Package: lassoinf (via r-universe)

October 20, 2026

Title Selective Inference for the Randomized Lasso

Version 0.1.0

Description Confidence intervals and p-values for parameters chosen by the (randomized) lasso, valid after selection. Implements the polyhedral approach of Lee, Sun, Sun and Taylor (2016) <doi:10.1214/15-AOS1371>, the randomized response of Tian and Taylor (2018) <doi:10.1214/17-AOS1564> and the exact randomized inference of Panigrahi, Fry and Taylor (2024) <doi:10.1093/biomet/asae019>, with the truncated bivariate normal computations of Liu (2023) <doi:10.48550/arXiv.2308.10346>. Includes inference for user-specified estimands, screening of inactive variables and problems fitted by 'glmnet'. Computations use a 'C++' backend with matrix-free linear operators for wide designs.

License BSD_3_clause + file LICENSE

Copyright See inst/COPYRIGHTS for the code adapted from Sifan Liu (MIT).

URL https://github.com/jonathan-taylor/lassoinf

BugReports https://github.com/jonathan-taylor/lassoinf/issues

Encoding UTF-8

Depends R (>= 3.5.0)

LinkingTo Rcpp, RcppEigen

Imports methods, mvtnorm, R6, Rcpp, stats

Suggests glmnet (>= 5.1), knitr, rmarkdown, testthat

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Jonathan Taylor [aut, cre, cph], Kevin Fry [aut], Snigdha Panigrahi [aut], Sifan Liu [ctb, cph] (truncated bivariate normal computations, see inst/COPYRIGHTS)

Maintainer Jonathan Taylor <jtaylo@stanford.edu>

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

Date/Publication 2026-10-20 18:00:02 UTC

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

RemoteRef HEAD

RemoteSha 34ba9fb465c44a098cd64e12cd9bd0a0f8f9393e

Contents

AffineConstraints 2
contrast_estimand 3
contrast_inference 4
custom_estimand_inference 4
dense_operator 6
estimand_summary 6
glmnet_inference 7
glmnet_problem 9
glmnet_scaling 11
inactive_summary 11
kkt_violation 12
LassoInference 13
ScreenedSelection 16
SelectionCoordinates 17
truncated_normal_inference 20
TruncBivariateNormal 21

Index **24**

`AffineConstraints` *C++ classes*

Description

Classes exposed from the C++ backend through an Rcpp module; construct them with `new()`.

`new(AffineConstraints, Z, Z_noisy, Sigma, Sigma_noise, scalar_noise = NaN)` The randomized selection setup: full-data score `Z`, the noisy score used for selection, and their covariances (matrices or `LinearOps`; `Sigma_noise` may be `NULL`, in which case `scalar_noise` scales `Sigma`). Methods `compute_contrast(v)`, `compute_covariance_contrast(...)` and `solve_contrast(...)`.

`new(DiscreteFamily, sufficient_stat, weights, theta)` Exponential family with reference measure `sum_j weights[j] delta(sufficient_stat[j])`. Methods `pdf`, `cdf`, `ccdf`, `equal_tailed_interval`, `get_theta`, `set_theta` and `get_partition`.

`new(TruncatedGaussian, estimate, sigma, smoothing_sigma, lower_bound, upper_bound, noisy_estimate, fact` Selection weight of a Gaussian estimate with randomized truncation; method `weight(x)`.

`new(WeightedGaussianFamily, estimate, sigma, weight_fns, num_sd, num_grid)` A Gaussian reweighted by the product of the functions in the list `weight_fns`, discretized on `num_grid` points within `num_sd` standard deviations. Methods `pvalue(null_value, alternative, basept)` and `interval(basept, level)`.

Examples

```
grid <- seq(-5, 5, length.out = 201)
fam <- new(DiscreteFamily, grid, dnorm(grid), 0)
fam$cdf(0, 1, 0)
```

`contrast_estimand` *Estimand specifications*

Description

`contrast_estimand(eta, offset)`: estimator `sum(eta * Z_full) + offset` for the estimand `sum(eta * E[Z_full]) + offset`. `covariance_estimand(theta_hat, variance, score_cov)`: estimator given by its value, `Var(theta_hat)` and `Cov(Z_full, theta_hat)`; requires `Sigma_noise`.

Usage

```
contrast_estimand(eta, offset = 0)

covariance_estimand(theta_hat, variance, score_cov)

estimand_value(estimand, Z)
```

Arguments

`eta`	Contrast vector.
`offset`	Constant added to the estimator and the estimand.
`theta_hat`	Observed value of the estimator.
`variance`	Variance of the estimator.
`score_cov`	Covariance of `Z_full` with the estimator.
`estimand`	A contrast estimand.
`Z`	A score vector.

