Package ‘OptimalDesign’

November 7, 2016

Version 0.2
Date 2016-11-03
Title Algorithms for D-, A-, and IV-Optimal Designs
Author Radoslav Harman, Lenka Filova
Maintainer Lenka Filova <OptimalDesignR@gmail.com>
Description Algorithms for D-, A- and IV-optimal designs of experiments. Some of the functions in this package require the 'gurobi' software and its accompanying R package. For their installation, please follow the instructions at <www.gurobi.com> and the file gurobi_inst.txt, respectively.
Enhances gurobi, slam, Matrix
License GPL-3
URL <http://www.iam.fmph.uniba.sk/design/ >
NeedsCompilation no
Repository CRAN
Date/Publication 2016-11-07 08:12:38

R topics documented:

OptimalDesign-package ........................................ 2
F.cube ......................................................... 2
F.simplex ...................................................... 3
od.AA .......................................................... 4
od.crit ......................................................... 7
od.infmat ...................................................... 9
od.IQP ......................................................... 10
od.KL .......................................................... 13
od.m1 .......................................................... 15
od.MISOCP ................................................... 18
od.plot ......................................................... 21
od.print ....................................................... 22
od.RC ......................................................... 24
od.RCs ......................................................... 27
od.SOCP ....................................................... 29
Description

Procedures for computing D-, A-, and IV-optimal approximate and exact designs of experiments on finite domains, for regression models with real-valued, uncorrelated observations.

Author(s)

Radoslav Harman, Lenka Filova

F.cube

Model matrix on a cube

Description

Creates the matrix of regressors for a model on a d-dimensional rectangular grid of points, i.e., for a factorial model with d quantitative factors.

Usage

F.cube(formula, lower, upper, n.levels)

Arguments

- **formula**: The formula of the model.
- **lower**: The real vector of length d determining the smallest values of factors.
- **upper**: The real vector of length d determining the largest values of factors.
- **n.levels**: The vector of length d consisting of the numbers greater than or equal to 2, determining the numbers of levels of each factor.

Details

The rules for creating the model formula are the same as in the `lm` function but: 1) the formula must not contain the dependent variable; 2) the d factors (independent variables) must be labeled x1, x2, ...

Value

The n times m matrix of regressors with the m-dimensional rows corresponding to the regressors of the n design points (n is equal to \(\text{prod}(n\text{.levels})\)).
F.simplex

Description

Creates the matrix of regressors for a model on a discretized (d-1)-dimensional probability simplex, i.e., for a mixture model with d mixture components.

Usage

F.simplex(formula, n.factors, n.levels)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>formula</td>
<td>The formula of the model.</td>
</tr>
<tr>
<td>n.factors</td>
<td>The integer number greater than or equal to 2, determining the number of factors.</td>
</tr>
<tr>
<td>n.levels</td>
<td>The integer number greater than or equal to 2, determining the numbers of levels of each factor (all factors have the same number of levels).</td>
</tr>
</tbody>
</table>

Details

The rules for creating the model formula are the same as in the lm function but: 1) the formula must not contain the dependent variable; 2) the d factors (independent variables) must be labeled x1, x2, ...
Value

The \( n \) times \( m \) matrix of regressors corresponding to \( m \) model parameters and \( n \) design points, where \( n \) is equal to \((n.\text{factors} + n.\text{levels} - 2)\) choose \((n.\text{factors} - 1)\).

Author(s)

Radoslav Harman, Lenka Filova

See Also

F.cube

Examples

```r
# The matrix of regressors for the Scheffe quadratic mixture model
# with 3 mixture components, each with levels \{0, 0.25, 0.5, 0.75, 1\}.
F.simplex(~x1 + x2 + x3 + I(x1*x2) + I(x1*x3) + I(x2*x3) - 1, 3, 5)

# The matrix of regressors for a "special cubic" model with 3 mixture
# components, each with levels \{0, 0.2, 0.4, 0.6, 0.8, 1\}.
F.simplex(~x1 + x2 + x3 + I(x1*x2) + I(x1*x3) + I(x2*x3) +
          I(x1*x2*x3) - 1, 3, 6)

# Note that one must be careful when choosing a model for a mixture
# experiment: Let us compute the matrix of regressors of the simple
# linear mixture model with 4 mixture components, each with levels
# \{0, 0.5, 1\}.
F1 <- F.simplex(~x1 + x2 + x3 + x4, 4, 3)

# The model has only 4 parameters and as many as 10 design points,
# but there is no design that guarantees estimability of the
# parameters. This can be shown by evaluating:
det(od.infmat(F1, rep(1, 10)))
```

Description

Computes an optimal approximate design under the standard (size) constraint using a mixture of methods.

Usage

```r
od.AA(F, N=1, crit="D", R=TRUE, w1=TRUE, alg="oom", lambda=0.9,
          tab=TRUE, graph=TRUE, eff=1e-9, t.max=120)
```
Arguments

\( F \)  The \( n \) times \( m \) matrix of real numbers. The rows of \( F \) represent the \( m \)-dimensional regressors corresponding to \( n \) design points. It is assumed that \( n \geq m \geq 2 \). Cf. \texttt{od.m1} for models with 1-dimensional regressors.

\( N \)  The required size of the design, i.e., the sum of the components of the resulting design. If \( N=1 \) the result will be in the standard form of a normalized (probability) measure representing proportions of trials in individual design points.

\( \text{crit} \)  The optimality criterion. Possible values are "D", "A", "IV".

\( R \)  The region of summation for the IV-optimality criterion. The argument \( R \) must be a subvector of \( 1:n \), or NULL. If \( R=NULL \), the procedure uses \( R=1:n \). Argument \( R \) is ignored if \( \text{crit}="D" \), or if \( \text{crit}="A" \).

\( w1 \)  The real vector of length \( n \) with non-negative components, representing the initial design. Before using, \( w1 \) is normalized. The information matrix of \( w1 \) must be nonsingular. The argument \( w1 \) can also be NULL; in that case the function generates a random initial design. Set all components of \( w1 \) to strictly positive numbers if you use multiplicative steps only.

\( \text{alg} \)  The methods ("d", "a", "m") in the order used within one iteration (see Details).

\( \lambda \)  A real number in the interval \((0,1)\) representing the parameter of the multiplicative algorithm used for computing A- and IV-optimal designs. Parameter \( \lambda \) is ignored if \( \text{crit}="D" \).

\( \text{tab} \)  A vector determining the regressor components to be printed with the resulting design. This argument should be a subvector of \( 1:n \), or a subvector of \texttt{colnames}(\( F \)), or it can be NULL. If \( \text{tab}=\text{NULL} \), the design is not printed.

\( \text{graph} \)  A vector determining the regressor components to be plotted with the resulting design. This argument should be a subvector of \( 1:n \), or a subvector of \texttt{colnames}(\( F \)), or it can be NULL. If \( \text{graph}=\text{NULL} \), the resulting design is not visualized.

\( \text{eff} \)  The efficiency for the stopping rule in the interval \((0,1)\). The algorithm will be stopped if a lower bound on the efficiency of the actual design is equal or greater than \( \text{eff} \).

\( t.max \)  The time limit for the computation.

Details

The function implements three algorithms for the computation of optimal approximate designs: the standard vertex-direction method ("d"), the standard multiplicative method ("m"), and a novel ordered vertex exchange method ("a"). The methods can be freely combined. For instance, setting \texttt{method}="doom" means that within each iteration the program will execute one vertex direction step, then two ordered vertex exchange steps, and then one multiplicative step. The justification of this approach is similar to that of Yu (2011); see the references.

The information matrix of \( w1 \) should have the reciprocal condition number of at least \( 1 \times 10^{-5} \). Even if no initial design is provided, the model should be non-singular in the sense that there exists an approximate design \( w \) with a well conditioned information matrix. If this requirement is not satisfied, the computation may fail, or it may produce a deficient design.
If the criterion of IV-optimality is selected, the region \( R \) should be chosen such that the associated matrix \( L \) (see the help page of the function `od.crit`) is non-singular, preferably with a reciprocal condition number of at least \( 1e^{-5} \). If this requirement is not satisfied, the computation may fail, or it may produce a deficient design.

