

Package: FielDHub (via r-universe)

October 12, 2026

Title A Shiny App for Design of Experiments in Life Sciences

Version 1.6.0

Description A shiny design of experiments (DOE) app that aids in the creation of traditional, un-replicated, augmented and partially-replicated designs applied to agriculture, plant breeding, forestry, animal and biological sciences.

Depends R (>= 4.1.0)

License MIT + file LICENSE

Imports dplyr (>= 1.0.0), blocksdesign, ggplot2 (>= 3.4.0), rlang, viridisLite, desplot (>= 1.11), shiny (>= 1.8.1), htmltools, DT, bslib, promises, shinyjs

Encoding UTF-8

RoxygenNote 7.3.2

Author Didier Murillo [cre, aut], Salvador Gezan [aut], Ana Heilman [ctb], Thomas Walk [ctb], Johan Aparicio [ctb], Matthew Seefeldt [ctb], Jean-Marc Montpetit [ctb], Richard Horsley [ctb], North Dakota State University [cph]

Maintainer Didier Murillo <didier.murilloflorez@ndsu.edu>

Suggests mirai, codetools, testthat (>= 3.2.0), spelling, knitr, rmarkdown

Config/testthat/edition 3

URL <https://github.com/DidierMurilloF/FielDHub>,
<https://didiermurillof.github.io/FielDHub/>

BugReports <https://github.com/DidierMurilloF/FielDHub/issues>

Language en-US

VignetteBuilder knitr

Config/pak/sysreqs cmake make libicu-dev libuv1-dev zlib1g-dev

Repository <https://didiermurillof.r-universe.dev>

Date/Publication 2026-10-12 19:27:00 UTC

RemoteUrl <https://github.com/didiermurillof/fieldhub>

RemoteRef HEAD

RemoteSha 720449effcebf525ef9cfaab128134f99c5ebda

Contents

alpha_lattice	3
CRD	5
design_arguments	7
design_formats	8
design_results	9
diagonal_arrangement	10
do_optim	14
field_layout	16
full_factorial	17
incomplete_blocks	19
latin_rectangle	21
latin_square	22
multi_location_prep	24
optimized_arrangement	27
partially_replicated	29
plot.FieldHub	33
print.FieldHub	34
print.fieldLayout	35
print.summary.FieldHub	35
RCBD	36
RCBD_augmented	39
read_design	41
rectangular_lattice	42
reproduce_design	44
reproduce_simulation	45
row_column	46
run_app	50
sparse_allocation	51
split_families	53
split_plot	54
split_split_plot	56
square_lattice	58
strip_plot	60
summary.FieldHub	62
swap_pairs	63
write_design	65

Index

alpha_lattice	<i>Generates an Alpha Design</i>
---------------	----------------------------------

Description

Randomly generates an alpha design like $\alpha(0,1)$ across multiple locations.

Usage

```
alpha_lattice(
  t = NULL,
  k = NULL,
  r = NULL,
  l = 1,
  plotNumber = 101,
  locationNames = NULL,
  seed = NULL,
  data = NULL,
  reps = NULL
)
```

Arguments

t	Number of treatments, or a character vector with the treatment labels.
k	Size of incomplete blocks (number of units per incomplete block).
r	Deprecated alias for reps; positional calls remain supported.
l	Number of locations. By default $l = 1$.
plotNumber	Numeric vector with the starting plot number for each location. By default $\text{plotNumber} = 101$.
locationNames	(optional) String with names for each of the l locations.
seed	(optional) Real number that specifies the starting seed to obtain reproducible designs.
data	(optional) Data frame with label list of treatments.
reps	Number of full resolvable replicates per location.

Value

A list with two elements.

- `infoDesign` is a list with information on the design parameters.
- `fieldBook` is a data frame with the alpha design field book.

Reproducibility

The result records effective inputs and the resolved seed in `metadata$parameters`, using `reps` for replication. Under the same package versions and RNG settings, rebuild a result `x` with `do.call(alpha_lattice, x$metadata$parameters)`.

Author(s)

Didier Murillo [aut], Salvador Gezan [aut], Ana Heilman [ctb], Thomas Walk [ctb], Johan Aparicio [ctb], Richard Horsley [ctb]

References

Edmondson., R. N. (2021). `blocksdesign`: Nested and crossed block designs for factorial and unstructured treatment sets. <https://CRAN.R-project.org/package=blocksdesign>

Examples

```
# Example 1: Generates an alpha design with 4 full blocks and 15 treatments.
# Size of IBlocks k = 3.
alphalattice1 <- alpha_lattice(t = 15,
                              k = 3,
                              reps = 4,
                              l = 1,
                              plotNumber = 101,
                              locationNames = "GreenHouse",
                              seed = 1247)

alphalattice1$infoDesign
head(alphalattice1$fieldBook, 10)

# Example 2: Generates an alpha design with 3 full blocks and 25 treatment.
# Size of IBlocks k = 5.
# In this case, we show how to use the option data.
treatments <- paste("G-", 1:25, sep = "")
ENTRY <- 1:25
treatment_list <- data.frame(list(ENTRY = ENTRY, TREATMENT = treatments))
head(treatment_list)
alphalattice2 <- alpha_lattice(t = 25,
                              k = 5,
                              reps = 3,
                              l = 1,
                              plotNumber = 1001,
                              locationNames = "A",
                              seed = 1945,
                              data = treatment_list)

alphalattice2$infoDesign
head(alphalattice2$fieldBook, 10)
```

CRD

*Generates a Completely Randomized Design (CRD)***Description**

It randomly generates a completely randomized design.

Usage

```
CRD(
  t = NULL,
  reps = NULL,
  plotNumber = 101,
  locationName = NULL,
  seed = NULL,
  data = NULL,
  locationNames = NULL
)
```

Arguments

t	An integer number with total number of treatments or a vector of dimension t with labels.
reps	One positive whole-number count of replicates per treatment.
plotNumber	Starting plot number. By default plotNumber = 101.
locationName	Deprecated spelling of locationNames. Existing positional calls remain supported, with a deprecation warning.
seed	(optional) Real number that specifies the starting seed to obtain reproducible designs.
data	(optional) Data frame whose first two columns contain unique treatment labels and positive whole-number replication counts. Extra columns are ignored; rows missing either selected value are omitted.
locationNames	(optional) Name of the single location. Supply only one of locationNames and locationName.

Details

The result records effective inputs and the resolved seed in `metadata$parameters`. Under the same package versions and RNG settings, rebuild a result `x` with `do.call(CRD, x$metadata$parameters)`. Data inputs are recorded after the existing column selection and normalization.

Value

A list with three elements.

- metadata records the design, schema/package versions, seed, random-number settings and effective input parameters.
- infoDesign is a list with information on the design parameters.
- fieldBook is a data frame with the CRD field book.

Author(s)

Didier Murillo [aut], Salvador Gezan [aut], Ana Heilman [ctb], Thomas Walk [ctb], Johan Aparicio [ctb], Richard Horsley [ctb]

References

Federer, W. T. (1955). Experimental Design. Theory and Application. New York, USA. The Macmillan Company.

Examples

```
# Example 1: Generates a CRD design with 10 treatments and 5 reps each.
crd1 <- CRD(
  t = 10,
  reps = 5,
  plotNumber = 101,
  seed = 1987,
  locationNames = "Fargo"
)
crd1$infoDesign
head(crd1$fieldBook, 10)

# Example 2: Generates a CRD design with 15 treatments and 6 reps each.
Gens <- paste("Wheat", 1:15, sep = "")
crd2 <- CRD(
  t = Gens,
  reps = 6,
  plotNumber = 1001,
  seed = 1654,
  locationNames = "Fargo"
)
crd2$infoDesign
head(crd2$fieldBook, 10)

# Example 3: Generates a CRD design with 12 treatments and 4 reps each.
# In this case, we show how to use the option data.
treatments <- paste("ND-", 1:12, sep = "")
treatment_list <- data.frame(list(TREATMENT = treatments, REP = 4))
head(treatment_list)
crd3 <- CRD(
  t = NULL,
  reps = NULL,
```

```

    plotNumber = 2001,
    seed = 1655,
    locationNames = "Cali",
    data = treatment_list
  )
  crd3$infoDesign
  head(crd3$fieldBook, 10)

```

design_arguments

Shared design arguments and compatibility

Description

Use named arguments when writing FieldHub scripts. The common names below describe the same concepts across the design families that support them. Each function's help page defines its supported inputs and constraints.

Common arguments

- t Treatments in classical designs: a count or, where supported, a vector of labels. Check-based designs use lines for the number of experimental entries excluding checks.
- reps Full replicates per location in classical, incomplete-block, lattice, row-column and strip-plot designs. In `latin_square()`, this is the number of independent squares. Partial and multi-location replication uses the design-specific allocation arguments instead.
- l Number of locations. `CRD()` and `latin_square()` generate one location and have no `l` argument.
- locationNames Location labels, in location order. `CRD()` takes one location name.
- plotNumber Starting plot number, or one start per location.
- seed Seed recorded in the result to reproduce randomization. When `seed` is `NULL`, one integer is drawn from the current random-number stream and recorded; the design's own randomization does not change the caller's stream.
- k Number of plots per incomplete block.
- checks A check count, entry identifiers, or labels; see "Dimensions, factors and checks" below.
- rep_checks Check replication: how many times each check is repeated. `RCBD()` takes one count per check label supplied in `checks` (recycled from a scalar). `optimized_arrangement()` takes either a total check count or one count per check label (this is `amountChecks` in 1.5.x scripts; see "Migrating existing scripts"). Multi-location allocations (`multi_location_prep()`, `do_optim(design = "prep")`) take the same one-count-per-check form. `sparse_allocation()` and `do_optim(design = "sparse")` have no `rep_checks`: every check is replicated once per location by construction.

Dimensions, factors and checks

`nrows` and `ncols` describe field dimensions in spatial designs. In `row_column()`, `nrows` is the number of rows per replicate; `field_layout()` controls how replicates are placed in the field. Factorial, split-plot and strip-plot designs keep their factor-specific arguments so whole plots, subplots and crossed strips remain distinct. The meaning of checks is design-specific: a count, entry identifiers, or labels. Consult the function's help before transferring a check vector between design families.

Partial and multi-location replication does not share one argument name across families; each keeps the allocation arguments that describe its own replication scheme: `partially_replicated()`'s `repGens` (how many entries get each replication level) and `repUnits` (those replication levels), and `RCBD_augmented()`'s `b` (number of augmented blocks) and `repsExpt` (replicates of the whole experiment). These are design-specific allocation arguments, not aliases of `reps` or `rep_checks`.

Boolean controls

Logical switches such as `continuous`, `factorLabels`, `spread_reps`, `allow_fillers`, and `randomizeH` require one nonmissing TRUE or FALSE. Numeric switches (0/1), strings, vectors, and arrays are not supported. Invalid switches signal a `fieldhub_input_error` before randomization, with argument, value, and options fields identifying the correction.

Migrating existing scripts

`CRD(locationName = ...)` remains supported; use `CRD(locationNames = ...)` in new code. In `incomplete_blocks()`, `alpha_lattice()`, `square_lattice()`, `rectangular_lattice()` and `row_column()`, replace the argument name `r` with `reps`. In `strip_plot()`, replace `b` with `reps`. In `optimized_arrangement()`, replace `amountChecks` with `rep_checks`. `RCBD_augmented(b = ...)` still denotes blocks and is unchanged.

Old names and positional calls retain their meaning and signal a warning of class `fieldhub_deprecated_warning`. Supplying both names for one argument is an error, even if their values agree. These aliases do not change seeded designs or the field names of saved results.

See Also

[CRD](#), [incomplete_blocks](#), [row_column](#), [strip_plot](#), [field_layout](#)

design_formats

Supported recorded-design file formats

Description

Lists the version and read/write capabilities of the built-in design format registry. Formats are selected explicitly, not guessed from filenames. CSV tables and app workflow ZIPs are separate exports; neither is silently interpreted as a complete recorded design by these functions.

Usage

`design_formats()`

Value

A data frame with format, extension, media_type, format_version, readable and writable columns.

Examples

```
design_formats()
```

design_results

Design results and reproducibility metadata

Description

FieldHub uses a versioned result contract. Schema 1 retains the established elements and storage types of each design family; validation checks those elements without coercing valid field books or removing user columns. Field, family, allocation and optimization results share the same construction boundary, with validation appropriate to each result type.

Field designs

Field designs inherit from FieldHub, with a first class such as fieldhub_rcbd identifying the engine. Test inheritance with inherits(x, "FieldHub"), not equality with class(x). Their fieldBook contains finite numeric ID and PLOT vectors and nonmissing atomic LOCATION identifiers. Integer and double storage, factors, and the existing character or numeric location conventions are preserved rather than normalized.

