

# Package ‘rclsp’

February 19, 2026

**Type** Package

**Title** A Modular Two-Step Convex Optimization Estimator for Ill-Posed Problems

**Version** 0.4.0

**Description** Convex Least Squares Programming (CLSP) is a two-step estimator for solving underdetermined, ill-posed, or structurally constrained least-squares problems. It combines pseudoinverse-based estimation with convex-programming correction methods inspired by Lasso, Ridge, and Elastic Net to ensure numerical stability, constraint enforcement, and interpretability. The package also provides numerical stability analysis and CLSP-specific diagnostics, including partial  $R^2$ , normalized RMSE (NRMSE), Monte Carlo t-tests for mean NRMSE, and condition-number-based confidence bands.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Language** en-US

**Depends** R ( $\geq 4.2$ )

**Imports** Matrix, stats, methods, CVXR, MASS

**Suggests** testthat ( $\geq 3.0.0$ )

**Config/testthat/edition** 3

**URL** <https://github.com/econcz/rclsp>

**BugReports** <https://github.com/econcz/rclsp/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](mailto:ilya.bolotov@vse.cz)>

**Repository** CRAN

**Date/Publication** 2026-02-19 15:10:02 UTC

## Contents

canonize . . . . .	2
clsp . . . . .	3
corr . . . . .	7
ttest . . . . .	8

<b>Index</b>	<b>10</b>
--------------	-----------

---

canonize	<i>Construct the canonical design matrix <math>A = [C S; M Q]</math> for CLSP.</i>
----------	--

---

### Description

This method assembles the constraint matrix  $A$  from user-supplied or internally generated components —  $C$ ,  $S$ ,  $M$ , and  $Q$  — and assigns the corresponding right-hand side vector  $b$ . It is a required pre-step before solving a Convex Least Squares Programming (CLSP) problem.

### Usage

```

canonize(
  object,
  problem = "",
  C = NULL,
  S = NULL,
  M = NULL,
  Q = NULL,
  b = NULL,
  m = NULL,
  p = NULL,
  i = 1L,
  j = 1L,
  zero_diagonal = FALSE
)

```

### Arguments

object	An object of class "clsp".
problem	Character, optional. Structural template for matrix construction. One of: <ul style="list-style-type: none"> <li>'ap' or 'tm': allocation or tabular matrix problem.</li> <li>'cmls' or 'rp': constrained modular least squares or RP-type.</li> <li>' ' or other: General CLSP problems (user-defined <math>C</math> and/or <math>M</math>).</li> </ul>
$C$ , $S$ , $M$	Numeric matrix or NULL. Blocks of the constraint matrix $A = [C S; M Q]$ . If $C$ and/or $M$ are provided, the matrix $A$ is constructed accordingly. If both are NULL and $A$ is not yet defined, an error is raised.

Q	Numeric matrix or NULL. Externally supplied residual slack matrix used to adjust inequality constraints in M. Required only when $r > 1$ . Encodes the sign pattern of residuals from the previous iteration and is used to construct the $[C S; M Q]$ canonical form. Defaults to a conformable zero matrix on the first iteration.
b	Numeric vector or NULL. Right-hand side vector. Must have as many rows as $A$ . Required.
m, p	Integer or NULL. Dimensions of $X \in \mathbb{R}^{m \times p}$ , relevant for allocation problems ('ap').
i, j	Integer, default = 1. Grouping sizes for row and column sum constraints in AP problems.
zero_diagonal	Logical, default = FALSE. If TRUE, enforces structural zero diagonals via identity truncation.

### Details

Depending on the specified problem type, it can generate allocation, tabular matrix, or modular constraints and enforce optional diagonal exclusions. All missing blocks are padded to ensure conformability.

### Value

An updated object of class "clsp".

### Attributes Set

- A Numeric matrix. Canonical design matrix constructed from (C, S, M, Q).
- C\_idx Integer vector of length 2 indicating the size of the C block.
- b Numeric vector. Conformable right-hand side vector.

---

clsp

*Convex Least Squares Programming (CLSP) estimator.*


---

### Description

The Convex Least Squares Programming (CLSP) estimator solves underdetermined, ill-posed, or structurally constrained least-squares problems using a modular two-step approach. The first step computes a pseudoinverse-based estimate, and the second step applies a convex correction (Lasso, Ridge, or Elastic Net) to ensure numerical stability, constraint enforcement, and interpretability.