The performance of the algorithms depends on the problem, on the choice of the parameters, and on the hardware used. However, if the method "o" is used at least once within each iteration, the function tends to reliably compute at least 99.9%-efficient approximate design for a problem with a ten thousand design points within seconds of computing time.

**Value**

A list with the following components:

- `method`: The method used for computing the design `w.best`.
- `w.best`: The best design found.
- `phi.best`: The value of the criterion of `w.best`.
- `eff.best`: The lower bound on the efficiency of `w.best`.
- `t.act`: The actual time taken by the computation.

**Author(s)**

Radoslav Harman, Lenka Filova

**References**


**See Also**

`od.SOCP`, `od.KL`, `od.RCs`

**Examples**

```r
# Create the regressors matrix for the third-degree trigonometric model
# on a "partial circle".
F.trig <- F.cube(-I(cos(x1)) + I(sin(x1)) +
    I(cos(2 * x1)) + I(sin(2 * x1)) +
    I(cos(3 * x1)) + I(sin(3 * x1)),
    -pi/2, pi/2, 200)

# Compute the D-, A-, and IV-optimal standardized approximate designs.
res.trig.D <- od.AA(F.trig, 1, alg = "doom", crit = "D",
    graph=1:7, t.max=4)
res.trig.A <- od.AA(F.trig, 1, alg = "doom", crit = "A",
    graph=1:7, t.max=4)
res.trig.IV <- od.AA(F.trig, 1, alg = "doom", crit = "IV",
    graph=1:7, t.max=4)

# Create a model with 2000 regressors generated from the 5 dimensional
```
# multivariate normal distribution. (A random linear regression model
# with 2000 design points and 5 parameters.)
F.norm <- matrix(rnorm(10000), ncol=5)

# Compute the D-, and A-optimal standardized approximate designs.
res.norm.D <- od.AA(F.norm, 1, alg = "om", crit = "D",
                   graph=1:5, t.max=4)
res.norm.A <- od.AA(F.norm, 1, alg = "om", crit = "A",
                   graph=1:5, t.max=4)

---

od.crit  

**Optimality criterion**

**Description**

Computes the value of the criterion of D-, A-, or IV-optimality of a given design.

**Usage**

od.crit(F, w, crit="D", R=NULL, tol=1e-12)

**Arguments**

- **F**: The n times m matrix of real numbers. Rows of F represent the m-dimensional regressors corresponding to n design points.
- **w**: The non-negative vector of length n representing the design.
- **crit**: The optimality criterion. Possible values are "D", "A", "IV".
- **R**: The region of summation for the IV-optimality criterion. The argument R must be a subvector of 1:n, or NULL. If R=NULL, the procedure uses R=1:n. Argument R is ignored if crit="D", or if crit="A".
- **tol**: A small positive number to determine singularity of the information matrix.

**Details**

Let w be a design with information matrix M, let n be the number of design points and let m be the number of parameters of the model.

For w, the value of the criterion of D-optimality is computed as \((\det(M))^{(1/n)}\) and the value of the criterion of A-optimality is computed as \(m/\text{trace}(M.inv)\), where M.inv is the inverse of M.

The IV-optimal design, sometimes called I-optimal or V-optimal, minimizes the integral of the variances of the BLUEs of the response surface over a region R, or the sum of the variances over R, if R is finite; see Section 10.6 in Atkinson et al. Let the matrix L be the integral (or the sum) of \(F(x) \times \text{cov}(F(x))\) over x in R. If the criterion of IV-optimality is selected, the region R should be chosen such that the associated matrix L is non-singular. Then, let \(L=t(C)C.inv\) be the Cholesky decomposition of L. The design w is IV-optimal in the model given by F, if and only if w is A-optimal for the model with the regressors matrix \(F \times C.inv\), where C.inv is the inverse of C.
For the purpose of this package, the value of the IV-criterion for $w$ is $m/\text{trace}(N.inv)$, where $N.inv$ is the inverse of the information matrix of $w$ in the model given by regressors matrix $F*\%C.inv$, and every computational problem of IV-optimality is converted to the corresponding problem of A-optimality. The argument $R$ is assumed to be a subset of $1:n$. If the application requires that $R$ is not a subset of the set of design points, the user should compute the matrix $C$, transform the model as described above, and use the procedures for A-optimality.

If the information matrix is singular, the value of all three criteria is zero. An information matrix is considered to be singular, if its minimal eigenvalue is smaller than $m*\text{tol}$.

Value

The value of the concave, positive homogeneous version of the selected real-valued criterion applied to the information matrix of the design $w$ in the linear regression model with $m$-dimensional regressors $F[1,\ldots,F[n]$ corresponding to $n$ design points.

Author(s)

Radoslav Harman, Lenka Filova

References


See Also

od.infmat, od.print, od.plot

Examples

```r
# The matrix of regressors for the spring balance weighing model with
# 6 weighed items.
F.sbw <- F.cube(~x1 + x2 + x3 + x4 + x5 + x6 ~ 1, rep(0, 6),
               rep(1, 6), rep(2, 6))

# The value of all 3 optimality criteria for the design of size 15
# that weighs each pair of items exactly once.
w2 <- rep(0, 64); w2[apply(F.sbw, 1, sum)==2] <- 1
od.crit(F.sbw, w2, "D")
od.crit(F.sbw, w2, "A")
od.crit(F.sbw, w2, "IV")

# The value of all 3 optimality criteria for the design of size 15 that
# weighs each quadruple of items exactly once.
w4 <- rep(0, 64); w4[apply(F.sbw, 1, sum)==4] <- 1
od.crit(F.sbw, w4, "D")
od.crit(F.sbw, w4, "A")
od.crit(F.sbw, w4, "IV")
```
**od.infmt**

---

### Information matrix

**Description**

Computes the information matrix of a given design.

**Usage**

```r
od.infmt(F, w)
```

**Arguments**

- `F` The n times m matrix of real numbers. Rows of `F` represent the m-dimensional regressors corresponding to the n design points.
- `w` The non-negative vector of length n representing the design.

**Details**

The information matrix of the design `w` is equal to $w[1]*M[1,] + \ldots + w[n]*M[n,]$, where $M[i,]$ is the elementary information matrix corresponding to the single trial in the i-th design point, that is, $M[i,]$ is the product of `F[i,]` and the transpose of `F[i,]`, $i=1,\ldots,n$.

Note: The actual computation of the information matrix uses an equivalent, but numerically more efficient formula.

**Value**

The m times m information matrix of the design `w` for the linear regression model with regressors $F[1,],\ldots,F[n,]$ and uncorrelated real-valued unit-variance observations.

**Author(s)**

Radoslav Harman, Lenka Filova

**See Also**

- `od.crit`, `od.print`, `od.plot`

**Examples**

```r
# The information matrix of an approximate design with weights 1/4
# in -1, -0.4, 0.4, 1 for the cubic model on a discretization of
# the interval [-1,1]
F.1D <- F.cube(-x1 + I(x1 ^ 2) + I(x1 ^ 3), -1, 1, 11)
round(od.infmt(F.1D, c(0.25,0,0,0.25,0,0,0.25,0,0,0.25)), 6)
```

```r
# The information matrix of a random exact design for the full quadratic
# model with 2 factors; the first with levels -1,0,1, and the second with
```
# levels -1, 0.5, 0.5, 1.
F.2D <- F.cube(-1*1^2 + I(1)*1^2) + I(x2^2), c(-1, -1), c(1, 1), c(3, 5))
o.d.infmat(F.2D, sample(0:1, dim(F.2D)[1], replace=TRUE))

# The matrix of the lattice design at levels 0, 0.5, 1 for the Scheffe
# quadratic mixture model with 3 mixture components, each with levels
# (0, 0.25, 0.5, 0.75, 1).
F.scheffe <- F.simplex(-1 + x2 + x3 + I(x1*x2) + I(x1*x3) +
                      I(x2*x3) - 1, 3, 5)
w.lattice <- rep(0, 15); w.lattice[c(1,3,5,10,12,15)] <- 1
o.d.infmat(F.scheffe, w.lattice)

---

**od.IQP**

**Efficient exact design using integer quadratic programming**

**Description**

Computes an efficient exact design under general linear constraints using the approach of integer quadratic programming.