Additional required columns depend on the design:

- CRD and RCBD: REP, TREATMENT. RCBD with repeated checks also requires ENTRY and CHECKS.
- Latin square: SQUARE, ROW, COLUMN, TREATMENT.
- Factorial: REP, TRT_COMB, and one FACTOR_<name> column per recorded factor.
- Split plot: REP, WHOLE_PLOT, SUB_PLOT, TRT_COMB; split-split plot also requires SUB_SUB_PLOT.
- Strip plot: REP, HSTRIP, VSTRIP, TRT_COMB.
- Incomplete blocks and lattices: REP, IBLOCK, UNIT, ENTRY, TREATMENT.
- Row-column: REP, ROW, COLUMN, ENTRY, TREATMENT.
- Diagonal, optimized, sparse, augmented and partially replicated: EXPT, YEAR, ROW, COLUMN, CHECKS, ENTRY, TREATMENT. Augmented RCBD adds BLOCK; partially replicated designs add REP.

Required extension columns are atomic vectors, not matrix or list columns. Numeric values must be finite. Missing CHECKS values remain supported, as do missing REP values on partially replicated filler plots with ENTRY = 0. Other required extension values cannot be missing. Extra columns are permitted. Use field_layout() to obtain the final coordinates and plot numbers for a chosen planting path and layout.

Family splits and allocation plans

`split_families()` returns `rowsEachlist` with location counts and `data_locations` with ENTRY, NAME, FAMILY and LOCATION. Counts must agree with the entry table, including locations with zero entries.

`do_optim()` retains the Sparse or MultiPrep class. Its allocation counts, location sizes and entry lists are validated; these plans are not field books and do not yet specify plot coordinates.

Standalone optimization

`swap_pairs()` retains its matrices, distances and stopping diagnostics, with the classes `fieldhub_pair_swap` and `fieldhub_optimization`. Its shared metadata includes the input matrix and every optimization control. The validator checks field geometry, entry counts and retained search steps. These results can be saved and replayed with `reproduce_design()`.

Reproducibility

metadata records design, schema_version, seed, rng_kind, package_version and, in newly generated results, parameters. Parameters use effective values after validation, including defaults, the resolved year and seed, and uploaded data. Automatic seeds are integers; explicit real-valued seeds retain the legacy behavior of R's `set.seed()`, which truncates to an integer.

Use `saveRDS()` and `readRDS()` to retain the exact result and its metadata, then `reproduce_design()` to reconstruct it. Exact replay of optimization may require the same dependency versions and platform. Layout selections and simulated responses added later by the app are not part of the core design parameters.

Older schema-1 metadata without parameters still validates, but cannot be replayed automatically. Objects saved by FieldHub 1.5.0 without metadata remain supported by the print, summary and plotting compatibility methods; they are not silently rewritten to the new schema.

See Also

[reproduce_design](#), [field_layout](#), [design_arguments](#)

diagonal_arrangement *Spatial Un-replicated Diagonal Arrangement Design*

Description

Randomly generates an spatial un-replicated diagonal arrangement design.

Usage

```
diagonal_arrangement(
  nrows = NULL,
  ncols = NULL,
  lines = NULL,
  checks = NULL,
```

```

planter = "serpentine",
l = 1,
plotNumber = 101,
kindExpt = "SUDC",
splitBy = "row",
seed = NULL,
blocks = NULL,
exptName = NULL,
locationNames = NULL,
multiLocationData = FALSE,
data = NULL,
year = NULL,
checksPercent = NULL,
sameEntries = FALSE,
spread_checks = FALSE,
dist_method = "euclidean"
)

```

Arguments

nrows	Number of rows in the field.
ncols	Number of columns in the field.
lines	Number of genotypes, experimental lines or treatments.
checks	Number of genotypes checks.
planter	Option for serpentine or cartesian plot arrangement. By default planter = 'serpentine'.
l	Number of locations or sites. By default l = 1.
plotNumber	Numeric vector with the starting plot number for each location. By default plotNumber = 101.
kindExpt	Type of diagonal design, with single options: Single Un-replicated Diagonal Checks 'SUDC' and Decision Blocks Un-replicated Design with Diagonal Checks 'DBUDC' for multiple experiments. By default kindExpt = 'SUDC'.
splitBy	Option to split the field when kindExpt = 'DBUDC' is selected. By default splitBy = 'row'.
seed	(optional) Real number that specifies the starting seed to obtain reproducible designs.
blocks	Number of experiments or blocks to generate an DBUDC design. If kindExpt = 'DBUDC' and data is null, blocks are mandatory.
exptName	(optional) Name of the experiment.
locationNames	(optional) Names each location.
multiLocationData	(optional) Option to pass an entry list for multiple locations. By default multiLocationData = FALSE.
data	(optional) Data frame with 2 columns: ENTRY NAME .

year	(optional) Year recorded in the YEAR column of the field book. By default the current year.
checksPercent	(optional) Percentage of checks, one of the options available for the field dimensions. By default the last (largest) option.
sameEntries	(optional) Logical. When TRUE and kindExpt = "DBUDC", every block holds the same entries, and all blocks must have the same size. By default sameEntries = FALSE.
spread_checks	Logical. For kindExpt = "SUDC", exchange check identities among the existing check plots to increase separation of repeated checks. Test entries, filler plots and each check's replication stay fixed. The default FALSE preserves the original randomization.
dist_method	Distance between field coordinates for check optimization: "euclidean" (default) or "manhattan". Grid steps have equal scale.

Details

Check optimization compares counts of same-check pairs at ascending distances, reducing the count at the shortest differing distance first. It is a deterministic local exchange heuristic, not a statistical-efficiency or global-optimality guarantee. Each location stops after a complete scan finds no improvement, 100 accepted swaps, or 100,000 candidate evaluations. One check identity, or checks occurring only once, leaves the layout unchanged. The selected check percentage and filler count are unchanged. With checksPercent = NULL, Single Diagonal Arrangement still selects the valid option with the most check plots and therefore the fewest fillers.

Value

A FieldHub design result with the following components.

- infoDesign is a list with information on the design parameters. With check optimization enabled, check_optimization records swaps, candidate evaluations, stopping reason and minimum distance before and after optimization for each location.
- check_spacing records per-check minimum, mean nearest and mean pairwise distances, and same-check neighbors within 5 grid steps, for Single Diagonal Arrangement. With optimization enabled, both the original and final assignments are recorded. print() shows the spacing overview and summary() adds per-check details without rerunning the optimizer. Distances use dist_method and equally scaled row/column coordinates, not physical plot dimensions. Overall distances average equally across checks with repeated occurrences; neighbor percentages include all check occurrences. These metrics measure spacing, not statistical precision.
- layoutRandom is a matrix with the randomization layout.
- plotsNumber is a matrix with the layout plot number.
- data_entry is a data frame with the data input.
- fieldBook is a data frame with field book design. This includes the index (Row, Column).

Reproducibility

The result records effective inputs and the resolved seed in metadata\$parameters. Under the same package versions and RNG settings, rebuild a result x with do.call(diagonal_arrangement, x\$metadata\$parameters).

Author(s)

Didier Murillo [aut], Salvador Gezan [aut], Ana Heilman [ctb], Thomas Walk [ctb], Johan Aparicio [ctb], Richard Horsley [ctb]

References

Clarke, G. P. Y., & Stefanova, K. T. (2011). Optimal design for early-generation plant breeding trials with unreplicated or partially replicated test lines. *Australian & New Zealand Journal of Statistics*, 53(4), 461–480.

Examples

```
# Example 1: Generates a spatial single diagonal arrangement design in one location
# with 270 treatments and 30 check plots for a field with dimensions 15 rows x 20 cols
# in a serpentine arrangement.
spatd <- diagonal_arrangement(
  nrows = 15,
  ncols = 20,
  lines = 270,
  checks = 4,
  plotNumber = 101,
  kindExpt = "SUDC",
  planter = "serpentine",
  seed = 1987,
  exptName = "20WRY1",
  locationNames = "MINOT"
)
spatd$infoDesign
# Optimize check identities while keeping the diagonal check plots fixed.
spread <- diagonal_arrangement(nrows = 15, ncols = 20, lines = 270, checks = 4,
                                seed = 1987, spread_checks = TRUE)
spread$infoDesign$check_optimization
spatd$layoutRandom
spatd$plotsNumber
head(spatd$fieldBook, 12)

# Example 2: Generates a spatial decision block diagonal arrangement design in one location
# with 720 treatments allocated in 5 experiments or blocks for a field with dimensions
# 30 rows x 26 cols in a serpentine arrangement. In this case, we show how to set up the data
# option with the entries list.
checks <- 5;expts <- 5
list_checks <- paste("CH", 1:checks, sep = "")
treatments <- paste("G", 6:725, sep = "")
treatment_list <- data.frame(list(ENTRY = 1:725, NAME = c(list_checks, treatments)))
head(treatment_list, 12)
tail(treatment_list, 12)
spatDB <- diagonal_arrangement(
  nrows = 30,
  ncols = 26,
  checks = 5,
  plotNumber = 1,
  kindExpt = "DBUDC",
```

```

    planter = "serpentine",
    splitBy = "row",
    blocks = c(150,155,95,200,120),
    data = treatment_list
  )
  spatDB$infoDesign
  spatDB$layoutRandom
  spatDB$plotsNumber
  head(spatDB$fieldBook,12)

# Example 3: Generates a spatial decision block diagonal arrangement design in one location
# with 270 treatments allocated in 3 experiments or blocks for a field with dimensions
# 20 rows x 15 cols in a serpentine arrangement. Which in turn is an augmented block (3 blocks).
spatAB <- diagonal_arrangement(
  nrows = 20,
  ncols = 15,
  lines = 270,
  checks = 4,
  plotNumber = c(1,1001,2001),
  kindExpt = "DBUDC",
  planter = "serpentine",
  exptName = c("20WRA", "20WRB", "20WRC"),
  blocks = c(90, 90, 90),
  splitBy = "column"
)
spatAB$infoDesign
spatAB$layoutRandom
spatAB$plotsNumber
head(spatAB$fieldBook,12)

```

do_optim

Generate the sparse or p-rep allocation to multiple locations.

Description

Generate the sparse or p-rep allocation to multiple locations.

Usage

```

do_optim(
  design = "sparse",
  lines,
  l,
  copies_per_entry,
  add_checks = FALSE,
  checks = NULL,
  rep_checks = NULL,
  force_balance = TRUE,

```

```

    seed,
    data = NULL
  )

```

Arguments

design	Type of experimental design. It can be prep or sparse
lines	Number of genotypes, experimental lines or treatments.
l	Number of locations or sites. By default l = 1.
copies_per_entry	Number of copies per plant. When design is sparse then copies_per_entry should be less than 1
add_checks	Option to add checks. Optional if design = "prep"
checks	Number of genotypes checks.
rep_checks	Replication for each check.
force_balance	Get balanced unbalanced locations. By default force_balance = TRUE.
seed	(optional) Real number that specifies the starting seed to obtain reproducible designs.
data	(optional) Data frame with 2 columns: ENTRY NAME . ENTRY must be numeric.

Value

A list with five elements, classed `c("fieldhub_sparse_optimization", "Sparse")` or `c("fieldhub_multi_prep_optimization", "MultiPrep")`: the legacy Sparse/MultiPrep class is retained for compatibility, with an additional `fieldhub_*` class alongside it.

- `multi_location_data` is a data frame with the entries of every location: LOCATION | ENTRY | NAME, with a REPS column for p-rep allocations.
- `list_locs` is a list with each location list of entries.
- `allocation` is a data frame of test-entry copy counts, with one column per location.
- `size_locations` is a named vector of test-entry copies per location, excluding checks.
- `metadata` records the allocation type, schema version, seed, random-number settings, package version and evaluated input parameters. With the same package versions and RNG settings, rebuild the allocation with `do.call(do_optim, x$metadata$parameters)`.

Author(s)

Didier Murillo [aut], Salvador Gezan [aut], Ana Heilman [ctb]

References

Edmondson, R.N. Multi-level Block Designs for Comparative Experiments. JABES 25, 500–522 (2020). <https://doi.org/10.1007/s13253-020-00416-0>

Examples

```
sparse_example <- do_optim(
  design = "sparse",
  lines = 120,
  l = 4,
  copies_per_entry = 3,
  add_checks = TRUE,
  checks = 4,
  seed = 15
)
```

field_layout

*Field layout of a design***Description**

Returns the field book of a design with the ROW and COLUMN of every plot, for the layout, planter and stacking chosen. This is the field book that plot() draws and that the app exports.

The field books of CRD(), RCBD(), latin_square(), full_factorial(), split_plot(), split_split_plot(), strip_plot(), incomplete_blocks(), row_column() and the lattice designs have no field coordinates: their plots can be arranged in several ways, and the layouts that renumber the plots along the planting path also change PLOT. The other designs place their plots when they are built; their field book already has ROW and COLUMN, and it is returned as it is.

