Package: IMR (via r-universe)

October 25, 2026

Type Package

Title Incomplete Matrix Regression

Version 1.0.0

Description A framework for matrix completion and regression on response matrices with missing values. The model estimates missing entries using any combination of intercepts, row and column covariates, and a low-rank matrix approximation. It applies Lasso penalties on the covariates and a nuclear norm penalty on the low-rank component. It also adjusts for correlation within the rows and columns of the target matrix using similarity matrices. The framework is described in Fouda, Labbe and Oualkacha (2026) <doi:10.48550/arXiv.2606.26325>.

License GPL (>= 3)

URL https://github.com/khaledfouda/IMR

BugReports https://github.com/khaledfouda/IMR/issues

Depends R (>= 3.5)

Imports fields, irlba, MASS, Matrix, methods, parallel, Rcpp, RSpectra, stats, utils

Suggests knitr, rmarkdown, spelling, testthat (>= 3.0.0)

LinkingTo Rcpp, RcppArmadillo

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData true

SystemRequirements C++17

NeedsCompilation yes

Author Khaled Fouda [aut, cre] (ORCID:
<https://orcid.org/0009-0000-3688-8767>), Aurélie Labbe [ths, ctb], Karim Oualkacha [ths, ctb], Korbinian Strimmer [cph, ctb] (Authored original fast.svd R implementation in the corpcor package)

Maintainer Khaled Fouda <khaled.fouda@hec.ca>

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

Date/Publication 2026-07-10 18:20:02 UTC

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

RemoteRef HEAD

RemoteSha e7b8dd494790eb9fcda68d656bd41cda0ab5d80f

Contents

as_incomplete . . . 3
Bixi_sample . . . 3
coef.imr_fit . . . 4
evaluate . . . 5
get_metric . . . 7
imr_convergence . . . 7
imr_data . . . 8
imr_fit . . . 10
imr_set_grid_limits . . . 12
imr_similarity . . . 14
imr_tune . . . 16
imr_tune_grid . . . 19
is_incomplete . . . 20
mc_with_means . . . 21
print.imr_convergence . . . 22
print.imr_data . . . 22
print.imr_fit . . . 23
print.imr_similarity . . . 25
print.imr_tune_grid . . . 26
reconstruct . . . 27
reconstruct_partial . . . 28
summary.imr_fit . . . 30
svd_opt . . . 31
update.imr_data . . . 33

Index **35**

as_incomplete *Clean and convert any matrix to a sparse object*

Description

Clean and convert any matrix to a sparse object

Usage

```
as_incomplete(x)
```

Arguments

x	numeric, A matrix with class `matrix` or `Matrix`

Value

A standard `dgCMatrix` with both NAs and 0s are treated as missing values.

Examples

```
# create sample data
Y <- matrix(
  c(2, NA, 0,
  4, .5, NA,
  NA, NA, 0), 3, byrow= TRUE
)
# make it sparse with both NAs and 0s dropped
Y <- as_incomplete(Y)
# verify
print(is_incomplete(Y))
```

Bixi_sample *Bixi Sample Data*

Description

A subset of data from Bixi (https://Bixi.com/fr/), a docked bike-sharing service in Montreal, Canada. We use the data compiled by Lei et al., 2025, which contains normalized daily departure counts for each of 579 stations over 196 days (April 15 to October 27, 2019). The data is accompanied by side information. We select two temporal variables (temperature and precipitation) and one spatial variable (park area). We also take a sample of 150 stations and the first 100 consecutive days. The data also contains two distance matrices: a spatial distance matrix derived from the geographic coordinates of the stations (columns) and a temporal distance matrix representing the elapsed days (rows)

Usage

```
Bixi_sample
```

Format

`Bixi_sample`**:**

A list of six matrices:

Y a 100 x 150 matrix of normalized departure counts. Missing values are set to zero. Each row is a day and each column is a station.

test a 100 x 150 test matrix where test indices are nonzero. All nonzero in `test` are set to zero in `Y` and vice-versa.

X a 100 x 2 matrix of two row covariates: temperature and precipitation. Both are normalized.

Z a 150 x 1 matrix of one column covariate: park area.

spatial_distance a 150 x 150 distance matrix between stations

temporal_distance a 100 x 100 distance matrix for days

References

Lei, M., Labbe, A., & Sun, L. (2025). Scalable Spatiotemporally Varying Coefficient Modeling with Bayesian Kernelized Tensor Regression. *Bayesian Analysis*, 20(3). doi:10.1214/24BA1428.

`coef.imr_fit` *Extract the fitted model's coefficients*

Description

Extract the fitted model's coefficients

Usage

```
## S3 method for class 'imr_fit'
coef(object, ...)
```

Arguments

`object`	An `imr_fit` object
`...`	Additional arguments to comply with generic function

Value

A named list of the estimated model coefficients (any of `u`, `d`, `v`, `beta`, `gamma`, `beta0`, `gamma0`), as stored in the `imr_fit` object.

See Also

`imr_fit()`

Examples

```
# create sample data
Y <- matrix(
  c(2, NA, 3,
  4, .5, NA,
  NA, NA, 5), 3, byrow= TRUE
)

# create covariate matrix of 2 variables
X <- matrix(rnorm(3*2), 3, 2)

# create the data object
data <- imr_data(Y =  Y, X = X, val_prop = 0.0)

# update the model to add row intercepts
data <- update(data, row_intercept = TRUE)

# fit the model
fit <- imr_fit(data, rank = 2)

# print the model fit summary
print(fit)

# print a summary of the fitted coefficients
summary(fit)

# extract the coefficients
coefs <- coef(fit)

# estimate the target matrix
target <- reconstruct(fit, data)$estimates

# compute estimates of the training data
estimates <- reconstruct_partial(fit, data, data$Y@i, data$Y@p, return_matrix = FALSE)

# compute the training Root Mean Squared Error
evaluate(estimates, data$Y@x, metric = "RMSE")
```

---

evaluate *Evaluate Model Predictions*

Description

Computes common error metrics between two vectors or matrices.

