Package 'minimaxdesign'

December 11, 2017

Type Package

Title Minimax and Minimax Projection Designs

Version 0.1.3

Author Simon Mak

Maintainer Simon Mak <smak6@gatech.edu>

Description Provides two main functions: mMcPSO() and miniMaxPro(), which generates minimax designs and minimax projection designs using a hybrid clustering - particle swarm optimization (PSO) algorithm. These designs can be used in a variety of settings, e.g., as space-filling designs for computer experiments or sensor allocation designs. A detailed description of the two designs and the employed algorithms can be found in Mak and Joseph (2017) <DOI:10.1080/10618600.2017.1302881>.

License GPL (>= 2)

LazyData TRUE

Imports Rcpp (>= 0.12.4), randtoolbox, DiceDesign, MaxPro, doParallel, doSNOW, gtools, nloptr, foreach, jpeg

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 6.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2017-12-11 08:12:12 UTC

R topics documented:

`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:	1.0
Date:	2017-04-13
License:	GPL (>= 2)

The 'minimaxdesign' package provides two main functions: `mMcPSO()` and `miniMaxPro()`, which generates minimax designs and minimax projection designs using a hybrid clustering - particle swarm optimization (PSO) algorithm. These designs can be used in a variety of settings, e.g., as space-filling designs for computer experiments or sensor allocation designs. A detailed description of the two designs and the employed algorithms can be found in Mak and Joseph (2017).

Author(s)

Simon Mak

Maintainer: Simon Mak <smak6@gatech.edu>

References

Mak, S., & Joseph, V.R. (2017). Minimax and minimax projection designs using clustering. *Journal of Computational and Graphical Statistics.* To appear.

Examples

```
## Not run:

#Generate and plot a minimax design with 7 points on the unit hypercube [0,1]^2
mMdes <- mMcPSO(N=7,p=2)

#Generate a miniMaxPro design of 20 points on the unit hypercube [0,1]^8
library(foreach)
mMPdes <- miniMaxPro(N=20,p=8)$miniMaxPro
```

```
## End(Not run)
```

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

```
CtoA(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 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)
```

miniMaxPro — *Compute minimax projection designs using clustering*

Description

miniMaxPro is the main function for generating minimax projection designs on a desired design space X. A formal exposition of minimax projection designs and the employed algorithm can be found in Mak and Joseph (2017). Currently only available when X is the unit hypercube $[0,1]^p$.

Usage

```
miniMaxPro(N,p,mMdes=NA, mM_tol=1e-3*p,
            refine_num=1e5, refine_pts=NA, refine_itmax=100, ...)
```

Arguments

N	Number of design points desired.
p	Dimension of design desired.
mMdes	Minimax design from mMcPSO().
refine_num	Number of points used to estimate X in the refinement step.
refine_pts	User-specified representative points for the refinement step. If NA, the algorithm generates these points.
refine_itmax	Maximum iterations for the refinement step.
mM_tol	Upper bound for minimax distance increase (since the refinement step may increase this distance slightly).
...	Parameter settings for mMcPSO.

Value

A list with two objects:

minimax	An N-by-p matrix for the minimax design from mMcPSO before projection refinement.
miniMaxPro	An N-by-p matrix for the minimax projection design.

Examples

```
## Not run:
#Generate a miniMaxPro design of 20 points on the unit hypercube [0,1]^8
des <- miniMaxPro(N=40,p=8,refine_itmax=100)
pairs(des$minimax,xlim=c(0,1),ylim=c(0,1),pch=16)
pairs(des$miniMaxPro,xlim=c(0,1),ylim=c(0,1),pch=16)

#Use the minimax design from mMc-PSO to warm start miniMaxPro at a new setting
mMdes <- mMcPSO(N=40,p=8)
newdes <- miniMaxPro(N=20,p=8,mMdes=mMdes,refine_itmax=100)
pairs(newdes$miniMaxPro,xlim=c(0,1),ylim=c(0,1),pch=16)


## End(Not run)
```

mMcPSO *Compute minimax designs using clustering*

Description

mMcPSO is the main function for generating minimax designs on a desired design space X using a hybrid clustering - particle swarm optimization (PSO) algorithm. Subfunctions for mMcPSO are written in C++ for speed. Users have the flexibility of adjusting a variety of algorithmic parameters, including particle swarm optimization (PSO) settings, termination conditions, number of approximating points, etc. A formal exposition of this algorithm can be found in Mak and Joseph (2017).

