Package: slideimp (via r-universe)

October 20, 2026

Type Package

Title Numeric Matrices K-NN and PCA Imputation

Version 1.2.0

Description Fast k-nearest neighbors (K-NN) and principal component analysis (PCA) imputation algorithms for missing values in epigenetic data or other high-dimensional numeric matrices. For PCA, a locally optimal block preconditioned conjugate gradient (LOBPCG) eigensolver with warm starts of both the eigenblock and search direction is also supported. Two complementary imputation strategies are available. Group-wise imputation (e.g., by chromosome) is recommended for Illumina DNA methylation microarrays (e.g., 450K, EPIC) and other matrices with groupable columns. A sliding window approach for K-NN or PCA imputation is recommended only for whole-genome methylation data such as whole-genome bisulfite sequencing (WGBS) or Enzymatic Methyl-seq (EM-seq). The package also supports hyperparameter tuning via repeated cross-validation. The K-NN algorithm is described in: Hastie, T., Tibshirani, R., Sherlock, G., Eisen, M., Brown, P. and Botstein, D. (1999) ``Imputing Missing Data for Gene Expression Arrays". The PCA imputation is an optimized reimplementation of the imputePCA() function from the 'missMDA' package described in: Josse, J. and Husson, F. (2016) <doi:10.18637/jss.v070.i01> ``missMDA: A Package for Handling Missing Values in Multivariate Data Analysis".

License GPL (>= 2)

URL https://github.com/hhp94/slideimp, https://hhp94.github.io/slideimp/

BugReports https://github.com/hhp94/slideimp/issues

Depends R (>= 4.3.0)

Imports bigmemory, checkmate, cli, collapse, mirai, Rcpp, stats, utils

Suggests knitr, missMDA, RhpcBLASctl, rmarkdown, testthat (>= 3.0.0), withr

LinkingTo Rcpp, RcppArmadillo, RcppThread

VignetteBuilder knitr

Config/Needs/website rmarkdown

Config/roxygen2/version 8.0.0

Config/testthat/edition 3

Encoding UTF-8

Language en-US

NeedsCompilation yes

Author Hung Pham [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0002-8271-9355>), Posit Software, PBC
[cph] (Copyright holder of code adapted from the 'carrier'
package, MIT licensed)

Maintainer Hung Pham <amser.hoanghung@gmail.com>

Repository https://cran.r-universe.dev

Date/Publication 2026-06-16 11:39:24 UTC

RemoteUrl https://github.com/cran/slideimp

RemoteRef HEAD

RemoteSha dfe3063fde4e7e8b0415aa23c1d99256cc77d6d1

Contents

col_vars . . . 3
compute_metrics . . . 4
group_imp . . . 5
knn_imp . . . 9
lobpcg_control . . . 11
mat_miss . . . 12
mean_imp_col . . . 13
pca_imp . . . 14
prep_groups . . . 16
print.slideimp_results . . . 18
print.slideimp_sim . . . 19
print.slideimp_tbl . . . 19
sample_na_loc . . . 20
sim_mat . . . 21
slide_imp . . . 23
slideimp_resolve_group . . . 27
tune_imp . . . 28

Index **33**

`col_vars` *Calculate Matrix Column Variances*

Description

Compute the sample variance for each column of a numeric matrix.

Usage

```
col_vars(obj, cores = 1)
```

Arguments

`obj`	A numeric matrix.
`cores`	Integer. Number of cores to use for parallel computation. Defaults to `1`.

Details

Columns with fewer than two non-missing values are assigned `NA`.

`NA`/`NaN` are treated as missing and dropped (equivalent to `var(x, na.rm = TRUE)`). `Inf`/`-Inf` are **not** missing. They enter the arithmetic, so a column's variance can be `NaN`, matching base R.

Value

A numeric vector of column variances, named if `obj` has column names.

Examples

```
set.seed(123)
obj <- matrix(rnorm(7 * 10), ncol = 7)
obj[1, 1] <- Inf
obj[1, 2] <- NA
obj[1:8, 3] <- NA
obj[8, 3] <- Inf
obj[1:8, 4] <- NA
obj[1:8, 5] <- NA
obj[9, 5] <- obj[10, 5]
obj[1:9, 6] <- NA
obj[, 7] <- NA
obj

col_vars(obj)
apply(obj, 2, var, na.rm = TRUE)
```

compute_metrics *Compute Prediction Accuracy Metrics*

Description

Compute prediction accuracy metrics for results from `tune_imp()`.

Usage

```
compute_metrics(results, metrics = c("mae", "rmse"))

## S3 method for class 'data.frame'
compute_metrics(results, metrics = c("mae", "rmse"))

## S3 method for class 'slideimp_tune'
compute_metrics(results, metrics = c("mae", "rmse"))
```

Arguments

`results`	A `slideimp_tune` data frame from `tune_imp()`. Must contain a `result` list-column whose elements are data frames with `truth` and `estimate` columns.
`metrics`	Character vector of metric names to compute. Defaults to `c("mae", "rmse")`. Available metrics are `"mae"`, `"rmse"`, `"mape"`, `"bias"`, `"rsq"`, and `"rsq_trad"`.

Value

A data frame containing the original parameter columns along with unnested metric columns: `.metric`, `.estimator`, and `.estimate`.

Examples

```
set.seed(1234)
obj <- sim_mat(20, 30)$input

results <- tune_imp(
  obj = obj,
  parameters = data.frame(k = 5),
  .f = "knn_imp",
  n_reps = 1,
  num_na = 10,
  .progress = FALSE
)

compute_metrics(results)
```

group_imp *Grouped K-NN or PCA Imputation*

Description

Perform K-NN or PCA imputation independently within feature groups.

Usage

```
group_imp(
  obj,
  group,
  subset = NULL,
  allow_unmapped = FALSE,
  k = NULL,
  ncp = NULL,
  method = NULL,
  cores = 1,
  .progress = TRUE,
  min_group_size = NULL,
  colmax = NULL,
  post_imp = NULL,
  dist_pow = NULL,
  scale = NULL,
  coeff.ridge = NULL,
  threshold = NULL,
  row.w = NULL,
  seed = NULL,
  nb.init = NULL,
  maxiter = NULL,
  miniter = NULL,
  solver = NULL,
  lobpcg_control = NULL,
  clamp = NULL,
  pin_blas = FALSE,
  na_check = TRUE,
  on_infeasible = c("error", "skip", "mean")
)
```

Arguments

obj	A numeric matrix with samples in rows and features in columns.
group	Specification of how features should be grouped for imputation. Accepted formats are: • A character scalar naming a supported Illumina platform; see Note. • A long-format `data.frame` with columns `group` and `feature`.

