p00	specified initial approximate allocation, default to NULL, if NULL, will generate a random initial approximate allocation
random	TRUE or FALSE, if TRUE then the function will run with additional "nram" number of initial allocation p00, default to be TRUE
nram	when random == TRUE, the function will generate nram number of initial points, default is 3

Value

p reported D-optimal approximate allocation

p0 the initial approximate allocation that derived the reported D-optimal design

Maximum the maximum of the determinant of the Fisher information matrix

Convergence TRUE or FALSE, whether the algorithm converges

itmax, maximum iterations

Examples

```
m=5
p=10
J=5
factor_x = matrix(c(-1,-25,199.96,-150,-100,16,1,23.14,196.35,0,-100,
16,1,-24.99,199.99,-150,0,16,-1,25,-200,0,0,16,-1,-25,-200,-150,0,16),ncol=6,byrow=TRUE)
Xi=rep(0,J*p*m); dim(Xi)=c(J,p,m)
hfunc.temp = function(y){
if(length(y) != 6){stop("Input should have length 6");}
```

```
 model.mat = matrix(NA, nrow=5, ncol=10, byrow=TRUE)
 model.mat[5,]=0
 model.mat[1:4,1:4] = diag(4)
 model.mat[1:4, 5] =((-1)*y[6])
 model.mat[1:4, 6:10] = matrix(((-1)*y[1:5]), nrow=4, ncol=5, byrow=TRUE)
 return(model.mat)
 }
for(i in 1:m) {
Xi[,,i]=hfunc.temp(factor_x[i,])
}
thetavec=c(-1.77994301, -0.05287782,  1.86852211, 2.76330779, -0.94437464, 0.18504420,
-0.01638597, -0.03543202, -0.07060306, 0.10347917)
liftoneDoptimal_MLM_func(m=m,p=p,Xi=Xi,J=J,thetavec=thetavec,
link="cumulative",p00=rep(1/5,5), random=FALSE)
```

`MLM_Exact_Design` *Approximation to exact design algorithm for multinomial logit model*

Description

Approximation to exact design algorithm for multinomial logit model

Usage

```
MLM_Exact_Design(
  J,
  k.continuous,
  design_x,
  design_p,
  det.design,
  p,
  ForLion,
  bvec,
  bvec_matrix,
  rel.diff,
  L,
  N,
  hfunc,
  link
)
```

Arguments

`J`	number of response levels in the multinomial logit model
`k.continuous`	number of continuous factors
`design_x`	the matrix with rows indicating design point which we got from the approximate design

design_p	D-optimal approximate allocation
det.design	the determinant of D-optimal approximate allocation
p	number of parameters
ForLion	TRUE or FALSE, TRUE: this approximate design was generated by ForLion algorithm, FALSE: this approximate was generated by EW ForLion algorithm
bvec	If ForLion==TRUE assumed parameter values of model parameters beta, same length of h(y)
bvec_matrix	If ForLion==FALSE the matrix of the bootstrap parameter values of beta
rel.diff	points with distance less than that will be merged
L	rounding factor
N	total number of observations
hfunc	function for obtaining model matrix h(y) for given design point y, y has to follow the same order as n.factor
link	link function, default "continuation", other choices "baseline", "cumulative", and "adjacent"

Value

x.design matrix with rows indicating design point

ni.design EW D-optimal or D-optimal exact allocation

rel.efficiency relative efficiency of the Exact and Approximate Designs

Examples

```
J=3
k.continuous=1
design_x<-c(0.0000,103.5451,149.2355)
design_p<-c(0.2027, 0.3981, 0.3992)
det.design=54016609
p=5
theta = c(-1.935, -0.02642, 0.0003174, -9.159, 0.06386)
hfunc.temp = function(y){
   matrix(data=c(1,y,y*y,0,0,0,0,0,1,y,0,0,0,0,0), nrow=3,
            ncol=5, byrow=TRUE)
}
link.temp = "continuation"
MLM_Exact_Design(J=J, k.continuous=k.continuous,design_x=design_x,
design_p=design_p,det.design=det.design,p=p,ForLion=TRUE,bvec=theta,
rel.diff=1,L=0.5,N=1000,hfunc=hfunc.temp,link=link.temp)
```

nu1_cauchit_self *Function to calculate first derivative of nu function given eta for cauchit link*

Description

Function to calculate first derivative of nu function given eta for cauchit link

Usage

```
nu1_cauchit_self(x)
```

Arguments

x	vector of eta, eta=X*beta

Value

the first derivative of nu function given eta for cauchit link

Examples

```
eta = c(1,2,3,4)
nu1_cauchit_self(eta)
```

nu1_identity_self *function to calculate first derivative of nu function given eta for identity link*

Description

function to calculate first derivative of nu function given eta for identity link

