Package 'rtmpinv'

December 3, 2025

Type Package

Title Tabular Matrix Problems via Pseudoinverse Estimation

Version 0.1.0

Description The Tabular Matrix Problems via Pseudoinverse Estimation (TMPinv) is a two-stage estimation method that reformulates structured table-based systems - such as allocation problems, transaction matrices, and input-output tables - as structured least-squares problems. Based on the Convex Least Squares Programming (CLSP) framework, TMPinv solves systems with row and column constraints, block structure, and optionally reduced dimensionality by (1) constructing a canonical constraint form and applying a pseudoinverse-based projection, followed by (2) a convex-programming refinement stage to improve fit, coherence, and regularization (e.g., via Lasso, Ridge, or Elastic Net).

License MIT + file LICENSE

Encoding UTF-8

Language en-US

Depends R (>= 4.2)

Imports rclsp

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

URL https://github.com/econcz/rtmpinv

BugReports https://github.com/econcz/rtmpinv/issues

RoxygenNote 7.3.3

NeedsCompilation no

Author Ilya Bolotov [aut, cre] (ORCID: <https://orcid.org/0000-0003-1148-7144>)

Maintainer Ilya Bolotov <ilya.bolotov@vse.cz>

Repository CRAN

Date/Publication 2025-12-03 20:30:08 UTC

Contents

`tmpinv` *Solve a tabular matrix estimation problem via Convex Least Squares Programming (CLSP).*

Description

Solve a tabular matrix estimation problem via Convex Least Squares Programming (CLSP).

Usage

```
tmpinv(
  S = NULL,
  M = NULL,
  b_row = NULL,
  b_col = NULL,
  b_val = NULL,
  i = 1L,
  j = 1L,
  zero_diagonal = FALSE,
  reduced = NULL,
  symmetric = FALSE,
  bounds = NULL,
  replace_value = NA_real_,
  tolerance = sqrt(.Machine$double.eps),
  iteration_limit = 50L,
  r = 1L,
  final = TRUE,
  alpha = NULL,
  ...
)
```

Arguments

`S` numeric matrix of size $(m + p) \times (m + p)$, optional. A diagonal sign-slack (surplus) matrix with entries in $\{0, \pm 1\}$.

- `0` enforces equality (== `b_row` or `b_col`),
- `1` enforces a lower-than-or-equal ($\leq$) condition,
- `-1` enforces a greater-than-or-equal ($\geq$) condition. The first `m` diagonal entries correspond to row constraints, and the remaining `p` correspond to column constraints.

M	numeric matrix of size $k \times (mp)$, optional. A model matrix, typically with entries in $\{0, 1\}$. Each row defines a linear restriction on the flattened solution matrix. The corresponding right-hand-side values must be supplied in b_val. This block encodes known cell values.
b_row	numeric vector of length m. Right-hand-side vector of row totals.
b_col	numeric vector of length p. Right-hand-side vector of column totals.
b_val	numeric vector of length k. Right-hand-side vector of known cell values.
i	integer, default = 1. Number of row groups.
j	integer, default = 1. Number of column groups.
zero_diagonal	logical scalar, default = FALSE. If TRUE, enforces a structural zero diagonal.
reduced	integer vector of length 2, optional. Dimensions of the reduced problem. If supplied, estimation is performed block-wise on contiguous submatrices. For example, reduced = c(6,6) yields 5×5 blocks with one slack row and one slack column (edge blocks may be smaller).
symmetric	logical scalar, default = FALSE. If TRUE, enforces symmetry of the estimated matrix via x <- 0.5 * (x + t(x)). This applies to tmpinv$x only. For symmetry in the model, add explicit symmetry rows to M instead of using this flag.
bounds	NULL, numeric(2), or list of numeric(2). Bounds on cell values. If a single pair c(low, high) is given, it is applied to all mp cells. Example: c(0, NA).
replace_value	numeric scalar or NA, default = NA. Final replacement value for any cell that violates the bounds by more than the given tolerance.
tolerance	numeric scalar, default = sqrt(.Machine$double.eps). Convergence tolerance for bounds.
iteration_limit	integer, default = 50. Maximum number of iterations allowed in the refinement loop.
r	integer scalar, default = 1 Number of refinement iterations for the first step of the CLSP estimator.
final	logical scalar, default = TRUE If FALSE, only the first step of the CLSP estimator is performed.
alpha	numeric scalar, numeric vector, or NULL, Regularization parameter for the second step of the CLSP estimator.
...	Additional arguments passed to the **rclsp** solver.

Value

An object of class "tmpinv" containing the fitted CLSP model (tmpinv$model) and solution matrix (tmpinv$x).