- A list-column `data.frame` with a `feature` list-column. Optional list-columns are `aux`, for auxiliary feature names, and `parameters`, for group-specific parameter lists.

`subset` Optional character vector of feature names to impute. If `NULL`, all grouped features are imputed. Features in a group but not in `subset` are demoted to auxiliary columns for that group. Groups left with zero features after demotion are dropped with a message.

`allow_unmapped` Logical. If `FALSE`, every column in `colnames(obj)` must appear in `group`. If `TRUE`, columns with no group assignment are left untouched and are not used as auxiliary columns.

`k` Integer or `NULL`. Number of nearest neighbors for K-NN imputation. If `NULL`, `k` may be supplied through `group$parameters`.

`ncp` Integer or `NULL`. Number of components for PCA imputation. If `NULL`, `ncp` may be supplied through `group$parameters`.

`method` Character or `NULL`. For K-NN imputation, one of `"euclidean"` or `"manhattan"`. For PCA imputation, one of `"regularized"` or `"EM"`. If `NULL`, the corresponding backend default is used unless overridden by `group$parameters`.

`cores` Integer. Number of cores for K-NN imputation only. For PCA imputation, use `mirai::daemons()` to parallelize across groups.

`.progress` Logical. If `TRUE`, show progress.

`min_group_size` Integer or `NULL`. Minimum total number of columns per group, counting both features and auxiliary columns. Groups smaller than this are padded with randomly sampled columns from `obj`.

`colmax` Numeric scalar between `0` and `1`. Columns with a missing-data proportion greater than `colmax` are excluded from the main imputation method. Excluded columns are left unchanged unless `post_imp = TRUE`, in which case remaining missing values are replaced by column means when possible.

`post_imp` Logical. If `TRUE`, replace missing values remaining after the main imputation method with column means when possible.

`dist_pow` Numeric. Power used to penalize more distant neighbors in the weighted average. `dist_pow = 0` gives an unweighted average of the nearest neighbors.

`scale` Logical. If `TRUE`, columns are scaled to unit variance.

`coeff.ridge` Numeric. Ridge regularization, used only when `method = "regularized"`. Values `< 1` move toward EM PCA; values `> 1` move toward mean imputation.

`threshold` Numeric. Convergence threshold.

`row.w` Row weights, normalized to sum to `1`. `NULL` (equal weights), a positive numeric vector of length `nrow(obj)`, or `"n_miss"` (down-weight rows with more missing values).

`seed` Integer, numeric, or `NULL`. Random seed for reproducibility.

`nb.init` Integer. Number of random initializations. The first initialization is always mean imputation.

`maxiter` Integer. Maximum number of iterations.

`miniter` Integer. Minimum number of iterations.

solver	Character. Eigensolver: "auto" (default), "exact", or "lobpcg". "auto" runs a short timed probe and picks "lobpcg" only when clearly faster. Consecutive EM calls warm-start LOBPCG with both the previous eigenblock and search direction. When nb.init > 1, the auto choice from the first init is reused. See Performance tips.
lobpcg_control	A list of LOBPCG eigensolver control options, usually created by lobpcg_control(). A plain named list is also accepted. Ignored when solver = "exact".
clamp	Optional numeric vector c(lower, upper) bounding PCA-imputed values (use -Inf/Inf for one-sided, NULL for none). E.g., c(0, 1) for DNAm beta values. Observed values are not clamped.
pin_blas	Logical. If TRUE, pin BLAS threads to 1 to reduce contention when using parallel PCA on systems linked with multithreaded BLAS.
na_check	Logical. If TRUE, check whether the returned matrix still contains missing values.
on_infeasible	Character. One of "error", "skip", or "mean". Controls behavior when a group is infeasible for imputation, for example when k or ncp exceeds the number of usable columns after applying colmax.

Details

group_imp() performs K-NN or PCA imputation on feature groups independently, which can substantially reduce runtime for large matrices.

Specify k and related arguments to use K-NN imputation, or ncp and related arguments to use PCA imputation. If both k and ncp are NULL, group$parameters must supply either k or ncp for every group.

Group-specific parameters in group$parameters take priority over global arguments. Global arguments fill in any missing values. All groups must use the same imputation method.

For method-specific arguments inherited from knn_imp() or pca_imp(), NULL means the backend default is used unless overridden by group$parameters.

Per-group k is capped at the number of usable columns in the group minus one. Per-group ncp is capped at the maximum feasible number of PCA components for that group's submatrix. A warning is issued when capping occurs.

Value

A numeric matrix of the same dimensions as obj, with missing values imputed. The returned object has class slideimp_results.

Parallelization

- K-NN: use the cores argument. If mirai daemons are active, cores is automatically set to 1 to avoid nested parallelism.
- PCA: use mirai::daemons() instead of cores.

When running PCA imputation in parallel with mirai, set pin_blas = TRUE in tune_imp() or group_imp() to prevent BLAS threads from oversubscribing CPU cores. This relies on RhpcBLASctl and works with OpenBLAS and MKL (typical on Linux, and on Windows after an OpenBLAS swap). pin_blas = TRUE may have no effect on macOS.

PCA Performance tips

Speed comes from three levers: `solver` (through LOBPCG with warm-start), `threshold`, and `scale`. Tune these first, then accuracy parameters (`ncp`, `coeff.ridge`) on a representative subset.

Exact vs. LOBPCG with warm-start. Whether `"lobpcg"` beats `"exact"` depends on size and low-rankness: `"lobpcg"` is preferred for large, approximately low-rank matrices with small `ncp`, and `"exact"` for small matrices (including `slide_imp()` windows), where it is faster and more robust. Separately, the warm-start makes each successive solve cheap: `pca_imp()` warm-starts LOBPCG with the previous eigenblock and search direction, so once imputed values stabilize, later solves converge in a few iterations. The payoff therefore grows with the number of EM iterations, independent of low-rankness. `solver = "auto"` (default) probes both and is a safe start.

