Package ‘SOAs’

August 10, 2023

Title Creation of Stratum Orthogonal Arrays
Version 1.4
Description Creates stratum orthogonal arrays (also known as strong orthogonal arrays). These are arrays with more levels per column than the typical orthogonal array, and whose low order projections behave like orthogonal arrays, when collapsing levels to coarser strata. Details are described in Groemping (2022) “A unifying implementation of stratum (aka strong) orthogonal arrays” <http://www1.bht-berlin.de/FB_I/II/reports/Report-2022-002.pdf>.
Depends R (>= 3.6.0), DoE.base (>= 1.2)
Imports stats, combinat, FrF2, igraph, lhs (>= 1.1.3), conf.design, sfsmisc, partitions
License GPL (>= 2)
Encoding UTF-8
URL https://github.com/bertcarnell/SOAs
BugReports https://github.com/bertcarnell/SOAs/issues
RoxygenNote 7.2.3
Suggests testthat (>= 3.0.0)
Config/testthat/edition 3
NeedsCompilation no
Author Ulrike Groemping [aut, cre], Rob Carnell [ctb]
Maintainer Ulrike Groemping <ulrike.groemping@bht-berlin.de>
Repository CRAN
Date/Publication 2023-08-10 17:50:02 UTC

R topics documented:

SOAs-package .................................................. 2
contr.FbHelmert ........................................... 4
contr.Power .................................................. 5
contr.TianXu ............................................... 6
SOAs-package

Description

Creates stratum orthogonal arrays (also known as strong orthogonal arrays).

Details

This package constructs arrays in $s^el$ levels from orthogonal arrays in $s$ levels. These are all based on equations of the type

$$D = s^{el-1}A_1 + \ldots + sA_{el-1} + A_{el},$$

or for $s^2$ levels,

$$D = sA + B$$

and for $s^3$ levels,

$$D = s^2A + sB + C.$$  

The constructions differ in how they obtain the ingredient matrices, and what properties can be guaranteed for the resulting $D$. Where a construction function guarantees orthogonal columns for all matrices $D$ it produces, its name starts with a OSOA, otherwise with SOA.

If optimization is requested (default TRUE), space filling properties of $D$ are improved using a level permutation algorithm by Weng (2014). This algorithm is applied for improving the $\text{phi}_p$ criterion, which is often a reasonable surrogate for increasing the minimum distance.

in unified notation. The constructions by Zhou and Tang (2019) and Li et al. (2021) are very close to each other and are both implemented in the three functions OSOAs, OSOAs_hadamard and OSOAs_regular.

Within the package, available SOA constructions for specific situations can be queried using the guide functions guide_SOAs and guide_SOAs_from_OA.

Besides the construction functions, properties of the resulting array D can be checked using the aforementioned function phi_p as well as check functions ocheck, ocheck3 for orthogonality and soacheck2D, soacheck3D for (O)SOA stratification properties, and Spattern for the space-filling pattern proposed by Tian and Xu (2022); the implementation of the latter will presumably become more important than the 2D and 3D check functions eventually.

There is one further construction, maximin distance level expansion (XiaoXuMDLE, MDLEs), that does not yield stratum (aka strong) orthogonal arrays and is available for comparison only (Xiao and Xu 2018).

Author(s)
Author: Ulrike Groemping, BHT Berlin. Contributor: Rob Carnell.

References


**See Also**

Useful links:
- [https://github.com/bertcarnell/SOAs](https://github.com/bertcarnell/SOAs)
- Report bugs at [https://github.com/bertcarnell/SOAs/issues](https://github.com/bertcarnell/SOAs/issues)

---

**contr.FFbHelmert**

*Full-factorial-based real-valued contrasts for $s^e$l levels*

**Description**

Full-factorial-based real-valued contrasts for $s^e$l levels
Full-factorial-based polynomial contrasts for $s^e$l levels

**Usage**

```r
contr.FFbHelmert(n, s, contrasts = TRUE, slowfirst = TRUE)
contr.FFbPoly(n, s, contrasts = TRUE, slowfirst = TRUE)
```

**Arguments**

- `n` integer or vector; either an integer number of levels of the factor for which contrasts are created, which must be a a power of `s`; or a factor whose number of levels is a power of `s`; or a vector of levels whose number of elements is a power of `s`.
- `s` positive integer, at least 2
- `contrasts` logical; must be TRUE
- `slowfirst` logical; default TRUE

**Details**

The functions implement real-valued full-factorial-based contrasts in the sense of Groemping (2023b) that can be used instead of the complex-valued contrasts from Tian and Xu (2022), as implemented in function `contr.TianXu`. Their main use is the calculation of the stratification pattern (also called space-filling pattern). Function `Spattern` uses function `contr.FFbHelmert` for this purpose, the internal function `Spattern_Poly` uses `contr.FFbPoly`.

**Value**

`contr.FFbHelmert` and `contr.FFbPoly` yield a matrix of real-valued contrasts. That matrix can be used in function `model.matrix` or in any statistical modeling functions.
### contr.Power

A contrast function based on regular factorials for number of levels a prime or prime power

#### Description

A contrast function based on regular factorials for number of levels a prime or prime power

#### Usage

```r
contr.Power(n, s = 2, contrasts = TRUE)
```

#### Arguments

- `n`: integer or vector; either an integer number of levels of the factor for which contrasts are created, which must be a a power of `s`; or a factor whose number of levels is a power of `s`; or a vector of levels whose number of elements is a power of `s`.
- `s`: integer; prime or prime power
- `contrasts`: logical; must be TRUE

#### Details

The function is a generalization (with slowest first instead of fastest first) of function `contr.FrF2` from package `DoE.base`. It is in this package because it needs Galois field functionality from package `lhs` for non-prime `s`. Its purpose is (was) the calculation of the stratification (or space-filling) pattern by Tian and Xu (2022), see also Groemping (2022). The package now calculates the pattern with function `contr.TianXu`.

#### Examples

```r
## the same n can yield different contrasts for different s
## Helmert variant
contr.FFbHelmert(16, 2)
round(contr.FFbHelmert(16, 4), 4)
round(contr.FFbHelmert(16, 16), 4)
## Poly variant
contr.FFbHelmert(16, 2)
round(contr.FFbHelmert(16, 4), 4)
round(contr.FFbHelmert(16, 16), 4)
```
Value
contr.TianXu yields a matrix of contrasts. It can be used in function model.matrix or anywhere where factors with the number of levels a power of $s^s$ are used with contrasts. The exponent for $s$ is determined from the number of levels.

References
Groemping (2022) Tian and Xu (2022)

Examples
## the same n can yield different contrasts for different s
contr.TianXu(16, 2)
contr.TianXu(16, 4)

contr.TianXu A complex-valued contrast function for $s^\text{el}$ levels based on powers of the $s$-th root of the unity

Description
A complex-valued contrast function for $s^\text{el}$ levels based on powers of the $s$-th root of the unity

Usage
contr.TianXu(n, s = 2, contrasts = TRUE)

Arguments
n integer or vector; either an integer number of levels of the factor for which contrasts are created, which must be a a power of $s$; or a factor whose number of levels is a power of $s$; or a vector of levels whose number of elements is a power of $s$.
s positive integer, at least 2
contrasts logical; must be TRUE

Details
The function implements the complex-valued contrasts from Tian and Xu (2022). Its sole use is the calculation of the stratification pattern (also called space-filling pattern). However, note that it is not used in function Spattern, but only in the internal function Spattern_TianXu, which yields exactly the same results as function Spattern.
The contrasts argument has been kept in order to be prepared in case the model.matrix function gains the ability to handle complex-valued contrasts.
The Tian and Xu contrasts are full-factorial-based contrasts in the sense of Groemping (2023b). Function Spattern uses a different type of full-factorial-based contrasts, the full-factorial-based Helmert contrasts provided in function contr.FFbHelmert.
**Value**

contr.TianXu yields a matrix of complex-valued contrasts. It can therefore NOT be used in function `model.matrix` or in statistical modeling functions.