Usage

```
field_layout(x, layout = 1, planter = "serpentine", stacked = "vertical")
```

Arguments

x	A design created by a FieldHub function.
layout	Layout option, a whole number. The options depend on the design, the planter and the stacking; an unavailable option is an error that lists the available ones.
planter	Order in which the plots are planted: "serpentine" (by default) or "cartesian".
stacked	How the reps are arranged: "vertical" (by default), "horizontal", or "grid_panel" for designs in incomplete blocks and split plots in complete blocks with more than two reps.

Value

A data frame: the field book of every location with the columns ID, LOCATION, PLOT, ROW and COLUMN first.

Author(s)

Didier Murillo [aut]

Examples

```
rcbd <- RCBD(t = 6, reps = 3, plotNumber = 101, seed = 1)
head(field_layout(rcbd))
# Blocks side by side, plots planted row by row
head(field_layout(rcbd, stacked = "horizontal", planter = "cartesian"))
```

full_factorial	<i>Generates a Full Factorial Design</i>
----------------	--

Description

It randomly generates a full factorial design across locations.

Usage

```
full_factorial(
  setfactors = NULL,
  reps = NULL,
  l = 1,
  type = 2,
  plotNumber = 101,
  continuous = FALSE,
  planter = "serpentine",
  seed = NULL,
  locationNames = NULL,
  factorLabels = TRUE,
  data = NULL
)
```

Arguments

setfactors	Numeric vector with levels of each factor.
reps	Number of replicates (full blocks).
l	Number of locations. By default l = 1.
type	Option for CRD or RCBD designs. Values are type = 1 (CRD) or type = 2 (RCBD). By default type = 2.
plotNumber	Numeric vector with the starting plot number for each location. By default plotNumber = 101.
continuous	Logical for plot number continuous or not. By default continuous = FALSE.
planter	Option for serpentine or cartesian plot arrangement. By default planter = 'serpentine'.
seed	(optional) Real number that specifies the starting seed to obtain reproducible designs.

locationNames (optional) Names for each location.

factorLabels (optional) If TRUE retain the levels labels from the original data set otherwise, numeric labels will be assigned. Default is factorLabels = TRUE.

data (optional) Data frame whose first two columns contain factor names and level labels. Levels must be unique within each factor, but a label may occur in different factors. Incomplete rows are omitted.

Value

A list with two elements.

- infoDesign is a list with information on the design parameters.
- fieldBook is a data frame with the full factorial field book.

Reproducibility

The result records effective inputs and the resolved seed in metadata\$parameters. Under the same package versions and RNG settings, rebuild a result x with `do.call(full_factorial, x$metadata$parameters)`.

Author(s)

Didier Murillo [aut], Salvador Gezan [aut], Ana Heilman [ctb], Thomas Walk [ctb], Johan Aparicio [ctb], Richard Horsley [ctb]

References

Federer, W. T. (1955). Experimental Design. Theory and Application. New York, USA. The Macmillan Company.

Examples

```
# Example 1: Generates a full factorial with 3 factors each with 2 levels.
# This in an RCBD arrangement with 3 reps.
fullFact1 <- full_factorial(setfactors = c(2,2,2), reps = 3, l = 1, type = 2,
                           plotNumber = 101,
                           continuous = TRUE,
                           planter = "serpentine",
                           seed = 325,
                           locationNames = "FARGO")

fullFact1$infoDesign
head(fullFact1$fieldBook,10)

# Example 2: Generates a full factorial with 3 factors and each with levels: 2,3,
# and 2, respectively. In this case, we show how to use the option data
FACTORS <- rep(c("A", "B", "C"), c(2,3,2))
LEVELS <- c("a0", "a1", "b0", "b1", "b2", "c0", "c1")
data_factorial <- data.frame(list(FACTOR = FACTORS, LEVEL = LEVELS))
print(data_factorial)
# This in an RCBD arrangement with 5 reps in 3 locations.
fullFact2 <- full_factorial(setfactors = NULL, reps = 5, l = 3, type = 2,
```

```

        plotNumber = c(101,1001,2001),
        continuous = FALSE,
        planter = "serpentine",
        seed = 326,
        locationNames = c("Loc1","Loc2","Loc3"),
        data = data_factorial)

fullFact2$infoDesign
head(fullFact2$fieldBook,10)

```

incomplete_blocks	<i>Generates a Resolvable Incomplete Block Design</i>
-------------------	---

Description

Randomly generates a resolvable incomplete block design (IBD) of characteristics (t, k, r). The randomization can be done across locations.

Usage

```

incomplete_blocks(
  t = NULL,
  k = NULL,
  r = NULL,
  l = 1,
  plotNumber = 101,
  locationNames = NULL,
  seed = NULL,
  data = NULL,
  reps = NULL
)

```

Arguments

t	Number of treatments, or a character vector with the treatment labels.
k	Size of incomplete blocks (number of units per incomplete block).
r	Deprecated alias for reps; positional calls remain supported.
l	Number of locations. By default l = 1.
plotNumber	Numeric vector with the starting plot number for each location. By default plotNumber = 101.
locationNames	(optional) Names for each location.
seed	(optional) Real number that specifies the starting seed to obtain reproducible designs.
data	(optional) Data frame with label list of treatments.
reps	Number of full resolvable replicates per location.

Value

A list with two elements.

- `infoDesign` is a list with information on the design parameters.
- `fieldBook` is a data frame with the incomplete block design field book.

Reproducibility

The result records effective inputs and the resolved seed in `metadata$parameters`, using `reps` for replication. Under the same package versions and RNG settings, rebuild a result `x` with `do.call(incomplete_blocks, x$metadata$parameters)`.

Author(s)

Didier Murillo [aut], Salvador Gezan [aut], Ana Heilman [ctb], Thomas Walk [ctb], Johan Aparicio [ctb], Richard Horsley [ctb]

References

Edmondson., R. N. (2021). `blocksdesign`: Nested and crossed block designs for factorial and unstructured treatment sets. <https://CRAN.R-project.org/package=blocksdesign>

Examples

```
# Example 1: Generates a resolvable IBD of characteristics (t,k,r) = (12,4,2).
# 1-resolvable IBDs
ibd1 <- incomplete_blocks(t = 12,
                          k = 4,
                          reps = 2,
                          seed = 1984)

ibd1$infoDesign
head(ibd1$fieldBook)

# Example 2: Generates a balanced resolvable IBD of characteristics (t,k,r) = (15,3,7).
# In this case, we show how to use the option data.
treatments <- paste("TX-", 1:15, sep = "")
ENTRY <- 1:15
treatment_list <- data.frame(list(ENTRY = ENTRY, TREATMENT = treatments))
head(treatment_list)
ibd2 <- incomplete_blocks(t = 15,
                          k = 3,
                          reps = 7,
                          seed = 1985,
                          data = treatment_list)

ibd2$infoDesign
head(ibd2$fieldBook)
```

latin_rectangle	<i>Randomized cyclic Latin rectangle</i>
-----------------	--

Description

Arrange treatments in fewer complete rows than a Latin square, with no treatment repeated in any column. Each row is a complete replicate.

Usage

```
latin_rectangle(
  t,
  rows = 3,
  l = 1,
  plotNumber = 101,
  planter = "serpentine",
  seed = NULL,
  locationNames = NULL
)
```

Arguments

t	Number of treatments (at least two), or distinct character labels.
rows	Number of complete rows, from two through the treatment count.
l	Number of independently randomized locations.
plotNumber	First plot number: one positive whole number, or one per location. Plot numbers restart at the supplied value for each location.
planter	Plot-number traversal, "serpentine" or "cartesian".
seed	Randomization seed; NULL draws and records one automatic seed.
locationNames	Distinct nonblank location labels, one per location; NULL generates LOC1, LOC2, and so on. Labels are preserved.

Details

The construction takes the first rows consecutive shifts of the cyclic Latin square, then independently permutes rows, columns and treatment labels at each location. It samples this cyclic construction family, not uniformly from all Latin rectangles, and does not optimize efficiency. It uses $O(1 * \text{rows} * t)$ construction work with no search.

Rows are complete blocks; columns are incomplete when $\text{rows} < t$. This is not necessarily a balanced incomplete-block or Youden design. Consecutive cyclic shifts keep the additive row/column/treatment model connected. Its residual degrees of freedom at one location are $(\text{rows} - 2) * (t - 1)$: two-row rectangles are saturated and cannot independently estimate residual variation under that model. Interactions are not separately estimable. Choose replication and analysis with the intended scientific use in mind.

Field coordinates and plot numbering are fixed at construction. The planter and stacked arguments to field_layout() do not rearrange a saved rectangle. Supplied seeds preserve the caller's RNG; an automatic seed consumes exactly one recorded draw.

Value

A schema-1 FieldHub result with infoDesign, a field book containing ID, LOCATION, PLOT, ROW, COLUMN, REP and TREATMENT, and replay metadata. REP is the complete row block. Use reproduce_design() for replay and field_layout() for the recorded coordinates.

References

Peter G. Doyle, *The number of Latin rectangles*. <https://math.dartmouth.edu/~doyle/docs/latin/latin.pdf>.

Examples

```
x <- latin_rectangle(t = 5, rows = 3, seed = 27)
x$fieldBook
identical(reproduce_design(x), x)
```

latin_square	<i>Generates a Latin Square Design</i>
--------------	--

Description

Randomly generates a latin square design of up 10 treatments.

Usage

```
latin_square(
  t = NULL,
  reps = 1,
  plotNumber = 101,
  planter = "serpentine",
  seed = NULL,
  locationNames = NULL,
  data = NULL
)
```

Arguments

- t Number of treatments.
- reps Number of full resolvable squares. By default reps = 1.
- plotNumber Starting plot number. By default plotNumber = 101.
- planter Option for serpentine or cartesian arrangement. By default planter = 'serpentine'.

seed	(optional) Real number that specifies the starting seed to obtain reproducible designs.
locationNames	(optional) Name for the location.
data	(optional) Data frame with label list of treatments.

Details

The randomized search is limited to 100,000 placement iterations per square. If it cannot complete a square within that budget, it raises a `fieldhub_search_error` with the square number and iteration limit. No incomplete design is returned; try a different seed. Designs completed within the limit retain their existing seeded output.

Value

A list with information on the design parameters.

Data frame with the latin square field book.

A list with two elements.

- `infoDesign` is a list with information on the design parameters.
- `fieldBook` is a data frame with the latin square field book.

Reproducibility

The result records effective inputs and the resolved seed in `metadata$parameters`. Under the same package versions and RNG settings, rebuild a result `x` with `do.call(latin_square, x$metadata$parameters)`.

Author(s)

Didier Murillo [aut], Salvador Gezan [aut], Ana Heilman [ctb], Thomas Walk [ctb], Johan Aparicio [ctb], Thiago de Paula Oliveira[ctb] Richard Horsley [ctb]

References

Federer, W. T. (1955). Experimental Design. Theory and Application. New York, USA. The Macmillan Company.

Examples

```
# Example 1: Generates a latin square design with 4 treatments and 2 reps.
latinSq1 <- latin_square(t = 4,
                        reps = 2,
                        plotNumber = 101,
                        planter = "cartesian",
                        seed = 1980)

print(latinSq1)
summary(latinSq1)
head(latinSq1$fieldBook)

# Example 2: Generates a latin square design with 5 treatments and 3 reps.
latin_data <- data.frame(list(ROW = paste("Period", 1:5, sep = "")),
```

```

                                COLUMN = paste("Cow", 1:5, sep = ""),
                                TREATMENT = paste("Diet", 1:5, sep = "")))
print(latin_data)
latinSq2 <- latin_square(t = NULL,
                        reps = 3,
                        plotNumber = 101,
                        planter = "cartesian",
                        seed = 1981,
                        data = latin_data)

latinSq2$squares
latinSq2$plotSquares
head(latinSq2$fieldBook)

```

multi_location_prep	<i>Optimized multi-location partially replicated design</i>
---------------------	---

Description

Optimized multi-location partially replicated design

Usage

```

multi_location_prep(
  lines,
  nrows,
  ncols,
  l,
  planter = "serpentine",
  plotNumber,
  desired_avg,
  copies_per_entry,
  checks = NULL,
  rep_checks = NULL,
  exptName,
  locationNames,
  optim_list,
  seed,
  spread_reps = TRUE,
  data = NULL,
  allow_fillers = FALSE,
  max_fillers = NULL,
  year = NULL
)

```

Arguments

lines	Number of genotypes, experimental lines or treatments.
-------	--

nrows	Numeric vector with the number of rows field at each location.
ncols	Numeric vector with the number of columns field at each location.
l	Number of locations. By default $l = 1$.
planter	Option for serpentine or cartesian movement. By default <code>planter = 'serpentine'</code> .
plotNumber	Numeric vector with the starting plot number for each location. By default <code>plotNumber = 101</code> .
desired_avg	(optional) Desired average of treatments across locations.
copies_per_entry	Number of total copies per treatment.
checks	Number of checks.
rep_checks	Number of replications per check.
exptName	(optional) Name of the experiment.
locationNames	(optional) Name for each location.
optim_list	(optional) A list object of class "MultiPrep" generated by <code>do_optim()</code> function.
seed	(optional) Real number that specifies the starting seed to obtain reproducible designs.
spread_reps	A logical value indicating whether to maximize the spatial distance between replicated treatments in the field. Default is TRUE.
data	(optional) Data frame with 2 columns: ENTRY NAME . ENTRY must be numeric.
allow_fillers	A logical value indicating whether field dimensions may exceed the number of experimental plots. Fillers occupy the terminal cells of the selected planter path. Default is FALSE.
max_fillers	Maximum number of filler plots considered when dimensions are selected automatically. When NULL, up to 10 filler plots are considered.
year	(optional) Year recorded in the YEAR column of the field book. By default the current year.