**Usage**

```r
od.IQP(F, b, A=NULL, w=NULL, crit="D", R=NULL, w1=NULL,
       kappa=1e-9, tab=NULL, graph=NULL, t.max=120)
```

**Arguments**

- **F**
  - The \( n \times m \) matrix of real numbers. The rows of \( F \) represent the \( m \)-dimensional regressors corresponding to \( n \) design points. It is assumed that \( n \geq m \geq 2 \). Use `od.m1` for models with 1-dimensional regressors.

- **b, A**
  - The real vector of length \( k \) and the \( k \times n \) matrix of reals numbers. The linear constraints \( A \times w \leq b \) define the set of permissible designs \( w \) (where \( w \) is a described below.) The argument \( A \) can also be `NULL`; in that case \( b \) must be a positive number and \( A \) is set to the \( 1 \times n \) matrix of ones.

- **w**
  - The non-negative vector of length \( n \) representing the design to be augmented. This argument can also be `NULL`; in that case, \( w \) is set to the vector of zeros.

- **crit**
  - The optimality criterion. Possible values are "D", "A", "IV".

- **R**
  - The region of summation for the IV-optimality criterion. The argument \( R \) must be a subvector of \( 1:n \), or `NULL`. If \( R=\text{NULL} \), the procedure uses \( R=1:n \). Argument \( R \) is ignored if \( \text{crit="D"} \), or if \( \text{crit="A"} \).

- **w1**
  - The non-negative vector of length \( n \) representing the optimal or nearly-optimal approximate design for the design problem (including the design constraints). The argument \( w1 \) can also be `NULL`. In that case the procedure computes \( w1 \) using an appropriate procedure.

- **kappa**
  - A small non-negative perturbation parameter.
tab A vector determining the regressor components to be printed with the resulting design. This argument should be a subvector of 1:n, or a subvector of colnames(f), or it can be NULL. If tab=NULL, the design is not printed.

graph A vector determining the regressor components to be plotted with the resulting design. This argument should be a subvector of 1:n, or a subvector of colnames(f), or it can be NULL. If graph=NULL, the resulting design is not visualized.

t.max The time limit for the computation.

Details

The procedure computes an efficient exact design by means of integer quadratic programming. The idea is to use a quadratic criterion, which approximates the target criterion in the neighborhood of the information matrix of the optimal approximate design. See the reference for details.

The model should be non-singular in the sense that there exists an exact design w satisfying the constraints \( \theta \leq w \theta \leq w \) and \( \lambda \% \% \% w \leq b \), with a non-singular information matrix, preferably with the reciprocal condition number of at least \( 1e^{-5} \). If this requirement is not satisfied, the computation may fail, or it may produce a deficient design.

If the criterion of IV-optimality is selected, the region \( R \) should be chosen such that the associated matrix \( L \) (see the help page of the function od.crit) is non-singular, preferably with a reciprocal condition number of at least \( 1e^{-5} \). If this requirement is not satisfied, the computation may fail, or it may produce a deficient design.

The perturbation parameter kappa can be used to add \( n \times m \) iid random numbers from the uniform distribution in \([-\kappa, \kappa]\) to the elements of \( f \) before the optimization is executed. This can be helpful for increasing the numerical stability of the computation or for generating a random design from the potentially large set of optimal or nearly-optimal designs.

The performance depends on the problem and on the hardware used, but in most cases the function can compute an optimal or nearly-optimal exact design for a problem with a thousand design points within minutes of computing time. Although mostly reliable and fast, there are some pathological problems, even small in size, where the resulting design is deficient. We advise the user to verify the quality of the resulting design by comparing it to the result of an alternative method (such as od.rrc or od.MISOCP) and/or by computing its efficiency relative to the corresponding optimal approximate design (e.g., by using od.SOCP).

Value

A list with the following components:

- **method** The method used for computing the design w.best.
- **w.best** the best permissible design found, or NULL. The value of w.best will be NULL if the computation fails. This can happen, if no permissible solution is found within the time limit, no permissible solution exists, or the problem is unbounded; see the status variable for more details. Note that even if w.best is a permissible design, then it still can have a singular information matrix; cf. the Phi.best variable.
- **Phi.best** The value of the criterion of optimality of the design w.best. If w.best has a singular information matrix or if the computation fails, the value of Phi.best will be 0.
status The status variable of the gurobi optimization procedure; see the gurobi solver documentation for details.
t.act The actual time taken by the computation.

Author(s)
Radoslav Harman, Lenka Filova

References

See Also
od.RC, od.MISOCP, od.SOCP

Examples

if(require("gurobi")){
# Consider the full quadratic model on the domain (-1,-0.9,...,0.9,1)^2,
# that is 2 factors, each with 11 levels -1,-0.9,...,0.9,1. Suppose that
# the cost of an observation for the combination (i1,i2) of factors is
# equal to (i1+1.1)+(i2+1.1) price units. Our time and budget constraints
# dictate that we cannot perform more than 18 observations and spend more
# than 28 price units. Moreover, we do not wish to perform more than one
# observation under the same combination of factors. Let us compute
# an IV-efficient design under these constraints.

# Create the matrix of regressors of the model.
F.quad <- F.cube(-x1 + x2 + I(x1^2) + I(x2^2) + I(x1 * x2),
               c(-1, -1), c(1, 1), c(11, 11))

# Create the matrix A and the vector b such that our constraints on the
# feasible design w are equivalent to A * w <= b.
A.quad <- matrix(0, nrow=123, ncol=121)
for (i in 1:121){
  A.quad[i, i] <- 1
  A.quad[2, i] <- sum(F.quad[i, 2:3]) + 2.2
  A.quad[i + 2, i] <- 1
}
b.quad <- c(18, 28, rep(1, 121))

# Compute an IV-efficient design under the constraints defined above.
resIV <- od.IQP(F.quad, b.quad, A.quad, crit="IV", graph = c("x1","x2"),
               t.max = 60)

# Verify the quality of the resulting design by computing its efficiency
# with respect to the IV-optimal approximate design.
resIV.approx <- od.IQP(F.quad, b.quad, A.quad, crit="IV")
resIV$Phi.best / resIV.approx$Phi.best
}
**od.KL**

*Efficient exact size-constrained design using the KL exchange heuristic*

**Description**

Computes an efficient exact design under the standard (size) constraint using the KL exchange heuristic.

**Usage**

```r
don.KL(F, N, crit="D", R=NULL, w1=NULL, K=NULL, L=NULL,
variant="a", kappa=0, tab=NULL, graph=NULL,
t.max=120)
```

**Arguments**

- **F**: The \(n \times m\) matrix of real numbers. The rows of \(F\) represent the \(m\)-dimensional regressors corresponding to \(n\) design points. It is assumed that \(n >= m = 2\). Cf. `od.m1` for models with \(Q\)-dimensional regressors.

- **N**: The required size of the design, i.e., the sum of the components of the resulting design. It is assumed that \(N >= m\).

- **crit**: The optimality criterion. Possible values are "D", "A", "IV".

- **R**: The region of summation for the IV-optimality criterion. The argument \(R\) must be a subvector of \(1:n\), or NULL. If \(R=\)NULL, the procedure uses \(R=1:n\). Argument \(R\) is ignored if \(crit="D"\), or if \(crit="A"\).

- **w1**: The real vector of length \(n\) with non-negative integer components used as an initial design for the forward phase of the algorithm. Required: \(\text{sum}(w1) <= N\). The argument \(w1\) can also be NULL; in that case the procedure generates a random initial design.

- **K, L**: The natural numbers representing tuning parameters of the procedure; see Details. One or both arguments \(K, L\) can also be NULL; then the function selects its own value of the parameter(s).

- **variant**: The variant of the exchange heuristic; implemented values are "a", "b".

- **kappa**: A small non-negative perturbation parameter.

- **tab**: A vector determining the regressor components to be printed with the resulting design. This argument should be a subvector of \(1:n\), or a subvector of `colnames(F)`, or it can be NULL. If `tab=NULL`, the design is not printed.

- **graph**: A vector determining the regressor components to be plotted with the resulting design. This argument should be a subvector of \(1:n\), or a subvector of `colnames(F)`, or it can be NULL. If `graph=NULL`, the resulting design is not visualized.

- **t.max**: The time limit for the computation.
Details

This implementation of the KL algorithm is generally based on the ideas described in Atkinson et al. (2007); see the references.