Threshold. The default `1e-6` is conservative; `1e-5` is often faster with very similar values.

Scale. For columns on a common scale (e.g., DNAm beta values in `[0, 1]`), `scale = FALSE` can be faster and more accurate.

Parallel and BLAS. In parallel via `tune_imp()` or `group_imp()` with a multithreaded BLAS, set `pin_blas = TRUE` to avoid thread oversubscription. On Windows, the stock BLAS can be slow. Advanced users can swap in OpenBLAS.

See Speeding up PCA imputation for the full workflow.

Note

A character scalar can be passed to `group` to name a supported Illumina platform, such as `"EPICv2"` or `"EPICv2_deduped"`. This requires the optional `slideimp.extra` package to be installed. Supported platforms are listed in the `slideimp_arrays` object in the `slideimp.extra` package.

See Also

`prep_groups()`, `knn_imp()`, `pca_imp()`

Examples

```
set.seed(1234)
to_test <- sim_mat(10, 20, perc_total_na = 0.05, perc_col_na = 1)
obj <- to_test$input
group <- to_test$col_group
head(group)

# Simple grouped K-NN imputation
results <- group_imp(obj, group = group, k = 2, .progress = FALSE)
results

# Impute only a subset of features
subset_features <- sample(to_test$col_group$feature, size = 10)
knn_subset <- group_imp(
  obj,
  group = group,
  subset = subset_features,
  k = 2,
  .progress = FALSE
)
```

```
# Use prep_groups() to inspect and edit per-group parameters
prepped <- prep_groups(colnames(obj), group)
prepped$parameters <- lapply(seq_len(nrow(prepped)), function(i) list(k = 2))
prepped$parameters[[2]]$k <- 4
knn_grouped <- group_imp(obj, group = prepped, .progress = FALSE)


# PCA imputation with mirai parallelism
mirai::daemons(2)
pca_grouped <- group_imp(obj, group = group, ncp = 2)
mirai::daemons(0)
pca_grouped
```

---

knn_imp *K-Nearest Neighbor Imputation for Numeric Matrices*

Description

Impute missing values in a numeric matrix using full K-nearest neighbors (K-NN).

Usage

```
knn_imp(
  obj,
  k,
  colmax = 0.9,
  method = c("euclidean", "manhattan"),
  cores = 1,
  post_imp = TRUE,
  subset = NULL,
  dist_pow = 0,
  na_check = TRUE,
  .progress = FALSE
)
```

Arguments

obj	A numeric matrix with samples in rows and features in columns.
k	Integer. Number of nearest neighbors to use for K-NN imputation.
colmax	Numeric scalar between 0 and 1. Columns with a missing-data proportion greater than colmax are excluded from the main imputation method. Excluded columns are left unchanged unless post_imp = TRUE, in which case remaining missing values are replaced by column means when possible.
method	Character. K-NN imputation distance method: either "euclidean" or "manhattan".
cores	Integer. Number of cores to use for K-NN imputation. Defaults to 1.

post_imp	Logical. If TRUE, replace missing values remaining after the main imputation method with column means when possible.
subset	Optional character or integer vector specifying columns to target for imputation. If NULL, all eligible columns are targeted.
dist_pow	Numeric. Power used to penalize more distant neighbors in the weighted average. dist_pow = 0 gives an unweighted average of the nearest neighbors.
na_check	Logical. If TRUE, check whether the returned matrix still contains missing values.
.progress	Logical. If TRUE, show imputation progress.

Details

knn_imp() performs imputation column-wise, treating rows as observations and columns as features.

Nearest neighbors are found using brute-force K-NN.

When dist_pow > 0, imputed values are computed as distance-weighted averages. Weights are inverse distances raised to the power of dist_pow.

Value

A numeric matrix of the same dimensions as obj, with missing values imputed. The returned object has class slideimp_results.

K-NN performance optimization

- Use subset when only specific columns need imputation.
- Use grouped or sliding-window imputation for very large matrices.

References

Troyanskaya O, Cantor M, Sherlock G, Brown P, Hastie T, Tibshirani R, Botstein D, Altman RB (2001). Missing value estimation methods for DNA microarrays. *Bioinformatics*, 17(6), 520-525. doi:10.1093/bioinformatics/17.6.520

Examples

```
set.seed(123)
obj <- sim_mat(20, 20, perc_col_na = 1)$input
sum(is.na(obj))

# Select `k` with `tune_imp()`.
result <- knn_imp(obj, k = 10, .progress = FALSE)
result
```

lobpcg_control *LOBPCG Eigensolver Control Options*

Description

Construct a validated list of control options for the LOBPCG eigensolver used by `pca_imp()`. Most users do not need to call this directly.

Usage

```
lobpcg_control(warmup_iters = 10L, tol = 1e-09, maxiter = 20)
```

Arguments

`warmup_iters`	Integer. Number of warm-up iterations before the main LOBPCG solve. Must be non-negative.
`tol`	Numeric. Convergence tolerance for the LOBPCG eigensolver. Must be non-negative and finite.
`maxiter`	Integer. Maximum number of LOBPCG iterations. Must be non-negative. In `pca_imp()`, `maxiter` must be positive when `solver = "auto"` or `solver = "lobpcg"`. Use `solver = "exact"` to force the exact solver.

Value

A named list of class `"slideimp_lobpcg_control"` containing `warmup_iters`, `tol`, and `maxiter`.

Examples

```
set.seed(123)
obj <- sim_mat(10, 10)$input

# Use all defaults
lobpcg_control()

# Override a single option
lobpcg_control(maxiter = 50)

# Force the exact solver from pca_imp()
pca_imp(obj, ncp = 2, solver = "exact")

# Pass directly to pca_imp()
pca_imp(obj, ncp = 2, lobpcg_control = lobpcg_control(tol = 1e-9))

# Or use a named list
pca_imp(obj, ncp = 2, lobpcg_control = list(maxiter = 50))
```

mat_miss *Column or Row Missing Counts and Proportions*

Description

Calculate the number or proportion of missing values per column or per row of a numeric matrix without allocating a full logical mask matrix.