Value

`contrast_estimand` and `covariance_estimand`: estimand specifications (lists of class `ContrastEstimand` and `CovarianceEstimand`), for `contrast_inference`, `custom_estimand_inference` and `estimand_summary`. `estimand_value`: the value of the estimator at `Z`.

Examples

```
e <- contrast_estimand(c(1, -1, 0), offset = 0.5)
estimand_value(e, c(2, 1, 5))
```

contrast_inference *Selective inference for a contrast estimand*

Description

Inference for `theta = sum(eta * E[Z_full]) + offset`, as in `LassoInference$summary()`.

Usage

```
contrast_inference(selection, estimand, level = NULL, null_value = 0)
```

Arguments

`selection`	A `LassoInference` or `ScreenedSelection`.
`estimand`	A `contrast_estimand` or a contrast vector `eta`.
`level`	Confidence level (default: the selection's level).
`null_value`	Null value for the p-value.

Value

A list with `estimate`, `lower_conf`, `upper_conf`, `p_value`, `variance` and `contrast`.

Examples

```
set.seed(1)
n <- 100; p <- 10
x <- matrix(rnorm(n * p), n, p)
y <- x[, 1] - x[, 2] + rnorm(n)
fit <- glmnet::glmnet(x, y, lambda = c(0.3, 0.2), control = list(thresh = 1e-14))
li <- glmnet_inference(fit, x, y, level = 0.9)   # index 0 is the intercept
# the sum of the coefficients of x1 and x2 in the selected model
eta <- li$contrasts[["1"]] + li$contrasts[["2"]]
contrast_inference(li, eta)[c("estimate", "lower_conf", "upper_conf", "p_value")]
```

custom_estimand_inference *Selective inference for an estimand given by its covariance with the score*

Description

For `(theta_hat, Z_full)` jointly Gaussian with known `Var(theta_hat)` and `Cov(Z_full, theta_hat)`. Requires `Sigma_noise`; with `scalar_noise`, express the estimand as a contrast.

Usage

```
custom_estimand_inference(
  selection,
  theta_hat,
  variance,
  score_cov,
  level = NULL,
  null_value = 0
)
```

Arguments

selection	A LassoInference or ScreenedSelection.
theta_hat	Observed value of the estimator.
variance	Variance of the estimator.
score_cov	Covariance of Z_full with the estimator.
level	Confidence level (default: the selection's level).
null_value	Null value for the p-value.

Value

A list as from contrast_inference: estimate, lower_conf, upper_conf, p_value, variance and contrast.

Examples

```
set.seed(1)
n <- 100; p <- 10
x <- matrix(rnorm(n * p), n, p)
y <- x[, 1] - x[, 2] + rnorm(n)
# randomized response y + omega with Var(omega) = kappa, sigma^2 = 1 known
kappa <- 0.5
y_noisy <- y + rnorm(n, sd = sqrt(kappa))
Q <- crossprod(x); Z <- drop(crossprod(x, y))
lam <- 2 * sqrt(n)
fit <- glmnet::glmnet(x, y_noisy, lambda = lam / n, intercept = FALSE,
                      standardize = FALSE, control = list(thresh = 1e-14))
b <- as.numeric(coef(fit))[-1]
G <- drop(crossprod(x, x %*% b - y_noisy))
li <- LassoInference$new(b, G, Q, rep(lam, p), NULL, NULL, Z, Q, kappa * Q, level = 0.9)
# the estimator sum(eta * Z), specified by its variance and covariance with Z
eta <- li$contrasts[[1]]
custom_estimand_inference(li, sum(eta * Z), sum(eta * (Q %*% eta)),
                          drop(Q %*% eta))[c("lower_conf", "upper_conf", "p_value")]
```

dense_operator *Linear operators*

Description

Matrix-free operators for `Q_hat`, `Sigma` and `Sigma_noise` in `LassoInference`. `dense_operator(M)` wraps a matrix; `xtvx_operator(X, weights)` represents `t(X) %*% diag(weights) %*% X` without forming it, which is what wide designs (n « p) need.

Usage

```
dense_operator(M)

xtvx_operator(X, weights = rep(1, nrow(X)))
```

Arguments

`M`	A square numeric matrix.
`X`	A numeric design matrix.
`weights`	Observation weights, length `nrow(X)`.

Value

A `LinearOp` with methods `multiply`, `multiply_transpose`, `rows`, `cols`, `diagonal`, `to_dense` and `scaled(c)` (c times the operator, sharing its data).