Value

A list of class `FieldHub` with several elements.

- `infoDesign` is a list with information on the design parameters.
- `layoutRandom` is a matrix with the randomization layout.
- `plotNumber` is a matrix with the layout plot number.
- `binaryField` is a matrix with the binary field.
- `fillerField` is a logical matrix identifying filler plots.
- `dataEntry` is a data frame with the data input.
- `genEntries` is a list with the entries for replicated and non-replicated parts.
- `fieldBook` is a data frame with field book design. This includes the index (Row, Column).
- `min_pairwise_distance` is a data frame with the minimum pairwise distance between each pair of locations.

- `reps_info` is a data frame with information on the number of replicated and non-replicated treatments at each location.
- `pairsDistance` is a data frame with the pairwise distances between each pair of treatments.
- `treatments_with_reps` is a list with the entries for the replicated part of the design.
- `treatments_with_no_reps` is a list with the entries for the non-replicated part of the design.
- `list_locs` is a list with each location list of entries.
- `multi_location_data` is a data frame with the entries of every location: LOCATION | ENTRY | NAME | REPS.
- `allocation` is a matrix with the allocation of treatments.
- `size_locations` is a named vector with the size of each location.

Reproducibility

The result records effective inputs and the resolved seed in `metadata$parameters`. Under the same package versions and RNG settings, rebuild a result `x` with `do.call(multi_location_prep, x$metadata$parameters)`.

Author(s)

Didier Murillo [aut], Salvador Gezan [aut], Jean-Marc Montpetit [ctb], Ana Heilman [ctb]

References

Edmondson, R.N. Multi-level Block Designs for Comparative Experiments. JABES 25, 500–522 (2020). <https://doi.org/10.1007/s13253-020-00416-0>

Examples

```
# Example 1: Generates a spatially optimized multi-location p-rep design with 142
# genotypes. The number of copies per plant available for this experiment is 9.
# This experiment is carried out in 5 locations, and there are seven seeds available
# for each plant to make replications.
# In this case, we add three controls (checks) with six reps each.
# With this setup, the experiment will have 142 treatments + 3 checks = 145
# entries and the number of plots per location after the allocation process
# will be 196.
# The average genotype allocation will be 1.5 copies per location.
## Not run:
optim_multi_prep <- multi_location_prep(
  lines = 150,
  l = 5,
  copies_per_entry = 7,
  checks = 3,
  rep_checks = c(6,6,6),
  locationNames = c("LOC1", "LOC2", "LOC3", "LOC4", "LOC5"),
  seed = 1234
)
field_book <- optim_multi_prep$fieldBook
head(subset(field_book, LOCATION == "LOC1"), 10)
```

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

optimized_arrangement *Generates an Spatial Un-replicated Optimized Arrangement Design*

Description

Randomly generates a spatial un-replicated optimized arrangement design, where the distance between checks is maximized in such a way that each row and column have control plots. Note that design generation needs the dimension of the field (number of rows and columns).

Usage

```
optimized_arrangement(
  nrows = NULL,
  ncols = NULL,
  lines = NULL,
  amountChecks = NULL,
  checks = NULL,
  planter = "serpentine",
  l = 1,
  plotNumber = 101,
  seed = NULL,
  exptName = NULL,
  locationNames = NULL,
  spread_reps = TRUE,
  data = NULL,
  year = NULL,
  rep_checks = NULL
)
```

Arguments

nrows	Number of rows in the field.
ncols	Number of columns in the field.
lines	Number of genotypes, experimental lines or treatments.
amountChecks	Deprecated alias for rep_checks.
checks	Number of genotypes as checks.
planter	Option for serpentine or cartesian arrangement. By default planter = 'serpentine'.
l	Number of locations. By default l = 1.
plotNumber	Numeric vector with the starting plot number for each location. By default plotNumber = 101.
seed	(optional) Real number that specifies the starting seed to obtain reproducible designs.

exptName	(optional) Name of the experiment.
locationNames	(optional) Name for each location.
spread_reps	A logical value indicating whether to maximize the spatial distance between replicated treatments in the field. Default is TRUE.
data	(optional) Data frame with 3 columns: ENTRY NAME REPS.
year	(optional) Year recorded in the YEAR column of the field book. By default the current year.
rep_checks	Integer with the total amount of checks, or a numeric vector with the replicates of each check label.

Value

A list with five elements.

- infoDesign is a list with information on the design parameters.
- layoutRandom is a matrix with the randomization layout.
- plotNumber is a matrix with the layout plot number.
- dataEntry is a data frame with the data input.
- genEntries is a list with the entries for replicated and no replicated part.
- fieldBook is a data frame with field book design. This includes the index (Row, Column).

Reproducibility

The result records effective inputs and the resolved seed in `metadata$parameters`. Under the same package versions and RNG settings, rebuild a result `x` with `do.call(optimized_arrangement, x$metadata$parameters)`.

Author(s)

Didier Murillo [aut], Salvador Gezan [aut], Ana Heilman [ctb], Thomas Walk [ctb], Johan Aparicio [ctb], Richard Horsley [ctb]

References

Clarke, G. P. Y., & Stefanova, K. T. (2011). Optimal design for early-generation plant breeding trials with unreplicated or partially replicated test lines. *Australian & New Zealand Journal of Statistics*, 53(4), 461–480.

Examples

```
# Example 1: Generates a spatial unreplicated optimized arrangement design in one location
# with 120 genotypes + 20 check plots (4 checks) for a field with dimension 14 rows x 10 cols.
## Not run:
optim_unrep1 <- optimized_arrangement(
  nrows = 14,
  ncols = 10,
  lines = 120,
```

```

    rep_checks = 20,
    checks = 1:4,
    planter = "cartesian",
    plotNumber = 101,
    exptName = "20RW1",
    locationNames = "CASSELTON",
    seed = 14124
  )
optim_unrep1$infoDesign
optim_unrep1$layoutRandom
optim_unrep1$plotNumber
head(optim_unrep1$fieldBook, 12)

## End(Not run)

# Example 2: Generates a spatial unreplicated optimized arrangement design in one location
# with 200 genotypes + 20 check plots (4 checks) for a field with dimension 10 rows x 22 cols.
# As example, we set up the data option with the entries list.
## Not run:
checks <- 4
list_checks <- paste("CH", 1:checks, sep = "")
treatments <- paste("G", 5:204, sep = "")
REPS <- c(5, 5, 5, 5, rep(1, 200))
treatment_list <- data.frame(list(ENTRY = 1:204, NAME = c(list_checks, treatments), REPS = REPS))
head(treatment_list, 12)
tail(treatment_list, 12)
optim_unrep2 <- optimized_arrangement(
  nrows = 10,
  ncols = 22,
  planter = "serpentine",
  plotNumber = 101,
  seed = 120,
  exptName = "20YWA2",
  locationNames = "MINOT",
  data = treatment_list
)
optim_unrep2$infoDesign
optim_unrep2$layoutRandom
optim_unrep2$plotNumber
head(optim_unrep2$fieldBook, 12)

## End(Not run)

```

partially_replicated *Generates a Spatial Partially Replicated Arrangement Design*

Description

Randomly generates a spatial partially replicated (p-rep) design for single or multiple locations.

Usage

```

partially_replicated(
  nrows = NULL,
  ncols = NULL,
  repGens = NULL,
  repUnits = NULL,
  planter = "serpentine",
  l = 1,
  plotNumber = 101,
  spread_reps = TRUE,
  seed = NULL,
  exptName = NULL,
  locationNames = NULL,
  multiLocationData = FALSE,
  dist_method = "euclidean",
  border_penalization = 0.5,
  data = NULL,
  allow_fillers = FALSE,
  year = NULL
)

```

Arguments

<code>nrows</code>	Numeric vector with the number of rows field at each location.
<code>ncols</code>	Numeric vector with the number of columns field at each location.
<code>repGens</code>	Numeric vector with the amount genotypes to replicate.
<code>repUnits</code>	Numeric vector with the number of reps of each genotype.
<code>planter</code>	Option for serpentine or cartesian movement. By default <code>planter = 'serpentine'</code> .
<code>l</code>	Number of locations. By default <code>l = 1</code> .
<code>plotNumber</code>	Numeric vector with the starting plot number for each location. By default <code>plotNumber = 101</code> .
<code>spread_reps</code>	A logical value indicating whether to maximize the spatial distance between replicated treatments in the field. Default is <code>TRUE</code> .
<code>seed</code>	(optional) Real number that specifies the starting seed to obtain reproducible designs.
<code>exptName</code>	(optional) Name of the experiment.
<code>locationNames</code>	(optional) Name for each location.
<code>multiLocationData</code>	(optional) Option to pass an entry list for multiple locations.
<code>dist_method</code>	The method used for distance calculation. Options are "euclidean" (default) and "manhattan". By default <code>multiLocationData = FALSE</code> .
<code>border_penalization</code>	A tuning parameter for the centrality penalty. Default is 0.5.
<code>data</code>	(optional) Data frame with 3 columns: ENTRY NAME REPS. If <code>multiLocationData = TRUE</code> then the data must have 4 columns: LOCATION ENTRY NAME REPS

allow_fillers	A logical value indicating whether field dimensions may exceed the number of experimental plots. Fillers occupy the terminal cells of the selected planter path. Default is FALSE.
year	(optional) Year recorded in the YEAR column of the field book. By default the current year.

Details

This function generates and optimizes a partially replicated (p-rep) experimental design for a given set of treatments and replication levels. The design is represented by a matrix and optimized using a pairwise distance metric. The function outputs various information about the optimized design including the field layout, replicated and unreplicated treatments, and pairwise distances between treatments. Note that the design generation needs the dimension of the field (number of rows and columns).

Value

A list with several elements.

- infoDesign is a list with information on the design parameters.
- layoutRandom is a matrix with the randomization layout.
- plotNumber is a matrix with the layout plot number.
- binaryField is a matrix with the binary field.
- fillerField is a logical matrix identifying filler plots.
- dataEntry is a data frame with the data input.
- genEntries is a list with the entries for replicated and non-replicated parts.
- fieldBook is a data frame with field book design. This includes the index (Row, Column).
- min_pairwise_distance is a data frame with the minimum pairwise distance between each pair of locations.
- reps_info is a data frame with information on the number of replicated and non-replicated treatments at each location.
- pairsDistance is a data frame with the pairwise distances between each pair of treatments.
- treatments_with_reps is a list with the entries for the replicated part of the design.
- treatments_with_no_reps is a list with the entries for the non-replicated part of the design.

Reproducibility

The result records effective inputs and the resolved seed in `metadata$parameters`. Under the same package versions and RNG settings, rebuild a result `x` with `do.call(partially_replicated, x$metadata$parameters)`.

Author(s)

Didier Murillo [aut], Salvador Gezan [aut], Ana Heilman [ctb], Thomas Walk [ctb], Johan Aparicio [ctb], Jean-Marc Montpetit [ctb], Richard Horsley [ctb]

References

Cullis, S., B. R., & Coombes, N. E. (2006). On the design of early generation variety trials with correlated data. *Journal of Agricultural, Biological, and Environmental Statistics*, 11, 381–393. <https://doi.org/10.1198/108571106X154443>

Examples

```
# Example 1: Generates a spatial optimized partially replicated arrangement design in one
# location with 335 genotypes for a field with dimensions 15 rows x 28 cols.
# Note that there are 250 genotypes unreplicated (only one time), 85 genotypes replicated
# two times, and three checks 8 times each.
## Not run:
prep_deseign1 <- partially_replicated(
  nrows = 12,
  ncols = 37,
  repGens = c(250, 85, 3),
  repUnits = c(1, 2, 8),
  planter = "cartesian",
  plotNumber = 101,
  seed = 77
)
prep_deseign1$infoDesign
prep_deseign1$layoutRandom
prep_deseign1$plotNumber
head(prep_deseign1$fieldBook, 12)

## End(Not run)

# Example 2: Generates a spatial optimized partially replicated arrangement design with 492
# genotypes in a field with dimensions 30 rows x 20 cols. Note that there 384 genotypes
# unreplicated (only one time), 108 genotypes replicated two times.
# In this case we don't have check plots.
# As example, we set up the data option with the entries list.
## Not run:
NAME <- paste("G", 1:492, sep = "")
repGens = c(108, 384); repUnits = c(2,1)
REPS <- rep(repUnits, repGens)
treatment_list <- data.frame(list(ENTRY = 1:492, NAME = NAME, REPS = REPS))
head(treatment_list, 12)
tail(treatment_list, 12)
prep_deseign2 <- partially_replicated(
  nrows = 30,
  ncols = 20,
  planter = "serpentine",
  plotNumber = 101,
  seed = 41,
  data = treatment_list
)
prep_deseign2$infoDesign
prep_deseign2$layoutRandom
prep_deseign2$plotNumber
head(prep_deseign2$fieldBook, 10)
```

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

plot.FieldHub	<i>Plot a FieldHub object</i>
---------------	-------------------------------

Description

Draw a field layout plot for a FieldHub object.