Usage

```
evaluate(predicted, actual, metric = "all", na.rm = TRUE)
```

Arguments

predicted, actual	Numeric vectors or matrices of the same dimension
metric	A character string specifying the single metric to compute (one of "rmse", "rrmse", "mae", "mape", "spearman"). If set to "all" (the default), the function computes and returns all available metrics.
na.rm	Logical. Should missing values be removed before computation? Defaults to TRUE. Note that removal is performed pairwise: if an NA is present at a specific index in either predicted or actual, that entire pair is excluded from the calculation.

Details

The function calculates the following metrics:

- **RMSE**: Root Mean Squared Error.
- **RRMSE**: Relative Root Mean Squared Error.
- **MAE**: Mean Absolute Error. The average of the absolute differences between predictions and actual observations.
- **MAPE**: Mean Absolute Percentage Error. Measures prediction accuracy as a percentage.
- **Spearman**: Spearman's rank correlation coefficient.

Value

If metric = "all", returns a one-row data-frame containing columns for each calculated metric. If a specific metric is requested, returns a single numeric value.

See Also

get_metric()

Examples

```
actual_vals <- c(10, 15, 20, NA, 30)
pred_vals <- c(11, 14, 22, 25, 28)

# Get one row of all metrics (NAs removed pairwise automatically)
evaluate(pred_vals, actual_vals)

# Get only the Root Mean Squared Error
evaluate(pred_vals, actual_vals, metric = "rmse")
```

get_metric *Returns a predefined error metric*

Description

get_metric() returns one of the common error metric functions that takes two arguments: predictions and true values.

Usage

```
get_metric(metric)
```

Arguments

metric	one of ("rmse", "rrmse", "mae", "mape", "spearman")

Value

A function that takes two arguments (vectors or matrices)

See Also

evaluate()

Examples

```
rmse_function <- get_metric("rmse")
rmse_function(predicted = c(1,3,NA), actual = c(1,2,3), na.rm = TRUE)
```

imr_convergence *Control Parameters for IMR Convergence*

Description

Defines and creates a structure for the convergence parameters used by the IMR fit function.

Usage

```
imr_convergence(maxit = 600, thresh = 1e-05, trace = FALSE, ls_initial = TRUE)
```

Arguments

maxit	Integer. The maximum number of iterations allowed for the fit function. Defaults to 600.
thresh	Numeric. The convergence threshold. Training stops early if the Frobenius difference between the new and old parameters falls below this value. Defaults to 1e-5.
trace	Logical. If TRUE, prints the Frobenius ratio and the loss function value at each iteration. Defaults to FALSE.
ls_initial	Logical. If TRUE (the default), least-squares initial values are used. If FALSE, random initialization is used. It is recommended to leave this as TRUE to speed up convergence.

Value

An object of class "imr_convergence", which is a list containing the specified control parameters.

See Also

imr_fit(), print.imr_convergence()

Examples

```
# Use the default convergence parameters
convergence <- imr_convergence()
print(convergence)
```

---

imr_data *Prepare Data and Model Structure for IMR*

Description

imr_data creates a structured data object for Incomplete Matrix Regression (IMR). It handles target matrix formatting, optional train/validation splitting, QR decomposition of covariates, and initializes a default model structure based on the provided inputs.

Usage

```
imr_data(
  Y,
  X = NULL,
  Z = NULL,
  similarity_rows = NULL,
  similarity_cols = NULL,
  val_prop = 0,
  seed = NULL
)
```

Arguments

Y	A numeric matrix (dense or sparse) representing the target/response matrix. This is the only required parameter. Missing values and Zeros are treated as unobserved.
X	A numeric matrix of row covariates. One of the dimensions must match the number of rows of Y. The other dimension must be strictly smaller. Defaults to NULL.
Z	A numeric matrix of column covariates. One of the dimensions must match the number of columns of Y The other dimension must be strictly smaller. Defaults to NULL.
similarity_rows	An "imr_similarity" object (see imr_similarity) describing the row similarity/information matrix. Defaults to NULL.
similarity_cols	An "imr_similarity" object (see imr_similarity) describing the column similarity/information matrix. Defaults to NULL.
val_prop	Numeric. The proportion of observed entries in Y to hold out for validation. If 0.0 (the default), no validation split is performed. Validation is required for hyperparameter tuning.
seed	Integer. An optional random seed for reproducibility when creating the train/validation split. Defaults to NULL.

Details

By default, imr_data assumes that if you provide X, Z, or similarity matrices, you want to include them in your IMR model. It sets $model logical flags to TRUE for any provided data. If you wish to change this model structure later without re-processing the data matrices, use the update() method.

Value

An object of class "imr_data", containing:

- Y, y_train, y_valid: The target matrix and, when val_prop > 0, its training/validation splits (as Incomplete objects).
- Xq, Xr, Zq, Zr: The QR decompositions of the covariates.
- similarity_rows, similarity_cols: The provided similarity matrices.
- meta: A list of dataset dimensions, sparsity, and variable counts.
- model: A list of logical flags defining the current model structure.

See Also

update.imr_data(), imr_similarity()

Examples

```
# create sample data
Y <- matrix(
  c(2, NA, 3,
  4, .5, NA,
  NA, NA, 5), 3, byrow= TRUE
)

# create covariate matrix of 2 variables
X <- matrix(rnorm(3*2), 3, 2)

# create the data object
data <- imr_data(Y =  Y, X = X, val_prop = 0.0)

# update the model to add row intercepts
data <- update(data, row_intercept = TRUE)

# print the metadata
print(data)
```

---

imr_fit *Fit an IMR Model*

Description

The main function for fitting an Incomplete Matrix Regression (IMR) model. It takes a prepared `imr_data` object and estimates the parameters based on the specified model structure and regularization parameters.