Examples

```
set.seed(1)
x <- matrix(rnorm(50 * 5), 50, 5)
v <- rnorm(5)
Q <- xtvx_operator(x)
all.equal(Q$multiply(v), drop(crossprod(x) %*% v))
all.equal(Q$scaled(2)$to_dense(), 2 * crossprod(x))
dense_operator(crossprod(x))$rows()
```

estimand_summary *Table of selective inference results*

Description

Table of selective inference results

Usage

```
estimand_summary(selection, estimands, level = NULL, null_value = 0)
```

Arguments

selection	A LassoInference or ScreenedSelection.
estimands	Named list of contrast_estimands, contrast vectors or covariance_estimands.
level	Confidence level (default: the selection's level).
null_value	Null value for the p-values.

Value

A data frame with columns index, estimate, lower_conf, upper_conf, p_value.

Examples

```
set.seed(1)
n <- 100; p <- 10
x <- matrix(rnorm(n * p), n, p)
y <- x[, 1] - x[, 2] + rnorm(n)
fit <- glmnet::glmnet(x, y, lambda = c(0.3, 0.2), control = list(thresh = 1e-14))
li <- glmnet_inference(fit, x, y, level = 0.9)   # index 0 is the intercept
estimand_summary(li, list(x1 = li$contrasts[["1"]],
                           sum = li$contrasts[["1"]] + li$contrasts[["2"]]))
```

---

glmnet_inference *Selective inference after a glmnet fit*

Description

Intervals and p-values for the coefficients selected by a glmnet fit at lambda = s, accounting for the selection. Without selection_rows the fit is on all the data and there is no randomization (the polyhedral approach). With selection_rows, the fit is on those rows only and inference uses all the data (carving): selecting on a proportion π of the data is randomization with scalar_noise = (1 - pi) / pi.

The selection score is Z_noisy = -G + Q_hat beta_hat from glmnet_problem_from_fit on the selection rows, and Sigma = dispersion * Q_hat / sum(weights). Without held-out data Z_full = Z_noisy. With held-out data, Z_full = -G_full(beta_tilde) + Q_hat beta_tilde, with G_full the full-data gradient and beta_tilde one Newton step for the full-data loss on the selected coordinates from beta_hat. Weights are treated as prior (frequency or precision) weights, as in glm.

Usage

```
glmnet_inference(
  fit,
  x,
  y,
  s = NULL,
  selection_rows = NULL,
```

```
  weights = NULL,
  offset = NULL,
  dispersion = NULL,
  level = 0.95,
  hessian = c("dense", "operator"),
  information = c("relaxed", "lasso"),
  envir = parent.frame()
)
```

Arguments

`fit`	A fitted `glmnet` object (gaussian, binomial or poisson).
`x, y`	The full data. With `selection_rows`, `fit` must have been fitted on `x[selection_rows, ]` and `y[selection_rows]`.
`s`	A value in `fit$lambda` (default: the last one).
`selection_rows`	Rows (indices or logical) the fit used for selection; `NULL` (default) for all of them, i.e. no randomization.
`weights, offset`	Full-data observation weights and offset; default those passed to `glmnet` (which must then be for the full data).
`dispersion`	Dispersion; default 1 for binomial and poisson and, for gaussian, the residual variance of the least squares fit on all the variables (requires `nrow(x) > ncol(x)`; supply it otherwise).
`level`	Confidence level.
`hessian, information`	As in `glmnet_problem`.
`envir`	Environment in which to evaluate the arguments of `fit$call`.

Value

A `LassoInference` object; `$summary()` gives the selective intervals and p-values. Indices are 0-based, with 0 the intercept when the fit has one.

References

Lee, J. D., Sun, D. L., Sun, Y. and Taylor, J. E. (2016). Exact post-selection inference, with application to the lasso. *The Annals of Statistics*, 44(3), 907–927. doi:10.1214/15AOS1371

Panigrahi, S., Fry, K. and Taylor, J. (2024). Exact selective inference with randomization. *Biometrika*, 111(4), 1109–1127. doi:10.1093/biomet/asae019

Examples

```
set.seed(1)
n <- 200; p <- 10
x <- matrix(rnorm(n * p), n, p)
y <- rbinom(n, 1, plogis(x[, 1] - x[, 2]))
fit <- glmnet::glmnet(x, y, family = "binomial", lambda = c(0.2, 0.1, 0.05),
                      control = list(thresh = 1e-14))
```

```
glmnet_inference(fit, x, y, level = 0.9)$summary()

# carving: select on half of the data, inference with all of it
rows <- sample(n, n / 2)
fit <- glmnet::glmnet(x[rows, ], y[rows], family = "binomial",
                      lambda = c(0.2, 0.1, 0.05), control = list(thresh = 1e-14))
glmnet_inference(fit, x, y, selection_rows = rows, level = 0.9)$summary()
```

---

glmnet_problem *The bounded LASSO problem glmnet solved*

Description

Gradient and Hessian of the smooth part at the solution, penalty weights and limits, in the original coordinates: the inputs of `LassoInference`. `glmnet_problem_from_fit` reads the settings from a fitted `glmnet` object.