Usage

```
## S3 method for class 'FieldHub'
plot(x, ...)
```

Arguments

x	a object inheriting from class FieldHub
...	further arguments passed to utility function plot_layout(). • layout a integer. Options available depend on the type of design and its characteristics • l a integer to specify the location to plot. • planter it can be serpentine or cartesian. It has no effect on split-plot and split-split-plot designs in complete blocks (type = 2), whose whole plots are numbered in a fixed order. • stacked it can be vertical or horizontal stacked layout.

Value

- a plot object inheriting from class fieldLayout
- field_book a data frame with the fieldbook that includes the coordinates ROW and COLUMN.

Author(s)

Didier Murillo [aut]

Examples

```
## Not run:
# Example 1: Plot a RCBD design with 24 treatments and 3 reps.
s <- RCBD(t = 24, reps = 3, plotNumber = 101, seed = 12)
plot(s)

## End(Not run)
```

<code>print.FieldHub</code>	<i>Print a FieldHub object</i>
-----------------------------	--------------------------------

Description

Prints the design parameters and the first rows of the field book of any FieldHub design, including results saved by earlier versions of FieldHub.

Usage

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

Arguments

<code>x</code>	an object inheriting from class
<code>n</code>	a single integer. If positive or zero, size for the resulting object: number of elements for a vector (including lists), rows for a matrix or data frame or lines for a function. If negative, all but the <code>n</code> last/first number of elements of <code>x</code> .
<code>...</code>	further arguments passed to head .

Value

an object inheriting from class FieldHub

Author(s)

Thiago de Paula Oliveira, <thiago.paula.oliveira@alumni.usp.br> [aut], Didier Murillo [aut]

Examples

```
# Example 1: Generates a CRD design with 5 treatments and 5 reps each.
crd1 <- CRD(t = 5, reps = 5, plotNumber = 101,
seed = 1985, locationNames = "Fargo")
crd1$infoDesign
print(crd1)
```

print.fieldLayout	<i>Print a fieldLayout plot object</i>
-------------------	--

Description

Prints a plot object of class fieldLayout.

Usage

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

Arguments

x	a plot object inheriting from class fieldLayout.
...	unused, for extensibility.

Value

a plot object inheriting from class fieldLayout.

Author(s)

Didier Murillo [aut]

print.summary.FieldHub	<i>Print the summary of a FieldHub object</i>
------------------------	---

Description

Print summary information on the design parameters, and data frame structure

Usage

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

Arguments

x	an object inheriting from class FieldHub
...	Unused, for extensibility

Value

an object inheriting from class FieldHub

Author(s)

Thiago de Paula Oliveira, <thiago.paula.oliveira@alumni.usp.br> [aut], Didier Murillo [aut]

RCBD

Generates a Randomized Complete Block Design (RCBD)

Description

It randomly generates a randomized complete block design (RCBD) across locations.

Usage

```
RCBD(
  t = NULL,
  reps = NULL,
  l = 1,
  plotNumber = 101,
  continuous = FALSE,
  planter = "serpentine",
  seed = NULL,
  locationNames = NULL,
  data = NULL,
  checks = NULL,
  rep_checks = NULL,
  spread_checks = TRUE
)
```

Arguments

t	An integer number with total number of treatments or a vector of dimension t with labels.
reps	Number of replicates (full blocks) of each treatment.
l	Number of locations. By default l = 1.
plotNumber	Numeric vector with the starting plot number for each location. By default plotNumber = 101.
continuous	Logical value for plot number continuous or not. By default continuous = FALSE.
planter	Option for serpentine or cartesian arrangement. By default planter = 'serpentine'.
seed	(optional) Real number that specifies the starting seed to obtain reproducible designs.
locationNames	(optional) Names for each location.
data	(optional) Data frame with the labels of treatments.

checks	(optional) Checks to repeat within every block. Either a positive integer <i>N</i> or a character vector of check labels. When data or a character vector <i>t</i> supplies a pool of entries, <i>N</i> takes that pool's first <i>N</i> entries as the checks; when no pool is supplied (<i>t</i> is a bare count), <i>N</i> instead generates its own check labels "CH1" . . "CHN" ahead of the auto-generated test entries "T1" . . "Tt". checks sits after data in the argument list (rather than next to <i>t</i> , where it might otherwise go) precisely so that data keeps its original positional slot and existing positional calls to <code>RCBD()</code> keep working unchanged. By default checks = NULL, which produces an ordinary RCBD.
rep_checks	(optional) Number of times each check is repeated within every block. A single value is recycled across all checks, or supply one value per check. By default rep_checks = NULL, which is treated as 1 for every check.
spread_checks	(optional) Logical. When TRUE (the default), the repeated copies of each check are spread across the block by placing one copy in each of rep_checks contiguous strata. When FALSE, the whole block is randomized without restriction.

Details

The result records effective inputs, the resolved seed and the starting plot numbers in `metadata$parameters`. Under the same package versions and RNG settings, rebuild a result *x* with `do.call(RCBD, x$metadata$parameters)`.

When checks is supplied, one or more checks are repeated multiple times within every block, while every test entry still appears exactly once. In a classical RCBD, the residual is the treatment-by-block interaction; repeating checks inside a block instead supplies a within-block estimate of error and a form of local control.

checks accepts either a single positive integer *N* or a character vector of check labels. When a pool of entries is supplied through data or a character *t*, an integer *N* takes the first *N* entries of that pool as the checks, and a character vector of labels must already exist in that pool; an unmatched label is an error that also names the closest case-insensitive match, if any. When no pool is supplied (*t* is a bare count), the labels are generated instead: a character checks is appended to the auto-generated test entries, while an integer *N* generates its own check labels "CH1" . . "CHN" ahead of the auto-generated test entries "T1" . . "Tt".

rep_checks sets how many times each check repeats within a block: a single value is recycled across all checks, or one value can be supplied per check.

A block (test entries plus repeated checks) larger than 10,000 plots is rejected.

With spread_checks = TRUE (the default), the repeated copies of a check are placed one per contiguous stratum of the block, so they are spread across it rather than clustered together. Two distinct density facts apply here: as soon as checks occupy more than 50 percent of the block, a warning is issued as an early caution, but randomization itself remains effectively unconstrained up to about 67 percent check density; only from about 78 percent and higher does the stratified-placement constraint leave few or no alternative positions, so a repeated check's placement can become nearly or fully deterministic rather than random. This is a property of the stratified-placement geometry, not a bug, and most field trials use far lower check density than either threshold.

Value

A list with five elements.

- metadata records the design, schema/package versions, seed, random-number settings and effective input parameters.
- infoDesign is a list with information on the design parameters.
- layoutRandom is the RCBD layout randomization for each location.
- plotNumber is the plot number layout for each location.
- fieldBook is a data frame with the RCBD field book design. Without checks it has columns ID, LOCATION, PLOT, REP and TREATMENT. When checks is supplied it gains ENTRY and CHECKS columns, ordered ID, LOCATION, PLOT, REP, ENTRY, CHECKS, TREATMENT.

Author(s)

Didier Murillo [aut], Salvador Gezan [aut], Ana Heilman [ctb], Thomas Walk [ctb], Johan Aparicio [ctb], Richard Horsley [ctb]

References

- Federer, W. T. (1955). Experimental Design. Theory and Application. New York, USA. The Macmillan Company.
- Lin, C.S., & Poushinsky, G. (1985). A modified augmented design (type 2) for rectangular plots. Canadian Journal of Plant Science, 65(3), 743-749.

Examples

```
# Example 1: Generates a RCBD design with 3 blocks and 20 treatments across 3 locations.
rcbd1 <- RCBD(t = LETTERS[1:20], reps = 5, l = 3,
  plotNumber = c(101,1001, 2001),
  continuous = TRUE,
  planter = "serpentine",
  seed = 1020,
  locationNames = c("FARGO", "MINOT", "CASSELTON"))
rcbd1$infoDesign
rcbd1$layoutRandom
rcbd1$plotNumber
head(rcbd1$fieldBook)

# Example 2: Generates a RCBD design with 6 blocks and 18 treatments in one location.
# In this case, we show how to use the option data.
treatments <- paste("ND-", 1:18, sep = "")
treatment_list <- data.frame(list(TREATMENT = treatments))
head(treatment_list)
rcbd2 <- RCBD(reps = 6, l = 1,
  plotNumber = 101,
  continuous = FALSE,
  planter = "serpentine",
  seed = 13,
  locationNames = "IBAGUE",
  data = treatment_list)
rcbd2$infoDesign
rcbd2$layoutRandom
rcbd2$plotNumber
```

```

head(rcbd2$fieldBook)

# Example 3: RCBD with two checks repeated twice in each of 3 blocks,
# alongside 18 test entries. Block size is 18 + 2 + 2 = 22 plots.
rcbd3 <- RCBD(t = 18, reps = 3,
             checks = c("CK1", "CK2"),
             rep_checks = c(2, 2),
             plotNumber = 101,
             seed = 1234,
             locationNames = "FARGO")
rcbd3$infoDesign
head(rcbd3$fieldBook)
# Each check appears twice per block, every test entry exactly once:
table(subset(rcbd3$fieldBook, REP == 1)$TREATMENT)

```

RCBD_augmented	<i>Generates an Augmented Randomized Complete Block Design (AR-CBD)</i>
----------------	---

Description

It randomly generates an augmented randomized complete block design across locations (ARCBd).

Usage

```

RCBD_augmented(
  lines = NULL,
  checks = NULL,
  b = NULL,
  l = 1,
  planter = "serpentine",
  plotNumber = 101,
  repsStack = c("vertical", "horizontal"),
  exptName = NULL,
  seed = NULL,
  locationNames = NULL,
  repsExpt = 1,
  random = TRUE,
  data = NULL,
  nrows = NULL,
  ncols = NULL,
  year = NULL
)

```

Arguments

lines	Treatments, number of lines for test.
-------	---------------------------------------

checks	Number of checks per augmented block.
b	Number of augmented blocks.
l	Number of locations. By default $l = 1$.
planter	Option for serpentine or cartesian arrangement. By default <code>planter = 'serpentine'</code> .
plotNumber	Numeric vector with the starting plot number for each location. By default <code>plotNumber = 101</code> .
repsStack	Option for horizontal or vertical layout By default <code>repsStack = 'vertical'</code> .
exptName	(optional) Name of experiment.
seed	(optional) Real number that specifies the starting seed to obtain reproducible designs.
locationNames	(optional) Name for each location.
repsExpt	(optional) Number of reps of experiment. By default <code>repsExpt = 1</code> .
random	Logical value to randomize treatments or not. By default <code>random = TRUE</code> .
data	(optional) Data frame with the labels of treatments.
nrows	(optional) Number of rows in the field.
ncols	(optional) Number of columns in the field.
year	(optional) Year recorded in the YEAR column of the field book. By default the current year.

Value

A list with five elements.

- `infoDesign` is a list with information on the design parameters.
- `layoutRandom` is the ARCBD layout randomization for the first location.
- `plotNumber` is the plot number layout for the first location.
- `exptNames` is the experiment names layout.
- `data_entry` is a data frame with the data input.
- `fieldBook` is a data frame with the ARCBD field book.

Reproducibility

The result records effective inputs and the resolved seed in `metadata$parameters`. Under the same package versions and RNG settings, rebuild a result `x` with `do.call(RCBD_augmented, x$metadata$parameters)`.

Author(s)

Didier Murillo [aut], Salvador Gezan [aut], Ana Heilman [ctb], Thomas Walk [ctb], Johan Aparicio [ctb], Richard Horsley [ctb]

References

Federer, W. T. (1955). Experimental Design. Theory and Application. New York, USA. The Macmillan Company.