Usage

```
imr_fit(
  data,
  rank = 2,
  lambda_m = 0,
  lambda_beta = 0,
  lambda_gamma = 0,
  convergence = imr_convergence(),
  warm_start = NULL,
  training = FALSE
)
```

Arguments

`data`	An object of class `"imr_data"`. This object contains the response matrix, optional covariates and similarities, and the specific model structure to be fitted.

`rank`	Integer. The rank of the low-rank component ($M = UDV^T$). This is required if the model structure includes a low-rank component. Defaults to `2`.
`lambda_m`	Numeric. The regularization parameter for the low-rank component. Required if the low-rank component is included in the model. Defaults to `0`.
`lambda_beta`	Numeric. The regularization parameter for the row covariates. Only used if row covariates are specified in the `imr_data` model structure. Defaults to `0`.
`lambda_gamma`	Numeric. The regularization parameter for the column covariates. Only used if column covariates are specified in the `imr_data` model structure. Defaults to `0`.
`convergence`	An object of class `"imr_convergence"`, generated by `imr_convergence()`. It dictates the maximum iterations, threshold, tracing, and initialization method.
`warm_start`	An optional object of class `"imr_fit"`. If provided, the coefficients from this previous fit are used as the starting values for the new optimization, overriding the initialization strategy in `convergence`. Defaults to `NULL`.
`training`	Logical. If `TRUE`, the model fits on `data$y_train` rather than the full `data$Y` matrix. This is primarily intended for internal use during cross-validation. Generally, users should leave this as `FALSE`. Defaults to `FALSE`.

Value

An object of class `"imr_fit"`. This is a list containing:

- `coefficients`: A list of the estimated model parameters, which may include the low-rank matrices (`u`, `d`, `v`), covariate effects (`beta`, `gamma`), and intercepts (`beta0`, `gamma0`).
- `residuals`: A sparse matrix of the fitted residuals.
- `Xr`, `Zr`: The pxp and qxq R matrices generated from the QR-decomposition of the row and column covariates. They are copied from `data` and used to transform the estimated coefficients to the original scale. (used for the `summary` method)
- `meta`: A list of fitting metadata, including the iteration count, convergence status, and sum of squares components (used for the `print` method).
- `convergence`: The convergence control parameters used.
- `model`: The logical model structure inherited from `data`.
- `meta_data`: Variable metadata inherited from `data`.

References

Fouda, K., Labbe, A., & Oualkacha, K. (2026). *Incomplete Matrix Regression*. https://arxiv.org/abs/2606.26325

See Also

`imr_data()`, `imr_convergence()`, `print.imr_fit()`, `summary.imr_fit()`, `coef.imr_fit()`

Examples

```
# create sample data
Y <- matrix(
  c(2, NA, 3,
  4, .5, NA,
  NA, NA, 5), 3, byrow= TRUE
)

# create covariate matrix of 2 variables
X <- matrix(rnorm(3*2), 3, 2)

# create the data object
data <- imr_data(Y =  Y, X = X, val_prop = 0.0)

# update the model to add row intercepts
data <- update(data, row_intercept = TRUE)

# fit the model
fit <- imr_fit(data, rank = 2)

# print the model fit summary
print(fit)

# print a summary of the fitted coefficients
summary(fit)

# extract the coefficients
coefs <- coef(fit)

# estimate the target matrix
target <- reconstruct(fit, data)$estimates

# compute estimates of the training data
estimates <- reconstruct_partial(fit, data, data$Y@i, data$Y@p, return_matrix = FALSE)

# compute the training Root Mean Squared Error
evaluate(estimates, data$Y@x, metric = "RMSE")
```

---

imr_set_grid_limits *Automatically Determine Hyperparameter Grid Maximum Values*

Description

Computes the optimal upper bounds for the hyperparameter tuning grid when specifications are set to "auto". The function identifies the minimal regularization parameter required to shrink all corresponding coefficients to zero.

Usage

```
imr_set_grid_limits(
  data,
  grid,
  default_rank = 2,
  default_lambda_m = 0,
  default_lambda_beta = 0,
  default_lambda_gamma = 0,
  convergence = imr_convergence(trace = FALSE, ls_initial = FALSE),
  bisection_iter = 5,
  verbose = 0
)
```

Arguments

`data`	An object of class `"imr_data"` containing the response matrices and covariate structures.
`grid`	An object of class `"imr_tune_grid"`, initialized via `imr_tune_grid()`.
`default_rank`, `default_lambda_m`	Integer,Numeric. The fixed rank and low-rank component penalty used as reference when estimating the maximum λ_β or λ_Γ. Default to `2` and `0`.
`default_lambda_beta`, `default_lambda_gamma`	Numeric. The row and column covariate penalties used as a reference when estimating the maximum λ_M. Defaults to `0`.
`convergence`	An `"imr_convergence"` object specifying the numerical tolerances and optimization parameters for internal model fits. Defaults to `imr_convergence(trace = FALSE, ls_initial = FALSE)`.
`bisection_iter`	Integer. The number of iterations for the bisection algorithm employed to refine the estimated upper bound. Defaults to `5`.
`verbose`	Integer. Level of diagnostic output. `0` (default) is silent, `1` reports final parameters, and `2` provides per-iteration updates.

Details

The procedure isolates each hyperparameter to determine its saturation point through a two-stage estimation process:

1. An initial theoretical upper bound is derived based on the Karush-Kuhn-Tucker (KKT) conditions.
2. The function executes a bisection search over `bisection_iter` iterations using KKT estimates as initial values. This identifies the minimum of the penalty values that result in a zero-solution for the targeted parameters.