Usage

```
glmnet_problem(
  x,
  y,
  coef,
  a0 = 0,
  lambda,
  family = "gaussian",
  weights = NULL,
  offset = NULL,
  alpha = 1,
  penalty.factor = rep(1, ncol(x)),
  exclude = NULL,
  lower.limits = -Inf,
  upper.limits = Inf,
  standardize = TRUE,
  intercept = TRUE,
  scaling = NULL,
  y_scale = NULL,
  hessian = c("dense", "operator"),
  information = c("relaxed", "lasso")
)

glmnet_problem_from_fit(
  fit,
  x,
  y,
  s = NULL,
  weights = NULL,
```

```
  offset = NULL,
  hessian = c("dense", "operator"),
  information = c("relaxed", "lasso"),
  envir = parent.frame()
)
```

Arguments

x	Design matrix.
y	Response (0/1 or a two-level factor for binomial).
coef	Fitted coefficients (original scale).
a0	Fitted intercept.
lambda	The lambda of the fit.
family	"gaussian", "binomial" or "poisson".
weights, offset	Observation weights and offset.
alpha, penalty.factor, exclude, lower.limits, upper.limits, standardize, intercept	As passed to glmnet (exclude 1-based).
scaling	Column scales; default glmnet's convention (glmnet_scaling).
y_scale	The ridge term is divided by y_scale; default glmnet's convention (scale of y for gaussian, 1 otherwise).
hessian	"dense" (matrix) or "operator" (matrix-free LinearOp).
information	Where Q_hat is evaluated: "relaxed" (default), a one-step relaxed fit, one Newton step for the unpenalized loss on the selected coordinates (the intercept and the nonzero coefficients) from the LASSO solution, beta_E - H_EE^{-1} grad_E with the Hessian H at the LASSO solution (limits and the ridge term are ignored); or "lasso", the LASSO solution. The LASSO solution is shrunk, so for binomial and poisson its information overstates the information at the truth and Q_hat / n understates Var(Z); the relaxed fit avoids this. The two agree for gaussian. G_hat is always at the LASSO solution.
fit	A fitted glmnet object.
s	A value in fit$lambda (default: the last one); refit at other values.
envir	Environment in which to evaluate the arguments of fit$call.

Value

A list with beta_hat, G_hat, Q_hat, D, L, U and intercept; coordinates are (intercept, coef) when intercept = TRUE.

Examples

```
set.seed(1)
x <- matrix(rnorm(100 * 5), 100, 5)
y <- rbinom(100, 1, plogis(x[, 1]))
fit <- glmnet::glmnet(x, y, family = "binomial", lambda = c(0.1, 0.05),
```

```
                        control = list(thresh = 1e-14))
prob <- glmnet_problem_from_fit(fit, x, y)
prob$beta_hat
max(kkt_violation(prob$beta_hat, prob$G_hat, prob$D, prob$L, prob$U))
```

`glmnet_scaling` *glmnet's column scales*

Description

Weighted standard deviations of the columns (with or without an intercept), 1 if not standardizing.

Usage

```
glmnet_scaling(x, weights = rep(1, nrow(x)), standardize = TRUE)
```

Arguments

`x`	Design matrix.
`weights`	Observation weights.
`standardize`	As passed to `glmnet`.

Value

The column scales, a vector of length `ncol(x)`.

Examples

```
x <- matrix(rnorm(40), 10, 4)
glmnet_scaling(x)
```

`inactive_summary` *Selective inference for inactive variables*

Description

Selective inference for inactive variables

Usage

```
inactive_summary(
  selection,
  estimand = "coef",
  basis = "refit",
  variables = NULL,
  level = NULL,
  null_value = 0,
  Q_diag = NULL
)
```

Arguments

selection	A LassoInference or ScreenedSelection.
estimand	"coef" (coefficient of j in model E u {j}) or "score".
basis	"refit" or "lasso".
variables	Inactive variables (0-based); defaults to screened for a ScreenedSelection and all inactive variables otherwise.
level	Confidence level (default: the selection's level).
null_value	Null value for the p-values.
Q_diag	Optional diagonal of Q_hat.

Value

A data frame as from estimand_summary, with integer index.

Examples

```
set.seed(1)
n <- 100; p <- 10
x <- matrix(rnorm(n * p), n, p)
y <- x[, 1] - x[, 2] + rnorm(n)
fit <- glmnet::glmnet(x, y, lambda = c(0.3, 0.2), control = list(thresh = 1e-14))
li <- glmnet_inference(fit, x, y, level = 0.9)   # index 0 is the intercept
head(inactive_summary(li))
```

---

kkt_violation *KKT violation of the bounded LASSO*

Description

Coordinatewise violation of 0 in G + D d|beta| + N_[L, U](beta); 0 at the exact solution. Coordinates with L = U are fixed and skipped.