### Usage

```
clsp(
  problem = "",
  C = NULL,
  S = NULL,
  M = NULL,
```

```

b = NULL,
m = NULL,
p = NULL,
i = 1L,
j = 1L,
zero_diagonal = FALSE,
r = 1L,
Z = NULL,
rcond = FALSE,
tolerance = NULL,
iteration_limit = NULL,
final = TRUE,
alpha = NULL,
...
)

```

### Arguments

problem	character scalar, optional Structural template for matrix construction. One of: <ul style="list-style-type: none"> <li>'ap' or 'tm': allocation or tabular matrix problem.</li> <li>'cmls' or 'rp': constrained modular least squares or RP-type.</li> <li>' ' or other: general CLSP problems (user-defined <math>C</math> and/or <math>M</math>).</li> </ul>
C, S, M	numeric matrix or NULL Blocks of the constraint matrix $A = \begin{bmatrix} C & S \\ M & Q \end{bmatrix}$ . If $C$ and/or $M$ are provided, the matrix $A$ is constructed accordingly. If both are NULL and $A$ is not yet defined, an error is raised.
b	numeric vector or NULL Right-hand-side vector. Must have as many rows as $A$ . Required.
m, p	integer scalar or NULL Dimensions of $X \in \mathbb{R}^{m \times p}$ , relevant for allocation problems ('ap').
i, j	integer scalar, default = 1 Grouping sizes for row and column-sum constraints in AP problems.
zero_diagonal	logical scalar, default = FALSE If TRUE, enforces structural zero diagonals via identity truncation.
r	integer scalar, default = 1 Number of refinement iterations for the pseudoinverse-based estimator. When $r > 1$ , the slack block $Q$ is updated iteratively to improve feasibility in underdetermined or ill-posed systems.
Z	numeric matrix or NULL A symmetric idempotent matrix (projector) defining the subspace for Bott–Duffin pseudoinversion. If NULL, the identity matrix is used, reducing to the Moore–Penrose case.
rcond	numeric scalar or logical scalar, default = FALSE Regularization parameter for the Moore–Penrose and Bott–Duffin inverses, providing numerically stable inversion and ensuring convergence of singular values. If TRUE, an automatic tolerance equal to <code>tolerance</code> is applied. If set to a numeric value, it specifies the relative cutoff below which small singular values are treated as zero.

tolerance	numeric scalar or NULL, default = NULL Convergence tolerance for NRMSE change between iterations.
iteration_limit	integer scalar or NULL, default = NULL Maximum number of iterations allowed in the refinement loop.
final	logical scalar, default = TRUE If TRUE, a convex programming problem is solved to refine $\hat{\mathbf{z}}$ . The resulting solution $\mathbf{z}$ minimizes a weighted $\ell_1/\ell_2$ norm around $\hat{\mathbf{z}}$ subject to $\mathbf{A}\mathbf{z} = \mathbf{b}$ .
alpha	numeric scalar, numeric vector, or NULL, default = NULL Regularization parameter: <ul style="list-style-type: none"> <li>• <math>\alpha = 0</math>: Lasso (<math>\ell_1</math> norm)</li> <li>• <math>\alpha = 1</math>: Ridge (<math>\ell_2</math> norm)</li> <li>• <math>0 &lt; \alpha &lt; 1</math>: Elastic Net. If a numeric scalar is provided, that value is used after clipping to <math>[0, 1]</math>. If a numeric vector is provided, each candidate is evaluated via a full solve, and the <math>\alpha</math> with the smallest NRMSE is selected. If NULL, <math>\alpha</math> is chosen automatically according to <math display="block">\alpha = \min \left( 1, \frac{\text{NRMSE}_{\alpha=0}}{\text{NRMSE}_{\alpha=0} + \text{NRMSE}_{\alpha=1} + \text{tolerance}} \right)</math> </li> </ul>
...	Optional. Additional arguments passed to the <b>CVXR</b> solver backend.

## Details

This estimator unifies pseudoinverse-based least squares with convex programming correction. The pseudoinverse step computes an initial solution  $\mathbf{z}^{(r)}$  iteratively via the Moore–Penrose or Bott–Duffin inverse. The convex step then refines  $\mathbf{z}$  by minimizing a mixed  $\ell_1/\ell_2$  norm under equality constraints  $\mathbf{A}\mathbf{z} = \mathbf{b}$ . The method supports allocation problems (AP), constrained modular least squares (CMLS), and general CLSP formulations.

## Value

An object of class "clsp" representing the fitted Convex Least Squares Programming (CLSP) model. The object is a named list containing all initialized fields and solver results. Class-specific methods such as `summary.clsp()`, `corr.clsp()`, and `ttest.clsp()` can be used to extract, analyze, and summarize the results.

## See Also

[CVXR](#)

## Examples

```
## Not run:
## Example: CMLS (RP) estimation with stationary-point constraints

set.seed(123456789)
```

```

# sample (dataset)
k <- 500L                                # number of observations
p <- 6L                                   # number of regressors
c0 <- 1                                   # sum of coefficients

D      <- matrix(NA_real_, nrow = k, ncol = p)
D[, 1] <- 1.0                             # constant
D[, 2:p] <- matrix(rnorm(k * (p - 1)), k, p - 1)

b_true <- rnorm(p)
b_true <- (b_true / sum(b_true)) * c0      # normalize to sum = c

e      <- matrix(rnorm(k), ncol = 1)
y      <- D %>% b_true + e

# build blocks for CLSP (CMLS)
b <- rbind(
  matrix(c0, ncol = 1),                    # sum of coefficients
  matrix(0, nrow = k - 2, ncol = 1),
  matrix(0, nrow = k - 1, ncol = 1),
  matrix(y, ncol = 1)
)

C <- rbind(
  matrix(1, nrow = 1, ncol = p),           # row of ones
  diff(D, differences = 2),               # 2nd differences
  diff(D, differences = 1)                # 1st differences
)

# diagonal sign-matrix for 2nd differences
S <- rbind(
  matrix(0, nrow = 1, ncol = k - 2),
  diag(sign(diff(as.numeric(y), differences = 2))),
  matrix(0, nrow = k - 1, ncol = k - 2)
)

# model
model <- rclsp::clsp(
  problem = "cmls",
  b       = b,
  C       = C,
  S       = S,
  M       = D,
  r       = 1L,                             # no refinement
  alpha   = 1.0                             # MNBLUE solution
)

# results
print("true beta (x_M):")
print(round(b_true, 4))

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