Examples

```
# Example 1: Generates an ARCBD with 6 blocks, 3 checks for each, and 50 treatments
# in two locations.
ARCBD1 <- RCBDAugmented(lines = 50, checks = 3, b = 6, l = 2,
                        planter = "cartesian",
                        plotNumber = c(1,1001),
                        seed = 23,
                        locationNames = c("FARGO", "MINOT"))

ARCBD1$infoDesign
ARCBD1$layoutRandom
ARCBD1$exptNames
ARCBD1$plotNumber
head(ARCBD1$fieldBook, 12)

# Example 2: Generates an ARCBD with 17 blocks, 4 checks for each, and 350 treatments
# in 3 locations.
# In this case, we show how to use the option data.
checks <- 4;
list_checks <- paste("CH", 1:checks, sep = "")
treatments <- paste("G", 5:354, sep = "")
treatment_list <- data.frame(list(ENTRY = 1:354, NAME = c(list_checks, treatments)))
head(treatment_list, 12)
ARCBD2 <- RCBDAugmented(lines = 350, checks = 4, b = 17, l = 3,
                        planter = "serpentine",
                        plotNumber = c(101,1001,2001),
                        seed = 24,
                        locationNames = LETTERS[1:3],
                        data = treatment_list)

ARCBD2$infoDesign
ARCBD2$layoutRandom
ARCBD2$exptNames
ARCBD2$plotNumber
head(ARCBD2$fieldBook, 12)
```

read_design

Read a validated recorded design

Description

Read a validated recorded design

Usage

```
read_design(file, format = "rds")
```

Arguments

file Local filename. Only read serialized files from trusted sources.

format Explicit format key from `design_formats()`; currently "rds" is the lossless import format.

Details

Imports validate the result schema, recorded parameters and known engine. They do not randomize, reconstruct a design, or execute an R script. Unknown schema versions are rejected, not silently coerced. Pre-metadata legacy objects remain usable through `readRDS()` but do not contain the recorded-input contract required by this importer.

Value

The complete saved result, unchanged. File/format input mistakes raise `fieldhub_input_error`; corrupt or incompatible contents raise `fieldhub_import_error`, with the original condition in parent.

Examples

```
file <- tempfile(fileext = ".rds")
x <- RCBD(t = 5, reps = 3, seed = 27)
write_design(x, file)
identical(read_design(file), x)
unlink(file)
```

<code>rectangular_lattice</code>	<i>Generates a Rectangular Lattice Design.</i>
----------------------------------	--

Description

It randomly generates a rectangular lattice design across locations.

Usage

```
rectangular_lattice(
  t = NULL,
  k = NULL,
  r = NULL,
  l = 1,
  plotNumber = 101,
  locationNames = NULL,
  seed = NULL,
  data = NULL,
  reps = NULL
)
```

Arguments

t	Number of treatments, or a character vector with the treatment labels.
k	Size of incomplete blocks (number of units per incomplete block).
r	Deprecated alias for reps; positional calls remain supported.
l	Number of locations. By default $l = 1$.
plotNumber	Numeric vector with the starting plot number for each location. By default <code>plotNumber = 101</code> .
locationNames	(optional) Names for each location.
seed	(optional) Real number that specifies the starting seed to obtain reproducible designs.
data	(optional) Data frame with label list of treatments.
reps	Number of full resolvable replicates per location.

Value

A list with two elements.

- `infoDesign` is a list with information on the design parameters.
- `fieldBook` is a data frame with the rectangular lattice design field book.

Reproducibility

The result records effective inputs and the resolved seed in `metadata$parameters`, using `reps` for replication. Under the same package versions and RNG settings, rebuild a result `x` with `do.call(rectangular_lattice, x$metadata$parameters)`.

Author(s)

Didier Murillo [aut], Salvador Gezan [aut], Ana Heilman [ctb], Thomas Walk [ctb], Johan Aparicio [ctb], Richard Horsley [ctb]

References

Edmondson., R. N. (2021). `blocksdesign`: Nested and crossed block designs for factorial and unstructured treatment sets. <https://CRAN.R-project.org/package=blocksdesign>

Examples

```
# Example 1: Generates a rectangular lattice design with 6 full blocks, 4 units per IBlock (k)
# and 20 treatments in one location.
rectangularLattice1 <- rectangular_lattice(t = 20, k = 4, reps = 6, l = 1,
                                           plotNumber = 101,
                                           locationNames = "FARGO",
                                           seed = 126)

rectangularLattice1$infoDesign
head(rectangularLattice1$fieldBook, 12)
```

```
# Example 2: Generates a rectangular lattice design with 5 full blocks, 7 units per IBlock (k)
# and 56 treatments across 2 locations.
# In this case, we show how to use the option data.
treatments <- paste("ND-", 1:56, sep = "")
ENTRY <- 1:56
treatment_list <- data.frame(list(ENTRY = ENTRY, TREATMENT = treatments))
head(treatment_list)
rectangularLattice2 <- rectangular_lattice(t = 56, k = 7, reps = 5, l = 2,
                                           plotNumber = c(1001, 2001),
                                           locationNames = c("Loc1", "Loc2"),
                                           seed = 127,
                                           data = treatment_list)

rectangularLattice2$infoDesign
head(rectangularLattice2$fieldBook, 12)
```

reproduce_design

Reconstruct a design from its recorded inputs

Description

Calls the design engine using the input parameters and random-number settings recorded in a result's metadata.

Usage

```
reproduce_design(x)
```

Arguments

x	A FieldHub design, allocation plan, or pair-swap optimization result with recorded parameters.
---	--

Details

The caller's RNG settings and `.Random.seed` are restored on exit, including when reconstruction fails. Arguments are passed as values, so language objects in the parameter list are not evaluated as code.

Reproduction requires the same software versions and platform for algorithms whose results depend on numerical optimization. A different FieldHub version signals a `fieldhub_reproduction_warning`; dependency or platform differences are not checked. Older results without recorded parameters cannot be reconstructed automatically, but remain usable for printing and plotting.

Value

A newly generated design or allocation plan, invisibly.

Examples

```
x <- RCBD(t = 5, reps = 3, seed = 38)
identical(reproduce_design(x), x)
```

reproduce_simulation	<i>Reconstruct recorded simulated responses</i>
----------------------	---

Description

Replays a classic truncated-normal or spatial AR1-by-AR1 simulation using its saved input field book, parameters, seed, and RNG settings.

Usage

```
reproduce_simulation(x)
```

Arguments

x	A simulation record containing input_field_book, field_book, and metadata. Metadata must contain model, schema_version, seed, rng_kind, package_version, and the named parameters used by that model.
---	---

Details

This reconstructs the responses, not the experimental design or its layout: it starts from the exact saved input field book. Use `reproduce_design()` separately to reconstruct the experimental design. The caller's RNG settings and `.Random.seed` are restored on exit. Recorded arguments are passed as values rather than evaluated as R code. A changed FieldHub version signals a `fieldhub_reproduction_warning`; reproduction across software versions or platforms is not guaranteed.

Spatial `min_value` and `max_value` determine the model's center and genetic-effect scale, not hard response bounds. Spatial records retain unrounded per-location simulation values separately from the field book, whose responses are rounded to two decimal places. Classic records use a truncated-normal model, with rounding also applied to the field book.

Value

The reconstructed simulation record, invisibly.

See Also

```
reproduce_design()
```

row_column

Generates a Resolvable Row-Column Design (RowColD)

Description

It randomly generates a resolvable row-column design (RowColD). The design is optimized in both rows and columns blocking factors. The randomization can be done across multiple locations.

Usage

```
row_column(
  t = NULL,
  nrows = NULL,
  r = NULL,
  l = 1,
  plotNumber = 101,
  locationNames = NULL,
  seed = NULL,
  iterations = NULL,
  data = NULL,
  method = c("onestage", "twostage"),
  latinize = FALSE,
  reps = NULL
)
```

Arguments

t	Number of treatments, or a character vector with the treatment labels.
nrows	Number of rows of a full resolvable replicate.
r	Deprecated alias for reps; positional calls remain supported.
l	Number of locations. By default l = 1.
plotNumber	Numeric vector with the starting plot number for each location. By default plotNumber = 101.
locationNames	(optional) Names for each location.
seed	(optional) Real number that specifies the starting seed to obtain reproducible designs.
iterations	Finite positive whole-number optimization budget, at most <code>.Machine\$integer.max</code> . Its meaning and default depend on method. For method = "onestage" it is passed to <code>blocksdesign::design()</code> as its number of searches (default 200; values beyond a few hundred rarely improve the design). For method = "twostage" it is the number of greedy row-swap iterations (default 1000). If NULL (the default), a value appropriate to the chosen method is used.
data	(optional) Data frame with label list of treatments

method	Optimization strategy for the row and column blocking factors. Either "onestage" (default) or "twostage". With "onestage" (also known as simultaneous) the rows and columns are optimized jointly with <code>blocksdesign::design()</code> , which usually yields a higher joint row-by-column A-Efficiency while leaving the per-factor row and column efficiencies almost unchanged (see Details and the Row-by-Column row of <code>blocksModel</code>); for "twostage" the <code>iterations</code> argument is passed as the number of searches to <code>blocksdesign::design()</code> . With "twostage" the columns are optimized first and the rows are then improved by a greedy search, keeping the columns fixed; this reproduces the designs generated by previous versions of FieldHub. "onestage" automatically falls back to "twostage" (with a warning) for very small designs whose joint one-stage model is over-parameterized, so "twostage" rarely has to be requested explicitly.
latinize	If TRUE, optimize the row and column blocking factors as crossed (latinized) effects shared across the replicates, so that each treatment tends to appear in different rows and columns across the replicates. By default FALSE. Only meaningful with <code>method = "onestage"</code> ; it is ignored for <code>method = "twostage"</code> (which cannot latinize across replicates), where passing <code>latinize = TRUE</code> is ignored with a warning.
reps	Number of full resolvable replicates per location.

Details

With `method = "twostage"`, the Row-Column design is built in two stages. The first step constructs the blocking factor Columns using Incomplete Block Units from an incomplete block design that sets the number of incomplete blocks as the number of Columns in the design, each of which has a dimension equal to the number of Rows. Once this design is generated, the Rows are used as the Row blocking factor that is optimized for A-Efficiency, but levels within the original Columns are fixed. To optimize the Rows while maintaining the current optimized Columns, we use a heuristic algorithm that swaps at random treatment positions within a given Column (Block) also selected at random. The algorithm begins by calculating the A-Efficiency on the initial design, performs a swap iteration, recalculates the A-Efficiency on the resulting design, and compares it with the previous one to decide whether to keep or discard the new design. This iterative process is repeated, by default, 1,000 times. This two-stage greedy search reproduces the designs generated by previous versions of FieldHub and is available via `method = "twostage"`.

When `method = "onestage"` (the default, also called simultaneous), the rows and columns are optimized jointly in a single stage with `blocksdesign::design()` instead of optimizing the columns first and the rows afterwards. The joint row-by-column A-Efficiency of the standard analysis model (rows and columns adjusted together) is reported as the Row-by-Column row of `blocksModel`. With the default `latinize = FALSE` the joint optimization targets this within-replicate criterion and typically increases it relative to "twostage"; with `latinize = TRUE` it instead targets the across-replicate latinization described below. The per-factor (marginal) Row and Column A-Efficiencies are already close to optimal under both methods and differ only slightly between them (and not systematically in one direction). The size of the joint gain is layout-dependent and tends to be largest for small designs with few replicates. For `method = "onestage"`, the `iterations` argument is passed to `blocksdesign::design()` as its number of searches. For a very small design, the joint one-stage model can be over-parameterized (no residual degrees of freedom); in that case `row_column()` issues a warning and automatically falls back to `method = "twostage"` rather than failing, so a design is still produced.

Setting `latinize = TRUE` (only available with `method = "onestage"`) codes the rows and columns as crossed blocking factors shared across the replicates, so that a treatment tends to occupy different rows and different columns in the different replicates (a latinized resolvable row-column design in the sense of John and Williams). This guards against the same treatment recurring in the same field row or column over the whole trial, not just within a replicate, and is useful when a field trend runs along the whole-field rows or columns. It optimizes a different (crossed) model than the default nested one, so the two are not directly comparable, and full latinization is only possible when there are at least as many rows and columns as replicates. By default (`latinize = FALSE`) the rows and columns stay nested within each replicate. `latinize` has no effect for `method = "twostage"`, which cannot latinize across replicates.

Value

A list with five elements.

- `infoDesign` is a list with information on the design parameters.
- `blocksModel` is a list (one element per location) with the efficiency of the design. In addition to the per-factor Rep, Row and Column A- and D-Efficiencies, it reports a Row-by-Column row with the joint A- and D-Efficiency of the standard resolvable row-column analysis model (rows and columns adjusted together).
- `resolvableBlocks` a list with the resolvable row columns blocks.
- `concurrency` is the concurrency matrix.
- `fieldBook` is a data frame with the row-column field book.

Reproducibility

The result records effective inputs and the resolved seed in `metadata$parameters`, using `reps` for replication. Under the same package versions and RNG settings, rebuild a result `x` with `do.call(row_column, x$metadata$parameters)`.

