

Package ‘rlppinv’

June 11, 2026

Type Package

Title Linear Programming via Regularized Least Squares

Version 2.0.0

Description The Linear Programming via Regularized Least Squares (LPPinv) is a two-stage estimation method that reformulates linear programs as structured least-squares problems. Based on the Convex Least Squares Programming (CLSP) framework, LPPinv solves linear inequality, equality, and bound constraints by (1) constructing a canonical constraint system and computing a pseudoinverse projection, followed by (2) a convex-programming correction stage to refine the solution under additional regularization (e.g., Lasso, Ridge, or Elastic Net). LPPinv is intended for underdetermined and ill-posed linear problems, for which standard solvers fail.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

Depends R (>= 4.3)

Imports recls (>= 2.0.0)

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

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

BugReports <https://github.com/econcz/rlppinv/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 2026-06-11 14:10:02 UTC

Contents

lppinv	2
Index	5

lppinv	<i>Solve a linear program via Convex Least Squares Programming (CLSP).</i>
--------	--

Description

Solve a linear program via Convex Least Squares Programming (CLSP).

Usage

```
lppinv(
  c = NULL,
  A_ub = NULL,
  b_ub = NULL,
  A_eq = NULL,
  b_eq = NULL,
  non_negative = TRUE,
  bounds = NULL,
  replace_value = NA_real_,
  tolerance = sqrt(.Machine$double.eps),
  final = TRUE,
  alpha = NULL,
  cond_tolerance = NULL,
  ...
)
```

Arguments

<code>c</code>	numeric vector of length p , optional. Objective-function coefficients. Included for API parity with Python's <code>pylppinv</code> ; not used by CLSP.
<code>A_ub</code>	numeric matrix of size $i \times p$, optional. Matrix of inequality constraints $\mathbf{A}_{ub}\mathbf{x} \leq \mathbf{b}_{ub}$.
<code>b_ub</code>	numeric vector of length i , optional. Right-hand side for the inequality constraints.
<code>A_eq</code>	numeric matrix of size $j \times p$, optional. Matrix of equality constraints $\mathbf{A}_{eq}\mathbf{x} = \mathbf{b}_{eq}$.
<code>b_eq</code>	numeric vector of length j , optional. Right-hand side for the equality constraints.
<code>non_negative</code>	logical scalar, default = TRUE. If FALSE, no default nonnegativity bound is applied.

bounds	NULL, numeric(2), or list of numeric(2). Bounds on variables. If a single pair $c(\text{low}, \text{high})$ is given, it is applied to all variables. If NULL, defaults to $c(0, \text{NA})$ for each variable (non-negativity).
replace_value	numeric scalar or NA, default = NA. Final replacement value for any variable that violates the bounds by more than the given tolerance.
tolerance	numeric scalar, default = $\sqrt{.Machine\$double.eps}$. Convergence tolerance for bounds.
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.
cond_tolerance	numeric scalar or NULL, default = NULL. Singular-value cutoff for the custom condition number function. If NULL, the implementation uses an internal relative cutoff of $1e-14$.
...	Additional arguments passed to the rclsp solver.

Value

An object of class "clsp" containing the fitted CLSP model.

See Also

[clsp](#)
[CVXR-package](#)

Examples

```
## Linear Programming via Regularized Least Squares (LPPinv)
## Underdetermined and potentially infeasible LP system

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

# sample (dataset)
A_ub <- matrix(rnorm(50 * 500), nrow = 50L, ncol = 500L)
A_eq <- matrix(rnorm(25 * 500), nrow = 25L, ncol = 500L)
b_ub <- matrix(rnorm(50),      ncol = 1L)
b_eq <- matrix(rnorm(25),      ncol = 1L)

# model (no default non-negativity, unique MNBLUE solution)
model <- lppinv(
  A_ub = A_ub,
  A_eq = A_eq,
  b_ub = b_ub,
  b_eq = b_eq,
  non_negative = FALSE,
  final = TRUE,
  alpha = 1.0
) # unique MNBLUE estimator
```

```

# coefficients
print("x hat (x_M hat):")
print(round(model$x, 4))

# numerical stability (if available)
if (!is.null(model$kappaC)) {
  cat("\nNumerical stability:\n")
  cat(" kappaC :", round(model$kappaC, 4), "\n")
}
if (!is.null(model$kappaB)) {
  cat(" kappaB :", round(model$kappaB, 4), "\n")
}
if (!is.null(model$kappaA)) {
  cat(" kappaA :", round(model$kappaA, 4), "\n")
}

# goodness-of-fit diagnostics (if available)
if (!is.null(model$normse)) {
  cat("\nGoodness-of-fit:\n")
  cat(" NRMSE :", round(model$normse, 6), "\n")
}
if (!is.null(model$x_lower)) {
  cat(" Diagnostic band (min):", round(min(model$x_lower), 4), "\n")
}
if (!is.null(model$x_upper)) {
  cat(" Diagnostic band (max):", round(max(model$x_upper), 4), "\n")
}

# bootstrap NRMSE t-test (if supported by rclsp)
if ("ttest" %in% names(model)) {
  cat("\nBootstrap t-test:\n")
  tt <- model$ttest(sample_size = 30L,
                    seed = 123456789,
                    distribution = "normal")
  for (nm in names(tt)) {
    cat(" ", nm, ": ", round(tt[[nm]], 6), "\n", sep = "")
  }
}

```

Index

`clsp`, [3](#)
`CVXR-package`, [3](#)
`lppinv`, [2](#)