Note

1. In the reduced model, S is ignored. Slack behaviour is inferred from block-wise marginal totals. Likewise, M must be a unique row subset of an identity matrix (diagonal-only). Non-diagonal model matrices cannot be mapped into reduced blocks.
2. Internal keyword arguments b_lim and C_lim are passed to .tmpinv.instance() and contain cell-value bounds. These arguments are ignored in the reduced model.

See Also

clsp

CVXR-package

Examples

```
## Example 1: AP/TM reconstruction on a symmetric 20x20 matrix
## (10 percent known entries)

set.seed(123456789)

m <- 20L
p <- 20L

# sample (dataset)
X_true <- abs(matrix(rnorm(m * p), nrow = m, ncol = p))
X_true <- 0.5 * (X_true + t(X_true))              # symmetric

idx <- sample.int(
    m * p,
    size = max(1L, floor(0.1 * (m * p))),         # 10 percent known
    replace = FALSE
)

M     <- diag(m * p)[idx, , drop = FALSE]
b_row <- rowSums(X_true)
b_col <- colSums(X_true)
b_val <- matrix(as.numeric(X_true)[idx], ncol = 1L)

# model (unique MNBLUE estimator)
result <- tmpinv(
    M = M,
    b_row = b_row,
    b_col = b_col,
    b_val = b_val,
    bounds = c(0, NA),                            # non-negativity
    symmetric = TRUE,
    r = 1L,
    alpha = 1.0
)

# coefficients
print("true X:")
print(round(X_true, 4))

print("X_hat:")
print(round(result$x, 4))

# numerical stability
print("\nNumerical stability:")
print(paste("  kappaC :", result$model$kappaC))
print(paste("  kappaB :", result$model$kappaB))
```

```
print(paste("  kappaA :", result$model$kappaA))

# diagnostics
print("\nGoodness-of-fit:")
print(paste("  NRMSE :", result$model$nrmse))
print(paste("  Diagnostic band (min):", min(result$model$x_lower)))
print(paste("  Diagnostic band (max):", max(result$model$x_upper)))

# bootstrap NRMSE t-test
tt <- rclsp::ttest(
    result$model,
    sample_size = 30L,
    seed = 123456789L,
    distribution = rnorm,
    partial = TRUE
)
print("\nBootstrap t-test:")
print(tt)

## Example 2: AP/TM reconstruction on a 40x40 matrix
## with zero diagonal and reduced (20,20) submodels
## (20 percent known entries)

set.seed(123456789)

m <- 40L
p <- 40L

# sample (dataset)
X_true <- abs(matrix(rnorm(m * p), nrow = m, ncol = p))
diag(X_true) <- 0                                  # zero diagonal

idx <- sample.int(
    m * p,
    size = max(1L, floor(0.2 * (m * p))),          # 20 percent known
    replace = FALSE
)

M     <- diag(m * p)[idx, , drop = FALSE]
b_row <- rowSums(X_true)
b_col <- colSums(X_true)
b_val <- matrix(as.numeric(X_true)[idx], ncol = 1L)

# model (reduced models of size 20x20)
result <- tmpinv(
    M = M,
    b_row = b_row,
    b_col = b_col,
    b_val = b_val,
    zero_diagonal = TRUE,
    reduced = c(20L, 20L),
    bounds = c(0, NA),
    r = 1L,
```

```
    alpha = 1.0
)

print("true X:")
print(round(X_true, 4))

print("X_hat:")
print(round(result$x, 4))

# numerical stability across submodels
kC <- sapply(result$model, function(CLSP) CLSP$kappaC)
kB <- sapply(result$model, function(CLSP) CLSP$kappaB)
kA <- sapply(result$model, function(CLSP) CLSP$kappaA)

print("\nNumerical stability (min-max across models):")
print(paste("  kappaC :", range(kC)))
print(paste("  kappaB :", range(kB)))
print(paste("  kappaA :", range(kA)))

# diagnostics (min-max)
nrmse <- sapply(result$model, function(CLSP) CLSP$nrmse)
x_low <- unlist(lapply(result$model, function(CLSP) CLSP$x_lower))
x_up  <- unlist(lapply(result$model, function(CLSP) CLSP$x_upper))

print("\nGoodness-of-fit (min-max across models):")
print(paste("  NRMSE :", range(nrmse)))
print(paste("  Diagnostic band (min):", range(x_low)))
print(paste("  Diagnostic band (max):", range(x_up)))

# bootstrap t-tests across all block models
print("\nBootstrap t-tests:")
tests <- lapply(
    result$model,
    function(CLSP) rclsp::ttest(
        CLSP,
        sample_size = 30L,
        seed = 123456789L,
        distribution = rnorm,
        partial = TRUE
    )
)
print(tests)
```

Index