Usage

```
kkt_violation(beta, G, D, L, U, tol = 1e-08)
```

Arguments

beta, G, D, L, U	Solution, gradient of the smooth part, penalty weights and limits (e.g. from glmnet_problem).
tol	Tolerance for being at 0 or at a limit.

Value

The violation for each coordinate, nonnegative.

Examples

```
# beta = (1, 0) solves min (beta - c(2, 0.5))^2 / 2 + |beta|_1
beta <- c(1, 0)
kkt_violation(beta, G = beta - c(2, 0.5), D = c(1, 1), L = c(-Inf, -Inf), U = c(Inf, Inf))
```

LassoInference *LassoInference Class*

Description

Computes post-selection inference for the lasso. This class mirrors the Python ‘lassoinf.LassoInference‘ dataclass, providing identical parameters and functional parity.

Public fields

`beta_hat` The lasso solution from the randomized objective.

`G_hat` The gradient at beta_hat of the unpenalized loss.

`Q_hat` The Hessian / design matrix crossprod: a matrix or a `LinearOp`.

`D` The penalty weights.

`L` Lower bounds.

`U` Upper bounds.

`Z_full` The unpenalized score on the full data.

`Sigma` Covariance of Z_full: a matrix or a `LinearOp`.

`Sigma_noise` Covariance of the randomization noise: a matrix, a `LinearOp` or `NULL`.

`scalar_noise` Scalar controlling variance of noise if Sigma_noise is not provided.

`A` Affine constraints, a matrix-free `LinearOp` (from C++ backend).

`b` Affine constraint vector (from C++ backend).

`E` The active set (selected indices).

`E_c` The inactive set.

`s_E` Signs of the active set coefficients.

`v_Ec` Values at the bounds for the inactive set.

`W` Inverse of Q_hat restricted to the active set.

`score` Inactive score operator `U_{-E}` (a `LinearOp`).

`Z_noisy` The score used for selection.

`level` Confidence level.

`si` AffineConstraints C++ wrapper instance.

`intervals` Selective inference intervals.

`splitting` Data splitting intervals.

`naive` Naive intervals.

`contrasts` The contrast vectors for inference.

`_naive` Data frame storing naive results.

`_splitting` Data frame storing data splitting results.

Methods

Public methods:

- `LassoInference$new()`
- `LassoInference$summary()`
- `LassoInference$clone()`

`LassoInference$new()`: Initialize the LassoInference object.

Usage:

```
LassoInference$new(
  beta_hat,
  G_hat,
  Q_hat,
  D,
  L = NULL,
  U = NULL,
  Z_full,
  Sigma,
  Sigma_noise = NULL,
  scalar_noise = NaN,
  tol = 1e-06,
  level = 0.95
)
```

Arguments:

`beta_hat` The lasso solution.

`G_hat` The gradient at beta_hat.

`Q_hat` The Hessian / design matrix crossprod.

`D` The penalty weights.

`L` Lower bounds for beta. Default is `NULL` (equivalent to `-Inf`).

`U` Upper bounds for beta. Default is `NULL` (equivalent to `Inf`).

`Z_full` The unpenalized score.

`Sigma` Covariance of Z_full.

`Sigma_noise` Covariance of Z_noisy. Default is `NULL`.

`scalar_noise` If `Sigma_noise` is `NULL`, the noise covariance is `scalar_noise * Sigma`. `scalar_noise = 0` is no randomization (the polyhedral lemma, with `Z_full` equal to the selection score `-G_hat + Q_hat beta_hat`); as in the Python package, values below 0.001 are raised to 0.001 for numerical stability. Default is `NaN`.

`tol` Tolerance for active set and KKT conditions. Default is `1e-6`.

`level` Confidence level. Default is 0.95.

`LassoInference$summary()`: Summary of post-selection inference results.

Usage:

```
LassoInference$summary()
```

Returns: A data frame containing the parameter indices, estimates, selective confidence intervals, and selective p-values.

