Package ‘minimaxdesign’

October 13, 2022

Type Package

Title Minimax and Minimax Projection Designs

Version 0.1.5

Author Simon Mak

Maintainer Simon Mak <smak6@gatech.edu>

Description Provides two main functions, minimax() and miniMaxPro(), for computing minimax
and minimax projection designs using the minimax clustering algorithm in Mak and
include the unit hypercube ("hypercube"), the unit simplex ("simplex"), the unit ball
("ball"), as well as user-defined constraints on the unit hypercube ("custom"). Minimax
designs can also be computed on user-provided images using the function minimax.map().
Design quality can be assessed using the function mMdist(), which computes the minimax
(fill) distance of a design.

License GPL (>= 2)

Imports Rcpp (>= 0.12.4), randtoolbox, DiceDesign, MaxPro, doParallel,
doSNOW, gtools, nloptr, foreach, jpeg, gmp, conf.design, pdist,
DoE.base, FrF2, geometry

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 6.1.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2021-07-12 22:10:02 UTC

R topics documented:

minimaxdesign-package .................................................. 2
CtoA ................................................................. 3
CtoB ................................................................. 4
minimax .............................................................. 5
minimax.map .......................................................... 6
minimax.seq .......................................................... 8
miniMaxPro ........................................................... 10
mMdist ............................................................... 11
minimaxdesign-package

An R package for computing Minimax and Minimax Projection Designs

Description

The 'minimaxdesign' package provides functions for generating minimax designs and minimax projection designs.

Details

Package: minimaxdesign
Type: Package
Version: 0.1.4
Date: 2019-05-03
License: GPL (>= 2)

Provides two main functions, minimax() and miniMaxPro(), for computing minimax and minimax projection designs using the minimax clustering algorithm in Mak and Joseph (2018) <DOI:10.1080/10618600.2017.1302881>. Current design region options include the unit hypercube ("hypercube"), the unit simplex ("simplex"), the unit ball ("ball"), as well as user-defined constraints on the unit hypercube ("custom"). Minimax designs can also be computed on user-provided images using the function minimax.map(). Design quality can be assessed using the function mMdist(), which computes the minimax (fill) distance of a design.

Author(s)

Simon Mak
Maintainer: Simon Mak <smak6@gatech.edu>

References


Examples

```r
## Not run:
#20-point minimax design on the hypercube [0,1]^2
D <- minimax(N=20,p=2)
plot(NULL,xlim=c(0,1),ylim=c(0,1),xlab="x1",ylab="x2") #set up plot
polygon(c(0,0,1,1),c(0,1,1,0),col="gray") #design space
points(D,xlim=c(0,1),ylim=c(0,1),xlab="x1",ylab="x2",pch=16) #design points
mM <- mMdist(D)
```
CtoA

Inverse Rosenblatt transformation from the unit hypercube to the unit simplex

Description

CtoA maps points on the unit hypercube in \( p \)-dimensions, \( C_p = [0, 1]^p \), to points on the unit simplex in \( p \)-dimensions, \( A_p \).

Usage

\[
\text{CtoA}(D, \text{by} = \text{ifelse(ncol(D)>2,1e-3,-1)}, \text{num}\_\text{proc}=\text{parallel::detectCores()})
\]

Arguments

- **D**: An \( N \)-by-\( p \) matrix representing \( N \) points in \( p \)-dimensions.
- **by**: Step-size used for approximating integrals.
- **num\_proc**: Number of processors to use.

Value

An \( N \)-by-\( p \) matrix for the inverse-Rosenblatt mapping of \( D \) onto the unit simplex \( A_p \).
Examples

## Not run:
# Map the first 100 points of the Sobol sequence in 3D
# onto the unit simplex in 3D
library(randtoolbox)
des <- sobol(100,3)
des_simp <- CtoA(des)
pairs(des_simp,xlim=c(0,1),ylim=c(0,1),pch=16)

## End(Not run)

---

CtoB  
Inverse Rosenblatt transformation from the unit hypercube to the unit ball

Description

CtoB maps points on the unit hypercube in \( p \)-dimensions, \( C_p = [0, 1]^p \), to points on the unit simplex in \( p \)-dimensions, \( B_p \).

Usage

CtoB(D, by=ifelse(ncol(D)>2,1e-3,-1), num_proc=parallel:::detectCores())

Arguments

- **D**: An \( N \)-by-\( p \) matrix representing \( N \) points in \( p \)-dimensions.
- **by**: Step-size used for approximating integrals.
- **num_proc**: Number of processors to use.