Usage

```
mMcPSO(N,p,q=10,region="uh",
        pso=list(w=0.72,c1=1.49,c2=1.49),
        part_num_pso=10,part_num_pp=5,
        point_num=1e5,eval_num=10*point_num,point=NA,eval_pts=NA,
        it_max_pso=200,it_max_pp=ifelse(region=="simp",0,50),it_max_inn=1e4,
        it_lim_pso=25,it_lim_pp=10,
        it_tol_pso=1e-4,it_tol_pp=1e-4,it_tol_inn=1e-4,
        regionby=ifelse(p>2,1e-3,-1),
        jit=ifelse(region=="simp",0,0.1/sqrt(N)),
        pp_flag=F)
```

Arguments

N	Number of design points desired.
p	Dimension of design desired.
q	The approximation constant used to estimate the minimax criterion; refer to paper for details. Larger values of q give a better approximation, but may cause numerical instability.
region	Option for non-hypercube design regions: Current choices include the unit hypercube "uh", the unit simplex "simp", and the unit ball "ball"
pso	PSO settings for particle momentum (w), local-best velocity (c1) and global-best velocity (c2).
part_num_pso,part_num_pp	Number of PSO particles for minimax clustering and post-processing.
point_num	Number of points used to estimate the design space X for minimax clustering.
eval_num	Number of points used to estimate the design space X for post-processing.
point,eval_pts	User-specified representative points for clustering and post-processing. If NA, the algorithm generates these points using low-discrepancy sequences.
it_max_pso,it_max_pp,it_max_inn	Maximum iterations of minimax clustering, post-processing and the inner-loop for computing C_q-centers.

`it_lim_pso,it_lim_pp,it_tol_pso,it_tol_pp,it_tol_inn`
: Algorithm terminates if the global-best objective does not improve by at least `it_tol` after `it_lim` iterations.

`regionby`
: Specifies step-size for approximating integrals in non-hypercube transformations.

`jit`
: Jitter radius for post-processing.

`pp_flag`
: Redundant; still in development.

Value

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

Examples

```
## Not run:
#Generate and plot a minimax design with 20 points on the unit hypercube [0,1]^2
desuh <- mMcPSO(N=20,p=2)
plot(desuh,xlim=c(0,1),ylim=c(0,1),pch=16)

#Generate and plot a minimax design with 20 points on the unit simplex A_2
# ... (CtoA provides the mapping from [0,1]^2 to A_2, see ?CtoA)
dessimp <- mMcPSO(N=20,p=2,region="simp")
plot(dessimp,xlim=c(0,1),ylim=c(0,1),pch=16)
abline(0,1)

#Generate and plot a minimax design with 20 points on the unit ball B_2
# ... (CtoB2 provides the mapping from [0,1]^2 to B_2, see ?CtoB2)
library(plotrix)
desball <- mMcPSO(N=20,p=2,region="ball")
plot(desball,xlim=c(-1,1),ylim=c(-1,1),pch=16)
draw.circle(0,0,1) #design boundaries

#Generate and plot a minimax design with 20 points on the unit hypercube [0,1]^4
desuh <- mMcPSO(N=20,p=4)
pairs(desuh,xlim=c(0,1),ylim=c(0,1),pch=16)

## End(Not run)
```

`mMcPSO_map` *Compute minimax designs using clustering on a user-provided map (provided as an image file).*

Description

`mMcPSO_map` generates minimax designs on a user-provided map (provided as an image file) using a hybrid clustering - particle swarm optimization (PSO) algorithm. This function uses the minimax clustering routine `mMcPSO` internally as a workhorse. A formal exposition of this algorithm can be found in Mak and Joseph (2017).

Usage

```
mMcPSO_map(N,img,p=2,q=10,
            pso=list(w=0.72,c1=1.49,c2=1.49),
            part_num_pso=10,part_num_pp=5,
            point_num=1e5,eval_num=10*point_num,point=NA,eval_pts=NA,
            it_max_pso=200,it_max_pp=50,it_max_inn=1e4,
            it_lim_pso=25,it_lim_pp=10,
            it_tol_pso=1e-4,it_tol_pp=1e-4,it_tol_inn=1e-4,
            jit=0.1/sqrt(N))
```

Arguments

N	Number of design points desired.
img	A 0-1 matrix corresponding to the image file of the map.
p	Dimension of design desired (default = 2).
q	The approximation constant used to estimate the minimax criterion; refer to paper for details. Larger values of q give a better approximation, but may cause numerical instability.
pso	PSO settings for particle momentum (w), local-best velocity (c1) and global-best velocity (c2).
part_num_pso,part_num_pp	Number of PSO particles for minimax clustering and post-processing.
point_num	Number of points used to estimate the design space X for minimax clustering.
eval_num	Number of points used to estimate the design space X for post-processing.
point,eval_pts	User-specified representative points for clustering and post-processing. If NA, the algorithm generates these points using low-discrepancy sequences.
it_max_pso,it_max_pp,it_max_inn	Maximum iterations of minimax clustering, post-processing and the inner-loop for computing C_q-centers.
it_lim_pso,it_lim_pp,it_tol_pso,it_tol_pp,it_tol_inn	Algorithm terminates if the global-best objective does not improve by at least it_tol after it_lim iterations.
jit	Jitter radius for post-processing.

Value

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

Examples

```
## Not run:
  #Generate and plot a minimax design with 20 points on the map of Georgia
  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 <- mMcPSO_map(n,img)
    points(des,pch=16)

## End(Not run)
```

Index