Author(s)

Didier Murillo [aut], Salvador Gezan [aut], Ana Heilman [ctb], Thomas Walk [ctb], Johan Aparicio [ctb], Richard Horsley [ctb]

References

Edmondson., R. N. (2021). `blocksdesign`: Nested and crossed block designs for factorial and un-structured treatment sets. <https://CRAN.R-project.org/package=blocksdesign>

Examples

```
# Example 1: Generates a row-column design with 2 full blocks and 24 treatments
# and 6 rows, for one location. This uses the default method = "onestage", for
# which a small number of iterations (passed as searches) already captures most
# of the joint-efficiency gain.
rowcold1 <- row_column(
  t = 24,
  nrow = 6,
```

```

    reps = 2,
    l = 1,
    plotNumber= 101,
    locationNames = "Loc1",
    iterations = 20,
    seed = 21
)
rowcold1$infoDesign
rowcold1$resolvableBlocks
head(rowcold1$fieldBook,12)

# Example 2: Generates a row-column design with 2 full blocks and 30 treatments
# and 5 rows, for one location, using the default method = "onestage".
# In this case, we show how to use the option data.
treatments <- paste("ND-", 1:30, sep = "")
ENTRY <- 1:30
treatment_list <- data.frame(list(ENTRY = ENTRY, TREATMENT = treatments))
head(treatment_list)
rowcold2 <- row_column(
  t = 30,
  nrows = 5,
  reps = 2,
  l = 1,
  plotNumber= 1001,
  locationNames = "A",
  seed = 15,
  iterations = 20,
  data = treatment_list
)
rowcold2$infoDesign
rowcold2$resolvableBlocks
head(rowcold2$fieldBook,12)

# Example 3: Same design as Example 1 but using the historical two-stage
# optimization (method = "twostage"), which reproduces the designs generated
# by previous versions of FieldHub. For twostage, iterations is the number of
# greedy row-swap iterations; its default is 1000, but a smaller value is
# used here to keep the example quick.
rowcold3 <- row_column(
  t = 24,
  nrows = 6,
  reps = 2,
  l = 1,
  plotNumber = 101,
  locationNames = "Loc1",
  iterations = 100,
  method = "twostage",
  seed = 21
)
rowcold3$infoDesign
# The Row-by-Column row reports the joint row-by-column A-Efficiency.
rowcold3$blocksModel
head(rowcold3$fieldBook, 12)

```

`run_app`*Run the Shiny Application*

Description

Run the Shiny Application

Usage

```
run_app(..., launch.browser = TRUE, workers = 0L)
```

Arguments

<code>...</code>	Unused, for extensibility
<code>launch.browser</code>	Logical. If 'TRUE', the application is launched in the system's default web browser.
<code>workers</code>	Non-negative integer. Background workers; the default 0L keeps synchronous execution and does not start any worker processes.

Details

Constructing the app does not change R options. On startup, uploads are limited to 100 MiB (100 * 1024^2 bytes); when the app stops, the previous upload-limit option is restored, including an unset option. Shiny uses a process-wide upload limit, shared by the app's sessions while it is running. Closing one session does not restore the limit for others.

The standard app packages are installed with FieldHub. If any are unavailable, this function signals a `fieldhub_dependency_error` with an installation command and the missing package names in its `packages` field. It never installs packages automatically. Only the background-worker backend is optional.

With `workers > 0`, an installed package and the optional **mirai** package run design jobs in a private background pool. Other sessions remain responsive while a design runs. The pool closes when the app stops. Without `mirai`, or with development source, execution stays synchronous with a message.

Value

A shiny app object

sparse_allocation	<i>Unreplicated designs using the sparse allocation approach</i>
-------------------	--

Description

Unreplicated designs using the sparse allocation approach

Usage

```
sparse_allocation(
  lines,
  nrows,
  ncols,
  l,
  planter = "serpentine",
  plotNumber,
  copies_per_entry,
  checks = NULL,
  exptName = NULL,
  locationNames,
  sparse_list,
  seed,
  data = NULL,
  year = NULL,
  checksPercent = NULL
)
```

Arguments

lines	Number of genotypes, experimental lines or treatments.
nrows	Number of rows in the field.
ncols	Number of columns in the field.
l	Number of locations or sites. By default l = 1.
planter	Option for serpentine or cartesian plot arrangement. By default planter = 'serpentine'.
plotNumber	Numeric vector with the starting plot number for each location. By default plotNumber = 101.
copies_per_entry	Number of copies per plant. When design is sparse then copies_per_entry < 1
checks	Number of genotypes checks.
exptName	(optional) Name of the experiment.
locationNames	(optional) Names each location.
sparse_list	(optional) A class "Sparse" object generated by do_optim() function.

seed	(optional) Real number that specifies the starting seed to obtain reproducible designs.
data	(optional) Data frame with 2 columns: ENTRY NAME . ENTRY must be numeric.
year	(optional) Year recorded in the YEAR column of the field book. By default the current year.
checksPercent	(optional) Percentage of checks in each location, one of the options available for the field. By default the last (largest) option.

Value

A list with eight elements.

- infoDesign is a list with information on the design parameters.
- layoutRandom is a list with the randomization layout of each location.
- plotsNumber is a list with the plot number layout of each location.
- data_entry is a data frame with the data input.
- fieldBook is a data frame with the field book of all locations.
- list_locs is a list with each location list of entries.
- allocation is a matrix with the allocation of treatments.
- size_locations is a named vector with the number of lines allocated to each location.

Reproducibility

The result records effective inputs and the resolved seed in metadata\$parameters. Under the same package versions and RNG settings, rebuild a result x with `do.call(sparse_allocation, x$metadata$parameters)`.

Author(s)

Didier Murillo [aut], Salvador Gezan [aut], Ana Heilman [ctb]

References

Edmondson, R.N. Multi-level Block Designs for Comparative Experiments. JABES 25, 500–522 (2020). <https://doi.org/10.1007/s13253-020-00416-0>

Examples

```
sparse <- sparse_allocation(
  lines = 120,
  l = 4,
  copies_per_entry = 3,
  checks = 4,
  locationNames = c("LOC1", "LOC2", "LOC3", "LOC4"),
  seed = 1234
)
```

split_families	<i>Split a population of genotypes randomly into several locations.</i>
----------------	---

Description

Split a population of genotypes randomly into several locations, with the aim of having approximately the same number of replicates of each genotype, line or treatment per location.

Usage

```
split_families(l = NULL, data = NULL, seed = NULL)
```

Arguments

l	Number of locations.
data	Data frame with the entry (ENTRY) and the labels of each treatment (NAME) and number of individuals per family group (FAMILY).
seed	(optional) A single real number specifying the random seed. When omitted, one integer is drawn from the current random-number stream and recorded in <code>infoDesign\$seed</code> and <code>metadata\$seed</code> ; the allocation's own randomization does not change the caller's stream.

Details

To reproduce an allocation previously made with `set.seed(s); split_families(l, data)`, use `split_families(l, data, seed = s)`.

Value

A list with two elements.

- `rowsEachlist` is a table with a summary of cases.
- `data_locations` is a data frame with the entries for each location

Reproducibility

The result records effective inputs and the resolved seed in `metadata$parameters`. Under the same package versions and RNG settings, rebuild a result `x` with `do.call(split_families, x$metadata$parameters)`.

Author(s)

Didier Murillo [aut], Salvador Gezan [aut], Ana Heilman [ctb], Thomas Walk [ctb], Johan Aparicio [ctb], Richard Horsley [ctb]

Examples

```
# Example 1: Split a population of 3000 and 200 families into 8 locations.
# Original dataset is been simulated.
set.seed(77)
N <- 2000; families <- 100
ENTRY <- 1:N
NAME <- paste0("SB-", 1:N)
FAMILY <- vector(mode = "numeric", length = N)
x <- 1:N
for (i in x) { FAMILY[i] <- sample(1:families, size = 1, replace = TRUE) }
gen.list <- data.frame(list(ENTRY = ENTRY, NAME = NAME, FAMILY = FAMILY))
head(gen.list)
# Now we are going to use the split_families() function.
split_population <- split_families(l = 8, data = gen.list, seed = 77)
print(split_population)
summary(split_population)
head(split_population$data_locations, 12)
```

split_plot

Generates a Split Plot Design

Description

It randomly generates a split plot design (SPD) across locations.

Usage

```
split_plot(
  wp = NULL,
  sp = NULL,
  reps = NULL,
  type = 2,
  l = 1,
  plotNumber = 101,
  seed = NULL,
  locationNames = NULL,
  factorLabels = TRUE,
  data = NULL
)
```

Arguments

wp	Number of whole plots, as an integer or a vector.
sp	Number of sub plots per whole plot, as an integer or a vector.
reps	Number of blocks (full replicates).

type	Option for CRD or RCBD designs. Values are type = 1 (CRD) or type = 2 (RCBD). By default type = 2.
l	Number of locations. By default l = 1.
plotNumber	Numeric vector with the starting plot number for each location. By default plotNumber = 101.
seed	(optional) Real number that specifies the starting seed to obtain reproducible designs.
locationNames	(optional) Names for each location.
factorLabels	(optional) If TRUE retain the levels labels from the original data set otherwise, numeric labels will be assigned. Default is factorLabels = TRUE.
data	(optional) Data frame with label list of treatments.

Value

A list with two elements.

- infoDesign is a list with information on the design parameters.
- fieldBook is a data frame with the split plot field book.

Reproducibility

The result records effective inputs and the resolved seed in metadata\$parameters. Under the same package versions and RNG settings, rebuild a result x with `do.call(split_plot, x$metadata$parameters)`.

Author(s)

Didier Murillo [aut], Salvador Gezan [aut], Ana Heilman [ctb], Thomas Walk [ctb], Johan Aparicio [ctb], Richard Horsley [ctb]

References

Federer, W. T. (1955). Experimental Design. Theory and Application. New York, USA. The Macmillan Company.

Examples

```
# Example 1: Generates a split plot design SPD with 4 whole plots, 2 sub plots per whole plot,
# and 4 reps in an RCBD arrangement. This in for a single location.
SPDExample1 <- split_plot(wp = 4, sp = 2, reps = 5, l = 1,
                          plotNumber = 101,
                          seed = 14,
                          type = 2,
                          locationNames = "FARGO")

SPDExample1$infoDesign
SPDExample1$layoutlocations
head(SPDExample1$fieldBook,12)

# Example 2: Generates a split plot design SPD with 5 whole plots
# (4 types of fungicide + one control), 10 sub plots per whole plot (10 bean varieties),
```

```
# and 6 reps in an RCBD arrangement. This in 3 locations or sites.
# In this case, we show how to use the option data.
wp <- c("NFunc", paste("Func", 1:4, sep = "")) # Fungicides (5 Whole plots)
sp <- paste("Beans", 1:10, sep = "")           # Beans varieties (10 sub plots)
split_plot_Data <- data.frame(list(WHOLPLOT = c(wp, rep(NA, 5)), SUBPLOT = sp))
head(split_plot_Data, 12)
SPDExample2 <- split_plot(reps = 6, l = 3,
                          plotNumber = c(101, 1001, 2001),
                          seed = 23,
                          type = 2,
                          locationNames = c("A", "B", "C"),
                          data = split_plot_Data)

SPDExample2$infoDesign
SPDExample2$layoutlocations
head(SPDExample2$fieldBook, 12)
```

split_split_plot

Generates a Split Split Plot Design

Description

It randomly generates a split split plot design (SSPD) across locations.

Usage

```
split_split_plot(
  wp = NULL,
  sp = NULL,
  ssp = NULL,
  reps = NULL,
  type = 2,
  l = 1,
  plotNumber = 101,
  seed = NULL,
  locationNames = NULL,
  factorLabels = TRUE,
  data = NULL
)
```

Arguments

wp	Number of whole plots, as an integer or a vector.
sp	Number of sub plots per whole plot, as an integer or a vector.
ssp	Number of sub-sub plots, as an integer or a vector.
reps	Number of blocks (full replicates).

type	Option for CRD or RCBD designs. Values are type = 1 (CRD) or type = 2 (RCBD). By default type = 2.
l	Number of locations. By default l = 1.
plotNumber	Numeric vector with the starting plot number for each location. By default plotNumber = 101.
seed	(optional) Real number that specifies the starting seed to obtain reproducible designs.
locationNames	(optional) Names for each location.
factorLabels	(optional) If TRUE retain the levels labels from the original data set otherwise, numeric labels will be assigned. Default is factorLabels = TRUE.
data	(optional) Data frame with label list of treatments.

Value

A list with two elements.

- infoDesign is a list with information on the design parameters.
- fieldBook is a data frame with the split split plot field book.

Reproducibility

The result records effective inputs and the resolved seed in metadata\$parameters. Under the same package versions and RNG settings, rebuild a result x with `do.call(split_split_plot, x$metadata$parameters)`.

Author(s)

Didier Murillo [aut], Salvador Gezan [aut], Ana Heilman [ctb], Thomas Walk [ctb], Johan Aparicio [ctb], Richard Horsley [ctb]

References

Federer, W. T. (1955). Experimental Design. Theory and Application. New York, USA. The Macmillan Company.