`LassoInference$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LassoInference$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Tian, X. and Taylor, J. (2018). Selective inference with a randomized response. *The Annals of Statistics*, 46(2), 679–710. doi:10.1214/17AOS1564

Panigrahi, S., Fry, K. and Taylor, J. (2024). Exact selective inference with randomization. *Biometrika*, 111(4), 1109–1127. doi:10.1093/biomet/asae019

Examples

```
set.seed(1)
n <- 100; p <- 10
x <- matrix(rnorm(n * p), n, p)
y <- x[, 1] - x[, 2] + rnorm(n)
# randomized gaussian LASSO 1/2 ||y + omega - x b||^2 + lam ||b||_1, sigma^2 = 1,
# Var(omega) = kappa
kappa <- 0.5
y_noisy <- y + rnorm(n, sd = sqrt(kappa))
lam <- 2 * sqrt(n)
fit <- glmnet::glmnet(x, y_noisy, lambda = lam / n, intercept = FALSE,
                      standardize = FALSE, control = list(thresh = 1e-14))
b <- as.numeric(coef(fit))[-1]
Q <- crossprod(x)
G <- drop(crossprod(x, x %*% b - y_noisy))
li <- LassoInference$new(b, G, Q, D = rep(lam, p), Z_full = drop(crossprod(x, y)),
```

```
                          Sigma = Q, scalar_noise = kappa, level = 0.9)
li$summary()
# for a glmnet fit, glmnet_inference() builds these arguments
```

ScreenedSelection *ScreenedSelection Class*

Description

LASSO selection followed by screening of inactive variables on the gradient at the LASSO solution, `G = grad(beta_hat)_{-E}`: keep variables with `|G_j| > threshold`, or the `top_k` by `|G_j|`. Inference conditions on the kept variables and the signs of their gradients; for `top_k` with `conditioning = "first_dropped"` (default) also on the largest dropped variable and its sign (O(p) constraints), with `"exact"` on all (kept, dropped) pairs. Can be passed wherever a `LassoInference` is accepted by the estimand functions.

Public fields

`lasso_inference` The underlying `LassoInference`.

`threshold` Screening threshold (or `NULL`).

`top_k` Number of variables kept (or `NULL`).

`conditioning` Conditioning for `top_k`.

`screened` Kept inactive variables (0-based), by decreasing `|G_j|`.

`screened_signs` Signs of their gradients.

`first_dropped` Largest dropped variable (0-based) for `"first_dropped"`, else `NULL`.

`first_dropped_sign` Sign of its gradient.

`A` Constraints of the combined selection event (`LinearOp`).

`b` Constraint offsets.

`A_screen` Screening constraints alone (`LinearOp`).

`b_screen` Screening constraint offsets.

`si` The `AffineConstraints` of the LASSO inference.

`level` Confidence level.

Methods

Public methods:

- `ScreenedSelection$new()`
- `ScreenedSelection$clone()`

`ScreenedSelection$new()`**:** Screen inactive variables.

Usage:

```
ScreenedSelection$new(
  lasso_inference,
  threshold = NULL,
  top_k = NULL,
  conditioning = "first_dropped"
)
```

Arguments:

`lasso_inference` A `LassoInference`.

`threshold` Keep `|G_j| > threshold`.

`top_k` Keep the `top_k` largest `|G_j|`.

`conditioning` `"first_dropped"` or `"exact"` (for `top_k`).

`ScreenedSelection$clone()`: The objects of this class are cloneable with this method.

Usage:

```
ScreenedSelection$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
set.seed(1)
n <- 100; p <- 10
x <- matrix(rnorm(n * p), n, p)
y <- x[, 1] - x[, 2] + rnorm(n)
fit <- glmnet::glmnet(x, y, lambda = c(0.3, 0.2), control = list(thresh = 1e-14))
li <- glmnet_inference(fit, x, y, level = 0.9)   # index 0 is the intercept
screen <- ScreenedSelection$new(li, top_k = 3)
screen$screened
inactive_summary(screen)
```

`SelectionCoordinates` *SelectionCoordinates Class*

Description

The coordinates in which the selection constraints are expressed. With `basis = "refit"`: `bar_beta_E = W Z_E` and the inactive scores `U_{-E} = Z_{-E} - Q_{-E,E} W Z_E` (in OLS, `t(X_{-E}) (I - P_E) y`). With `basis = "lasso"`: the LASSO solution `beta_hat_E` and the negative gradient at it, `-grad_{-E}`; these differ from the refit coordinates by constants, so they give the same contrast with a shifted value. Everything is computed with `Q_hat` matrix-vector products.

Public fields

`cpp` The C++ object.

`E` Active set (0-based).

`E_c` Inactive set (0-based).

Methods

Public methods:

- SelectionCoordinates$new()
- SelectionCoordinates$contrast()
- SelectionCoordinates$estimand()
- SelectionCoordinates$inactive_score()
- SelectionCoordinates$inactive_coef()
- SelectionCoordinates$inactive_S()
- SelectionCoordinates$refit_coef()
- SelectionCoordinates$refit_score()
- SelectionCoordinates$lasso_offsets()
- SelectionCoordinates$score_operator()
- SelectionCoordinates$clone()

`SelectionCoordinates$new()`: Create the coordinates for a selection.