The information matrix of \( w_1 \) should preferably have the reciprocal condition number of at least \( 1 \times 10^{-5} \). Even if no initial design is provided, the model should be non-singular in the sense that there exists an exact design \( w \) of size \( N \) with a well conditioned information matrix. If this requirement is not satisfied, the computation may fail, or it may produce a deficient design.

If the criterion of IV-optimality is selected, the region \( R \) should be chosen such that the associated matrix \( \lambda \) (see the help page of the function \( \text{od.crit} \)) is non-singular, preferably with a reciprocal condition number of at least \( 1 \times 10^{-5} \). If this requirement is not satisfied, the computation may fail, or it may produce a deficient design.

The tuning parameter \( K \) is the (upper bound on the) number of "least promising" support points of the current design, for which exchanges are attempted. The tuning parameter \( L \) is the (upper bound on the) number of "most promising" candidate design points for which exchanges are attempted.

Variant "a" means that the exchanges are performed in an efficient order and any improving exchange is immediately executed. Variant "b" means that all permissible exchanges are evaluated and the best one of the exchanges (or a random one of the best exchanges) is executed.

If the algorithm stops in a local optimum before the allotted time elapsed, the computation is restarted. The result is the best design found within all restarts.

The perturbation parameter \( \kappa \) can be used to add \( n \times m \) iid random numbers from the uniform distribution in \([\kappa - \kappa, \kappa]\) to the elements of \( F \) before the optimization is executed. This can be helpful for generating a random design from the potentially large set of optimal or nearly-optimal designs.

The performance of the function depends on the problem, on the chosen parameters, and on the hardware used, but in most cases the function can compute a nearly-optimal exact design for a problem with a thousand design points within seconds of computing time. Because this is only a heuristic, we advise the user to verify the quality of the resulting design by comparing it to the result of an alternative method (such as \( \text{od.RCs} \)) and/or by computing its efficiency relative to the corresponding optimal approximate design (computed using \( \text{od.AA} \)).

Value

A list with the following components:

- \( \text{method} \): The method used for computing the design \( w.\text{best} \).
- \( w.\text{best} \): The best design found.
- \( \Phi.\text{best} \): The value of the criterion of \( w.\text{best} \).
- \( t.\text{act} \): The actual time taken by the computation.

Author(s)

Radoslav Harman, Lenka Filova

References

See Also

od.RCs, od.AA

Examples

# Consider the quadratic Scheffe mixture model with 3 mixture components,
# each one with permissible levels 0,0.02,...,0.7, i.e., without high
# proportions of components.
# We will calculate an A-efficient exact design of size 18.

# Compute the regressors for the mixture model without constraints
# on the region.
# (Note: Here, the constraints are on the design region; it is generally
# much simpler to find an optimal design under non-standard constraints
# on the design region than find an optimal design under non-standard
# constraints on the design itself.)
F.scheffe <- F.simplex(-x1 + x2 + x3 + I(x1 * x2) + I(x1 * x3) +
                      I(x2 * x3) - 1, 3, 5)

# Remove the trials with high values of the mixture components.
# The resulting design space will have 966 design points.
F.scheffe <- F.scheffe[apply(F.scheffe[, 1:3], 1, max) <= 0.7,]

# Compute an A-efficient exact designs with 18 observations.
res.exact <- od.KL(F.scheffe, 18, crit = "A", tab=1:3,
                   graph=1:3, t.max=4)

# Verify the quality of the resulting design by computing its efficiency
# relative to the A-optimal approximate size-constrained design.
res.approx <- od.AA(F.scheffe, 18, crit = "A", eff=1-1e-9)
res.exact$Phi.best / res.approx$Phi.best

---

Optimal design for models with a one-dimensional parameter

Description

Computes an optimal approximate or exact design under general linear constraints for a model with
a one-dimensional parameter.

Usage

od.m1(f, b, A=NULL, w0=NULL, type="exact", kappa=1e-9,
       tab=NULL, graph=NULL, t.max=120)
Arguments

- **F**
  The \( n \) times 1 matrix of regressors corresponding to \( n \) design points and 1 model parameter.

- **b**, **A**
  The real vector of length \( k \) and the \( k \) times \( n \) matrix of reals numbers. The linear constraints \( A \times w = b \), \( w \geq 0 \) define the set of permissible designs \( w \) (where \( w \geq 0 \) is a described below.) The argument \( A \) can also be NULL; in that case \( b \) must be a positive number and \( A \) is set to the \( 1 \) times \( n \) matrix of ones.

- **w0**
  The non-negative vector of length \( n \) representing the design to be augmented. This argument can also be NULL; in that case, \( w0 \) is set to the vector of zeros.

- **type**
  Specifies whether exact or approximate design is to be computed.

- **kappa**
  A small non-negative perturbation parameter.

- **tab**
  A vector determining the regressor components to be printed with the resulting design. This argument should be a subvector of \( 1:n \), or a subvector of \( colnames(F) \), or it can be NULL. If \( \text{tab} = \text{NULL} \), the design is not printed.

- **graph**
  A vector determining the regressor components to be plotted with the resulting design. This argument should be a subvector of \( 1:n \), or a subvector of \( colnames(F) \), or it can be NULL. If \( \text{graph} = \text{NULL} \), the resulting design is not visualized.

- **t.max**
  The time limit for the computation.

Details

For the case of \( m = 1 \) parameter, the problem of optimal design is much simpler than for \( m > 2 \). First, for \( m = 1 \) all standardized information criteria coincide, therefore the D-, A-, and IV-optimal designs are the same, and we can call them just "optimal". Second, the information matrix is a real number, therefore we can call it just "information". Under the most common size constraint, the optimal design is any design supported on the set of maxima of \( F[1,1]^2, \ldots, F[n,1]^2 \), which is straightforward to find. Under a non-standard linear constraint, however, the problem becomes a less trivial knapsack problem, which is here solved by the integer linear programming solver of gurobi.

The model should be non-singular in the sense that there exists an exact design \( w \) satisfying the constraints \( 0 \leq w \leq w^* \) and \( A \times w = b \), with a nonzero information. If this requirement is not satisfied, the computation may fail, or produce a deficient design.

If the criterion of IV-optimality is selected, the region \( R \) should be chosen such that the associated matrix \( L \) (in this case a real number; see the help page of the function \( \text{od.crit} \)) is non-zero. If this requirement is not satisfied, the computation may fail, or it may produce a deficient design.

The perturbation parameter \( \kappa \) can be used to add \( n \times m \) iid random numbers from the uniform distribution in \([-\kappa, \kappa]\) to the elements of \( F \) before the optimization is executed. This can be helpful for increasing the numerical stability of the computation or for generating a random design from the potentially large set of optimal or nearly-optimal designs.

The performance depends on the problem and on the hardware used, but in most cases the function can compute an optimal exact design for a problem with a thousand design points within seconds of computing time.
A list with the following components:

- **method**
  - The method used for computing the design w.best.

- **w.best**
  - The best permissible design found, or NULL. The value of w.best will be NULL if the computation fails. This can happen, if no permissible solution is found within the time limit, no permissible solution exists, or the problem is unbounded; see the status variable for more details. Note that even if w.best is a permissible design, then it still can have a singular information matrix; cf. the Phi.best variable.

- **phi.best**
  - The value of the criterion of optimality of the design w.best. If w.best has a singular information matrix or if the computation fails, the value of phi.best will be 0.

- **status**
  - The status variable of the gurobi optimization procedure; see the gurobi solver documentation for details.

- **t.act**
  - The actual time taken by the computation.