**References**

Groemping (2023b) Tian and Xu (2022)

**Examples**

```r
## the same n can yield different contrasts for different s
contr.TianXu(16, 2)
contr.TianXu(16, 4)
round(contr.TianXu(16, 16), 4)
```

---

**createSaturated**  
*Function to create a regular saturated strength 2 array*

**Description**

produces an OA($s^k$, $(s^k-1)/(s-1)$, $s$, 2) (Rao-Hamming construction)

**Usage**

```r
createSaturated(s, k = 2)
```

**Arguments**

- `s`  
  the prime or prime power to use
- `k`  
  integer; determines the run size: the resulting array will have $s^k$ runs

**Details**

For many situations, the saturated fractions produced by this function are not the best choice for direct use in experimentation, because they heavily confound main effects with interactions. If not all columns are needed, using the last $m$ columns may yield better results than using the first $m$ columns. If possible, stronger OAs from other sources can be used, e.g. from package `FrF2` for 2-level factors or from package `DoE.base` for factors with more than 2 levels.

**Value**

createSaturated returns an $s^k$ times $(s^k-1)/(s-1)$ matrix (saturated regular OA with $s$-level columns)

**Examples**

```r
createSaturated(3, k=3)  ## 27 x 13 array in 3 levels
```
guide_SOAs

Utility function for inspecting available SOAs for which the user need not provide an OA

Description

Utility function for inspecting available SOAs for which the user need not provide an OA

Usage

guide_SOAs(s = 2, el = 3, m = NULL, n = NULL, ...)

Arguments

s  required (default: 2); prime or prime power on which the SOA is based
e1  required (default: 3); the power to which s is to be taken, i.e. the SOA will have columns with s^e1 levels
m  the number of columns needed (optional)
n  the maximum number of runs that are acceptable (optional); should be a multiple of s^e1; must not be smaller than m+1, if m is specified
...  currently unused

Details

The function provides the possible creation variants of an SOA that has m columns in s^e1 levels in up to n runs. It is permitted to specify m OR n only; in that case the function provides constructions with the smallest n or the largest m, respectively.

If both m and n are omitted, the function returns the smallest possible (O)SOA constructions for s^e1 levels that can be obtained without providing an OA.

Value

The function returns a data frame, each row of which contains a possibility; if no SOAs exist, the data.frame has zero rows. There is example code for constructing the SOA. Code details must be adjusted by the user (see the documentation of the respective functions). #'

Author(s)

Ulrike Groemping

References

For full detail, see SOAs-package.
Groemping (2023a)
He, Cheng and Tang (2018)
Li, Liu and Yang (2021)
Shi and Tang (2020)
Zhou and Tang (2019)

See Also

guide_SOAs_from_OA

Examples

```r
## guide_SOAs
## There is a Zhou and Tang type SOA with 4-level columns in 8 runs
guide_SOAs(2, 2, n=8)
## There are no SOAs with 8-level columns in 8 runs
guide_SOAs(2, 3, n=8)
## What SOAs based on s=2 in s^3 levels with 7 columns
## can be construct without providing an OA?
guide_SOAs(2, 3, m=7)
## pick the Shi and Tang family 3 design
myST_3plus <- SOAs_8level(n=32, m=7, constr='ShiTang_alphabeta')
## Note that the design has orthogonal columns and strength 3+,
## i.e., very good balance properties.
```

**guide_SOAs_from_OA**

Utility function for inspecting SOAs obtainable from an OA

**Description**

Utility function for inspecting SOAs obtainable from an OA

**Usage**

```r
guide_SOAs_from_OA(s, nOA, mOA, tOA, el = tOA, ...)
```

**Arguments**

- `s` required; the unique number of levels of the columns of a given OA (need not be prime or prime power)
- `nOA` required; the number of runs of the OA
- `mOA` required; the number of columns of the OA
- `tOA` required; the strength of the OA; strengths larger than 5 are reduced to 5; `el` must not be larger than the (reduced) strength, except for `tOA=2` with `el=3`, which is supported by the LLY algorithm
- `el` the power to which `s` is to be taken, i.e. the SOA will have columns with `s^el` levels; default: `tOA`, except for `tOA=2` and `el=3`, `el` can be chosen smaller than `tOA`, but not larger. If `el` is smaller than `tOA`, `tOA` is internally reduced before working out the possibilities.
... currently unused

Details

The function provides the possible creation variants of an SOA from a strength $tO_A$ OA with $mO_A$ $s$-level columns in $nO_A$ runs, for an SOA that has columns in $s^{e_1}$ levels. Note that the SOA may have $nO_A$ runs or $s*nO_A$ runs, depending on the construction.

Value

The function returns a data frame, each row of which contains a possibility. There is example code for constructing the SOA. The code assumes that a given OA has the name $O_A$; this can of course be modified by the user. Further code details can also be adjusted by the user (see the documentation of the respective functions).

Author(s)

Ulrike Groemping

References

For full detail, see SOAs-package.
Groemping (2023a)
He and Tang (2013)
He, Cheng and Tang (2018)
Liu and Liu (2015)
Li, Liu and Yang (2021)
Shi and Tang (2020)
Zhou and Tang (2019)

See Also

guide_SOAs

Examples

```r
>> guide_SOAs_from_OA
>> there is an OA(81, 3^10, 3) (L81.3.10 in package DoE.base)
>> inspect what can be done with it:
guide_SOAs_from_OA(s=3, mO_A=10, nO_A=81, tO_A=3)
>> the output shows that a strength 3 OSOA
>> with 4 columns of 27 levels each can be obtained in 81 runs
>> and provides the necessary code (replace O_A with L81.3.10)
>> optimize=FALSE reduces example run time
OSOAs_LiLiu(L81.3.10, t=3, optimize=FALSE)
>> or that an SOA with 9 non-orthogonal columns can be obtained
>> in the same number of runs
SOAs(L81.3.10, t=3)
```
Description

bound for number of columns for LiuLiu OSOAs

Usage

mbound_LiuLiu(moa, t)

Arguments

moa  
number of oa columns

t  
strength used in the construction in function OSOAs_LiuLiu (it is assumed that the oa used has at least that strength)

Value

the maximum number of columns that can be obtained by the command OSOAs_LiuLiu(oa, t=t) where oa has at least strength t and consists of moa columns

Author(s)

Ulrike Groemping

References

For full detail, see SOAs-package.
Liu and Liu 2015

Examples

```r
## moa is the number of columns of an oa
moa <- rep(seq(4,40),3)
## t is the strength used in the construction
## the oa must have at least this strength
t <- rep(2:4, each=37)
## numbers of columns for the combination
mbounds <- mapply(mbound_LiuLiu, moa, t)
## depending on the number of levels
## the number of runs can be excessive
## for larger values of moa with larger t!
## t=3 and t=4 have the same number of columns, except for moa=4*j+3
plot(moa, mbounds, pch=t, col=t)
```
**MDLEs**

*Function to create maximin distance level expanded arrays*

**Description**

Maximin distance level expansion similar to Xiao and Xu is implemented, using an optimization algorithm that is less demanding than the TA algorithm of Xiao and Xu.