Value

An \( N \)-by-\( p \) matrix for the inverse-Rosenblatt mapping of \( D \) onto the unit ball \( B_p \).

Examples

## Not run:
# Map the first 100 points of the Sobol sequence in 3D
# onto the unit ball in 3D
library(randtoolbox)
des <- sobol(100,3)
des_ball <- CtoB(des)
pairs(des_ball,xlim=c(-1,1),ylim=c(-1,1),pch=16)

## End(Not run)
minimax

Compute minimax designs using clustering on constrained design regions

Description

minimax computes minimax designs using the minimax clustering algorithm in Mak and Joseph (2018). Current design region options include the unit hypercube ("hypercube"), the unit simplex ("simplex"), the unit ball ("ball"), the inequality-constrained unit hypercube ("ineq"), and custom clustering points ("custom").

Usage

minimax(N,p,q=10,region="hypercube",ini=NA,const=NA,clust_pts=NA,
params_pso=list(w=0.72,c1=1.49,c2=1.49),
npart=5,nclust=1e5,neval=nclust,
max_pso=50,max_pp=100,max_inn=1e4,jit=0.1/sqrt(N))

Arguments

N Number of design points.
p Dimension of design region.
q Power parameter for approximating the minimax criterion (see paper for details). Larger values of q give a better approximation, but may cause numerical instability.
region String for desired design region. Current options include "hypercube", "simplex", "ball", "ineq" and "custom".
ini String for initial design. Current choices include NA (automatic choice), "sobol" (Sobol' sequence), and "ff" (fractional factorial).
const Function for desired constraints (inequalities) on design space. See examples for implementation.
clust_pts Custom clustering points (used only if region == "custom").
params_pso Particle swarm optimization parameters (particle momentum (w), local-best velocity (c1) and global-best velocity (c2)).
npart Number of particles for particle swarm optimization.
nclust,neval Number of sample points for minimax clustering and post-processing.
max_pso,max_pp,max_inn Maximum number of iterations for minimax clustering, post-processing and inner optimization.
jit Jitter radius for post-processing.

Value

An N-by-p matrix for the minimax design.
Examples

```r
## Not run:
#20-point minimax design on the hypercube [0,1]^2
D <- minimax(N=20,p=2)
plot(NULL,xlim=c(0,1),ylim=c(0,1),xlab="x1",ylab="x2") # set up plot
polygon(c(0,0,1,1),c(0,1,1,0),col="gray") # design space
points(D,xlim=c(0,1),ylim=c(0,1),xlab="x1",ylab="x2",pch=16) # design points
mM <- mMdist(D)
mM$dist # minimax (fill) distance
lines(rbind(mM$far.pt,mM$far.despt),col="red",lty=2,lwd=2) # plot farthest point

#20-point minimax design on design space [0,1]^2 constrained by given inequalities
ineqs <- function(xx){ # user-defined inequalities
  bool.vec <- rep(TRUE,2)
  return(all(bool.vec))
}
D <- minimax(N=20,p=2,region="ineq",const=ineqs)
plot(NULL,xlim=c(0,1),ylim=c(0,1),xlab="x1",ylab="x2") # set up plot
polygon(c(0,2/3,1),c(0,2/3,0),col="gray") # design space
points(D,pch=16) # design points
mM <- mMdist(D,region="custom",const=ineqs)
mM$dist # minimax (fill) distance
lines(rbind(mM$far.pt,mM$far.despt),col="red",lty=2,lwd=2) # plot farthest point

#20-point minimax design on custom clustering points
p <- 1000
NN <- 10000
clust_pts <- matrix(runif(NN*p),nrow=NN,ncol=p)
D <- minimax(N=20,p=2,region="custom",clust_pts=clust_pts)
plot(NULL,xlim=c(0,1),ylim=c(0,1),xlab="x1",ylab="x2") # set up plot
points(clust_pts,xlim=c(0,1),ylim=c(0,1),col="gray",pch=4,cex=0.5) # clustering points
points(D,xlim=c(0,1),ylim=c(0,1),xlab="x1",ylab="x2",pch=16) # design points
mM <- mMdist(D,eval_pts=clust_pts)
mM$dist # minimax (fill) distance
lines(rbind(mM$far.pt,mM$far.despt),col="red",lty=2,lwd=2) # plot farthest point

## End(Not run)
```

minimax.map

Compute minimax designs using clustering on a user-provided image

Description

minimax.map computes minimax designs on a user-provided binary (0-1) image, using the minimax clustering algorithm in Mak and Joseph (2018).