Usage:

```
SelectionCoordinates$new(selection, Q_diag = NULL)
```

Arguments:

`selection` A `LassoInference` or `ScreenedSelection`.

`Q_diag` Optional diagonal of `Q_hat`; lets `inactive_S` use `|E|` matvecs for many variables. Taken from `Q_hat` when known.

`SelectionCoordinates$contrast()`: Contrast `eta` with `sum(eta * Z) = sum(a_E * bar_beta_E) + sum(a_Ec * U_{-E})`.

Usage:

```
SelectionCoordinates$contrast(a_E = NULL, a_Ec = NULL)
```

Arguments:

`a_E` Coefficients on `bar_beta_E` (or `NULL`).

`a_Ec` Coefficients on `U_{-E}` (or `NULL`).

`SelectionCoordinates$estimand()`: Estimand linear in the selection coordinates.

Usage:

```
SelectionCoordinates$estimand(a_E = NULL, a_Ec = NULL, basis = "refit")
```

Arguments:

`a_E` Coefficients on the active coordinates.

`a_Ec` Coefficients on the inactive scores.

`basis` `"refit"` or `"lasso"`.

`SelectionCoordinates$inactive_score()`: Score of inactive variable `j`.

Usage:

```
SelectionCoordinates$inactive_score(j, basis = "refit")
```

Arguments:

`j` Inactive variable (0-based).
`basis` `"refit"` or `"lasso"`.

`SelectionCoordinates$inactive_coef()`: Coefficient of inactive variable `j` in the model `E u {j}`.

Usage:

```
SelectionCoordinates$inactive_coef(j, basis = "refit")
```

Arguments:

`j` Inactive variable (0-based).
`basis` `"refit"` or `"lasso"`.

`SelectionCoordinates$inactive_S()`: `S_jj = Q_jj - Q_{j,E} W Q_{E,j}` for inactive variables.

Usage:

```
SelectionCoordinates$inactive_S(variables)
```

Arguments:

`variables` Inactive variables (0-based).

`SelectionCoordinates$refit_coef()`: `bar_beta_E` for a score vector.

Usage:

```
SelectionCoordinates$refit_coef(Z)
```

Arguments:

`Z` A score vector.

`SelectionCoordinates$refit_score()`: `U_{-E}` for a score vector.

Usage:

```
SelectionCoordinates$refit_score(Z)
```

Arguments:

`Z` A score vector.

`SelectionCoordinates$lasso_offsets()`: Offsets of the lasso basis: active coefficients and inactive scores.

Usage:

```
SelectionCoordinates$lasso_offsets()
```

`SelectionCoordinates$score_operator()`: The inactive score operator.

Usage:

```
SelectionCoordinates$score_operator()
```

`SelectionCoordinates$clone()`: The objects of this class are cloneable with this method.

Usage:

```
SelectionCoordinates$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
set.seed(1)
n <- 100; p <- 10
x <- matrix(rnorm(n * p), n, p)
y <- x[, 1] - x[, 2] + rnorm(n)
fit <- glmnet::glmnet(x, y, lambda = c(0.3, 0.2), control = list(thresh = 1e-14))
li <- glmnet_inference(fit, x, y, level = 0.9)   # index 0 is the intercept
coords <- SelectionCoordinates$new(li)
coords$E_c
# coefficient of the first inactive variable in the model E u {j}
res <- contrast_inference(li, coords$inactive_coef(coords$E_c[1]))
res[c("lower_conf", "upper_conf", "p_value")]
```

---

truncated_normal_inference

Truncated normal inference from the polyhedral lemma

Description

Equal-tailed interval and two-sided p-value for `H0: theta = null_value`.

Usage

```
truncated_normal_inference(
  contrast,
  variance,
  A,
  b,
  level = 0.95,
  null_value = 0
)
```

Arguments

`contrast`	An `AffineConstraintsContrast`.
`variance`	Variance of the estimator.
`A`	Constraint matrix or `LinearOp`.
`b`	Constraint offsets.
`level`	Confidence level.
`null_value`	Null value of the estimand.

Value

Named vector `c(lower, upper, p_value)`.

Examples

```
set.seed(1)
n <- 100; p <- 10
x <- matrix(rnorm(n * p), n, p)
y <- x[, 1] - x[, 2] + rnorm(n)
fit <- glmnet::glmnet(x, y, lambda = c(0.3, 0.2), control = list(thresh = 1e-14))
li <- glmnet_inference(fit, x, y, level = 0.9)   # index 0 is the intercept
v <- li$contrasts[["1"]]   # the coefficient of x1 in the selected model
contrast <- li$si$compute_contrast(v)
truncated_normal_inference(contrast, contrast$naive_variance, li$A, li$b, level = 0.9)
```

TruncBivariateNormal *TruncBivariateNormal Class*

Description

Evaluates exact bounds and inference parameters for the selectively truncated bivariate normal distribution. Follows the bivariate normal computations of Liu (2023).