**Usage**

```r
MDLEs(
  oa,
  ell,
  noptim.rounds = 1,
  optimize = TRUE,
  noptim.oa = 1,
  dmethod = "manhattan",
  p = 50
)
```

**Arguments**

- `oa` matrix or data.frame that contains an ingoing symmetric OA. Levels must be denoted as 0 to s-1 or as 1 to s.
- `ell` the multiplier for each number of levels
- `noptim.rounds` the number of optimization rounds; optimization may take very long, therefore the default is 1, although more rounds are beneficial.
- `optimize` logical: if FALSE, suppress optimization of expansion levels
- `noptim.oa` integer: number of optimization rounds applied to initial oa itself before starting expansion
- `dmethod` distance method for `phi_p`, "manhattan" (default) or "euclidean"
- `p` p for `phi_p` (the larger, the closer to maximin distance)

**Details**

The ingoing oa is possibly optimized for space-filling, using function `phi_optimize` with `noptim.oa` optimization rounds. The expansions themselves are again optimized for improving `phi_p`, using an algorithm which is a variant of Weng (2014), instead of the more powerful but also much more demanding algorithm proposed by Xiao and Xu.

**Value**

A matrix of class `MDLE` with attributes

- `phi_p` the `phi_p` value that was achieved
- `type` MDLE
**optimized** logical: same as the input parameter
**call** the call that produced the matrix

**permpick** matrix of lists of length s with elements from 0 to ell-1; matrix element (i,j) contains the sequence of replacements used in function DcFromDp for constructing the level expansion of the ith level in the jth column

**Author(s)**

Ulrike Groemping

**References**

For full detail, see [SOAs-package](https://example.com).
Weng (2014)
Xiao and Xu (2018)

**Examples**

```r
dim(aus <- MDLEs(DoE.base::L16.4.5, 2, noptim.rounds = 1))
permpicks <- attr(aus, "permpick")
## for people interested in internal workings:
## the code below produces the same matrix as MDLEs
SOAs::DcFromDp(L16.4.5-1, 4,2, lapply(1:5, function(obj) permpicks[,obj]))
```

---

### ocheck

**functions to evaluate low order projection properties of (O)SOAs**

#### Description

ocheck and ocheck3 evaluate pairwise or 3-orthogonality of columns, count_npairs evaluates the number of level pairs in 2D projections, count_nallpairs calculates corresponding total numbers of pairs.

#### Usage

```r
ocheck(D, verbose = FALSE)
ocheck3(D, verbose = FALSE)
count_npairs(D, minn = 1)
count_nallpairs(ns)
```
Arguments

D a matrix with factor levels or an object of class SOA;
   factor levels can start with 0 or with 1, and need to be consecutively numbered
verbose logical; if TRUE, additional information is printed (table of correlations)
minn small integer number; the function counts pairs that are covered at least minn times
ns vector of numbers of levels for each column

Value

Functions ocheck and ocheck3 return a logical.

Functions count_npairs returns a vector of counts for level combinations covered in factor pairs
   (in the order of the columns of DoE.base::nchoosek(ncol(D),2)) for the array in D,
function count_nallpairs provides the total number of level combinations for designs with num-
   bers of levels given in ns (and thus can be used to obtain a denominator for count_npairs).

Author(s)

Ulrike Groemping

Examples

#' ## Shi and Tang strength 3+ construction in 7 8-level factors for 32 runs
D <- SOAs_8level(32, optimize=FALSE)
   ## is an OSOA
   ocheck(D)

   ## an OSOA of strength 3 with 3-orthogonality
   ## 4 columns in 27 levels each
   ## second order model matrix
D.o <- OSOAs.LiuLiu(DoE.base::L81.3.10, optimize=FALSE)
ocheck3(D.o)

   ## benefit of 3-orthogonality for second order linear models
   colnames(D.o) <- paste0("X", 1:4)
y <- stats::rnorm(81)
mylm <- stats::lm(y~(X1+X2+X3+X4)^2 + I(X1^2)+I(X2^2)+I(X3^2)+I(X4^2),
   data=as.data.frame(scale(D.o, scale=FALSE)))
crossprod(stats::model.matrix(mylm))

OSOAs

Function to create an OSOA from an OA

Description

An OSOA in ns runs of strength 2^s (s^3 levels) or 2+ (s^2 levels) is created from an OA(n,m,s,2).
Usage

```r
OSOAs(
  oa, 
  el = 3, 
  m = NULL, 
  noptim.rounds = 1, 
  noptim.repeats = 1, 
  optimize = TRUE, 
  dmethod = "manhattan", 
  p = 50 
)
```

Arguments

- `oa`: matrix or data.frame that contains an ingoing symmetric OA. Levels must be denoted as 0 to \( s-1 \) or as 1 to \( s \).
- `el`: the exponent of the number of levels, \( el=3 \) yields a strength 2* OSOA in \( s^3 \) levels, \( el=2 \) a strength 2+ OSOA in \( s^2 \) levels
- `m`: the desired number of columns of the resulting array; odd values of \( m \) will be reduced by one, so specify the next largest even \( m \), if you need an odd number of columns (the function will do so, if possible; if \( m=NULL \), the maximum possible value is used.
- `noptim.rounds`: the number of optimization rounds for each independent restart
- `noptim.repeats`: the number of independent restarts of optimizations with `noptim.rounds` rounds each
- `optimize`: logical: should space filling be optimized by level permutations?
- `dmethod`: distance method for `phi_p`, "manhattan" (default) or "euclidean"
- `p`: \( p \) for `phi_p` (the larger, the closer to maximin distance)

Details

The function implements the algorithms proposed by Zhou and Tang 2018 (\( s^2 \) levels) or Li, Liu and Yang 2021 (\( s^3 \) levels). Both are enhanced with the modification for matrix A by Groemping 2022. Level permutations are optimized using an adaptation of the algorithm by Weng (2014).

Suitable OAs for argument `oa` can e.g. be constructed with OA creation functions from package `lhs` or can be obtained from arrays listed in R package `DoE.base`.

Value

matrix of class SOA with the attributes that are listed below. All attributes can be accessed using function `attributes`, or individual attributes can be accessed using function `attr`. These are the attributes:

- `type`: the type of array (SOA or OSOA)
- `strength`: character string that gives the strength
- `phi_p`: the `phi_p` value (smaller=better)
optimized logical indicating whether optimization was applied
perms2pickfrom optional element, when optimization was conducted: the overall permutation list to which the numbers in permlist refer
call the call that created the object

Author(s)
Ulrike Groemping

References
For full detail, see SOAs-package.
Groemping (2023a)
Li, Liu and Yang (2021)
Weng (2014)
Zhou and Tang (2019)

Examples
## run with optimization for actual use!
## 54 runs with seven 9-level columns
OSOAs(DoE.base::L18[,3:8], el=2, optimize=FALSE)

## 54 runs with six 27-level columns
OSOAs(DoE.base::L18[,3:8], el=3, optimize=FALSE)

## 81 runs with four 9-level columns
OSOAs(DoE.base::L27.3.4, el=2, optimize=FALSE)
## An OA with 9-level factors (L81.9.10)
## has complete balance in 2D,
## however does not achieve 3D projection for
## all four collapsed triples
## It is up to the user to decide what is more important.
## I would go for the OA.