**Usage**

```r
minimax.map(N, img, p=2, q=10,
    params_pso=list(w=0.72, c1=1.49, c2=1.49),
    npart=5, nclus=nclus, neval=nclus,
    itmax_pso=50, itmax_pp=100, itmax_inn=1e4, jit=0.1/sqrt(N))
```

**Arguments**

- **N** Number of design points.
- **img** A binary 0-1 matrix, with 1 indicating the desired design region.
- **p** Dimension of design region.
- **q** Power parameter for approximating the minimax criterion (see paper for details). Larger values of q give a better approximation, but may cause numerical instability.
- **params_pso** Particle swarm optimization parameters (particle momentum (w), local-best velocity (c1) and global-best velocity (c2)).
- **npart** Number of particles for particle swarm optimization.
- **nclus, neval** Number of sample points for minimax clustering and post-processing.
- **itmax_pso, itmax_pp, itmax_inn** Maximum number of iterations for minimax clustering, post-processing and inner optimization.
- **jit** Jitter radius for post-processing.

**Value**

An N-by-p matrix for the minimax design.

**Examples**

```r
## Not run:
#20-point minimax design on the hypercube [0,1]^2
library(jpeg)
n <- 25
img <- readJPEG(system.file("img", "gamap.jpg", package="minimaxdesign"))[,1]
image(t(img)[,nrow(img):1],col=gray.colors(12,start=0.6),main="Georgia")
img <- t(img)[,nrow(img):1] #Invert image due to reading distortion
des <- minimax.map(n, img)
points(des,pch=16)

## End(Not run)
```
minimax.seq  
Compute sequential minimax designs using clustering on constrained design regions

Description

minimax.seq computes sequential minimax designs using the minimax clustering algorithm in Mak and Joseph (2018). Current design region options include the unit hypercube ("hypercube"), the unit simplex ("simplex"), the unit ball ("ball"), a user-defined active subspace ("subspace"), and user-defined constraints on the unit hypercube ("custom").

Usage

```R
minimax.seq(D,Nseq,q=10,region="hypercube",const=NA,subsp=NA,wt.subsp=FALSE,
params_pso=list(w=0.72,c1=1.49,c2=1.49),
npart=5,nclust=1e5,neval=nclust,
itmax_pso=50,itmax_pp=100,itmax_inn=1e4,jit=0.1/sqrt(Nseq+nrow(D)))
```

Arguments

- **D**  
  Initial design.
- **Nseq**  
  Number of sequential design points.
- **q**  
  Power parameter for approximating the minimax criterion (see paper for details). Larger values of q give a better approximation, but may cause numerical instability.
- **region**  
  String for desired design region. Current options include "hypercube", "simplex", "ball", "subspace", and "custom".
- **const**  
  Function for desired constraints (inequalities) on design space. See examples for implementation.
- **subsp**  
  A p' x p matrix for the desired active subspace, where p' and p are the dimensions of the full space and active subspace. Only used when region is "subspace".
- **wt.subsp**  
  TRUE if a weighted design is desired over the active subspace, FALSE if an unweighted (uniform) is desired. Only used when region is "subspace".
- **params_pso**  
  Particle swarm optimization parameters (particle momentum (w), local-best velocity (c1) and global-best velocity (c2)).
- **npart**  
  Number of particles for particle swarm optimization.
- **nclust,neval**  
  Number of sample points for minimax clustering and post-processing.
- **itmax_pso, itmax_pp, itmax_inn**  
  Maximum number of iterations for minimax clustering, post-processing and inner optimization.
- **jit**  
  Jitter radius for post-processing.
Value

An Nseq-by-p matrix for the minimax design.