Usage

```
mat_miss(obj, col = TRUE, prop = FALSE)
```

Arguments

obj	A numeric matrix.
col	Logical. If TRUE, compute column-wise. If FALSE, compute row-wise.
prop	Logical. If FALSE, return missing-value counts. If TRUE, return missing-value proportions.

Value

A numeric vector containing missing-value counts or proportions for columns or rows, named when the corresponding dimension names are present.

Examples

```
obj <- matrix(c(1, NA, 3, 4, NA, 6, NA, 8, 9), nrow = 3)
obj

# column missing counts
mat_miss(obj)

# row missing counts
mat_miss(obj, col = FALSE)

# column missing proportions
mat_miss(obj, prop = TRUE)
```

mean_imp_col *Column Mean Imputation*

Description

Impute missing values in a matrix by replacing them with the mean of their respective columns.

Usage

```
mean_imp_col(obj, subset = NULL, cores = 1)
```

Arguments

obj	A numeric matrix.
subset	Optional character or integer vector specifying columns to impute. If NULL, all columns are imputed.
cores	Integer. Number of cores to use for parallel computation. Defaults to 1.

Details

Columns with no observed values cannot be imputed by their column mean and are left unchanged.

Value

A numeric matrix of the same dimensions as obj, with missing values in the selected columns replaced by column means.

Examples

```
obj <- matrix(c(1, 2, NA, 4, NA, 6, NA, 8, 9, NA, NA, NA), nrow = 3)
colnames(obj) <- c("A", "B", "C", "D")
obj

# impute missing values with column means
mean_imp_col(obj)

# impute only specific columns by name
mean_imp_col(obj, subset = c("A", "C"))

# impute only specific columns by index
mean_imp_col(obj, subset = c(1, 3))
```

pca_imp *PCA Imputation for Numeric Matrices*

Description

Impute missing values in a numeric matrix using regularized or expectation-maximization (EM) PCA imputation. Supports warm-start LOBPCG with both the previous eigenblock and search direction.

Usage

```
pca_imp(
  obj,
  ncp = 2,
  scale = TRUE,
  method = c("regularized", "EM"),
  coeff.ridge = 1,
  row.w = NULL,
  threshold = 1e-06,
  seed = NULL,
  nb.init = 1,
  maxiter = 1000,
  miniter = 5,
  solver = c("auto", "exact", "lobpcg"),
  lobpcg_control = NULL,
  colmax = 0.9,
  post_imp = TRUE,
  na_check = TRUE,
  clamp = NULL,
  .progress = FALSE
)
```

Arguments

obj	A numeric matrix with samples in rows and features in columns.
ncp	Integer. Number of principal components used to predict missing entries.
scale	Logical. If TRUE, columns are scaled to unit variance.
method	Character. PCA imputation method: either "regularized" or "EM".
coeff.ridge	Numeric. Ridge regularization, used only when method = "regularized". Values < 1 move toward EM PCA; values > 1 move toward mean imputation.
row.w	Row weights, normalized to sum to 1. NULL (equal weights), a positive numeric vector of length nrow(obj), or "n_miss" (down-weight rows with more missing values).
threshold	Numeric. Convergence threshold.
seed	Integer, numeric, or NULL. Random seed for reproducibility.

nb.init	Integer. Number of random initializations. The first initialization is always mean imputation.
maxiter	Integer. Maximum number of iterations.
miniter	Integer. Minimum number of iterations.
solver	Character. Eigensolver: "auto" (default), "exact", or "lobpcg". "auto" runs a short timed probe and picks "lobpcg" only when clearly faster. Consecutive EM calls warm-start LOBPCG with both the previous eigenblock and search direction. When nb.init > 1, the auto choice from the first init is reused. See Performance tips.
lobpcg_control	A list of LOBPCG eigensolver control options, usually created by lobpcg_control(). A plain named list is also accepted. Ignored when solver = "exact".
colmax	Numeric scalar between 0 and 1. Columns with a missing-data proportion greater than colmax are excluded from the main imputation method. Excluded columns are left unchanged unless post_imp = TRUE, in which case remaining missing values are replaced by column means when possible.
post_imp	Logical. If TRUE, replace missing values remaining after the main imputation method with column means when possible.
na_check	Logical. If TRUE, check whether the returned matrix still contains missing values.
clamp	Optional numeric vector c(lower, upper) bounding PCA-imputed values (use -Inf/Inf for one-sided, NULL for none). E.g., c(0, 1) for DNAm beta values. Observed values are not clamped.
.progress	Logical. If TRUE, show imputation progress.

Details

This function reimplements the PCA imputation method from the missMDA package by Francois Husson and Julie Josse, based on Josse and Husson (2016).

Value

A numeric matrix of the same dimensions as obj, with missing values imputed. The returned object has class slideimp_results.

PCA Performance tips

Speed comes from three levers: solver (through LOBPCG with warm-start), threshold, and scale. Tune these first, then accuracy parameters (ncp, coeff.ridge) on a representative subset.

Exact vs. LOBPCG with warm-start. Whether "lobpcg" beats "exact" depends on size and low-rankness: "lobpcg" is preferred for large, approximately low-rank matrices with small ncp, and "exact" for small matrices (including slide_imp() windows), where it is faster and more robust. Separately, the warm-start makes each successive solve cheap: pca_imp() warm-starts LOBPCG with the previous eigenblock and search direction, so once imputed values stabilize, later solves converge in a few iterations. The payoff therefore grows with the number of EM iterations, independent of low-rankness. solver = "auto" (default) probes both and is a safe start.

Threshold. The default 1e-6 is conservative; 1e-5 is often faster with very similar values.

Scale. For columns on a common scale (e.g., DNAm beta values in `[0, 1]`), `scale = FALSE` can be faster and more accurate.

Parallel and BLAS. In parallel via `tune_imp()` or `group_imp()` with a multithreaded BLAS, set `pin_blas = TRUE` to avoid thread oversubscription. On Windows, the stock BLAS can be slow. Advanced users can swap in OpenBLAS.

See Speeding up PCA imputation for the full workflow.

References

Josse J, Husson F (2013). Handling missing values in exploratory multivariate data analysis methods. *Journal de la SFdS*, 153(2), 79-99.