## 81 runs with four 27-level columns
OSOAs(DoE.base::L27.3.4, el=3, optimize=FALSE)

OSOAs_hadamard function to create a strength 3 OSOA with 8-level columns or a strength 3- OSOA with 4-level columns from a Hadamard matrix

Description
A Hadamard matrix in k runs is used for creating an OSOA in n=2k runs for at most m=k-2 columns (8-level) or m=k-1 columns (4-level).
Usage

```
OSOAs_hadamard(
  m = NULL,
  n = NULL,
  el = 3,
  noptim.rounds = 1,
  noptim.repeats = 1,
  optimize = TRUE,
  dmethod = "manhattan",
  p = 50
)
```

Arguments

- **m**: the number of columns to be created; if `n` is also given, `m` must be compatible with it; at present, `m` can be at most 98.
- **n**: the number of runs to be created; `n` must be a multiple of 8 and can (at present) be at most 200; if `m` is also given, `n` must be compatible with it.
- **el**: exponent for 2, can be 2 or 3: the OSOA will have columns with $2^\text{el}$ (4 or 8) levels
- **noptim.rounds**: the number of optimization rounds for each independent restart
- **noptim.repeats**: the number of independent restarts of optimizations with `noptim.rounds` rounds each
- **optimize**: logical: should space filling be optimized by level permutations?
- **dmethod**: distance method for `phi_p`, "manhattan" (default) or "euclidean"
- **p**: `p` for `phi_p` (the larger, the closer to maximin distance)

Details

At least one of `m` or `n` must be provided. For `el=2`, Zhou and Tang (2019) strength 3- designs are created, for `el=3` strength 3 designs by Li, Liu and Yang (2021). Li et al.’s creation of the matrix A has been enhanced by using a column specific fold-over, which is beneficial for the space-filling properties (see Groemping 2022).

Value

matrix of class SOA with the attributes that are listed below. All attributes can be accessed using function `attributes`, or individual attributes can be accessed using function `attr`. These are the attributes:

- **type**: the type of array (SOA or OSOA)
- **strength**: character string that gives the strength
- **phi_p**: the `phi_p` value (smaller=better)
- **optimized**: logical indicating whether optimization was applied
- **permpick**: matrix that lists the id numbers of the permutations used
permsth2pickfrom optional element, when optimization was conducted: the overall permutation list to which the numbers in permlist refer

call the call that created the object

Author(s)
Ulrike Groemping

References
For full detail, see SOAs-package.
Groemping (2023a)
Li, Liu and Yang (2021)
Weng (2014)
Zhou and Tang (2019)

Examples

```r
dim(OSOAs_hadamard(9, optimize=FALSE)) # 9 8-level factors in 24 runs
dim(OSOAs_hadamard(n=16, optimize=FALSE)) # 6 8-level factors in 16 runs
OSOAs_hadamard(n=24, m=6, optimize=FALSE) # 6 8-level factors in 24 runs
## (though 10 would be possible)
dim(OSOAs_hadamard(m=35, optimize=FALSE)) # 35 8-level factors in 80 runs
```

---

**OSOAs_LiuLiu**

*Function to create OSOAs of strengths 2, 3, or 4 from an OA*

**Description**

Creates OSOAs from an OA according to the construction by Liu and Liu (2015). Strengths 2 to 4 are covered. Strengths 3 and 4 guarantee 3-orthogonality.

**Usage**

```r
OSOAs_LiuLiu(
  oa,
  t = NULL,
  m = NULL,
  noptim.rounds = 1,
  noptim.repeats = 1,
  optimize = TRUE,
  dmethod = "manhattan",
  p = 50
)
```
Arguments

- **oa**: matrix or data.frame; a symmetric orthogonal array of strength at least \( t \)
- **t**: the requested strength of the OSOA
- **m**: the requested number of columns of the OSOA (at most \( \text{mbound}_{LiuLiu}(\text{ncol}(oa), t) \)).
- **noptim.rounds**: the number of optimization rounds for each independent restart
- **noptim.repeats**: the number of independent restarts of optimizations with \( \text{noptim.rounds} \) rounds each
- **optimize**: logical: should space filling be optimized by level permutations?
- **dmethod**: distance method for \( \phi_p \); "manhattan" (default) or "euclidean"
- **p**: \( p \) for \( \phi_p \) (the larger, the closer to maximin distance)

Details

The number of columns goes down dramatically with the requested strength. However, the strength 3 or 4 arrays may be worthwhile, because they guarantee 3-orthogonality, which implies that (quantitative) linear models with main effects and second order effects can be robustly estimated.

Optimization is less successful for this construction of OSOAs; for small arrays, the level permutations make (almost) no difference.

Function \( \text{mbound}_{LiuLiu}(\text{moa}, t) \) calculates the number of columns that can be obtained from a strength \( t \) OA with \( \text{moa} \) columns (if such an array exists, the function does not check that).

Ingoing arrays can be obtained from \( \text{oa} \)-generating functions of R package **lhs** like \( \text{createBoseBush} \), or from OAs in R package **DoE.base**, or from 2-level designs created with R package **FrF2** (see example section).

Value

matrix of class `SOA` with the attributes that are listed below. All attributes can be accessed using function `attributes`, or individual attributes can be accessed using function `attr`. These are the attributes:

- **type**: the type of array (SOA or OSOA)
- **strength**: character string that gives the strength
- **\( \phi_p \)**: the \( \phi_p \) value (smaller=better)
- **optimized**: logical indicating whether optimization was applied
- **permpick**: matrix that lists the id numbers of the permutations used
- **perms2pickfrom**: optional element, when optimization was conducted: the overall permutation list to which the numbers in permlist refer
- **call**: the call that created the object

Author(s)

Ulrike Groemping
References

For full detail, see SOAs-package.

Liu and Liu (2015)
Weng (2014)

Examples

## strength 2, very small (four 9-level columns in 9 runs)
OSOA9 <- OSOAs_LiuLiu(DoE.base::L9.3.4)

## strength 3, from a Plackett-Burman design of FrF2
## 10 8-level columns in 40 runs with OSOA strength 3
oa <- suppressWarnings(FrF2::pb(40)[,c(1:19,39)])
### columns 1 to 19 and 39 together are the largest possible strength 3 set
OSOA40 <- OSOAs_LiuLiu(oa, optimize=FALSE) ## strength 3, 8 levels
### optimize would improve phi_p, but suppressed for saving run time

## 9 8-level columns in 40 runs with OSOA strength 3
oa <- FrF2::pb(40,19)
### 9 columns would be obtained without the final column in oa
mbound_LiuLiu(19, t=3) ## example for which q=3
### t=3 has one more column than t=4
OSOA40_2 <- OSOAs_LiuLiu(oa, optimize=FALSE) ## strength 3, 8 levels
### optimize would improve phi_p, but suppressed for saving run time

## starting from a strength 4 OA
oa <- FrF2::FrF2(64,8)
## four 16 level columns in 64 runs with OSOA strength 4
OSOA64 <- OSOAs_LiuLiu(oa, optimize=FALSE) ## strength 4, 16 levels