**Author(s)**

Radoslav Harman, Lenka Filova

**Examples**

```r
if(require("gurobi")){
  # We will demonstrate the procedure on a simple randomly generated
  # knapsack problem. Here, the squares of elements of F correspond
  # to the values of n available items, the elements of the 1 times n
  # matrix A correspond to the weights of the n items, and the real
  # number b is the upper limit on the total weight of the items that
  # can be put into the knapsack.
  # The resulting binary "optimal design" determines which of the items
  # should we put into the knapsack to steal the highest possible value.

  n <- 200  # There are this many items to choose from.
  F.square <- matrix(sample(1:10, n, replace=TRUE), ncol=1)
  # Generate random prices of items.
  A <- matrix(sample(1:10, n, replace=TRUE), nrow=1)
  # Generate random weights of items in kgs.
  A <- rbind(A, diag(n))
  # We assume that there is just one copy of each item.
  b <- c(n / 4, rep(1,n))
  # The capacity of the knapsack is n/4 kgs.

  # Compute the optimal design, which in this case determines how many
  # (0 or 1) of each of the n items should we put into the knapsack.
  od.ml(sqrt(F.square), b, A)

  # Note: one can compare the result with a specialized function
  # as follows:
  # library(adagio); knapsack(A[1,], F.square[,1], n / 4)
```
However, od.m1 is more general than the standard knapsack functions.
Suppose, for instance, that the uncle asks that we must be sure to
take the items number 1, 13 and 66. We will compute the most valuable
selection of items that fit into our knapsack and contain the
required items.

```r
w0 <- rep(0, n); w0[c(1, 13, 66)] <- 1
od.m1(sqrt(F.square), b, A, w0, t.max=2)
```

---

**Description**

Computes an efficient exact experimental design under general linear constraints using the approach
of mixed integer second-order cone programming.

**Usage**

```r
od.MISOCP(F, b, A=NULL, w0=NULL, crit="D", R=NULL, kappa=1e-9,
          tab=NULL, graph=NULL, t.max=120)
```

**Arguments**

- **F**
  The \( n \times m \) matrix of real numbers. The rows of \( F \) represent the \( m \)-dimensional regressors corresponding to \( n \) design points. It is assumed that \( n \geq m \geq 2 \). Use `od.m1` for models with 1-dimensional regressors. For D-optimality, the current implementation supports the models with \( m \leq 10 \).

- **b, A**
  The real vector of length \( k \) and the \( k \times n \) matrix of reals numbers. The linear constraints \( A%*%w0<w \) define the set of permissible designs \( w \) (where \( w0 \) is a described below.) The argument \( A \) can also be \( NULL \); in that case \( b \) must be a non-negative number and \( A \) is set to the \( 1 \times n \) matrix of ones.

- **w0**
  The non-negative vector of length \( n \) representing the design to be augmented. This argument can also be \( NULL \); in that case, \( w0 \) is set to the vector of zeros.

- **crit**
  The optimality criterion. Possible values are "D", "A", "IV".

- **R**
  The region of summation for the IV-optimality criterion. The argument \( R \) must be a subvector of \( 1:n \), or \( NULL \). If \( R=NULL \), the procedure uses \( R=1:n \). Argument \( R \) is ignored if \( crit="D" \), or if \( crit="A" \).

- **kappa**
  A small non-negative perturbation parameter.

- **tab**
  A vector determining the regressor components to be printed with the resulting design. This argument should be a subvector of \( 1:n \), or a subvector of `colnames(F)`, or it can be \( NULL \). If `tab=NULL`, the design is not printed.
graph

A vector determining the regressor components to be plotted with the resulting design. This argument should be a subvector of \(1:n\), or a subvector of \(\text{colnames}(F)\), or it can be \text{NULL}. If \text{graph}=\text{NULL}, the resulting design is not visualized.

t.max

The time limit for the computation.

Details

The procedure computes an efficient exact design by converting the optimal design problem to a specific problem of mixed integer second-order cone programming; see the reference for details. The advantage of this approach is the possibility to construct exact designs under a general system of linear constraints.

The model should be non-singular in the sense that there exists an exact design \(w\) satisfying the constraints \(0<=w_{0}<=w\) and \(A_{1}w<=b\), with a non-singular information matrix, preferably with the reciprocal condition number of at least \(1e-5\). If this requirement is not satisfied, the computation may fail, or it may produce a deficient design.

If the criterion of IV-optimality is selected, the region \(R\) should be chosen such that the associated matrix \(L\) (see the help page of the function \text{od.crit}) is non-singular, preferably with a reciprocal condition number of at least \(1e-5\). If this requirement is not satisfied, the computation may fail, or it may produce a deficient design.

The perturbation parameter \(\kappa\) can be used to add \(nm\) iid random numbers from the uniform distribution in \([-\kappa, \kappa]\) to the elements of \(F\) before the optimization is executed. This can be helpful for increasing the numerical stability of the computation or for generating a random design from the potentially large set of optimal or nearly-optimal designs.

The performance strongly depends on the problem and on the hardware used, but in most cases the function can compute an optimal or nearly-optimal exact design for a problem with a hundred design points within minutes of computing time. We advise the user to verify the quality of the resulting design by comparing it to the result of an alternative method (such as \text{od.IQP} and \text{od.RC}) and/or by computing its efficiency relative to the corresponding optimal approximate design (computed using \text{od.SOC}P). In the special case of the single constraint on the size, it is generally more efficient to use the function \text{od.KL}, or the function \text{od.RCs}.

Value

A list with the following components:

- \text{method}
  - The method used for computing the design \(w\_\text{best}\).
- \text{w.best}
  - the best permissible design found, or \text{NULL}. The value of \(w\_\text{best}\) will be \text{NULL} if the computation fails. This can happen, if no permissible solution is found within the time limit, no permissible solution exists, or the problem is unbounded; see the \text{status} variable for more details. Note that even if \(w\_\text{best}\) is a permissible design, then it still can have a singular information matrix; cf. the \text{Phi.condition} variable.
- \text{Phi.best}
  - The value of the criterion of optimality of the design \(w\_\text{best}\). If \(w\_\text{best}\) has a singular information matrix or if the computation fails, the value of \text{Phi.condition} will be \text{NULL}. 

status

The status variable of the gurobi optimization procedure; see the gurobi solver documentation for details.

t.act

The actual time taken by the computation.

Author(s)

Radoslav Harman, Lenka Filova

References


See Also

od.IQP, od.RC, od.SOCP, od.KL, od.RCs

Examples

if(require("gurobi")){
  # Consider a dose-response study where both efficacy and toxicity are
  # observed as 0/1 outcomes for each patient, where the probability of
  # the outcome 1 under the dose x is modeled by the logistic function:
  # exp(ae+be*x)/(1+exp(ae+be*x)) for the efficacy, and
  # exp(at+bt*x)/(1+exp(at+bt*x)) for the toxicity. We can choose the
  # doses x in the range 1 mg to 150 mgs.
  # The aim is to estimate the parameters ae,be using the D-optimal
  # design, or using the A-optimal design.
  # Because this is a non-linear model, the optimal designs will depend
  # on the unknown values of the parameters. We will use the approach of
  # local optimality with the following nominal values of the parameters:
  tle <- c(-10, 0.2)
  tlt <- c(-20, 0.2)

  # It is simple to show that the localized information matrix for
  # (ae,be) is the information matrix of the standard model with
  # the following regressors:
  F.logistic <- matrix(0, nrow=150, ncol=2)
  for (i in 1:150)
    F.logistic[i, ] <-
      c(sqrt(exp(tle[1]+tle[2]*i))/(1+exp(tle[1]+tle[2]*i)),

  # The constraints on the experiment are twofold: We can have at most
  # N=100 subjects and we also require that the expected number of
  # "failed" trials is at most 10. A trial is considered to be a failure
  # if it leads to either a toxic response, or if it is not efficacious.
  # These constraints can be expressed as A*x<=b:
  efficacy.prob <- function(x)
  toxicity.prob <- function(x)
\[
\text{exp(tlt[1]+tlt[2]*x)/(1+\text{exp(tlt[1]+tlt[2]*x))}
\]

\begin{verbatim}
failure.prob <- function(x)
  1 - (1 - toxicity.prob(x)) * efficacy.prob(x)
b <- c(100, 10); A <- rbind(rep(1,150), failure.prob(1:150))

# Now we can compute the designs:
res.D <- od.MISOCP(F.logistic, b, A, crit="D")
res.A <- od.MISOCP(F.logistic, b, A, crit="A")

# Let us verify the quality of the designs by computing their efficiency
# relative to the approximate optimal designs:
res.D$phi.best / od.SOCP(F.logistic, b, A, crit="D")$phi.best
res.A$phi.best / od.SOCP(F.logistic, b, A, crit="A")$phi.best

# We can plot the failure probability curve (red), the toxicity
# probability curve (black), the efficacy probability curve (green),
# the D-optimal design (orange) and the A-optimal design (blue):
plot(failure.prob(1:150), type="l",
ylab="probability / proportion of subjects",
lwd=3, col="red")
lines(toxicity.prob(1:150), col="black")
lines(efficacy.prob(1:150), col="green")
lines(res.D$w.best/100, type="h", col="orange")
lines(res.A$w.best/100, type="h", col="blue", lwd=2)

# Note that both designs perform the observations generally at two
# different levels, one of which is a dose which leads to a 50-percent
# failure of efficacy. Based on these designs, none of the patients
# are put on dangerously high levels of doses.
\end{verbatim}

---

**od.plot**

**Plot of a design**

**Description**

Plots a graph of a given design.

