

Package ‘rtmpinvi’

May 17, 2026

Type Package

Title Interactive Tabular Matrix Problems via Pseudoinverse Estimation

Version 1.0.0

Description Provides an interactive wrapper for the 'tmpinv()' function from the 'rtmpinv' package with options extending its functionality to pre- and post-estimation processing and streamlined incorporation of prior cell information. 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 rtmpinv (>= 1.0.0)

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

URL <https://github.com/econcz/rtmpinvi>

BugReports <https://github.com/econcz/rtmpinvi/issues>

Config/roxygen2/version 8.0.0

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 2026-05-16 23:30:02 UTC

Contents

tmpinvi	2
Index	5

tmpinvi	<i>Solve an interactive tabular matrix estimation problem via Convex Least Squares Programming (CLSP).</i>
---------	--

Description

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

Usage

```
tmpinvi(
  ival = NULL,
  ibounds = NULL,
  preestimation = NULL,
  postestimation = NULL,
  update = FALSE,
  ...
)
```

Arguments

ival	NULL, numeric matrix, or data.frame. Prior information on known cell values. If supplied and not entirely missing, ival is flattened and used to construct b_val and the corresponding identity-subset model matrix M internally. Missing entries (NA) are ignored. If all entries of ival are NA, no prior information is used and b_val and M are set to NULL. When ival is provided, it overrides any b_val or M arguments supplied through ...
ibounds	NULL, numeric(2), or list of numeric(2). Dynamic cell-value bounds passed to tmpinv(bounds = ...). The object supplied to ibounds may be created or modified programmatically (for example within preestimation()). If a single pair c(low, high) is provided, it is applied uniformly to all cells. Alternatively, a list of pairs may be supplied to specify cell-specific bounds with others set to NA. When ibounds is non-NULL, it overrides any bounds argument supplied through ...
preestimation	NULL or function. A function executed prior to model estimation. If supplied, it is called as preestimation(ival) and may perform arbitrary preparatory steps, such as constructing dynamic bounds or modifying objects in the calling environment. The return value is ignored.

postestimation NULL or function. A function executed after model estimation. For a full model, it is called as `postestimation(model)`. For reduced (block-wise) models, it is called as `postestimation(model[[i]], i)` for each block index `i`. The return value is ignored.

update logical scalar, default = FALSE. If TRUE and `ival` is supplied, missing entries (NA) in `ival` are replaced by the corresponding fitted values from `tmpinvi$result$x`. The updated matrix is returned in the `tmpinvi$data` component. If FALSE, the data component contains the fitted solution matrix `tmpinvi$result$x`.

... Additional arguments passed to `rtmpinv::tmpinv()`.

Value

An object of class "tmpinvi" with components:

- `result`: a fitted object of class "tmpinv".
- `data`: the processed matrix (either the fitted solution `x` or the updated `ival`, depending on `update`).

See Also

[tmpinv](#)

Examples

```

RNGkind("L'Ecuyer-CMRG")
set.seed(123456789)

iso2 <- c("CN", "DE", "JP", "NL", "US")
T <- 10L
year <- (as.integer(format(Sys.Date(), "%Y")) - T) + seq_len(T)
m <- length(iso2)

df <- expand.grid(year = year, iso2 = iso2, KEEP.OUT.ATTRS = FALSE)
df <- df[order(df$year, df$iso2), ]

ex_cols <- paste0("EX_", iso2)
df[ex_cols] <- NA_real_
df$EX <- NA_real_
df$IM <- NA_real_

X_true <- vector("list", length(year))
names(X_true) <- as.character(year)

for (t in seq_along(year)) {
  scale <- 1000 * (1.05^(t - 1L))
  X <- matrix(runif(m * m, 0, scale), m, m)
  diag(X) <- 0
  X_true[[t]] <- X

  rows <- ((t - 1L) * m + 1L):((t - 1L) * m + m)
  df$EX[rows] <- rowSums(X)
}

```

```

df$IM[rows] <- colSums(X)

miss <- matrix(runif(m * m) > 0.5, m, m)
X[miss] <- NA_real_
df[rows, ex_cols] <- X
}

cv <- qnorm(0.975)

for (nm in ex_cols) {
  fit <- lm(df[[nm]] ~ year * iso2, data = df, na.action = na.exclude)
  pr <- predict(fit, df, se.fit = TRUE)
  ub <- pr$fit + cv * pr$se.fit
  ub[ub < 0] <- NA_real_
  df[[paste0("_", nm, "_lb")]] <- 0
  df[[paste0("_", nm, "_ub")]] <- ub
}

make_bounds <- function(lb, ub)
  Map(function(a, b) c(a, b), lb, ub)

df_out <- df

for (step in 1:2) {
  for (y in year) {
    idx <- df_out$year == y
    d <- df_out[idx, ]
    ival <- as.matrix(d[ex_cols])

    lb <- as.vector(t(as.matrix(d[paste0("_EX_", iso2, "_lb")])))
    ub <- as.vector(t(as.matrix(d[paste0("_EX_", iso2, "_ub")])))

    fit <- tmpinvi(
      ival = ival,
      ibounds = make_bounds(lb, ub),
      b_row = d$EX,
      b_col = d$IM,
      alpha = 1.0,
      update = TRUE
    )

    df_out[idx, ex_cols] <- fit$data
  }
}

drop_cols <- grep("^_EX_.*(lb|ub)$", names(df_out), value = TRUE)
df_out[drop_cols] <- NULL
df_out

```

Index

tmpinv, 3
tmpinvi, 2