### reducing the strength to 3 does not increase the number of columns
mbound_LiuLiu(8, t=3)
### reducing the strength to 2 doubles the number of columns
mbound_LiuLiu(8, t=2)

## eight 4-level columns in 64 runs with OSOA strength 2
OSOA64_2 <- OSOAs_LiuLiu(oa, t=2, optimize=FALSE)
### fulfills the 2D strength 2 property
soacheck2D(OSOA64_2, s=2, el=2, t=2)
### fulfills also the 3D strength 3 property
soacheck3D(OSOA64_2, s=2, el=2, t=3)

### fulfills also the 4D strength 4 property
DoE.base::GWLP(OSOA64_2)
### but not the 3D strength 4 property
soacheck3D(OSOA64_2, s=2, el=2, t=4)
### and not the 2D 4x2 and 2x4 stratification balance
soacheck2D(OSOA64_2, s=2, el=2, t=3)

## six 36-level columns in 72 runs with OSOA strength 2
oa <- DoE.base::L72.2.5.3.3.4.1.6.7[,10:16]
OSOA72 <- OSOAs_LiuLiu(oa, t=2, optimize=FALSE)
OSOAs_regular  

Function to create an OSOA in \( s^2 \) or \( s^3 \) levels and \( s^k \) runs from a basic number of levels \( s \) and a power \( k \)

Description

The OSOA in \( s^k \) runs accommodates at most \( m = (s^k(s-1) - 1)/2 \) columns in \( s^2 \) levels or \( m' = 2 \times \text{floor}(m/2) \) columns in \( s^3 \) levels.

Usage

```r
OSOAs_regular(
  s,
  k,
  el = 3,
  m = NULL,
  noptim.rounds = 1,
  noptim.repeats = 1,
  optimize = TRUE,
  dmethod = "manhattan",
  p = 50
)
```

Arguments

- **s**: the prime or prime power to use (do not use for \( s=2 \), because other method is better); the resulting array will have pairwise orthogonal columns in \( s^t \) levels
- **k**: integer \( \geq 3 \); determines the run size: the resulting array will have \( s^k \) runs
- **el**: 2 or 3; the exponent of the number of levels, \( el=3 \) yields a strength 2* OSOA in \( s^3 \) levels, \( el=2 \) a strength 2+ or 3- OSOA in \( s^2 \) levels
- **m**: the desired number of columns of the resulting array; for \( el=3 \), odd values of \( m \) will be reduced by one, so specify the next largest even \( m \), if you need an odd number of columns (the function will do so, if possible); if \( m=NULL \), the maximum possible value is used. This is at most \( (s^k(s-1) - 1)/(s-1) \), or one less if this is odd and \( el=3 \).
- **noptim.rounds**: the number of optimization rounds for each independent restart
- **noptim.repeats**: the number of independent restarts of optimizations with \( \text{noptim.rounds} \) rounds each
- **optimize**: logical: should space filling be optimized by level permutations?
- **dmethod**: distance method for \( \phi_p \), "manhattan" (default) or "euclidean"
- **p**: \( p \) for \( \phi_p \) (the larger, the closer to maximin distance)
Details

The function implements the algorithms proposed by Zhou and Tang 2018 (s^2 levels) or Li, Liu and Yang 2021 (s^3 levels), enhanced with the modification for matrix A by Groemping (2023a). Level permutations are optimized using an adaptation of the algorithm by Weng (2014).

If m is specified, the function uses the last m columns of a saturated OA produced by function `createSaturated(s, k-1)`.

If m is small enough that a resolution IV / strength 3 OA for s levels in s^(k-1) runs exists, function `OSOAs` should be used with such an OA (which can be obtained from package `FrF2` for s=2 or from package `DoE.base` for s>2). For s=2, function `OSOAs_hadamard` may also be a better choice than `OSOAs_regular` for up to 192 runs.

Value

matrix of class SOA with the attributes that are listed below. All attributes can be accessed using function `attributes`, or individual attributes can be accessed using function `attr`. These are the attributes:

- **type** the type of array (SOA or OSOA)
- **strength** character string that gives the strength
- **phi_p** the phi_p value (smaller=better)
- **optimized** logical indicating whether optimization was applied
- **permpick** matrix that lists the id numbers of the permutations used
- **perms2pickfrom** optional element, when optimization was conducted: the overall permutation list to which the numbers in permlist refer
- **call** the call that created the object

Author(s)

Ulrike Groemping

References

For full detail, see SOAs-package. Groemping (2023a)
Li, Liu and Yang (2021)
Weng (2014)
Zhou and Tang (2019)

Examples

```r
## 13 columns in 9 levels each
OSOAs_regular(3, 4, el=2, optimize=FALSE) ## 13 columns, phi_p about 0.117
# optimizing level permutations typically improves phi_p a lot
# OSOAs_regular(3, 4, el=2) ## 13 columns, phi_p typically below 0.055
```
phi_optimize

function to optimize the phi_p value of an array by level permutation

Description

takes an n x m array and returns an n x m array with improved phi_p value (if possible)

Usage

phi_optimize(
  D,
  noptim.rounds = 1,
  noptim.repeats = 1,
  dmethod = "manhattan",
  p = 50
)

Arguments

D numeric matrix or data.frame with numeric columns, n x m. A symmetric array (e.g. an OA) with nl levels for each columns. Levels must be coded as 0 to nl - 1 or as 1 to nl. levels from

noptim.rounds number of rounds in the Weng algorithm

noptim.repeats number of independent repeats of the Weng algorithm

dmethod distance method for phi_p, "manhattan" (default) or "euclidean"

p p for phi_p (the larger, the closer to maximin distance)

Details

The function uses the algorithm proposed by Weng (2014) for SOA optimization:

It starts with a random permutation of column levels.

Initially, individual columns are randomly permuted (m permuted matrices, called one-neighbours), and the best permutation w.r.t. the phi_p value (manhattan distance) is made the current optimum. This continues, until the current optimum is not improved by a set of randomly drawn one-neighbours.

Subsequently, pairs of columns are randomly permuted (choose(m,2) permuted matrices, called two-neighbours). If the current optimum can be improved or the number of optimization rounds has not yet been exhausted, a new round with one-neighbours is started with the current optimum. Otherwise, the current optimum is returned, or an independent repeat is initiated (if requested).

Limited experience suggests that an increase of noptim.rounds from the default 1 is often helpful, whereas an increase of noptim.repeats did not yield as much improvement.