Examples

```r
## Not run:
# 20+20-point sequential minimax design on the hypercube [0,1]^2
Nini <- 20
Nseq <- 20
Dini <- matrix(runif(Nini*2), ncol=2) # random initial design
Dseq <- minimax.seq(Dini, Nseq) # sequential minimax design

plot(NULL, xlim=c(0,1), ylim=c(0,1), xlab="x1", ylab="x2") # set up plot
polygon(c(0,0,1,1), c(0,0,1,0), col="gray") # design space
points(Dini, xlim=c(0,1), ylim=c(0,1), xlab="x1", ylab="x2", pch=16) # original design
points(Dseq, xlim=c(0,1), ylim=c(0,1), xlab="x1", ylab="x2", pch=16, col="blue") # sequential design

# 20+20-point sequential minimax design on active subspace
set.seed(1)
library(rstiefel)
library(geometry)
p.full <- 10 # full dimension
p.as <- 2 # active subspace dimension
Nini <- 20
Nseq <- 20
Dini <- matrix(runif(Nini*p.full), ncol=p.full) # random initial design
A <- rustiefel(p.full, p.as) # random active subspace
Dini <- t(t(A)%*%t(Dini)) # initial design projected on active subspace
Dseq <- minimax.seq(Dini, Nseq=20, region="subspace", subsp=A, wt.subsp=TRUE)

# Compute active subspace polygon
Nplot <- 1e5
plot.pts <- matrix(runif(Nplot*p.full), ncol=p.full)
plot.pts <- t(t(A)%*%t(plot.pts))
hull <- convhulln(plot.pts)
for (i in 2:nrow(hull)) {
  newidx <- setdiff(c(which(hull[,2]==hull[i-1,2]), which(hull[,1]==hull[i-1,2])), i-1)
  if (length(newidx) > 0)
    hull[newidx,1] <- hull[i-1,2]
    hull[newidx,2] <- hull[i,1]
    hull[i,] <- tmp
  else
    hull[i,] <- rev(hull[newidx,])
}

# Visualize
plot(NULL, xlim=c(-2,2), ylim=c(-2,2), xlab="x1", ylab="x2") # set up plot
```
miniMaxPro

Compute minimax projection designs using clustering on constrained design regions

Description

miniMaxPro.

Usage

miniMaxPro(N, p, mMdes = NA, mMtol = 1e-3*p, neval = 1e5, itmax_refine = 100, ...)

Arguments

N
   Number of design points.
p
   Dimension of design region.
mMdes
   Minimax design from minimax().
mMtol
   Tolerance for fill distance increase (actual increase may be slightly higher).
neval
   Number of sample points for refinement.
itmax_refine
   Maximum number of iterations for refinement.
... Parameters for minimax().

Value

A list with two objects:

minimax An N-by-p matrix for the minimax design.
miniMaxPro An N-by-p matrix for the minimax projection design.

Examples

## Not run:
#30-point miniMaxPro design on the hypercube [0,1]^6
D <- minimax(N=30,p=6)
D <- miniMaxPro(N=30,p=6,mMdes=D)
mMdist(D$minimax)$dist
mMdist(D$miniMaxPro)$dist  #slightly higher fill distance
pairs(D$minimax,xlim=c(0,1),ylim=c(0,1),pch=16)
pairs(D$miniMaxPro,xlim=c(0,1),ylim=c(0,1),pch=16)  #... but better projections

## End(Not run)
Computes the minimax (fill) distance of a design

Description

`mMdist` computes the minimax (fill) distance of a design (see Mak and Joseph (2018) for definition). Current design region options include the unit hypercube ("hypercube"), the unit simplex ("simplex"), the unit ball ("ball"), and user-defined constraints on the unit hypercube ("custom").

Usage

```r
mMdist(D, neval = 1e5, method = "lattice", region = "hypercube", const = NA, eval_pts = NA)
```

Arguments

- `D`: An N-by-p design matrix.
- `neval`: Number of sample points for approximation.
- `method`: Method for generating approximation points. Current options include "lattice" (lattice rule) and "sobol" (Sobol' sequence).
- `region`: String for desired design region. Current options include "hypercube", "simplex", "ball", and "custom".
- `const`: Function for desired constraints (inequalities) on design space.
- `eval_pts`: Custom clustering points (used only if `region` == "custom").

Value

A list with two objects:

- `dist`: Minimax (fill) distance.
- `dist.vec`: Minimax (fill) distance for each design point.
- `far.pt`: A p-vector for the farthest point in design region to design.
- `far.despt`: A p-vector for the design point closest to `far.pt`.

Examples

```r
# Not run:
# 20-point minimax design on the hypercube [0,1]^2
D <- minimax(N=20, p=2)
plot(NULL, xlim=c(0,1), ylim=c(0,1), xlab="x1", ylab="x2") #set up plot
polygon(c(0,0,1,1),c(0,1,1,0),col="gray") #design space
points(D, xlim=c(0,1), ylim=c(0,1), xlab="x1", ylab="x2", pch=16) #design points
mM <- mMdist(D)
mM$dist #minimax (fill) distance
lines(rbind(mM$far.pt, mM$far.despt), col="red", lty=2, lwd=2) #plot farthest point

# End(Not run)
```
Index

* package
  minimaxdesign-package, 2

CtoA, 3
CtoB, 4

minimax, 5
minimax.map, 6
minimax.seq, 8
minimaxdesign (minimaxdesign-package), 2
minimaxdesign-package, 2
miniMaxPro, 10
mMdist, 11