During the estimation of the maximum λ_β or λ_Γ, the other covariate penalty is treated as infinite, while the low-rank component is set to `default_rank` and `default_lambda_m`. Conversely, estimation of the maximum λ_M (nuclear norm penalty) assumes the covariate penalties are fixed at `default_lambda_beta` and `default_lambda_gamma`.

Value

A modified "imr_tune_grid" object where all "auto" placeholders are replaced by the numerically determined maximum values.

See Also

imr_tune_grid(), imr_tune(), imr_data()

Examples

```
# create sample data
Y <- matrix(
  c(2, NA, 3, 4,
    4, .5, NA, 4,
    NA, NA, 5, 3), 3, byrow= TRUE
)
# create a data object
data <- imr_data(Y =  Y, val_prop = 0.2)
# create a grid of hyperparameters
grid <- imr_tune_grid(nuclear = c(0, NA, 5, 2),
                      rank = c(2, 5, 1, 2))

# get the KKT max value for the nuclear parameter
grid <- imr_set_grid_limits(data, grid, bisection_iter=0)

# tune the parameters lambda_m and r on the model Y = M
cv_out <- imr_tune(data, grid, fast_nuclear = TRUE, n_cores = 1)
```

---

imr_similarity *Generate Similarity or Information Matrix Object*

Description

Creates an "imr_similarity" object containing the Eigenvalue-decomposition of a similarity or information matrix. This object is required for incorporating row or column similarities in an IMR model. The function can accept a matrix or generate a covariance matrix from distance data using specified kernels.

Usage

```
imr_similarity(
  x,
  normalize = TRUE,
  invert = TRUE,
  jitter = 0.2,
  d = NULL,
  matern_smoothness = 1.5,
```

```
  matern_range = 1,
  rbf_ell = 1
)
```

Arguments

x	Either a square numeric matrix, or a character string specifying a kernel method ("matern" or "rbf"). If a matrix is provided, it should be a similarity or information matrix (if invert = FALSE), or a covariance matrix (if invert = TRUE).
normalize	Logical. Should the kernel be symmetrically normalized by its degree matrix ($S \leftarrow D^{-1/2}SD^{-1/2}$, where D is the diagonal matrix of row sums)? This assumes a non-negative kernel and is highly recommended for better performance. See the appendix of the accompanying paper (Fouda et al., 2026; https://arxiv.org/abs/2606.26325) for details. Defaults to TRUE.
invert	Logical. Should the kernel be inverted? The IMR model expects an information matrix. If your input (or generated kernel) represents a covariance matrix, you must set this to TRUE to invert it. Defaults to TRUE.
jitter	Numeric. A small positive value added to the diagonal of the matrix to improve numerical stability and reduce the condition number. This is applied before any inversion. A value between 0 and 1 and appropriately away from the boundaries is recommended. Defaults to 0.2.
d	A numeric distance matrix. This is strictly required if x is set to "matern" or "rbf". Defaults to NULL.
matern_smoothness, matern_range	the parameters for the Matern kernel: smoothness and range. Defaults to (smoothness = 1.5, range = 1).
rbf_ell	The length-scale parameter for the Radial Basis Function (RBF) kernel. Defaults to 1.

Value

An object of class "imr_similarity". This is a list containing:

- U: A matrix of eigenvectors.
- d: A vector of eigenvalues.
- meta: A list of metadata.

References

Fouda, K., Labbe, A., & Oualkacha, K. (2026). *Incomplete Matrix Regression*. https://arxiv.org/abs/2606.26325

See Also

imr_data(), print.imr_similarity()

Examples

```
# generate 5 random spatial locations
coords <- data.frame(
x = runif(5, 0, 10),
y = runif(5, 0 , 10))

# compute the distance matrix
distance_matrix <- as.matrix(dist(coords))

# generate the similarity object of a 5/2 matern kernel
sim <- imr_similarity(x = "matern", d = distance_matrix, matern_smoothness = 1.5)

# print the matrix's metadata
print(sim)
```

---

imr_tune *Hyperparameter Tuning for IMR Models*

Description

Executes hyperparameter optimization for Incomplete Matrix Regression (IMR) models. The procedure evaluates predictive performance on a validation set (`y_valid`) while estimating the model on a training set (`y_train`).

Usage

```
imr_tune(
  data,
  grid,
  final_fit = TRUE,
  use_warm_in_final = TRUE,
  fast_nuclear = TRUE,
  convergence = imr_convergence(),
  error_function = get_metric("rmse"),
  warm_start = NULL,
  verbose = 1,
  n_cores = 4,
  seed = NULL,
  nuclear_log_scale = TRUE,
  tune_maxit = 10,
  tune_tol = 1e-04
)
```

Arguments

`data`	An object of class `"imr_data"` containing the training and validation partitions.

`grid` An object of class `"imr_tune_grid"`. **Note:** Any `"auto"` specifications for maximum values within this grid must be resolved via `imr_set_grid_limits` prior to invoking `imr_tune`.

`final_fit` Logical. If `TRUE`, the function performs a final model estimation on the complete matrix (`Y`) using the identified optimal hyperparameters. Defaults to `TRUE`.

`use_warm_in_final` Logical. If `TRUE` (default), the model fit of the best parameter set will given as `warm_start` input for the final model estimation. If `FALSE`, the initialization procedure specified in `convergence` will be used.

`fast_nuclear` Logical. Specifies the algorithmic approach for tuning the low-rank component (nuclear norm penalty). Defaults to `TRUE`.

`convergence` An `"imr_convergence"` object specifying numerical tolerances for internal model estimation.

`error_function` A function used to evaluate prediction error on the validation set. Must accept two arguments, `(predicted, actual)`. Defaults to `get_metric("rmse")`.

`warm_start` An optional object of class `"imr_fit"`. If provided, the coefficients from this previous fit are used as the starting values for the new optimization, overriding the initialization strategy in `convergence`. Defaults to `NULL`.