Value

an n x m matrix
phi_p

Functions to evaluate space filling of an array

Author(s)
Ulrike Groemping

References
For full detail, see SOAs-package.
Weng (2014)

Examples
oa <- lhs::createBoseBush(8,16)
print(phi_p(oa, dmethod="manhattan"))
oa_optimized <- phi_optimize(oa)
print(phi_p(oa_optimized, dmethod="manhattan"))

phi_p

phi_p calculates the discrepancy

phi_p calculates the discrepancy

Usage
phi_p(D, dmethod = "manhattan", p = 50)

mindist(D, dmethod = "manhattan")

phi_p(D, dmethod = "manhattan", p = 50)

Arguments
D an array or an object of class SOA or MDLE
dmethod the distance to use, "manhattan" (default) or "euclidean"
p the value for p to use in the formula for phi_p

Details
Small values of phi_p tend to be associated with good performance on the maximin distance criterion, i.e. with a larger minimum distance.
small values of phi_p are associated with good performance on the maximin distance criterion

Value
both functions return a number
a number
Author(s)

Ulrike Groemping
Ulrike Groemping

Examples

A <- DoE.base::L25.5.6  ## levels 1:5 for each factor
phi_p(A)
mindist(A)  # 5
A2 <- phi_optimize(A)
phi_p(A2)  ## improved
mindist(A2)  ## 6, improved
A <- DoE.base::L16.4.5  ## levels 1:4 for each factor
phi_p(A)
phi_p(A, dmethod="euclidean")
A2 <- A
A2[,4] <- c(2,4,3,1)[A[,4]]
phi_p(A2)
## Not run:
## A2 has fewer minimal distances
par(mfrow=c(2,1))
hist(dist(A), xlim=c(2,6), ylim=c(0,40))
hist(dist(A2), xlim=c(2,6), ylim=c(0,40))
## End(Not run)

Description

Print Methods

Usage

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

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

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

## S3 method for class 'dimwt_tab'
print(x, ...)
SOAs

Arguments

x object to be printed (SOA, OSOA, MDLE, Spattern)
... further arguments for function print

Value

no value is returned

Examples

myOSOA <- OSOAs_reg(s=3, k=3, optimize=FALSE)
myOSOA
str(myOSOA) ## structure for comparison
Spat <- Spattern(myOSOA, s=3)
dim_wt_tab(Spat) ## print method prints NAs as .
print(dim_wt_tab(Spat), na.print=" ")

SOAs function to create SOAs of strength t with the GOA construction by He and Tang.

Description

takes an OA(n,m,s,t) and creates an SOA(n,m',s^t',t') with t'<=t.

Usage

SOAs(
  oa,
  t = 3,
  m = NULL,
  noptim.rounds = 1,
  noptim.repeats = 1,
  optimize = TRUE,
  dmethod = "manhattan",
  p = 50
)

Arguments

oa matrix or data.frame that contains an ingoing symmetric OA. Levels must be denoted as 0 to s-1 or as 1 to s.

m the requested number of columns (see details for permitted numbers of columns)

noptim.rounds the number of optimization rounds for each independent restart
SOAs

noptim.repeats the number of independent restarts of optimizations with noptim.rounds rounds each
optimize logical, default TRUE; if FALSE, suppresses optimization
dmethod method for the calculation of phi_p, "manhattan" (default) or "euclidean"
p p for phi_p (the larger, the closer to maximin distance)

Details
The resulting SOA will have at most m' columns in s^t levels and will be of strength t. m'(m, t) is a function of the number of columns of oa (denoted as m) and the strength t: m'(m,2)=m, m'(m,3)=m-1, m'(m,4)=floor(m/2), m'(m,5)=floor((m-1)/2).

Suitable OAs for argument oa can e.g. be constructed with OA creation functions from package lhs or can be obtained from arrays listed in R package DoE.base.

Value
matrix of class SOA with the attributes that are listed below. All attributes can be accessed using function attributes, or individual attributes can be accessed using function attr. These are the attributes:

  type the type of array (SOA or OSOA)
  strength character string that gives the strength
  phi_p the phi_p value (smaller=better)
  optimized logical indicating whether optimization was applied
  permpick matrix that lists the id numbers of the permutations used
  perms2pickfrom optional element, when optimization was conducted: the overall permutation list to which the numbers in permlist refer
  call the call that created the object

Author(s)
Ulrike Groemping

References
For full detail, see SOAs-package.
He and Tang (2013)
Weng (2014)

Examples
aus <- SOAs(DoE.base::L27.3.4, optimize=FALSE) ## t=3 is the default
dim(aus)
soacheck2D(aus, s=3, el=3) ## check for 2*
soacheck3D(aus, s=3, el=3) ## check for 3
aus2 <- SOAs(DoE.base::L27.3.4, t=2, optimize=FALSE)
 SOAs2plus_regular  

**Description**

creates an array in \(s^k\) runs with columns in \(s^2\) levels for prime or prime power \(s\)

**Usage**

```r
SOAs2plus_regular(
  s,
  k,
  m = NULL,
  orth = TRUE,
  old = FALSE,
  noptim.rounds = 1,
  noptim.repeats = 1,
  optimize = TRUE,
  dmethod = "manhattan",
  p = 50
)
```

**Arguments**

- **s**
  - prime or prime power
- **k**
  - array will have \(n=s^k\) runs; for \(s=2\), \(k\geq 4\) is needed, for \(s>2\), \(k\geq 3\) is sufficient
- **m**
  - optional integer: number of columns requested; if NULL, the maximum possible number of columns is created, which is \((s^k-1)/(s-1) - ((s-1)^k-1)/(s-2)\) for \(s>2\) and \(s^k-s^k1 - s^{(k-k1)} + 2\), with \(k1=\text{floor}(k/2)\), for \(s=2\); specifying a smaller \(m\) is beneficial not only for run time but also for possibly achieving a column-orthogonal array (see Details section)
- **orth**
  - logical: if FALSE, suppresses attempts for orthogonal columns and selects the first permissible column for each column of \(B\) (see Details section)
- **old**
  - logical, relevant for \(orth=TRUE\) only: if TRUE, limits possible columns for \(B\) to the columns not eligible for \(A\) (instead of the columns not used in \(A\)); should only be used for reproducing designs created by version 1.1 or earlier
- **noptim.rounds**
  - the number of optimization rounds for each independent restart
- **noptim.repeats**
  - the number of independent restarts of optimizations with **noptim.rounds** rounds each
optimize logical: should optimization be applied? default TRUE
dmethod method for the distance in phi_p, "manhattan" (default) or "euclidean"
p p for phi_p (the larger, the closer to maximin distance)

Details

The construction is by He, Cheng and Tang (2018), Prop.1 (C2) / Theorem 2 for s=2 and Theorem 4 for s>2.
B is chosen as an OA of strength 2, if possible, which yields orthogonal columns according to Zhou and Tang (2019). This is implemented using a matching algorithm for bipartite graphs from package igraph: the smaller m, the more likely that orthogonality can be achieved. However, strength 2+ SOAs are not usually advisable for m small enough that a strength 3 OA exists.
Optimization according to Weng has been added (separate level permutations in columns of A and B, noptim.rounds times). Limited tests suggest that a single round (noptim.rounds=1) often does a very good job (e.g. for s=2 and k=4), and further rounds do not yield too much improvement; there are also cases (e.g. s=5 with k=3), for which the unoptimized array has a better phi_p than what can be achieved by most optimization attempts from a random start.
The search for orthogonal columns can take a long time for larger arrays, even without optimization. If this is prohibitive (or not considered valuable), orth=FALSE causes the function to create the matrix B for equation D=2A+B with less computational effort.
The subsequent optimization, if not switched off, is of the same complexity, regardless of the value for orth. Its duration heavily depends on the number of optimization steps that are needed before the algorithm stops. This has not been systematically investigated; cases for which the total run time with optimization is shorter for orth=TRUE than for orth=FALSE have been observed.

With package version 1.2, the creation of SOAs has changed: Up to version 1.1, the columns of B were chosen only from those columns that were not eligible for A, whereas the new version chooses them from those columns that are not used for A. This increases the chance to achieve geometrically orthogonal columns.

Users who want to reproduce a design from an earlier version can use argument old.