Josse J, Husson F (2016). missMDA: A Package for Handling Missing Values in Multivariate Data Analysis. *Journal of Statistical Software*, 70(1), 1-31. doi:10.18637/jss.v070.i01

Examples

```
set.seed(123)
obj <- sim_mat(10, 10)$input
sum(is.na(obj))
obj[1:4, 1:4]

# Randomly initialize missing values 5 times. The first initialization is
# mean imputation. Select `ncp` with `tune_imp()`.
pca_imp(obj, ncp = 2, nb.init = 5, seed = 123)
```

`prep_groups` *Prepare Groups for Imputation*

Description

Normalize and validate a grouping specification for use with `group_imp()`.

Usage

```
prep_groups(
  obj_cn,
  group,
  subset = NULL,
  min_group_size = 0,
  allow_unmapped = FALSE,
  seed = NULL
)
```

Arguments

obj_cn	Character vector of column names from the data matrix, usually colnames(obj).
group	Specification of how features should be grouped for imputation. Accepted formats are: • A character scalar naming a supported Illumina platform; see Note. • A long-format data.frame with columns group and feature. • A list-column data.frame with a feature list-column. Optional list-columns are aux, for auxiliary feature names, and parameters, for group-specific parameter lists.
subset	Optional character vector of feature names to impute. If NULL, all grouped features are imputed. Features in a group but not in subset are demoted to auxiliary columns for that group. Groups left with zero features after demotion are dropped with a message.
min_group_size	Integer or NULL. Minimum total number of columns per group, counting both features and auxiliary columns. Groups smaller than this are padded with randomly sampled columns from obj_cn. If NULL or 0, no padding is performed.
allow_unmapped	Logical. If FALSE, every column in colnames(obj) must appear in group. If TRUE, columns with no group assignment are left untouched and are not used as auxiliary columns.
seed	Integer, numeric, or NULL. Random seed used when padding small groups.

Details

prep_groups() converts long-format or list-column group specifications into a validated slideimp_tbl, enforces feature and auxiliary-column set relationships, prunes dropped columns, and optionally pads small groups.

Let A be obj_cn and B be the union of all feature and auxiliary names in group. When allow_unmapped = FALSE, the function enforces $A \subseteq B$: every matrix column must appear somewhere in the grouping specification.

Elements in B but not in A, such as previously dropped probes, are pruned from each group. Groups left with zero features after pruning are removed with a diagnostic message.

Value

A data frame of class slideimp_tbl containing:

- group: original group labels, if provided, or sequential group labels.
- feature: a list-column of character vectors containing feature names.
- aux: a list-column of character vectors containing auxiliary names.
- parameters: a list-column of per-group configuration lists.

See Also

group_imp()

Examples

```
sim <- sim_mat(n = 10, p = 20)
prepped <- prep_groups(colnames(sim$input), sim$col_group)
prepped
```

`print.slideimp_results` *Print a* `slideimp_results` *Object*

Description

Print the output of `knn_imp()`, `pca_imp()`, `group_imp()`, or `slide_imp()`.

Usage

```
## S3 method for class 'slideimp_results'
print(x, n = 6L, p = 6L, ...)
```

Arguments

`x`	A `slideimp_results` object.
`n`	Number of rows to print.
`p`	Number of columns to print.
`...`	Not used.

Value

`x`, invisibly.

Examples

```
set.seed(1234)
mat <- sim_mat(n = 10, p = 10)
result <- knn_imp(mat$input, k = 5, .progress = FALSE)
class(result)
print(result, n = 6, p = 6)
```

print.slideimp_sim *Print a* slideimp_sim *Object*

Description

Print the output of sim_mat().

Usage

```
## S3 method for class 'slideimp_sim'
print(x, n = 6L, p = 6L, ...)
```

Arguments

x	A slideimp_sim object.
n	Number of rows of each component to show.
p	Number of columns of input to show.
...	Not used.

Value

x, invisibly.

Examples

```
set.seed(123)
sim_data <- sim_mat(n = 50, p = 10, rho = 0.5)
class(sim_data)
print(sim_data)
```

print.slideimp_tbl *Print a* slideimp_tbl *Object*

Description

Print slideimp_tbl objects, which inherit from data.frame, with compact display of list-columns.

Usage

```
## S3 method for class 'slideimp_tbl'
print(x, n = NULL, ...)
```

Arguments

x	A slideimp_tbl object.
n	Number of rows to show. If NULL, a default is used.
...	Not used.

Value

x, invisibly.

Examples

```
sim <- sim_mat(n = 10, p = 20)
tbl <- prep_groups(colnames(sim$input), sim$col_group)
class(tbl)
print(tbl)
```

---

sample_na_loc *Sample Missing-Value Locations with Constraints*

Description

Sample matrix indices for NA injection while respecting row and column missingness limits and avoiding zero-variance columns.

Usage

```
sample_na_loc(
  obj,
  n_cols = NULL,
  n_rows = 2L,
  num_na = NULL,
  n_reps = 1L,
  rowmax = 0.9,
  colmax = 0.9,
  na_col_subset = NULL,
  max_attempts = 100
)
```

Arguments

obj	A numeric matrix.
n_cols	Integer or NULL. Number of columns to receive injected missing values. Must be supplied when num_na = NULL.
n_rows	Integer. Target number of missing values to inject per selected column.

num_na	Integer or NULL. Total number of missing values to inject per repetition. If supplied, n_cols is derived from num_na and n_rows, and missing values are distributed as evenly as possible across columns.
n_reps	Integer. Number of independent repetitions.
rowmax	Numeric scalar between 0 and 1. Maximum allowed missing-data proportion per row after injection.
colmax	Numeric scalar between 0 and 1. Maximum allowed missing-data proportion per column after injection.
na_col_subset	Optional integer or character vector restricting which columns are eligible for missing-value injection.
max_attempts	Integer. Maximum number of resampling attempts per repetition before giving up.

Details

The function uses a greedy stochastic search for valid NA locations. It ensures that:

- Total missingness per row and column does not exceed rowmax and colmax.
- At least two distinct observed values are preserved in every affected column.

Value

A list of length n_reps. Each element is a two-column integer matrix with row and column indices for sampled NA locations.