`verbose` Integer. Controls the level of diagnostic progress output. Defaults to `1`.

`n_cores` Integer. Number of CPU cores allocated for parallel execution across covariate grids. Defaults to `4`.

`seed` Integer. Seed for random number generation to ensure reproducibility. Defaults to `NULL`.

`nuclear_log_scale` Logical. If `TRUE`, the λ_M (`lambda_m`) grid is constructed on a logarithmic scale, increasing the density of evaluation points near the lower bound. Recommended when the maximum λ_M is large. Defaults to `TRUE`.

`tune_maxit` Integer. Maximum number of alternating optimization iterations permitted when simultaneously tuning all three parameters (Scenario 3). Defaults to `10`.

`tune_tol` Numeric. Relative tolerance threshold for early stopping during alternating optimization. Optimization terminates if the absolute relative change in validation error falls below this limit. Defaults to `1e-4`.

Details

Tuning Scenarios: The function selects an optimization trajectory based on the complexity of the `grid` and model specification:

1. If λ_β and λ_Γ are fixed or absent from the model, optimization proceeds sequentially over the low-rank component parameters.
2. If exactly one covariate penalty (λ_β or λ_Γ) requires tuning, the function parallelizes execution across the corresponding covariate grid. For each value, Scenario 1 is executed to optimize the nuclear parameters.
3. When λ_β, λ_Γ, and the nuclear component all require tuning, an alternating loop is utilized. The algorithm fixes λ_Γ and optimizes λ_β + nuclear (Scenario 2), then fixes λ_β and optimizes λ_Γ + nuclear. This iterates until `tune_tol` or `tune_maxit` is reached.

Nuclear Tuning Modes (`fast_nuclear`):

- **Fast Mode (`TRUE`):** Commences at the maximum λ_M and minimum rank. The algorithm simultaneously decrements λ_M while increasing the rank, terminating when validation performance stagnates according to the grid's patience parameter.
- **Slow Mode (`FALSE`):** Constructs a nested grid. For each λ_M value, the function iteratively evaluates ranks until predictive performance degrades. This mode provides a comprehensive mapping of the parameter space.

Value

A list comprising:

- `all_params`: A data frame recording the optimal parameter configurations identified at each major iteration of the tuning process.
- `history`: A comprehensive data frame of all evaluated parameter combinations and their respective validation errors.
- `fit`: The final estimated `"imr_fit"` object evaluated at the global optimal parameters (if `final_fit = TRUE`).
- `params`: A data frame containing the global optimal hyperparameter combination.
- `time_secs`: Total execution time in seconds.

See Also

`imr_tune_grid()`, `imr_set_grid_limits()`, `imr_data()`

Examples

```
# create sample data
Y <- matrix(
  c(2, NA, 3, 4,
    4, .5, NA, 4,
    NA, NA, 5, 3), 3, byrow= TRUE
)
# create a data object
data <- imr_data(Y =  Y, val_prop = 0.2)
# create a grid of hyperparameters
grid <- imr_tune_grid(nuclear = c(0, NA, 5, 2),
                      rank = c(2, 5, 1, 2))

# get the KKT max value for the nuclear parameter
grid <- imr_set_grid_limits(data, grid, bisection_iter=0)

# tune the parameters lambda_m and r on the model Y = M
cv_out <- imr_tune(data, grid, fast_nuclear = TRUE, n_cores = 1)
```

`imr_tune_grid` *Define Hyperparameter Tuning Grid for IMR*

Description

Constructs a configuration object specifying the search space for hyperparameter optimization within an Incomplete Matrix Regression (IMR) framework. This object facilitates the execution of the `imr_tune` function by defining the domain for identifying optimal hyperparameters.

Usage

```
imr_tune_grid(
  beta = c(0, NA, 20),
  gamma = c(0, NA, 20),
  nuclear = c(0, NA, 20, 2),
  rank = c(2, 30, 2, 2)
)
```

Arguments

`beta`	A numeric vector defining the search grid for the row-wise covariate regularization parameter (λ_β). Expected format: `c(min, max, length)`. Defaults to `c(0, NA, 20)`.
`gamma`	A numeric vector defining the search grid for the column-wise covariate regularization parameter (λ_Γ). Expected format: `c(min, max, length)`. Defaults to `c(0, NA, 20)`.
`nuclear`	A numeric vector defining the search grid for the nuclear norm penalty (λ_M) applied to the low-rank component. Expected format: `c(min, max, length, streaks)`, where `streaks` specifies the patience threshold for early stopping. Defaults to `c(0, NA, 20, 2)`.
`rank`	A numeric vector defining the search grid for the rank of the low-rank component. Expected format: `c(min, max, step, streaks)`, where `streaks` specifies the patience threshold for early stopping. Defaults to `c(2, 30, 2, 2)`.

Details

Specifications for `beta`, `gamma`, and `nuclear` parameters are subject to the following conventions:

- If the second element (`max`) is specified as `NA`, the upper bound of the grid is determined internally via `imr_set_grid_limits`. You need to call that function before calling `imr_tune`.
- Providing a vector of length 1 (e.g., `beta = 0.5`) constrains the parameter to that value throughout the tuning process.
- If a vector of length 2 is provided, the elements are interpreted as the minimum and maximum bounds, respectively, with remaining grid attributes reverting to their default values.

Value

An object of class "imr_tune_grid", a list of the parameters and their values.

See Also

print.imr_tune_grid(), imr_set_grid_limits()

Examples

```
# Initialize default grid with automated limit calculation
default_grid <- imr_tune_grid()

# Fix beta at 0 and define custom search spaces for gamma, nuclear, and rank
custom_grid <- imr_tune_grid(
  beta = 0,
  gamma = c(0.01, 1, 10),
  nuclear = c(0, NA, 15, 3), # 15 points, patience threshold of 3
  rank = c(2, 10, 2, 2)
)

# print the grid's information
print(custom_grid)
```

---

is_incomplete *Has the matrix be processed by* as_incomplete()*?*

Description

Has the matrix be processed by as_incomplete()?