Public fields

`a_coeff` Internal coefficient.

`b_coeff` Internal coefficient.

`L` Lower bound of the truncation interval.

`U` Upper bound of the truncation interval.

`sig_omega` Standard deviation of the noise component.

`sig_x` Standard deviation of the target statistic.

Methods

Public methods:

- `TruncBivariateNormal$new()`
- `TruncBivariateNormal$.get_stats()`
- `TruncBivariateNormal$ccdf()`
- `TruncBivariateNormal$cdf()`
- `TruncBivariateNormal$E()`
- `TruncBivariateNormal$Var()`
- `TruncBivariateNormal$equal_tailed_interval()`
- `TruncBivariateNormal$clone()`

`TruncBivariateNormal$new()`**:** Initialize the TruncBivariateNormal object.

Usage:

```
TruncBivariateNormal$new(
  a_coeff,
  b_coeff,
  L,
  U,
  sig_omega,
  sig_x = 1,
  theta = 0
)
```

Arguments:

`a_coeff` The 'a' coefficient for the conditional statistic.

`b_coeff` The 'b' coefficient for the noise statistic.

`L` Lower bound of the condition.

`U` Upper bound of the condition.

`sig_omega` Noise standard deviation.

`sig_x` Target statistic standard deviation. Default is 1.0.

`theta` True underlying parameter. Default is 0.0.

`TruncBivariateNormal$.get_stats()`: Get the computed statistics given the parameter.

Usage:

```
TruncBivariateNormal$.get_stats(theta, x = NULL)
```

Arguments:

`theta` True parameter value.

`x` Observed value. Default is NULL.

Returns: A list containing the computed stats.

`TruncBivariateNormal$ccdf()`: Complementary CDF.

Usage:

```
TruncBivariateNormal$ccdf(theta, x = NULL, gamma = 0)
```

Arguments:

`theta` Parameter.

`x` Observed value.

`gamma` Ignored.

Returns: The complementary CDF value.

`TruncBivariateNormal$cdf()`: Cumulative Distribution Function.

Usage:

```
TruncBivariateNormal$cdf(theta, x = NULL, gamma = 1)
```

Arguments:

`theta` Parameter.

`x` Observed value.

`gamma` Ignored.

Returns: The CDF value.

`TruncBivariateNormal$E()`: Expected value.

Usage:

```
TruncBivariateNormal$E(theta, func)
```

Arguments:

`theta` Parameter.

`func` A function returning identity or x^2.

Returns: Expected value.

`TruncBivariateNormal$Var()`: Variance.

Usage:

```
TruncBivariateNormal$Var(theta, func)
```

Arguments:

`theta` Parameter.

`func` Identity function.

Returns: The conditional variance.

`TruncBivariateNormal$equal_tailed_interval()`: Find the equal tailed confidence interval by inverting the CDF.

Usage:

```
TruncBivariateNormal$equal_tailed_interval(observed, alpha = 0.05, tol = 1e-06)
```

Arguments:

`observed` The observed value.

`alpha` Type-I error rate. Default is 0.05.

`tol` Tolerance for uniroot.

Returns: A length 2 vector containing the lower and upper bounds.

`TruncBivariateNormal$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TruncBivariateNormal$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Liu, S. (2023). An exact sampler for inference after polyhedral model selection. *arXiv preprint*. doi:10.48550/arXiv.2308.10346

Examples

```
tbn <- TruncBivariateNormal$new(a_coeff = 1, b_coeff = 1, L = 1, U = Inf, sig_omega = 1)
tbn$cdf(theta = 0, x = 1.5)
tbn$equal_tailed_interval(1.5, alpha = 0.1)
```

Index

AffineConstraints, 2

contrast_estimand, 3
contrast_inference, 4
covariance_estimand
(contrast_estimand), 3
custom_estimand_inference, 4

dense_operator, 6
DiscreteFamily (AffineConstraints), 2

estimand_summary, 6
estimand_value (contrast_estimand), 3

glmnet_inference, 7
glmnet_problem, *8*, 9
glmnet_problem_from_fit
(glmnet_problem), 9
glmnet_scaling, 11

inactive_summary, 11

kkt_violation, 12

LassoInference, *8*, 13

ScreenedSelection, 16
SelectionCoordinates, 17

truncated_normal_inference, 20
TruncatedGaussian (AffineConstraints), 2
TruncBivariateNormal, 21

WeightedGaussianFamily
(AffineConstraints), 2

xtvx_operator (dense_operator), 6