Examples

```
set.seed(123)
mat <- matrix(runif(100), nrow = 10)

# Sample 5 missing values across 5 columns
locs <- sample_na_loc(mat, n_cols = 5, n_rows = 1)
locs

# Inject the missing values from the first repetition
mat[locs[[1]]] <- NA
mat
```

---

sim_mat *Simulate a Matrix with Metadata*

Description

sim_mat() generates random normal data with optional compound-symmetric column correlation. Values can optionally be scaled to the interval [0, 1] column-wise. The function also creates feature metadata for columns and sample metadata for rows, and can inject NA values into a specified proportion of matrix cells across a specified proportion of columns.

Usage

```
sim_mat(
  n = 100,
  p = 100,
  rho = 0.5,
  n_col_groups = 2,
  n_row_groups = 1,
  perc_total_na = 0.1,
  perc_col_na = 0.5,
  beta = TRUE
)
```

Arguments

n	Integer. Number of rows, interpreted as samples. Defaults to 100.
p	Integer. Number of columns, interpreted as features. Defaults to 100.
rho	Numeric. Compound-symmetric column correlation before optional scaling. Defaults to 0.5.
n_col_groups	Integer. Number of groups to assign to features. Defaults to 2.
n_row_groups	Integer. Number of groups to assign to samples. Defaults to 1.
perc_total_na	Numeric scalar between 0 and 1. Proportion of all matrix cells to set to NA. Defaults to 0.1.
perc_col_na	Numeric scalar between 0 and 1. Proportion of columns across which injected NA values are spread. Defaults to 0.5.
beta	Logical. If TRUE, scale values to the interval [0, 1] column-wise.

Details

Generate a numeric matrix with optional row and column metadata and added missing values.

Value

An object of class slideimp_sim. This is a list containing:

- input: a numeric matrix of dimension $n \times p$ containing the simulated values and injected missing values.
- col_group: a data frame with p rows mapping each feature to a group.
- row_group: a data frame with n rows mapping each sample to a group.

Examples

```
set.seed(123)
sim_data <- sim_mat(n = 50, p = 10, rho = 0.5)
sim_data
```

slide_imp *Sliding-Window K-NN or PCA Imputation*

Description

Perform sliding-window K-NN or PCA imputation on a numeric matrix whose columns are meaningfully ordered. Not intended for Illumina DNA methylation microarrays, use `group_imp()` instead.

Usage

```
slide_imp(
  obj,
  location,
  window_size,
  overlap_size = 0,
  flank = FALSE,
  min_window_n,
  subset = NULL,
  dry_run = FALSE,
  k = NULL,
  cores = 1,
  dist_pow = 0,
  ncp = NULL,
  scale = TRUE,
  coeff.ridge = 1,
  threshold = 1e-06,
  seed = NULL,
  row.w = NULL,
  nb.init = 1,
  maxiter = 1000,
  miniter = 5,
  solver = c("auto", "exact", "lobpcg"),
  lobpcg_control = NULL,
  clamp = NULL,
  method = NULL,
  .progress = TRUE,
  colmax = 0.9,
  post_imp = TRUE,
  na_check = TRUE,
  on_infeasible = c("skip", "error", "mean")
)
```

Arguments

obj	A numeric matrix with samples in rows and features in columns.

location	A sorted numeric vector of length ncol(obj) giving the position of each column, such as genomic coordinates.
window_size	Numeric. Window width in the same units as location.
overlap_size	Numeric. Overlap between consecutive windows in the same units as location. Must be less than window_size. Ignored when flank = TRUE.
flank	Logical. If FALSE, imputation uses sliding windows across the full matrix. If TRUE, one window of width window_size is created for each feature listed in subset; overlap_size is ignored. subset must be supplied when flank = TRUE.
min_window_n	Integer. Minimum number of columns a window must contain to be considered for imputation. For non-dry runs, the selected k or ncp value must be feasible for the usable columns in each retained window after applying colmax.
subset	Optional character or integer vector specifying columns to impute. If NULL, all eligible columns are imputed. Required when flank = TRUE.
dry_run	Logical. If TRUE, skip imputation and return a slideimp_tbl describing the windows that would be used after all filtering rules are applied. In this mode, k and ncp are not required.
k	Integer. Number of nearest neighbors to use for K-NN imputation.
cores	Integer. Number of cores to use for K-NN imputation.
dist_pow	Numeric. Power used to penalize more distant neighbors in the weighted average. dist_pow = 0 gives an unweighted average of the nearest neighbors.
ncp	Integer. Number of principal components used to predict missing entries.
scale	Logical. If TRUE, columns are scaled to unit variance.
coeff.ridge	Numeric. Ridge regularization, used only when method = "regularized". Values < 1 move toward EM PCA; values > 1 move toward mean imputation.
threshold	Numeric. Convergence threshold.
seed	Integer, numeric, or NULL. Random seed for reproducibility.
row.w	Row weights, normalized to sum to 1. NULL (equal weights), a positive numeric vector of length nrow(obj), or "n_miss" (down-weight rows with more missing values).
nb.init	Integer. Number of random initializations. The first initialization is always mean imputation.
maxiter	Integer. Maximum number of iterations.
miniter	Integer. Minimum number of iterations.
solver	Character. Eigensolver: "auto" (default), "exact", or "lobpcg". "auto" runs a short timed probe and picks "lobpcg" only when clearly faster. Consecutive EM calls warm-start LOBPCG with both the previous eigenblock and search direction. When nb.init > 1, the auto choice from the first init is reused. See Performance tips.
lobpcg_control	A list of LOBPCG eigensolver control options, usually created by lobpcg_control(). A plain named list is also accepted. Ignored when solver = "exact".

clamp	Optional numeric vector c(lower, upper) bounding PCA-imputed values (use -Inf/Inf for one-sided, NULL for none). E.g., c(0, 1) for DNAm beta values. Observed values are not clamped.
method	Character or NULL. For K-NN imputation, one of "euclidean" or "manhattan". For PCA imputation, one of "regularized" or "EM". If NULL, the corresponding backend default is used.
.progress	Logical. If TRUE, show imputation progress.
colmax	Numeric scalar between 0 and 1. Columns with a missing-data proportion greater than colmax are excluded from the main imputation method. Excluded columns are left unchanged unless post_imp = TRUE, in which case remaining missing values are replaced by column means when possible.
post_imp	Logical. If TRUE, replace missing values remaining after the main imputation method with column means when possible.
na_check	Logical. If TRUE, check whether the returned matrix still contains missing values.
on_infeasible	Character. One of "skip", "error", or "mean". Controls behavior when a window is infeasible for imputation, for example when k or ncp exceeds the number of usable columns after applying colmax.