Value

matrix of class SOA with the attributes that are listed below. All attributes can be accessed using function attributes, or individual attributes can be accessed using function attr. These are the attributes:

- **type** the type of array (SOA or OSOA)
- **strength** character string that gives the strength
- **phi_p** the phi_p value (smaller=better)
- **optimized** logical indicating whether optimization was applied
- **permpick** matrix that lists the id numbers of the permutations used
- **perms2pickfrom** optional element, when optimization was conducted: the overall permutation list to which the numbers in permlist refer
- **call** the call that created the object
Note

Strength 2+ SOAs can accommodate a large number of factors with reasonable stratified balance behavior. Note that their use is not usually advisable for m small enough that a strength 3 OA with \(s^2\) level factors exists.

Author(s)

Ulrike Groemping

References

For full detail, see SOAs-package.


Weng (2014)

Zhou and Tang (2019)

Examples

```r
## unoptimized OSOA with 8 16-level columns in 64 runs
## (maximum possible number of columns)
plan64 <- SOAs2plus_regular(4, 3, optimize=FALSE)
ocheck(plan64)  ## the array has orthogonal columns

## optimized SOA with 20 9-level columns in 81 runs
## (up to 25 columns are possible)
plan <- SOAs2plus_regular(3, 4, 20)
## many column pairs have only 27 level pairs covered
count_npairs(plan)
## an OA would exist for 10 9-level factors (DoE.base::L81.9.10)
## it would cover all pairs
## (SOAs are not for situations for which pair coverage
## is of primary interest)
```

---

**SOAs_8level**

Function to create 8-level SOAs according to Shi and Tang 2020

Description

creates strength 3 or 3+ SOAs with 8-level factors in \(2^k\) runs, \(k\) at least 4. These SOAs have at least some more balance than guaranteed by strength 3.
SOAs_8level

Usage

SOAs_8level(
  n,
  m = NULL,
  constr = "ShiTang_alphabeta",
  noptim.rounds = 1,
  noptim.repeats = 1,
  optimize = TRUE,
  dmethod = "manhattan",
  p = 50
)

Arguments

- **n**: run size of the SOA; power of 2, at least 16
- **m**: number of columns; at most \(\frac{5n}{16}\) for \(\text{constr} = \text{"ShiTang_alpha"}\) (exception: only 9 for \(n=32\)), at most \(\frac{n}{4}\) for \(\text{constr} = \text{"ShiTang_alphabeta"}\); for \(m=\text{NULL}\), defaults are \(m=\frac{5n}{16}\) and \(m=\frac{n}{4}-1\), respectively; the latter yields strength 3+.
- **constr**: construction method. Must be one of "ShiTang_alphabeta", "ShiTang_alpha". See Details section
- **noptim.rounds**: the number of optimization rounds for each independent restart
- **noptim.repeats**: the number of independent restarts of optimizations with noptim.rounds rounds each
- **optimize**: logical: should space filling be optimized by level permutations?
- **dmethod**: distance method for \(\phi_p\), "manhattan" (default) or "euclidean"
- **p**: \(p\) for \(\phi_p\) (the larger, the closer to maximin distance)

Details

The construction is implemented as described in Groemping (2023a).

The 8-level SOAs created by this construction have strength 3 and at least the additional property alpha, which means that all pairs of columns achieve perfect 4x4 balance, if consecutive level pairs (01, 23, 45, 67) are collapsed.

The "ShiTang_alphabeta" construction additionally yields perfect 4x2x2 balance, if one column is collapsed to 4 levels, while two further columns are collapsed to 2 levels (0123 vs 4567). With \(m = \frac{n}{4}\) columns, the "ShiTang_alphabeta" construction has a single pair of correlated columns, all other columns are uncorrelated, due to a modification of Shi and Tang’s column allocation that was proposed in Groemping (2023a).

For \(m \leq \frac{n}{4} - 1\), the "ShiTang_alphabeta" construction also yields perfect balance for 8x2 projections in 2D (i.e. if one original column with another column collapsed to two levels). Thus, it yields all strength 4 properties in 2D and 3D, which is called strength 3+. Furthermore, Groemping (2023a) proposed an improved choice of columns for matrix C that implies orthogonal columns in this case.
Value

matrix of class SOA with the attributes that are listed below. All attributes can be accessed using function `attributes`, or individual attributes can be accessed using function `attr`. These are the attributes:

- **type** the type of array (SOA or OSOA)
- **strength** character string that gives the strength
- **phi_p** the phi_p value (smaller=better)
- **optimized** logical indicating whether optimization was applied
- **permpick** matrix that lists the id numbers of the permutations used
- **perms2pickfrom** optional element, when optimization was conducted: the overall permutation list to which the numbers in permlist refer
- **call** the call that created the object

Author(s)

Ulrike Groemping

References

For full detail, see **SOAs-package**.

Groemping (2023a)
Shi and Tang (2020)
Weng (2014)

Examples

```r
## use with optimization for actually using such designs
## n/4 - 1 = 7 columns, strength 3+
SOAs_8level(32, optimize=FALSE)

## n/4 = 8 columns, strength 3 with alpha and beta
SOAs_8level(32, m=8, optimize=FALSE)

## 9 columns (special case n=32), strength 3 with alpha
SOAs_8level(32, constr="ShiTang_alpha", optimize=FALSE)

## 5*n/16 = 5 columns, strength 3 with alpha
SOAs_8level(16, constr="ShiTang_alpha", optimize=FALSE)
```
Spattern functions to evaluate stratification properties of (O)SOAs and GSOAs

Description
soacheck2D and soacheck3D evaluate 2D and 3D projections, Spattern calculates the stratification pattern by Tian and Xu (2022), and dim_wt_tab extracts and formats the dim_wt_tab attribute of Spattern.

Usage
Spattern(D, s, maxwt = 4, maxdim = NULL, verbose = FALSE, ...)
dim_wt_tab(pat, dimlim = NULL, wtlim = NULL, ...)
soacheck2D(D, s = 3, el = 3, t = 3, verbose = FALSE)
soacheck3D(D, s = 3, el = 3, t = 3, verbose = FALSE)

Arguments
D a matrix with factor levels or an object of class SOA or a data.frame object with numeric columns.
Functions soacheck2D and soacheck3D require levels that are consecutively numbered (starting with 0 or 1).
Function Spattern also works, if all columns of D have the same number of unique numeric values; the function will code them using power contrasts.
s the prime or prime power according to which the array is checked
maxwt maximum weight to be considered for the pattern (default: 4; see Details);
if the specified limit is larger than maxdim*el, it is reduced accordingly (where el is such that s^el is the number of levels)
maxdim maximum dimension to be considered for the pattern (default: NULL implies that maxdim=min(maxwt, ncol(D)); see also Details);
if the specified limit is larger than m=ncol(D), it is reduced to m
verbose logical; if TRUE, additional information is printed (for Spattern, status information during run time; for the SOAcheck... functions, confounded pair or triple projections with A2 or A3, respectively, or table of correlations)
... currently not used
pat an object of class Spattern
dimlim integer; limits the returned dimension rows to the rows from 1 up to dimlim; the bottom margin continues to include all dimensions that were used in calculating pat
wtlim integer; limits the returned weight columns to the columns from 1 up to wtlim; the right margin continues to include all weights that were used in calculating pat
the exponent so that the number of levels of the array is $s^{e_1}$ (if $s$ is not NULL).

t

the strength for which to look (2, 3, or 4), equal to the sum of the exponents in
the stratification dimensions; for example, soacheck2D considers

$s^2 x s$ 2D projections with $t=2$,

$s^2 x s$ and $s x s^2$ projections with $t=3$,

and $s^3 x s$, $s^2 x s^2$ and $s x s^3$ projections with $t=4$.