**Usage**

\[
\text{od.plot}(w, X=NULL, main="", del=0.001)
\]

**Arguments**

- **w**
  - The non-negative vector of length \(n\) representing the design.

- **X**
  - An \(n \times d\) real matrix giving the \(d\) coordinates of each of the \(n\) design points. The argument \(X\) can also be \(NULL\); in that case, the design weights will be plotted against the index vector \(1:n\).
main

A character string used for the main title of the plot.

del

A small positive number. Each design weight smaller than del will be treated as zero.

Details

If supplied, the rows of X represent coordinates of the design points in the d-dimensional design region. The procedure generates a plot of all basic two-dimensional projections of the design w, such that the circles corresponding to the same design points have the same color, and the areas of the circles are approximately proportional to the weights of the design points.

Author(s)

Radoslav Harman, Lenka Filova

See Also

od.crit, od.infmat, od.print

Examples

# Plot of a uniform design on a discretized line segment.
od.plot(rep(1:21, 21), seq(-1, 1, by=0.1), "Discrete uniform design")

# Plot of a non-uniform simplex lattice design.
F <- F.simplex(-x1 + x2 + x3 + I(x1*x2) + I(x1*x3) + I(x2*x3) - 1, 3, 4)
od.plot(c(4, 1, 1, 4, 1, 2, 1, 1, 1, 4), F[,c(1, 2, 3)],
"Non-uniform simplex lattice design")

# Plot of a Hadamard design on the cube.
F <- F.cube(-x1 + x2 + x3 - 1, rep(-1, 3), rep(1, 3), rep(11, 3))
w <- rep(0, 11^3); w[1+121, 11^3-10, 11^3-110] <- 1
od.plot(w, F, "Hadamard design (standard view")
U <- qr.Q(qr(matrix(rnorm(9), ncol=3)))
od.plot(w, F %*% U, "Hadamard design (randomly rotated view")

od.print

Print of a design

Description

Prints a table of a given design.

Usage

od.print(w, X=NULL, del=1e-6)
Arguments

- **w**: The non-negative vector of length \( n \) representing the design.
- **X**: An \( n \) times \( d \) real matrix giving the \( d \) coordinates of each of the \( n \) design points. The argument \( X \) can also be NULL; in that case, the non-zero design weights will be printed with their index in \( 1:n \).
- **del**: A small positive number. Each design weight smaller than **del** will be treated as zero, therefore it will not be displayed in the table.

Details

If supplied, the rows of \( X \) represent coordinates of the design points in the \( d \)-dimensional design region. The procedure prints a compact table of the design \( w \), excluding the coordinates smaller than **del**. This is often useful, because the support size of the experimental design (the number of non-zero design weights) is usually much smaller than the size \( n \) of the design space.

Author(s)

Radoslav Harman, Lenka Filova

See Also

- `od.crit`, `od.infmat`, `od.plot`

Examples

```r
# Print the uniform approximate design on 24 design points in [0, 2*pi].
w1 <- rep(1/24, 24); od.print(w1, seq(0, (46 / 24)* pi, length=24))

# Print an approximate design on 24 design points in [0, 2*pi]
# with 8 support points.
w2 <- rep(c(1/8, 0, 0), 8);
   od.print(w2, seq(0, (46 / 24)* pi, length=24))

# Print an approximate design on 24 design points in [0, 2*pi]
# with 3 support points.
w3 <- rep(c(1/3, rep(0,7)), 3);
   od.print(w3, seq(0, (46 / 24)* pi, length=24))

# For the trigonometric model of the first degree on the "full circle"
# discretized to 24 equispaced points, all three designs have the same
# information matrix:
F.trig <- F.cube(-I(cos(x1)) + I(sin(x1)), 0, (46 / 24)* pi, 24)
round(od.infmat(F.trig, w1), 6)
round(od.infmat(F.trig, w2), 6)
round(od.infmat(F.trig, w3), 6)
```
**od.RC**

*Efficient exact design using the RC heuristic*

**Description**

Computes an efficient exact design under linear resource constraints using the RC heuristic.

**Usage**

```
od.RC(F, b, A=NULL, w0=NULL, crit="D", R=NULL, w1=NULL, kappa=0, tab=NULL, graph=NULL, t.max=120)
```

**Arguments**

- **F**
  
  The \( n \) times \( m \) matrix of real numbers. The rows of \( F \) represent the \( m \)-dimensional regressors corresponding to \( n \) design points. It is assumed that \( n \geq m \geq 2 \). Use \( \text{od.m1} \) for models with 1-dimensional regressors.

- **b, A**
  
  The vector of length \( k \) with positive real components and the \( k \) times \( n \) matrix of non-negative reals numbers. Each column of \( A \) must have at least one strictly positive element. The linear constraints \( A*\omega \leq b \), \( \omega \leq w \) define the set of permissible designs \( w \) (where \( \omega \) is a described below.) The argument \( A \) can also be \( \text{NULL} \); in that case \( b \) must be a positive number and \( A \) is set to the \( Q \times n \) matrix of ones.

- **w0**
  
  The \( n \) times 1 vector of nonnegative numbers representing the design to be augmented. Design \( w0 \) must satisfy \( A*\omega \leq b \) and it is also required that \( w0 \) is not the only permissible exact design. The argument \( w0 \) can also be \( \text{NULL} \); in that case, \( w0 \) is set to the vector of zeros.

- **crit**
  
  The optimality criterion. Possible values are "D", "A", "IV".

- **R**
  
  The region of summation for the IV-optimality criterion. The argument \( R \) must be a subvector of \( 1:n \), or \( \text{NULL} \). If \( R=\text{NULL} \), the procedure uses \( R=1:n \). Argument \( R \) is ignored if \( \text{crit}="D" \), or if \( \text{crit}="A" \).

- **w1**
  
  The \( n \) times 1 nonnegative vector which represents the initial design. The design \( w1 \) must satisfy \( w0 \leq w1 \) and \( A*w1 \leq b \). The argument \( w1 \) can also be \( \text{NULL} \). In that case the procedure sets \( w1 \) to be \( w0 \).

- **kappa**
  
  A small non-negative perturbation parameter.

- **tab**
  
  A vector determining the regressor components to be printed with the resulting design. This argument should be a subvector of \( 1:n \), or a subvector of \( \text{colnames}(F) \), or it can be \( \text{NULL} \). If \( \text{tab}=\text{NULL} \), the design is not printed.

- **graph**
  
  A vector determining the regressor components to be plotted with the resulting design. This argument should be a subvector of \( 1:n \), or a subvector of \( \text{colnames}(F) \), or it can be \( \text{NULL} \). If \( \text{graph}=\text{NULL} \), the resulting design is not visualized.

- **t.max**
  
  The time limit for the computation.
Details

This is an implementation of the algorithm proposed by Harman et al. employing the tabu search principle, and related to the DETMAX procedure; see References. The inequalities $A_w x \leq b$, $w \leq w$ with the specific properties mentioned above, form the so-called resource constraints which encompass many practical restrictions on the design, always permit a feasible solution, and lead to a bounded set of feasible solutions.

The information matrix of $w \hat{1}$ should preferably have the reciprocal condition number of at least $10^{-5}$. Note that the floor of an optimal approximate design (computed using od.SOCP) is sometimes a good initial design. Alternatively, the initial design can be the result of a different optimal design procedure, such as od.IQP. Even if no initial design is provided, the model should be non-singular in the sense that there exists an exact design $w$ with a well conditioned information matrix, satisfying all constraints. If this requirement is not satisfied, the computation may fail, or it may produce a deficient design.

If the criterion of IV-optimality is selected, the region $R$ should be chosen such that the associated matrix $L$ (see the help page of the function od.crit) is non-singular, preferably with a reciprocal condition number of at least $10^{-5}$. If this requirement is not satisfied, the computation may fail, or produce a deficient design.