Usage

```
is_incomplete(x)
```

Arguments

x	Any object

Value

Boolean

See Also

as_incomplete()

Examples

```
# create sample data
Y <- matrix(
  c(2, NA, 0,
  4, .5, NA,
  NA, NA, 0), 3, byrow= TRUE
)
# make it sparse with both NAs and 0s dropped
Y <- as_incomplete(Y)
# verify
print(is_incomplete(Y))
```

---

mc_with_means *Matrix Completion using Row and Column Means*

Description

A naive matrix completion method that imputes missing entries by filling them with the average of their corresponding row and column means.

Usage

```
mc_with_means(mat)
```

Arguments

mat	A matrix object. This can be an Incomplete matrix, a standard sparse matrix, or a base R dense matrix. If it is not already of class Incomplete, it will be converted internally prior to imputation.

Details

The function efficiently computes row and column means using an internal C++ functions, excluding missing entries. It replaces each missing value at index (i, j) with the mean of the i-th row mean and the j-th column mean.

Value

A dense matrix of class Incomplete.

Examples

```
# Create a sample matrix with missing values (NAs)
mat <- matrix(c(1, NA, 3, 4, 5, NA, 7, 8, 9), nrow = 3)

# Impute the missing values using row and column averages
mc_with_means(mat)
```

print.imr_convergence *Summary of the model's convergence parameters*

Description

Summary of the model's convergence parameters

Usage

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

Arguments

x	An imr_convergence object
...	Additional arguments to comply with generic function

Value

The input imr_convergence object x, invisibly. Called for its side effect of printing the convergence settings to the console.

See Also

imr_convergence() imr_fit()

Examples

```
# Use the default convergence parameters
convergence <- imr_convergence()
print(convergence)
```

print.imr_data *Print an IMR data object*

Description

Print an IMR data object

Usage

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

Arguments

x	An imr_data object
...	Additional arguments to comply with generic function

Value

The input imr_data object x, invisibly. Called for its side effect of printing the data dimensions, split, and model configuration to the console.

See Also

imr_data() update.imr_data() imr_similarity()

Examples

```
# create sample data
Y <- matrix(
  c(2, NA, 3,
  4, .5, NA,
  NA, NA, 5), 3, byrow= TRUE
)

# create covariate matrix of 2 variables
X <- matrix(rnorm(3*2), 3, 2)

# create the data object
data <- imr_data(Y =  Y, X = X, val_prop = 0.0)

# update the model to add row intercepts
data <- update(data, row_intercept = TRUE)

# print the metadata
print(data)
```

---

print.imr_fit *Summary of the model's fit operation*

Description

Summary of the model's fit operation

Usage

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

Arguments

x	An imr_fit object
...	Additional arguments to comply with generic function

Value

The input imr_fit object x, invisibly. Called for its side effect of printing a compact overview of the fitted model to the console.

See Also

imr_fit(), summary.imr_fit()

Examples

```
# create sample data
Y <- matrix(
  c(2, NA, 3,
  4, .5, NA,
  NA, NA, 5), 3, byrow= TRUE
)

# create covariate matrix of 2 variables
X <- matrix(rnorm(3*2), 3, 2)

# create the data object
data <- imr_data(Y =  Y, X = X, val_prop = 0.0)

# update the model to add row intercepts
data <- update(data, row_intercept = TRUE)

# fit the model
fit <- imr_fit(data, rank = 2)

# print the model fit summary
print(fit)

# print a summary of the fitted coefficients
summary(fit)

# extract the coefficients
coefs <- coef(fit)

# estimate the target matrix
target <- reconstruct(fit, data)$estimates

# compute estimates of the training data
estimates <- reconstruct_partial(fit, data, data$Y@i, data$Y@p, return_matrix = FALSE)

# compute the training Root Mean Squared Error
evaluate(estimates, data$Y@x, metric = "RMSE")
```

`print.imr_similarity` *Print an IMR similarity object*

Description

Print an IMR similarity object

Usage

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

Arguments

`x`	An `imr_similarity` object
`...`	Additional arguments to comply with generic function

Value

The input `imr_similarity` object `x`, invisibly. Called for its side effect of printing the decomposition metadata to the console.

See Also

`imr_similarity()`

Examples

```
# generate 5 random spatial locations
coords <- data.frame(
x = runif(5, 0, 10),
y = runif(5, 0 , 10))

# compute the distance matrix
distance_matrix <- as.matrix(dist(coords))

# generate the similarity object of a 5/2 matern kernel
sim <- imr_similarity(x = "matern", d = distance_matrix, matern_smoothness = 1.5)

# print the matrix's metadata
print(sim)
```

`print.imr_tune_grid` *Summary of the hyperparameter grid*

Description

Summary of the hyperparameter grid

Usage

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

Arguments

`x`	An `imr_tune_grid` object
`...`	Additional arguments to comply with generic function

Value

The input `imr_tune_grid` object `x`, invisibly. Called for its side effect of printing the configured search ranges to the console.

See Also

`imr_tune_grid()`

Examples

```
# Initialize default grid with automated limit calculation
default_grid <- imr_tune_grid()

# Fix beta at 0 and define custom search spaces for gamma, nuclear, and rank
custom_grid <- imr_tune_grid(
  beta = 0,
  gamma = c(0.01, 1, 10),
  nuclear = c(0, NA, 15, 3), # 15 points, patience threshold of 3
  rank = c(2, 10, 2, 2)
)

# print the grid's information
print(custom_grid)
```

`reconstruct` *Reconstruct Model Estimates and Components*

Description

Calculates the estimated response matrix from a fitted IMR model. In addition to the final predicted values, it computes and returns the individual structural components of the model, such as the low-rank matrix, covariate effects, and intercepts.