```

```

print(model)

# bootstrap t-test
tt <- rclsp::ttest(
  model,
  sample_size = 30L,
  seed        = 123456789L,
  distribution = rnorm,
  partial     = TRUE
)

print("Bootstrap t-test:")
print(tt)

## End(Not run)

```

---

corr

---

*Compute the structural correlogram of the CLSP constraint system.*


---

### Description

This method performs a row-deletion sensitivity analysis on the canonical constraint matrix  $[C|S]$ , denoted as  $C_{\text{canon}}$ , and evaluates the marginal effect of each constraint row on numerical stability, angular alignment, and estimator sensitivity.

### Usage

```
corr(object, reset = FALSE, threshold = 0)
```

### Arguments

object	An object of class "c1sp".
reset	Logical, default = FALSE. If TRUE, forces recomputation of all diagnostic values.
threshold	Numeric, default = 0. If positive, limits the output to constraints with $\text{RMSA}_i \geq \text{threshold}$ .

### Details

For each row  $i$  in  $C_{\text{canon}}$ , it computes:

- The Root Mean Square Alignment ( $\text{RMSA}_i$ ) with all other rows  $j \neq i$ .
- The change in condition numbers  $\kappa(C)$ ,  $\kappa(B)$ , and  $\kappa(A)$  when row  $i$  is deleted.
- The effect on estimation quality: changes in NRMSE,  $\hat{z}$ ,  $z$ , and  $x$ .

Additionally, it computes the total RMSA statistic across all rows, summarizing the overall angular alignment of the constraint block.

**Value**

A named list containing per-row diagnostic values:

**constraint** Vector of constraint indices (1-based).

**rmsa\_i** List of  $\text{RMSA}_i$  values.

**rmsa\_dkappaC** List of  $\Delta\kappa(C)$  after deleting row  $i$ .

**rmsa\_dkappaB** List of  $\Delta\kappa(B)$  after deleting row  $i$ .

**rmsa\_dkappaA** List of  $\Delta\kappa(A)$  after deleting row  $i$ .

**rmsa\_dnormse** List of  $\Delta\text{NRMSE}$  after deleting row  $i$ .

**rmsa\_dzhat** List of  $\Delta\hat{z}$  after deleting row  $i$ .

**rmsa\_dz** List of  $\Delta z$  after deleting row  $i$ .

**rmsa\_dx** List of  $\Delta x$  after deleting row  $i$ .

---

ttest	<i>Perform bootstrap or Monte Carlo t-tests on the NRMSE statistic from the CLSP estimator.</i>
-------	---

---

**Description**

This function either (a) resamples residuals via a nonparametric bootstrap to generate an empirical NRMSE sample, or (b) produces synthetic right-hand side vectors  $b$  from a user-defined or default distribution and re-estimates the model. It tests whether the observed NRMSE significantly deviates from the null distribution of resampled or simulated NRMSE values.

**Usage**

```
ttest(
  object,
  reset = FALSE,
  sample_size = 50L,
  seed = NULL,
  distribution = NULL,
  partial = FALSE,
  simulate = FALSE
)
```

**Arguments**

object	An object of class "clsp".
reset	Logical, default = FALSE. If TRUE, forces recomputation of the NRMSE null distribution.
sample_size	Integer, default = 50. Size of the Monte Carlo simulated sample under $H_0$ .
seed	Integer or NULL, default = NULL. Optional random seed to override the default.

distribution	Function or NULL, default = NULL. Distribution for generating synthetic b vectors. One of: rnorm, runif, or a custom RNG function. Defaults to standard normal.
partial	Logical, default = FALSE. If TRUE, runs the t-test on the partial NRMSE: during simulation, the C-block entries are preserved and the M-block entries are simulated.
simulate	Logical, default = FALSE. If TRUE, performs a parametric Monte Carlo simulation by generating synthetic right-hand side vectors b. If FALSE (default), executes a nonparametric bootstrap procedure on residuals without re-estimation.

### Value

A named list containing test results and null distribution statistics:

**p\_one\_left**  $P(\text{nmse} \leq \text{null mean})$

**p\_one\_right**  $P(\text{nmse} \geq \text{null mean})$

**p\_two\_sided** 2-sided t-test p-value

**nmse** Observed value

**mean\_null** Mean of null distribution

**std\_null** Standard deviation of null distribution

# Index

canonize, 2

clsp, 3

corr, 7

CVXR, 5

ttest, 8