The perturbation parameter $\kappa$ can be used to add $n m$ iid random numbers from the uniform distribution in $[-\kappa, \kappa]$ to the elements of $F$ before the optimization is executed. This can be helpful for generating a random design from the potentially large set of optimal or nearly-optimal designs. However, the RC heuristic uses a tabu principle based on the criterion values of designs, therefore in some problems a nonzero $\kappa$ can be detrimental to the optimization process.

The procedure always returns a permissible design, but in some cases, especially if $t_{\text{max}}$ is too small, the resulting design can be inefficient. The performance depends on the problem and on the hardware used, but in most cases the function can compute a nearly-optimal exact design for a problem with a hundred design points within minutes of computing time. Because this is a heuristic method, we advise the user to verify the quality of the resulting design by comparing it to the result of an alternative method (such as od.IQP and od.MISOCP) and/or by computing its efficiency relative to the corresponding optimal approximate design (computed using od.SOCP). In the special case of the single constraint on the size, it is generally more efficient to use the functions od.KL, or od.RCs.

Value

A list with the following components:

- **method** The method used for computing the design $w_{\text{best}}$.
- **w.best** The best design found.
- **Phi.best** The value of the criterion of $w_{\text{best}}$.
- **t.act** The actual time taken by the computation.

Author(s)

Radoslav Harman, Lenka Filova
References


See Also

od.IQP, od.MISOCP, od.SOCP, od.KL, od.RCs

Examples

# Consider the spring balance weighing model with 6 items of unknown
# weight. Suppose that we have already performed one weighing of each
# item separately. We will compute an A-efficient augmentation designs
# with 20 additional weighings. Then we will compute A-efficient designs
# under the additional restriction that no item can be used more
# than 8 times.

# Create the matrix of regressors of the model and the design
# to be augmented.
F.sbw <- F.cube(~x1 + x2 + x3 + x4 + x5 + x6 - 1, rep(0, 6),
               rep(1, 6), rep(2, 6))
w0 <- rep(0, 64); w0[apply(F.sbw, 1, sum) == 1] <- 1

# Compute an exact A-efficient augmentation design with 26
# total weighings.
b.sbw <- 26; A.sbw <- matrix(1, nrow=1, ncol=64)
res <- od.RC(F.sbw, b.sbw, A.sbw, w0=w0, crit="A", tab=1:6, t.max = 30)

# There are many A-optimal designs for this problem, which is possible
# to see by running the line above several times with a very small
# non-zero kappa. Note that each of the A-optimal experiments uses each
# item exactly 11 times. Suppose that we can use each item at
# most 8 times.

# Create the constraints A.eight * w <= b.eight representing
# the restriction that no item can be used more than eight times
# in the entire experiment.
b.eight <- rep(8, 6); A.eight <- t(F.sbw)

# Compute an exact A-efficient design with 26 total weighings under
# all constraints defined above.
b.sbw <- c(b.eight, 26); A.sbw <- rbind(A.eight, rep(1,64))
res <- od.RC(F.sbw, b.sbw, A.sbw, w0=w0, crit="A", tab=1:6, t.max = 60)

# To find a lower bound on the true efficiency of the resulting design,
# let us compare it to the A-optimal approximate design.
res.approx <- od.SOCP(F.sbw, b.sbw, A.sbw, w0=w0, crit="A",
                     tab=1:6, t.max = 20)
res$Phi.best / res.approx$Phi.best
Efficient exact size-constrained design using the RC heuristic

Description

Computes an efficient exact design under standard (size) constraints using the RC heuristic.

Usage

```r
od.RCs(F, N, w0=NULL, crit="D", R=NULL, w1=NULL, kappa=0,
     tab=NULL, graph=NULL, t.max=120)
```

Arguments

- **F**
  The \( n \times m \) matrix of real numbers. The rows of \( F \) represent the \( m \)-dimensional regressors corresponding to \( n \) design points. It is assumed that \( n \geq m \geq 2 \). Cf. `od.m1` for models with \( Q \)-dimensional regressors.

- **N**
  The required size of the design, i.e., the sum of the components of the resulting design. It is assumed that \( N \geq m \).

- **w0**
  The \( n \times 1 \) vector representing the design to be augmented. Design \( w0 \) must satisfy \( \sum(w0) \leq N-1 \). The argument \( w0 \) can also be `NULL`. In that case \( w0 \) is set to the vector of zeros.

- **crit**
  The optimality criterion. Possible values are "D", "A", "IV".

- **R**
  The region of summation for the IV-optimality criterion. The argument \( R \) must be a subvector of \( 1:n \), or `NULL`. If \( R=NULL \), the procedure uses \( R=1:n \). Argument \( R \) is ignored if \( crit="D" \), or if \( crit="A" \).

- **w1**
  The \( n \times 1 \) nonnegative vector representing the initial design. Design \( w1 \) must satisfy \( w0 \leq w1 \) and \( \sum(w1) \leq N \). Argument \( w1 \) can also be `NULL`. In that case the procedure sets \( w1 \) to be \( w0 \).

- **kappa**
  A small non-negative perturbation parameter.

- **tab**
  A vector determining the regressor components to be printed with the resulting design. This argument should be a subvector of \( 1:n \), or a subvector of `colnames(F)`, or it can be `NULL`. If `tab=NULL`, the design is not printed.

- **graph**
  A vector determining the regressor components to be plotted with the resulting design. This argument should be a subvector of \( 1:n \), or a subvector of `colnames(F)`, or it can be `NULL`. If `graph=NULL`, the resulting design is not visualized.

- **t.max**
  The time limit for the computation.

Details

This is a numerically more efficient implementation of the general algorithm `od.RC` for the case of the standard (size) constraint, which is a special case of the resource constraints. See the reference for details.
The information matrix of \( w_1 \) should preferably have the reciprocal condition number of at least \( 1e^{-5} \). Note that the floor of an optimal approximate design of size \( N \) (computed using \( \text{od.AA} \)) is often a good initial design. Alternatively, the initial design can be the result of a different optimal design procedure, such as \( \text{od.KL} \). In any case, the model should be non-singular in the sense that there exists an exact design \( w \) with a well conditioned information matrix. If this requirement is not satisfied, the computation may fail, or it may produce a deficient design.

If the criterion of IV-optimality is selected, the region \( R \) should be chosen such that the associated matrix \( L \) (see the help page of the function \( \text{od.crit} \)) is non-singular, preferably with a reciprocal condition number of at least \( 1e^{-5} \). If this requirement is not satisfied, the computation may fail, or it may produce a deficient design.

The perturbation parameter \( \kappa \) can be used to add \( n \times m \) iid random numbers from the uniform distribution in \([-\kappa, \kappa]\) to the elements of \( F \) before the optimization is executed. This can be helpful for generating a random design from the potentially large set of optimal or nearly-optimal designs. However, the RC heuristic uses a tabu principle based on the criterion values of designs, therefore in some problems a nonzero \( \kappa \) can be detrimental to the optimization process.

The procedure always returns a permissible design, but in some cases, especially if \( t_{\text{max}} \) is too small, the resulting design can be inefficient. The performance depends on the problem and on the hardware used, but in most cases the function can compute a nearly-optimal exact design for a problem with a hundred design points within minutes of computing time. Because this is a heuristic method, we advise the user to verify the quality of the resulting design by comparing it to the result of an alternative method (such as \( \text{od.KL} \)) and/or by computing its efficiency relative to the corresponding optimal approximate design (computed using \( \text{od.AA} \)).

### Value

A list with the following components:

- `method` The method used for computing the design \( w.\text{best} \).
- `w.\text{best}` The best design found.
- `phi.\text{best}` The value of the criterion of \( w.\text{best} \).
- `t.\text{act}` The actual time taken by the computation.