Usage

```
reconstruct(fit, data, trace = TRUE)
```

Arguments

`fit`	An object of class `"imr_fit"`, generated by `imr_fit`.
`data`	An object of class `"imr_data"`, generated by `imr_data`.
`trace`	Logical. If `TRUE` (the default), prints progress messages for each reconstruction step.

Details

During the model fitting process, covariates are made orthogonal using QR decomposition. The `reconstruct` function reverses this process, using the `R` matrices (`Xr` and `Zr`) to back-transform the estimated coefficients (`beta` and `gamma`) to the original scale of the input data.

The final `estimates` matrix is computed by summing the active components based on the model structure: the low-rank matrix ($M = UDV^T$), row covariate effects ($X\beta$), column covariate effects (ΓZ), and any intercepts. The function automatically handles matrix dimension sweeping for shared effects

Value

A list containing the reconstructed components and final estimates:

- `beta`: The row covariate coefficients transformed back to the original scale.
- `gamma`: The column covariate coefficients transformed back to the original scale.
- `M`: The reconstructed low-rank component matrix.
- `xbeta`: The matrix of row covariate effects.
- `gammaz`: The matrix of column covariate effects.
- `beta0`: The estimated row intercepts.
- `gamma0`: The estimated column intercepts.
- `estimates`: The final predicted response matrix (the sum of all active components).

See Also

`reconstruct_partial()`

Examples

```
# create sample data
Y <- matrix(
  c(2, NA, 3,
  4, .5, NA,
  NA, NA, 5), 3, byrow= TRUE
)

# create covariate matrix of 2 variables
X <- matrix(rnorm(3*2), 3, 2)

# create the data object
data <- imr_data(Y =  Y, X = X, val_prop = 0.0)

# update the model to add row intercepts
data <- update(data, row_intercept = TRUE)

# fit the model
fit <- imr_fit(data, rank = 2)

# print the model fit summary
print(fit)

# print a summary of the fitted coefficients
summary(fit)

# extract the coefficients
coefs <- coef(fit)

# estimate the target matrix
target <- reconstruct(fit, data)$estimates

# compute estimates of the training data
estimates <- reconstruct_partial(fit, data, data$Y@i, data$Y@p, return_matrix = FALSE)

# compute the training Root Mean Squared Error
evaluate(estimates, data$Y@x, metric = "RMSE")
```

`reconstruct_partial` *Partially Reconstruct Model Estimates for Specific Indices*

Description

Computes the estimated response matrix from a fitted IMR model, but *only* for a specific subset of matrix entries. This is highly memory-efficient and fast when you only need to evaluate predictions on a test set or validation mask, as it avoids computing the full dense prediction matrix.

Usage

```
reconstruct_partial(
  fit,
  data,
  irow,
  pcol,
  trace = FALSE,
  return_matrix = FALSE
)
```

Arguments

fit	An object of class "imr_fit", generated by imr_fit.
data	An object of class "imr_data", generated by imr_data.
irow, pcol	Integer vectors corresponding to the @i and @p slots of a "CsparseMatrix" (dgCMatrix), denoting the indices of the elements to be estimated. The returned values correspond to the @x slot of that object.
trace	Logical. If TRUE, prints progress messages for each reconstruction step. Defaults to FALSE.
return_matrix	Logical. If TRUE then it returns a CsparseMatrix with the following slots: @i=irow, @p=pcol, @x=estimates. If FALSE, only the @x slot is returned as a vector. Defaults to FALSE.

Details

Unlike the full reconstruct function, which allocates and calculates every cell of an $n \times m$ matrix, reconstruct_partial relies on optimized C++ routines to calculate only the targeted entries.

Value

Either a CsparseMatrix object or a numeric vector (see above).

See Also

reconstruct()

Examples

```
# create sample data
Y <- matrix(
  c(2, NA, 3,
  4, .5, NA,
  NA, NA, 5), 3, byrow= TRUE
)

# create covariate matrix of 2 variables
X <- matrix(rnorm(3*2), 3, 2)

# create the data object
```

```
data <- imr_data(Y =  Y, X = X, val_prop = 0.0)

# update the model to add row intercepts
data <- update(data, row_intercept = TRUE)

# fit the model
fit <- imr_fit(data, rank = 2)

# print the model fit summary
print(fit)

# print a summary of the fitted coefficients
summary(fit)

# extract the coefficients
coefs <- coef(fit)

# estimate the target matrix
target <- reconstruct(fit, data)$estimates

# compute estimates of the training data
estimates <- reconstruct_partial(fit, data, data$Y@i, data$Y@p, return_matrix = FALSE)

# compute the training Root Mean Squared Error
evaluate(estimates, data$Y@x, metric = "RMSE")
```

`summary.imr_fit` *Summary of the fitted model's parameters*

Description

Summary of the fitted model's parameters

Usage

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

Arguments

`object`	An `imr_fit` object
`...`	Additional arguments to comply with generic function

Value

The input `imr_fit` object, invisibly. Called for its side effect of printing goodness-of-fit statistics, a variance decomposition, and the estimated coefficients to the console.