Usage

```
nu1_identity_self(x)
```

Arguments

x	vector of eta, eta=X*beta

Value

the first derivative of nu function given eta for identity link

Examples

```
eta = c(1,2,3,4)
nu1_identity_self(eta)
```

---

nu1_logit_self *function to calculate the first derivative of nu function given eta for logit link*

Description

function to calculate the first derivative of nu function given eta for logit link

Usage

```
nu1_logit_self(x)
```

Arguments

x	vector of eta, eta=X*beta

Value

the first derivative of nu function given eta for logit link

Examples

```
eta = c(1,2,3,4)
nu1_logit_self(eta)
```

---

nu1_loglog_self *function to calculate the first derivative of nu function given eta for log-log link*

Description

function to calculate the first derivative of nu function given eta for log-log link

Usage

```
nu1_loglog_self(x)
```

Arguments

x	vector of eta, eta=X*beta

Value

the first derivative of nu function given eta for log-log link

Examples

```
eta = c(1,2,3,4)
nu1_loglog_self(eta)
```

---

nu1_log_self *function to calculate first derivative of nu function given eta for log link*

Description

function to calculate first derivative of nu function given eta for log link

Usage

```
nu1_log_self(x)
```

Arguments

x	vector of eta, eta=X*beta

Value

the first derivative of nu function given eta for log link

Examples

```
eta = c(1,2,3,4)
nu1_log_self(eta)
```

---

nu1_probit_self *function to calculate the first derivative of nu function given eta for probit link*

Description

function to calculate the first derivative of nu function given eta for probit link

Usage

```
nu1_probit_self(x)
```

Arguments

x	vector of eta, eta=X*beta

Value

the first derivative of nu function for probit link

Examples

```
eta = c(1,2,3,4)
nu1_probit_self(eta)
```

---

nu2_cauchit_self *function to calculate the second derivative of nu function given eta for cauchit link*

Description

function to calculate the second derivative of nu function given eta for cauchit link

Usage

```
nu2_cauchit_self(x)
```

Arguments

x	vector of eta, eta=X*beta

Value

the second derivative of nu function for cauchit link

Examples

```
eta = c(1,2,3,4)
nu2_cauchit_self(eta)
```

`nu2_identity_self` *function to calculate the second derivative of nu function given eta for identity link*

Description

function to calculate the second derivative of nu function given eta for identity link

Usage

```
nu2_identity_self(x)
```

Arguments

x	vector of eta, eta=X*beta

Value

the second derivative of nu function for identity link

Examples

```
eta = c(1,2,3,4)
nu2_identity_self(eta)
```

`nu2_logit_self` *function to calculate the second derivative of nu function given eta for logit link*

Description

function to calculate the second derivative of nu function given eta for logit link

Usage

```
nu2_logit_self(x)
```

Arguments

x	vector of eta, eta=X*beta

Value

the second derivative of nu function for logit link

Examples

```
eta = c(1,2,3,4)
nu2_logit_self(eta)
```

`nu2_loglog_self` *function to calculate the second derivative of nu function given eta for loglog link*

Description

function to calculate the second derivative of nu function given eta for loglog link

Usage

```
nu2_loglog_self(x)
```

Arguments

x	vector of eta, eta=X*beta

Value

the second derivative of nu function for loglog link

Examples

```
eta = c(1,2,3,4)
nu2_loglog_self(eta)
```

`nu2_log_self` *function to calculate the second derivative of nu function given eta for log link*

Description

function to calculate the second derivative of nu function given eta for log link

Usage

```
nu2_log_self(x)
```

Arguments

x	vector of eta, eta=X*beta

Value

the second derivative of nu function for log link

Examples

```
eta = c(1,2,3,4)
nu2_log_self(eta)
```

`nu2_probit_self` *function to calculate the second derivative of nu function given eta for probit link*

Description

function to calculate the second derivative of nu function given eta for probit link

Usage

```
nu2_probit_self(x)
```

Arguments

x	vector of eta, eta=X*beta

Value

the second derivative of nu function for probit link

Examples

```
eta = c(1,2,3,4)
nu2_probit_self(eta)
```

`nu_cauchit_self` *function to calculate w = nu(eta) given eta for cauchit link*

Description

function to calculate w = nu(eta) given eta for cauchit link

Usage

```
nu_cauchit_self(x)
```

Arguments

x a list of eta - X*beta

Value

diagonal element of W matrix which is nu(eta)

Examples

```
eta = c(1,2,3,4)
nu_cauchit_self(eta)
```

---

nu_identity_self *Function to calculate w = nu(eta) given eta for identity link*

Description

Function to calculate w = nu(eta) given eta for identity link

Usage

```
nu_identity_self(x)
```

Arguments

x Numeric vector of eta, eta = X*beta.

Value

A numeric vector representing the diagonal elements of the W matrix (nu(eta)).