### Author(s)

Radoslav Harman, Lenka Filova

### References


### See Also

\( \text{od.RC}, \ \text{od.KL}, \ \text{od.AA} \)
Examples

# We will compute the D-optimal design for the block model with additive
# effects of blocks and treatments. Suppose that we have 15 blocks of
# size two and 10 treatments. The problem is equivalent to the problem
# of D-optimality in the standard model with 45 design points, 15 trials,
# and the matrix of regressors F.block computed as below
# (see the reference for details).

k <- 0
T1 <- T2 <- rep(0, 45)
F.aux <- matrix(0, nrow=45, ncol=10)
for(i in 1:9){
  for(j in (i + 1):10){
    k <- k + 1
    T1[k] <- i
    T2[k] <- j
    F.aux[k, i] <- 1
    F.aux[k, j] <- (-1)
  }
}
F.block <- F.aux %*% eigen(matrix(1, ncol=10, nrow=10))$vectors[, 2:10]

# Compute the D-optimal exact design of size 15 using the RCs procedure.
res <- od.RCs(F.block, 15, crit = "D", t.max=2.5)
res$Phi.best

# Solve the same problem using the KL procedure to check the obtained
# criterion value.
od.KL(F.block, 15, crit = "D", t.max=1.5)$Phi.best

# Display the treatments in the 15 blocks of the obtained design.
data.frame(T1=T1[as.logical(res$w.best)], T2=T2[as.logical(res$w.best)])

# Note that the concurrence graph of the design is the Petersen graph.

od.SOCP

Optimal approximate design using second-order cone programming

Description

Computes an efficient approximate experimental design under general linear constraints using the
approach of second-order cone programming.

Usage

od.SOCP(F, b, A=NULL, w0=NULL, crit="D", R=NULL, kappa=1e-9,
         tab=NULL, graph=NULL, t.max=120)
Arguments

F    The n times m matrix of real numbers. The rows of F represent the m-dimensional regressors corresponding to n design points. It is assumed that n>=m>=2. Use od.m1 for models with 1-dimensional regressors. For D-optimality, the current implementation supports the models with m<=10.

b, A The real vector of length k and the k times n matrix of real numbers. The linear constraints A*w<=b, w^0<=w define the set of permissible designs w (where w^0 is a described below.) The argument A can also be NULL; in that case b must be a positive number and A is set to the 1 times n matrix of ones.

w^0 The non-negative vector of length n representing the design to be augmented. This argument can also be NULL; in that case, w^0 is set to the vector of zeros.

crit The optimality criterion. Possible values are "D", "A", "IV".

R    The region of summation for the IV-optimality criterion. The argument R must be a subvector of 1:n, or NULL. If R=NULL, the procedure uses R=1:n. Argument R is ignored if crit="D", or if crit="A".

kappa A small non-negative perturbation parameter.

tab A vector determining the regressor components to be printed with the resulting design. This argument should be a subvector of 1:n, or a subvector of colnames(F), or it can be NULL. If tab=NULL, the design is not printed.

graph A vector determining the regressor components to be plotted with the resulting design. This argument should be a subvector of 1:n, or a subvector of colnames(F), or it can be NULL. If graph=NULL, the resulting design is not visualized.

t.max The time limit for the computation.

Details

The procedure computes an efficient approximate design by converting the optimal design problem to a specific problem of second-order cone programming; see the reference for details. The advantage of this approach is the possibility to construct approximate designs under a general system of linear constraints. In particular, this function provides means of computing informative lower bounds on the efficiency of the linearly constrained exact designs computed using methods such as od.RC, od.IQP, and od.MISOCP.

The model should be non-singular in the sense that there exists an approximate design w satisfying the constraints 0<=w^0<=w and A*w<=b, with a non-singular information matrix, preferably with the reciprocal condition number of at least 1e-5. If this requirement is not satisfied, the computation may fail, or it may produce a deficient design.

If the criterion of IV-optimality is selected, the region R should be chosen such that the associated matrix L (see the help page of the function od.crit) is non-singular, preferably with a reciprocal condition number of at least 1e-5. If this requirement is not satisfied, the computation may fail, or it may produce a deficient design.

The perturbation parameter kappa can be used to add n*m iid random numbers from the uniform distribution in [-kappa, kappa] to the elements of F before the optimization is executed. This can be helpful for increasing the numerical stability of the computation or for generating a random design from the potentially large set of optimal or nearly-optimal designs.
The performance depends on the problem and on the hardware used, but in most cases the function can compute an optimal or nearly-optimal exact design for a problem with a thousand design points within minutes of computing time. If the only constraint on the design is the standard constraint on the size, the function `od.AA` should be a preferred choice.

**Value**

A list with the following components:

- **method**: The method used for computing the design `w.best`.
- **w.best**: the best permissible design found, or NULL. The value of `w.best` will be NULL if the computation fails. This can happen, if no permissible solution is found within the time limit, no permissible solution exists, or the problem is unbounded; see the `status` variable for more details. Note that even if `w.best` is a permissible design, then it still can have a singular information matrix; cf. the `phi.best` variable.
- **Phi.best**: The value of the criterion of optimality of the design `w.best`. If `w.best` has a singular information matrix or if the computation fails, the value of `Phi.best` will be 0.
- **status**: The status variable of the gurobi optimization procedure; see the gurobi solver documentation for details.
- **t.act**: The actual time taken by the computation.

**Author(s)**

Radoslav Harman, Lenka Filova

**References**


**See Also**

`od.AA`, `od.MISOCP`, `od.IQP`, `od.RC`

**Examples**

```r
if(require("gurobi")){
  # Suppose that we model the mean values of the observations of
  # a circadian rhythm by a third-degree trigonometric model on
  # a discretization of the interval [0, 2*pi]. We would like to
  # construct a D-efficient design.
  # However, the distance of successive times of observations should not be
  # smaller than the (1/72)-th of the interval (20 minutes, if the interval
  # represents one day). Also, we cannot perform more than 8 observations
  # in total, and more than 4 observations in the interval [2*pi/3, 2*pi]
  # (i.e., during the 16 non-working hours).

  # Create the matrix of regressors.
}
F.trig <- F.\text{cube}(\text{-}I(\cos(x)) + I(\sin(x)) + I(\cos(2 \times x)) + I(\sin(2 \times x)) + I(\cos(3 \times x)) + I(\sin(3 \times x)),
1 / 144, 2 \times \pi, 288)

# Create the constraints.
b.trig <- c(rep(1, 285), 12, 4)
A.trig <- matrix(0, nrow=287, ncol=288)
for(i in 1:285) A.trig[i, i:(i+3)] <- 1
A.trig[286,] <- 1; A.trig[287, 97:288] <- 1

# Compute the D-optimal approximate design under the constraints.
res.trig <- od.SOCP(F.trig, b.trig, A.trig, crit="D")

# Inspect the resulting approximate design.
od.plot(res.trig$\text{w.best})
od.print(round(res.trig$\text{w.best},2))

# It is clear that a very efficient exact design of size 8 satisfying
# the constraints performs the observations in design points
# 1, 34, 63, 96, 134, 173, 212, 251, i.e.
w.exact <- rep(0, 288)
w.exact[c(1, 34, 63, 96, 134, 173, 212, 251)] <- 1

# Indeed, the efficiency of this exact design relative to the optimal
# approximate design is:
od.crit(F.trig, w.exact) / od.crit(F.trig, res.trig$\text{w.best})

# Of course, it is also possible to directly use an exact-design function
# such as od.MISOCP for this problem, or it is possible to use
# the optimal approximate design to decrease the support size of
# the candidate design points set, and then use an exact-design
# procedure.

# See also the examples in the help files of functions od.RC, od.IQP
# and od.MISOCP, where od.SOCP is used to compute informative lower
# bounds on the efficiencies of exact designs.
Index

*Topic Optimal Design, A-optimality, D-optimality, IV-optimality

OptimalDesign-package, 2

*Topic

OptimalDesign-package, 2

F.cube, 2, 4
F.simplex, 3, 3

od.AA, 4, 14, 15, 28, 31
od.crit, 6, 7, 9, 14, 16, 19, 22, 23, 25, 28, 30
od.infmt, 8, 9, 22, 23
od.IQP, 10, 19, 20, 25, 26, 30, 31
od.KL, 6, 13, 19, 20, 25, 26, 28
od.m1, 5, 10, 13, 15, 18, 24, 27, 30
od.MISOCP, 12, 18, 25, 26, 30, 31
od.plot, 8, 9, 21, 23
od.print, 8, 9, 22, 22
od.RC, 12, 19, 20, 24, 27, 28, 30, 31
od.RCs, 6, 14, 15, 19, 20, 25, 26, 27
od.SOCP, 6, 12, 19, 20, 25, 26, 29
OptimalDesign (OptimalDesign-package), 2
OptimalDesign-package, 2