See Also

imr_fit(), print.imr_fit()

Examples

```
# create sample data
Y <- matrix(
  c(2, NA, 3,
  4, .5, NA,
  NA, NA, 5), 3, byrow= TRUE
)

# create covariate matrix of 2 variables
X <- matrix(rnorm(3*2), 3, 2)

# create the data object
data <- imr_data(Y =  Y, X = X, val_prop = 0.0)

# update the model to add row intercepts
data <- update(data, row_intercept = TRUE)

# fit the model
fit <- imr_fit(data, rank = 2)

# print the model fit summary
print(fit)

# print a summary of the fitted coefficients
summary(fit)

# extract the coefficients
coefs <- coef(fit)

# estimate the target matrix
target <- reconstruct(fit, data)$estimates

# compute estimates of the training data
estimates <- reconstruct_partial(fit, data, data$Y@i, data$Y@p, return_matrix = FALSE)

# compute the training Root Mean Squared Error
evaluate(estimates, data$Y@x, metric = "RMSE")
```

---

svd_opt *Optimized Singular Value Decomposition*

Description

A general-purpose wrapper for Singular Value Decomposition (SVD) that selects the most computationally efficient backend based on the matrix dimensions, sparsity, and the desired number of singular components.

Usage

```
svd_opt(mat, k = NULL, tol = NULL)
```

Arguments

mat	A numeric matrix. Can be a base R dense matrix or a Sparse matrix.
k	Integer. The number of singular values and corresponding eigenvectors to compute or retain. If NULL or greater than or equal to the maximum possible rank of the matrix, a full SVD is computed.
tol	Numeric. A tolerance threshold for eigenvalue truncation. After computation, any singular values less than or equal to tol (and their corresponding eigenvectors in u and v) are removed. Defaults to NULL (no threshold applied).

Details

To minimize computation time, svd_opt routes the SVD request to different algorithms depending on the scenario:

- Thin Matrices: If the matrix is wide (ncol > 2 * nrow) or tall (nrow > 2 * ncol), it utilizes internal C++ functions (svd_small_nr_cpp or svd_small_nc_cpp).
- Full SVD: If k is NULL or requests the full rank, it uses base R's svd function.
- Partial SVD (Sparse or k <= 5): For large matrices where only a few components are needed or if the matrix is sparse, it uses the irlba.
- Partial SVD (Dense and k >= 5): For dense matrices where a larger number of components are requested, it uses the svds).

Value

A list containing the SVD components:

- d: A vector containing the computed singular values.
- u: A matrix whose columns contain the left singular vectors.
- v: A matrix whose columns contain the right singular vectors.

Examples

```
x <- matrix(rnorm(100),10, 10)
# return the first singular value and vectors
svd_opt(x, k = 1)
```

update.imr_data *Update IMR Model Structure*

Description

Updates the logical model structure of an existing "imr_data" object. This allows you to toggle model components (like intercepts, covariates, or low-rank components) on or off without needing to re-process the underlying data matrices.

Usage

```
## S3 method for class 'imr_data'
update(
  object,
  row_covariates = NULL,
  col_covariates = NULL,
  low_rank_component = NULL,
  row_similarity = NULL,
  col_similarity = NULL,
  row_intercept = NULL,
  col_intercept = NULL,
  shared_beta = NULL,
  shared_gamma = NULL,
  ...
)
```

Arguments

object	An object of class "imr_data".
row_covariates	Logical. Include row covariates in the model? (Requires X to have been provided to imr_data).
col_covariates	Logical. Include column covariates in the model? (Requires Z to have been provided to imr_data).
low_rank_component	Logical. Include the low-rank component ($M = UDV^T$)?
row_similarity	Logical. Include row similarity penalties? (Requires similarity_rows to have been provided to imr_data).
col_similarity	Logical. Include column similarity penalties? (Requires similarity_cols to have been provided to imr_data).
row_intercept	Logical. Include a row intercept/bias term?
col_intercept	Logical. Include a column intercept/bias term?
shared_beta	Logical. Should row covariate effects (beta) be shared across all columns?
shared_gamma	Logical. Should column covariate effects (gamma) be shared across all rows?
...	Additional arguments (ignored).

Value

The modified "imr_data" object with updated $model flags.

See Also

imr_data(), imr_similarity()

Examples

```
# create sample data
Y <- matrix(
  c(2, NA, 3,
  4, .5, NA,
  NA, NA, 5), 3, byrow= TRUE
)

# create covariate matrix of 2 variables
X <- matrix(rnorm(3*2), 3, 2)

# create the data object
data <- imr_data(Y =  Y, X = X, val_prop = 0.0)

# update the model to add row intercepts
data <- update(data, row_intercept = TRUE)

# print the metadata
print(data)
```

Index

∗ **datasets**
    Bixi_sample, 3

as_incomplete, 3
as_incomplete(), *20*

Bixi_sample, 3

coef.imr_fit, 4
coef.imr_fit(), *11*

evaluate, 5
evaluate(), *7*

get_metric, 7
get_metric(), *6*

imr_convergence, 7
imr_convergence(), *11*, *22*
imr_data, 8, *27*, *29*
imr_data(), *11*, *14*, *15*, *18*, *23*, *34*
imr_fit, 10, *27*, *29*
imr_fit(), *4*, *8*, *22*, *24*, *31*
imr_set_grid_limits, 12, *17*
imr_set_grid_limits(), *18*, *20*
imr_similarity, *9*, 14
imr_similarity(), *9*, *23*, *25*, *34*
imr_tune, 16
imr_tune(), *14*
imr_tune_grid, 19
imr_tune_grid(), *14*, *18*, *26*
irlba, *32*
is_incomplete, 20

mc_with_means, 21

print.imr_convergence, 22
print.imr_convergence(), *8*
print.imr_data, 22
print.imr_fit, 23
print.imr_fit(), *11*, *31*
print.imr_similarity, 25
print.imr_similarity(), *15*
print.imr_tune_grid, 26
print.imr_tune_grid(), *20*

reconstruct, 27, *29*
reconstruct(), *29*
reconstruct_partial, 28
reconstruct_partial(), *27*

summary.imr_fit, 30
summary.imr_fit(), *11*, *24*
svd, *32*
svd_opt, 31
svds, *32*

update.imr_data, 33
update.imr_data(), *9*, *23*