Examples

```
# Example 1: Generates a split split plot design SSPD with 5 whole plots, 2 sub-plots,
# 3 sub-sub plots, and 3 reps in an RCBD arrangement. This is for one location.
SSPD1 <- split_split_plot(wp = 4, sp = 2, ssp = 3, reps = 5, l = 1,
                          plotNumber = 101,
                          seed = 23,
                          type = 2,
                          locationNames = "FARGO")

SSPD1$infoDesign
head(SSPD1$fieldBook, 12)

# Example 2: Generates a split split plot design SSPD with 2 whole plot
# (Irrigation, No irrigation), 5 sub plots (4 types of fungicide + one control), and
```

```
# 10 sub-sub plots (Ten varieties of beans), and 4 reps in an RCBD arrangement.
# This is for 3 locations. In this case, we show how to use the option data.
wp <- paste("IRR_", c("NO", "Yes"), sep = "") #Irrigation (2 Whole plots)
sp <- c("NFung", paste("Fung", 1:4, sep = "")) #Fungicides (5 Sub plots)
ssp <- paste("Beans", 1:10, sep = "") #Beans varieties (10 Sub-sub plots)
split_split_plot_Data <- data.frame(list(WHOLPLOT = c(wp, rep(NA, 8)),
                                         SUBPLOT = c(sp, rep(NA, 5)),
                                         SUB_SUBPLOTS = ssp))

head(split_split_plot_Data, 10)
SSPD2 <- split_split_plot(reps = 4, l = 3,
                          plotNumber = c(101, 1001, 2001),
                          seed = 23,
                          type = 2,
                          locationNames = c("A", "B", "C"),
                          data = split_split_plot_Data)

SSPD2$infoDesign
head(SSPD2$fieldBook, 12)
```

square_lattice

Generates a Square Lattice Design.

Description

It randomly generates a square lattice design across locations.

Usage

```
square_lattice(
  t = NULL,
  k = NULL,
  r = NULL,
  l = 1,
  plotNumber = 101,
  locationNames = NULL,
  seed = NULL,
  data = NULL,
  reps = NULL
)
```

Arguments

t	Number of treatments, or a character vector with the treatment labels.
k	Size of incomplete blocks (number of units per incomplete block).
r	Deprecated alias for reps; positional calls remain supported.
l	Number of locations. By default l = 1.
plotNumber	Numeric vector with the starting plot number for each location. By default plotNumber = 101.


```
seed = 1986,
data = treatment_list)

squareLattice2$infoDesign
head(squareLattice2$fieldBook,12)
```

strip_plot	<i>Strip Plot Design</i>
------------	--------------------------

Description

It randomly generates a strip plot design across locations.

Usage

```
strip_plot(
  Hplots = NULL,
  Vplots = NULL,
  b = 1,
  l = 1,
  plotNumber = NULL,
  planter = "serpentine",
  locationNames = NULL,
  seed = NULL,
  factorLabels = TRUE,
  randomizeH = TRUE,
  randomizeV = FALSE,
  data = NULL,
  reps = 1
)
```

Arguments

Hplots	Number of horizontal factors, as an integer or a vector.
Vplots	Number of vertical factors, as an integer or a vector.
b	Deprecated alias for reps; positional calls remain supported.
l	Number of locations. By default l = 1.
plotNumber	Numeric vector with the starting plot number for each location. By default plotNumber = 101.
planter	Option for serpentine or cartesian arrangement. By default planter = 'serpentine'.
locationNames	(optional) Names for each location.
seed	(optional) Real number that specifies the starting seed to obtain reproducible designs.
factorLabels	(optional) If TRUE retain the levels labels from the original data set otherwise, numeric labels will be assigned. Default is factorLabels =TRUE.

randomizeH	Logical flag: TRUE to randomize horizontal strips separately in each replicate; FALSE to randomize once per location and reuse.
randomizeV	Logical flag: TRUE to randomize vertical strips separately in each replicate; FALSE to randomize once per location and reuse.
data	(optional) data frame with the labels of vertical and horizontal plots.
reps	Number of blocks (full replicates) per location. Default is one.

Value

A list with four elements.

- infoDesign is a list with information on the design parameters.
- stripsBlockLoc is a list with the strip blocks for each location.
- plotLayouts is a list with the layout plot numbers for each location.
- fieldBook is a data frame with the strip plot field book.

Reproducibility

The result records effective inputs and the resolved seed in `metadata$parameters`. Under the same package versions and RNG settings, rebuild a result `x` with `do.call(strip_plot, x$metadata$parameters)`.

Author(s)

Didier Murillo [aut], Salvador Gezan [aut], Ana Heilman [ctb], Thomas Walk [ctb], Johan Aparicio [ctb], Richard Horsley [ctb]

References

Federer, W. T. (1955). Experimental Design. Theory and Application. New York, USA. The Macmillan Company.

Examples

```
# Example 1: Generates a strip plot design with 5 vertical strips and 4 horizontal strips,
# with 3 reps in one location.
H <- paste("H", 1:4, sep = "")
V <- paste("V", 1:5, sep = "")
strip1 <- strip_plot(
  Hplots = H,
  Vplots = V,
  reps = 3,
  l = 1,
  plotNumber = 101,
  planter = "serpentine",
  locationNames = "A",
  seed = 333
)
strip1$infoDesign
strip1$stripsBlockLoc
```

```

strip1$plotLayouts
head(strip1$fieldBook, 12)

# Example 2: Generates a strip plot design with 5 vertical strips and 5 horizontal strips,
# with 6 reps across to 3 locations. In this case, we show how to use the option data.
Hplots <- LETTERS[1:5]
Vplots <- LETTERS[1:4]
strip_data <- data.frame(list(HPLOTS = Hplots, VPLOTS = c(Vplots, NA)))
head(strip_data)
strip2 <- strip_plot(
  Hplots = 5,
  Vplots = 5,
  reps = 6,
  l = 3,
  plotNumber = c(101, 1001, 2001),
  planter = "cartesian",
  locationNames = c("A", "B", "C"),
  seed = 222,
  data = strip_data
)
strip2$infoDesign
strip2$stripsBlockLoc
strip2$plotLayouts
head(strip2$fieldBook, 12)

```

summary.FieldHub

Summary a FieldHub object

Description

Summarise information on the design parameters, and data frame structure

Usage

```
## S3 method for class 'FieldHub'
summary(object, ...)
```

Arguments

object	an object inheriting from class FieldHub
...	Unused, for extensibility

Value

an object inheriting from class summary.FieldHub

Author(s)

Thiago de Paula Oliveira, <thiago.paula.oliveira@alumni.usp.br>

Examples

```
# Example 1: Generates a CRD design with 5 treatments and 5 reps each.
crd1 <- CRD(t = 5, reps = 5, plotNumber = 101,
seed = 1985, locationNames = "Fargo")
crd1$infoDesign
summary(crd1)
```

swap_pairs

Swap pairs in a matrix of integers

Description

Attempts to separate repeated entries by swapping cells of different entries. Distance thresholds increase from `starting_dist` in steps of one. Candidate sampling, mean pairwise distance, and a centrality penalty guide the search. The last successful threshold layout is returned; if none succeeds, the original matrix is returned. A requested minimum distance is not guaranteed.

Usage

```
swap_pairs(
  X,
  starting_dist = 3,
  stop_iter = 10,
  lambda = 0.5,
  dist_method = "euclidean",
  candidate_sample_size = 4,
  seed = NULL
)
```

Arguments

<code>X</code>	A numeric matrix of whole-number entry identifiers within R's integer range. Missing cells are inactive positions and never move.
<code>starting_dist</code>	First distance threshold; a finite nonnegative number. Default is 3.
<code>stop_iter</code>	Maximum complete swap sweeps per threshold, as a nonnegative whole number within R's integer range. Default is 10.
<code>lambda</code>	Finite nonnegative weight for the centrality penalty. Default is 0.5.
<code>dist_method</code>	Coordinate distance used for candidate filtering, scoring, stopping, and reported distances: "euclidean" (default) or "manhattan".
<code>candidate_sample_size</code>	Maximum candidates evaluated per swap, as a positive whole number within R's integer range. Default is 4.
<code>seed</code>	Optional randomization seed. When omitted, one integer is drawn from the current random-number stream and recorded; the swap's own randomization does not change the caller's stream.

Details

The finite threshold range is bounded by the field geometry. For Euclidean distance without missing cells, the historical bound is $\sqrt{\text{nrow}(X)^2 + \text{ncol}(X)^2}$; with missing cells it is the maximum distance between active positions. For Manhattan distance it is the maximum Manhattan distance between active positions. There are at most $\text{floor}(\text{bound} - \text{starting_dist}) + 1$ thresholds when the bound is at least `starting_dist`, and none otherwise. Each threshold permits at most `stop_iter` complete sweeps. Diagnostics distinguish an exhausted threshold budget from completing or skipping the distance range.

Standalone calls use the shared seed contract. To reproduce a previous `set.seed(s); swap_pairs(X)` call, with `X` already constructed, pass `seed = s`. Calls without a seed now select an automatic seed, so their layouts can differ from previous versions. Optimization performed inside field-design engines retains its existing draw sequence. Use `reproduce_design()` to replay a recorded optimization result.

Value

A list containing:

<code>optim_design</code>	The modified matrix.
<code>designs</code>	A list of all intermediate designs, starting from the input matrix.
<code>distances</code>	A list of all pair distances for each intermediate design.
<code>min_distance</code>	The minimum distance between pairs of occurrences of the same integer in the final design.
<code>pairwise_distance</code>	A data frame with the pairwise distances for the final design.
<code>rows_incidence</code>	Row-repetition counts for retained threshold steps, or for the original matrix when no step succeeds.
<code>diagnostics</code>	The distance metric, stop reason, total completed sweeps (iterations), per-threshold budget, number of attempted thresholds, last threshold, last attempted minimum distance, and retained minimum distance. Stop reasons are <code>iteration_limit</code> , <code>distance_range_complete</code> , and <code>no_distance_thresholds</code> . A failed attempt is not retained in <code>optim_design</code> .
<code>metadata</code>	The design identifier, schema and package versions, RNG settings, resolved seed and complete input parameters, including <code>X</code> . The result inherits from <code>fieldhub_optimization</code> .

Examples

```
set.seed(123)
X <- matrix(sample(c(rep(1:10, 2), 11:50), replace = FALSE), ncol = 10)
B <- swap_pairs(
  X,
  starting_dist = 3,
  stop_iter = 50,
  lambda = 0.5,
  dist_method = "euclidean",
  candidate_sample_size = 3,
  seed = 123
```

```
)
B$optim_design
```

write_design	<i>Write a recorded design or standalone replay script</i>
--------------	--

Description

Write a recorded design or standalone replay script

Usage

```
write_design(x, file, format = "rds", overwrite = FALSE)
```

Arguments

x	A validated design, allocation or optimization result with recorded parameters. App-selected layouts and simulations are separate artifacts.
file	Local output filename; its parent directory must already exist.
format	Explicit exporter key from <code>design_formats()</code> : "rds" saves the exact result and metadata; "r" saves a standalone script assigning the reconstructed result to design.
overwrite	One logical value. Existing files are protected by default.

Details

Serialization completes in a temporary file before the destination is copied. This protects an existing file from serialization failures, but is not an atomic filesystem transaction. Disk/copy failures still require normal backups. The temporary file is removed on exit.

R scripts embed only portable data and never evaluate recorded arguments during export. Large uploads or nonportable values require the RDS format. Reconstruction can depend on matching software versions and platform; the RDS format preserves the actual saved result independently of replay.

Value

file, invisibly. Unsupported formats and overwrite mistakes raise `fieldhub_input_error`; write failures raise `fieldhub_export_error`.

Examples

```
file <- tempfile(fileext = ".rds")
write_design(CRD(t = 4, reps = 2, seed = 27), file)
read_design(file)$metadata
unlink(file)
```

Index

`alpha_lattice`, [3](#)
`CRD`, [5](#), [8](#)
`design_arguments`, [7](#), [10](#)
`design_formats`, [8](#)
`design_results`, [9](#)
`diagonal_arrangement`, [10](#)
`do_optim`, [14](#)

`field_layout`, [8](#), [10](#), [16](#)
`full_factorial`, [17](#)

`head`, [34](#)

`incomplete_blocks`, [8](#), [19](#)

`latin_rectangle`, [21](#)
`latin_square`, [22](#)

`multi_location_prep`, [24](#)

`optimized_arrangement`, [27](#)

`partially_replicated`, [29](#)
`plot.FieldHub`, [33](#)
`print.FieldHub`, [34](#)
`print.fieldLayout`, [35](#)
`print.summary.FieldHub`, [35](#)

`RCBD`, [36](#)
`RCBD_augmented`, [39](#)
`read_design`, [41](#)
`rectangular_lattice`, [42](#)
`reproduce_design`, [10](#), [44](#)
`reproduce_simulation`, [45](#)
`row_column`, [8](#), [46](#)
`run_app`, [50](#)

`sparse_allocation`, [51](#)
`split_families`, [53](#)
`split_plot`, [54](#)

`split_split_plot`, [56](#)
`square_lattice`, [58](#)
`strip_plot`, [8](#), [60](#)
`summary.FieldHub`, [62](#)
`swap_pairs`, [63](#)

`write_design`, [65](#)