Details

The sliding-window approach divides the input matrix into smaller segments based on location values and applies imputation to each window independently. Values in overlapping regions are averaged across windows to produce the final imputed result.

Two windowing modes are supported:

- flank = FALSE: greedily partition location into windows of width window_size with the requested overlap_size between consecutive windows.
- flank = TRUE: create one window per feature in subset, centered on that feature using the supplied window_size.

Specify k and related arguments to use knn_imp(), or ncp and related arguments to use pca_imp().

Value

If dry_run = FALSE, a numeric matrix of the same dimensions as obj, with missing values imputed. The returned object has class slideimp_results.

If dry_run = TRUE, a data frame of class slideimp_tbl with columns start, end, and window_n, plus subset_local and, when flank = TRUE, target.

PCA Performance tips

Speed comes from three levers: solver (through LOBPCG with warm-start), threshold, and scale. Tune these first, then accuracy parameters (ncp, coeff.ridge) on a representative subset.

Exact vs. LOBPCG with warm-start. Whether "lobpcg" beats "exact" depends on size and low-rankness: "lobpcg" is preferred for large, approximately low-rank matrices with small ncp, and "exact" for small matrices (including slide_imp() windows), where it is faster and more

robust. Separately, the warm-start makes each successive solve cheap: `pca_imp()` warm-starts LOBPCG with the previous eigenblock and search direction, so once imputed values stabilize, later solves converge in a few iterations. The payoff therefore grows with the number of EM iterations, independent of low-rankness. `solver = "auto"` (default) probes both and is a safe start.

Threshold. The default `1e-6` is conservative; `1e-5` is often faster with very similar values.

Scale. For columns on a common scale (e.g., DNAm beta values in `[0, 1]`), `scale = FALSE` can be faster and more accurate.

Parallel and BLAS. In parallel via `tune_imp()` or `group_imp()` with a multithreaded BLAS, set `pin_blas = TRUE` to avoid thread oversubscription. On Windows, the stock BLAS can be slow. Advanced users can swap in OpenBLAS.

See Speeding up PCA imputation for the full workflow.

Examples

```
set.seed(1234)

# Example data with 20 samples and 100 ordered columns
beta_matrix <- sim_mat(20, 100)$input
location <- 1:100

# First perform a dry run to inspect the calculated windows
window_statistics <- slide_imp(
  beta_matrix,
  location = location,
  window_size = 50,
  overlap_size = 10,
  min_window_n = 10,
  dry_run = TRUE,
  .progress = FALSE
)
window_statistics

# Sliding-window K-NN imputation
imputed_knn <- slide_imp(
  beta_matrix,
  location = location,
  k = 5,
  window_size = 50,
  overlap_size = 10,
  min_window_n = 10,
  .progress = FALSE
)
imputed_knn

# Sliding-window PCA imputation
imputed_pca <- slide_imp(
  beta_matrix,
  location = location,
  ncp = 2,
  window_size = 50,
  overlap_size = 10,
```

```
  min_window_n = 10,
  .progress = FALSE
)
imputed_pca

# K-NN imputation with flanking windows
imputed_flank <- slide_imp(
  beta_matrix,
  location = location,
  k = 2,
  window_size = 30,
  flank = TRUE,
  subset = c(10, 30, 70),
  min_window_n = 5,
  .progress = FALSE
)
imputed_flank
```

`slideimp_resolve_group` *Resolve a Group Specification to a Data Frame*

Description

Convert a group specification to the canonical data-frame form expected by `prep_groups()`. This S3 generic is exported so that extension packages, such as `slideimp.extra`, can register additional methods.

Usage

```
slideimp_resolve_group(x)

## S3 method for class 'data.frame'
slideimp_resolve_group(x)

## Default S3 method:
slideimp_resolve_group(x)
```

Arguments

`x`	A group specification. `slideimp` provides a method for `data.frame` objects. The optional `slideimp.extra` package provides a method for character platform names.

Value

A data frame group specification, or an error for unsupported inputs.

Note

This is primarily an extension hook for `slideimp.extra`.

Examples

```
df <- data.frame(feature = c("cg1", "cg2"), group = c(1, 1))
slideimp_resolve_group(df)
```

`tune_imp` *Tune Imputation Method Parameters*

Description

Tune method specific hyperparameters by repeatedly masking observed values, imputing them, and comparing the imputed values with the original values.

Usage

```
tune_imp(
  obj,
  parameters = NULL,
  .f,
  na_loc = NULL,
  num_na = NULL,
  n_reps = 1,
  n_cols = NULL,
  n_rows = 2,
  rowmax = 0.9,
  colmax = 0.9,
  na_col_subset = NULL,
  max_attempts = 100,
  .progress = TRUE,
  cores = 1,
  location = NULL,
  pin_blas = FALSE
)
```

Arguments

`obj`	A numeric matrix.
`parameters`	A `data.frame` specifying parameter combinations to tune. Each column should be a parameter accepted by `.f`, excluding `obj`. List-columns are supported for complex parameters. Duplicate rows are removed. `NULL` is treated as a single parameter set with no additional arguments, which is useful for functions whose required arguments all have defaults.

.f	One of "knn_imp", "pca_imp", or "slide_imp", or a custom imputation function.
na_loc	Optional predefined missing-value locations. Accepted formats are a two-column integer matrix of row and column indices, a numeric vector of linear positions, or a list whose elements are either of those formats.
num_na	Integer or NULL. Total number of missing values to inject per repetition. If supplied, n_cols is derived from num_na and n_rows, and missing values are distributed as evenly as possible across columns.
n_reps	Integer. Number of independent repetitions.
n_cols	Integer or NULL. Must be supplied when both num_na and na_loc are NULL, unless the automatic default applies.
n_rows	Integer. Target number of missing values to inject per selected column.
rowmax	Numeric scalar between 0 and 1. Maximum allowed missing-data proportion per row after injection.
colmax	Numeric scalar between 0 and 1. Maximum allowed missing-data proportion per column after injection.
na_col_subset	Optional integer or character vector restricting which columns are eligible for missing-value injection.
max_attempts	Integer. Maximum number of resampling attempts per repetition before giving up.
.progress	Logical. If TRUE, show progress during tuning.
cores	Integer. Number of cores to use for K-NN and sliding-window K-NN imputation. For other methods, use mirai::daemons().
location	Numeric vector of column locations. Required when .f = "slide_imp".
pin_blas	Logical. If TRUE, pin BLAS threads to 1 during parallel tuning to reduce thread contention.