Examples

```
eta = c(1,2,3,4)
nu_identity_self(eta)
```

`nu_logit_self` *function to calculate w = nu(eta) given eta for logit link*

Description

function to calculate w = nu(eta) given eta for logit link

Usage

```
nu_logit_self(x)
```

Arguments

x	vector of eta, eta=X*beta

Value

diagonal element of W matrix which is nu(eta)

Examples

```
eta = c(1,2,3,4)
nu_logit_self(eta)
```

`nu_loglog_self` *function to calculate w = nu(eta) given eta for loglog link*

Description

function to calculate w = nu(eta) given eta for loglog link

Usage

```
nu_loglog_self(x)
```

Arguments

x	vector of eta, eta=X*beta

Value

diagonal element of W matrix which is nu(eta)

Examples

```
eta = c(1,2,3,4)
nu_loglog_self(eta)
```

`nu_log_self` *Function to calculate w = nu(eta) given eta for log link*

Description

Function to calculate w = nu(eta) given eta for log link

Usage

```
nu_log_self(x)
```

Arguments

x	Numeric vector of eta, eta = X*beta.

Value

A numeric vector representing the diagonal elements of the W matrix (nu(eta)).

Examples

```
eta = c(1,2,3,4)
nu_log_self(eta)
```

`nu_probit_self` *function to calculate w = nu(eta) given eta for probit link*

Description

function to calculate w = nu(eta) given eta for probit link

Usage

```
nu_probit_self(x)
```

Arguments

x	vector of eta, eta=X*beta

Value

diagonal element of W matrix which is nu(eta)

Examples

```
eta = c(1,2,3,4)
nu_probit_self(eta)
```

---

print.design_output *Print Method for Design Output from ForLion Algorithm*

Description

Custom print method for a list containing design information.

Usage

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

Arguments

x	An object of class ‘design_output‘.
...	Additional arguments (ignored).

Value

Invisibly returns ‘x‘.

---

print.list_output *Print Method for list_output Objects*

Description

Custom print method for objects of class ‘list_output‘.

Usage

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

Arguments

`x`	An object of class ‘list_output‘.
`...`	Additional arguments (ignored).

Value

Invisibly returns ‘x‘ (the input object).

`svd_inverse` *SVD Inverse Of A Square Matrix This function returns the inverse of a matrix using singular value decomposition. If the matrix is a square matrix, this should be equivalent to using the solve function. If the matrix is not a square matrix, then the result is the Moore-Penrose pseudo inverse.*

Description

SVD Inverse Of A Square Matrix This function returns the inverse of a matrix using singular value decomposition. If the matrix is a square matrix, this should be equivalent to using the solve function. If the matrix is not a square matrix, then the result is the Moore-Penrose pseudo inverse.

Usage

```
svd_inverse(x)
```

Arguments

`x`	the matrix for calculation of inverse

Value

the inverse of the matrix x

Examples

```
x = diag(4)
svd_inverse(x)
```

xmat_discrete_self *Generate GLM random initial designs within ForLion algorithm*

Description

Generate GLM random initial designs within ForLion algorithm

Usage

```
xmat_discrete_self(xlist, rowmax = NULL)
```

Arguments

xlist	a list of factor levels within ForLion algorithm, for example, a binary factor might be c(-1,1), a continuous factor within range of (25,45) will be c(25, 45).
rowmax	maximum number of rows of the design matrix

Value

design matrix of all possible combinations of discrete factors levels with min and max of the continuous factors.

Examples

```
#define list of factor levels for one continuous factor, four binary factors
factor.level.temp = list(c(25,45),c(-1,1),c(-1,1),c(-1,1),c(-1,1))
xmat_discrete_self(xlist = factor.level.temp)
```

Xw_maineffects_self *function for calculating X=h(x) and w=nu(beta^T h(x)) given a design point x = (x1,...,xd)^T*

Description

function for calculating X=h(x) and w=nu(beta^T h(x)) given a design point x = (x1,...,xd)^T

Usage

```
Xw_maineffects_self(x, b, link = "logit", h.func = NULL)
```

Arguments

`x`	x=(x1,...,xd) – design point/experimental setting
`b`	b=(b1,...,bp) – assumed parameter values
`link`	link = "logit" – link function, default: "logit", other links: "probit", "cloglog", "loglog", "cauchit", "log"
`h.func`	function h(x)=(h1(x),...,hp(x)), default (1,x1,...,xd)

Value

X=h(x)=(h1(x),...,hp(x)) – a row for design matrix

w – nu(b^t h(x))

link – link function applied

Examples

```
# y -> h(y)=(y1,y2,y3,y4,y5,y4*y5,1) in hfunc
hfunc.temp = function(y) {c(y,y[4]*y[5],1);};
link.temp="logit"
x.temp = c(25,1,1,1,1)
b.temp = c(-7.533386, 1.746778, -0.1937022, -0.09704664, 0.1077859, 0.2729715, 0.4293171)
Xw_maineffects_self(x.temp, b.temp, link=link.temp, h.func=hfunc.temp)
```

Index