If $t=4$ and $e_1=2$, property gamma ($s^3 x s$ and $s x s^3$) is obviously impossible
and will not be part of the checks.

Details

Function Spattern calculates the stratification pattern or S pattern as proposed in Tian and Xu (2022) (under the name space-filling pattern); the details and the implementation in this function
are described in Groemping (2023b); the function uses the full-factorial-based Helmert contrasts.

Position $j$ in the S pattern shows the imbalance when considering $s^j$ strata. $j$ is also called the (total) weight. $j=1$ can occur for an individual column only. $j=2$ can be obtained either for an $s^2$ level version of an individual column or for the crossing of $s^1$ level versions of two columns, and so forth.

Obtaining the entire S pattern can be computationally demanding. The arguments maxwt and maxdim limit the effort (choose NULL for no limit):

maxwt gives an upper limit for the weight $j$ of the previous paragraph; if NULL, maxwt is set to maxdim+$e_1$.

maxdim limits the number of columns that are considered in combination.

When using a non-null maxdim, pattern entries for $j$ larger than maxdim can be smaller than if one
would not have limited the dimension. Otherwise, dimensionality is unlimited, which is equivalent
to specifying maxdim as the minimum of maxwt and $\text{ncol}(D)$.

Spattern with maxdim=2 and maxwt=t can be used as an alternative to soacheck2D,
and analogously Spattern with maxdim=2 and maxwt=t can be used as an alternative to soacheck3D.

An Spattern object object can be post-processed with function dim_wt_tab. That function splits
the S pattern into contributions from effect column groups of different dimensions, arranged with a
row for each dimension and a column for each weight. If Spattern was called with maxdim=NULL
and maxwt=NULL, the output object shows the GWLP in the right margin and the S pattern in the
bottom margin. If Spattern was called with relevant restrictions on dimensions (maxdim, default 4) and/or weights (maxwt, default 4), sums in the margins can be smaller than they would be for unconstrained dimension and weights.

Functions soacheck2D and soacheck3D were available before function Spattern; many of their
use cases can now be handled with Spattern instead. The functions are often fast to yield a FALSE outcome, but can be very slow to yield a TRUE outcome for larger designs.

The functions inspect 2D and 3D stratification, respectively. Each column must have $s^{e_1}$ levels. $t$
specifies the degree of balance the functions are asked to look for.

Function soacheck2D,

• with el=2, t=2, looks for strength 2 conditions ($s^2$ levels, $s x s$ balance),

• with el=2, t=3, looks for strength 2+ / 3- conditions ($s^2$ levels, $s^2 x s$ balance),

• with el=2, t=4, looks for the enhanced strength 2+ / 3- property alpha ($s^2$ levels, $s^2 x s^2$ balance).
• and with $el=3$, $t=4$, looks for strength 3+ / 4 conditions ($s^3$ levels, $s^3xs$ and $s^2x^2s^2$ balance).

Function soacheck3D,
• with $el=2$, $t=3$, looks for strength 3- conditions ($s^2$ levels, $sxss$ balance),
• with $el=t=3$, looks for strength 3 conditions ($s^3$ levels, $sxss$ balance),
• and with $el=3$, $t=4$, looks for strength 3+ / 4 conditions ($s^3$ levels, $s^2x^2s$ balance).

If `verbose=TRUE`, the functions print the pairs or triples that violate the projection requirements for 2D or 3D.

**Value**

Function Spattern returns an object of class Spattern that is a named vector with attributes:
- The attribute `call` holds the function call (and thus documents, e.g., limits set on dimension and/or weight).
- The attribute `dim_wt_tab` holds a table of contributions split out by dimension (rows) and weights (columns), which has class `dim_wt_tab` and the further attribute `Spattern-class`.

Function `dim_wt_tab` returns the `dim_wt_tab` attribute of an object of class Spattern; note that the object contains NA values for combinations of dimension and weight that cannot occur.

Function `dim_wt_tab` postprocesses an Spattern object and produces a table that holds the S pattern entries separated by the dimension of the contributing effect column group (rows) and the weight of the effect column micro group (columns). The margin shows row and column sums (see Details section for caveats).

**References**

For full detail, see SOAs-package.
- Groemping (2023a)
- Groemping (2023b)
- He and Tang (2013)
- Shi and Tang (2020)
- Tian and Xu (2022)

**Examples**

```r
nullcase <- matrix(0:7, nrow=8, ncol=4)
soacheck2D(nullcase, s=2)
soacheck3D(nullcase, s=2)
Spattern(nullcase, s=2)
Spattern(nullcase, s=2, maxdim=2)
  ## the non-zero entry at position 2 indicates that
  ## soacheck2D does not comply with $t=2$
(Spat <- Spattern(nullcase, s=2, maxwt=4))
  ## comparison to maxdim=2 indicates that
  ## the contribution to S_4 from dimensions
  ## larger than 2 is 1
## postprocessing Spat
dim_wt_tab(Spat)
```
## Shi and Tang strength 3+ construction in 7 8-level factors for 32 runs

D <- SOAs_8level(32, optimize=FALSE)

## check for strength 3+ (default el=3 is OK)
## 2D check
soacheck2D(D, s=2, t=4)

## 3D check
soacheck3D(D, s=2, t=4)

## using Spattern (much faster for many columns)
## does not have strength 4
Spattern(D, s=2)

## but complies with strength 4 for dim up to 3
Spattern(D, s=2, maxwt=4, maxdim=3)

## inspect more detail
Spat <- (Spattern(D, s = 2, maxwt=5))
dim_wt_tab(Spat)
Index

* array
  - guide_SOAs, 8
  - guide_SOAs_from_OA, 9
  - 'SOAs-package' (SOAs-package), 2
  - _PACKAGE (SOAs-package), 2

attr, 15, 17, 19, 22, 27, 29, 32
attributes, 15, 17, 19, 22, 27, 29, 32

contr.FFbHelmer, 4, 6
contr.FFbPoly (contr.FFbHelmer), 4
contr.Power, 5
contr.Tianxu, 4, 5, 6
count_nallpairs (ocheck), 13
count_npairs (ocheck), 13
createSaturated, 7, 22

dim_wt_tab (Spattern), 33
DoE.base, 7

Frf2, 7

guide_SOAs, 3, 8, 10
guide_SOAs_from_OA, 3, 9, 9

mbound_luilli, 11
MDLEs, 3, 12
mindist (phi_p), 24

ocheck, 3, 13
ocheck3, 3
ocheck3 (ocheck), 13
OSOAs, 3, 14, 22
OSOAs_hadamard, 3, 16, 22
OSOAs_luilli, 2, 18
OSOAs_regular, 3, 21, 22

phi_optimize, 12, 23
phi_p, 2, 3, 12, 15, 17, 19, 21, 23, 24, 27, 29, 31

print.dim_wt_tab (print.SOA), 25
print.MDLE (print.SOA), 25
print.SOA, 25
print.Spatter (print.SOA), 25
socheck2D, 3
socheck2D (Spatter), 33
socheck3D, 3
socheck3D (Spatter), 33
SOAs, 2, 26
SOAs-package, 2
SOAs2plus_regular, 2, 28
SOAs_8level, 2, 30
Spatter, 3, 4, 6, 33
XiaoXuMDLE, 3