Details

Built-in methods can be selected by passing .f = "knn_imp", .f = "pca_imp", or .f = "slide_imp". A custom function can also be supplied. Custom functions must accept obj as their first argument and return a numeric matrix with the same dimensions as obj.

When na_loc is supplied, num_na, n_cols, n_rows, and na_col_subset are ignored.

When .f is a character string, columns in parameters are validated against the selected method:

- "knn_imp" requires k.
- "pca_imp" requires ncp.
- "slide_imp" requires window_size, overlap_size, and min_window_n, plus exactly one of k or ncp.

To tune parameters for grouped imputation, tune knn_imp() or pca_imp() on representative groups, then pass the selected parameters to group_imp().

The top-level rowmax and colmax arguments control random missing-value injection performed by sample_na_loc(). To tune or pass an imputation method's own colmax argument, include a colmax column in parameters.

Tuning results can be summarized with compute_metrics() or evaluated with external packages such as yardstick.

Value

A data frame of class slideimp_tune containing:

- columns originally provided in parameters;
- param_set, an integer ID for each unique parameter combination;
- rep_id, an integer repetition index;
- result, a list-column where each element is a data frame containing truth and estimate columns;
- error, a character column containing the error message if the iteration failed, otherwise NA.

Parallelization

- K-NN: use the cores argument. If mirai daemons are active, cores is automatically set to 1 to avoid nested parallelism.
- PCA: use mirai::daemons() instead of cores.

When running PCA imputation in parallel with mirai, set pin_blas = TRUE in tune_imp() or group_imp() to prevent BLAS threads from oversubscribing CPU cores. This relies on RhpcBLASctl and works with OpenBLAS and MKL (typical on Linux, and on Windows after an OpenBLAS swap). pin_blas = TRUE may have no effect on macOS.

PCA Performance tips

Speed comes from three levers: solver (through LOBPCG with warm-start), threshold, and scale. Tune these first, then accuracy parameters (ncp, coeff.ridge) on a representative subset.

Exact vs. LOBPCG with warm-start. Whether "lobpcg" beats "exact" depends on size and low-rankness: "lobpcg" is preferred for large, approximately low-rank matrices with small ncp, and "exact" for small matrices (including slide_imp() windows), where it is faster and more robust. Separately, the warm-start makes each successive solve cheap: pca_imp() warm-starts LOBPCG with the previous eigenblock and search direction, so once imputed values stabilize, later solves converge in a few iterations. The payoff therefore grows with the number of EM iterations, independent of low-rankness. solver = "auto" (default) probes both and is a safe start.

Threshold. The default 1e-6 is conservative; 1e-5 is often faster with very similar values.

Scale. For columns on a common scale (e.g., DNAm beta values in [0, 1]), scale = FALSE can be faster and more accurate.

Parallel and BLAS. In parallel via tune_imp() or group_imp() with a multithreaded BLAS, set pin_blas = TRUE to avoid thread oversubscription. On Windows, the stock BLAS can be slow. Advanced users can swap in OpenBLAS.

See Speeding up PCA imputation for the full workflow.

Examples

```
set.seed(123)

# Simulate some data
obj <- sim_mat(10, 50)$input

# Tune K-NN imputation with random missing-value injection.
# Use larger `num_na` and `n_reps` values for real analyses.
params_knn <- data.frame(k = c(2, 4))
results <- tune_imp(
  obj,
  params_knn,
  .f = "knn_imp",
  n_reps = 1,
  num_na = 10,
  .progress = FALSE
)
compute_metrics(results)

# Tune with fixed missing-value positions
na_positions <- list(
  matrix(c(1, 2, 3, 1, 1, 1), ncol = 2),
  matrix(c(2, 3, 4, 2, 2, 2), ncol = 2)
)

results_fixed <- tune_imp(
  obj,
  data.frame(k = 2),
  .f = "knn_imp",
  na_loc = na_positions,
  .progress = FALSE
)

# Custom imputation function
custom_fill <- function(obj, val = 0) {
  obj[is.na(obj)] <- val
  obj
}

tune_imp(
  obj,
  data.frame(val = c(0, 1)),
  .f = custom_fill,
  num_na = 10,
  .progress = FALSE
)


# Parallel tuning with mirai
mirai::daemons(2)

parameters_custom <- data.frame(mean = c(0, 1), sd = c(1, 1))
```

```
custom_imp <- function(obj, mean, sd) {
  na_pos <- is.na(obj)
  obj[na_pos] <- stats::rnorm(sum(na_pos), mean = mean, sd = sd)
  obj
}

results_p <- tune_imp(
  obj,
  parameters_custom,
  .f = custom_imp,
  n_reps = 1,
  num_na = 10,
  .progress = FALSE
)

mirai::daemons(0)
```

Index

col_vars, 3
compute_metrics, 4
compute_metrics(), *30*

group_imp, 5
group_imp(), *7*, *16–18*, *23*, *29, 30*

knn_imp, 9
knn_imp(), *7, 8*, *18*, *25*, *29*

lobpcg_control, 11
lobpcg_control(), *7*, *15*, *24*

mat_miss, 12
mean_imp_col, 13

pca_imp, 14
pca_imp(), *7, 8*, *11*, *18*, *25*, *29*
prep_groups, 16
prep_groups(), *8*, *27*
print.slideimp_results, 18
print.slideimp_sim, 19
print.slideimp_tbl, 19

sample_na_loc, 20
sample_na_loc(), *29*
sim_mat, 21
sim_mat(), *19*
slide_imp, 23
slide_imp(), *18*
slideimp_resolve_group, 27

tune_imp, 28
tune_imp(), *4*, *7*, *30*