Package ‘cirls’

July 22, 2025

Title Constrained Iteratively Reweighted Least Squares

Version 0.3.1

Description Routines to fit generalized linear models with constrained coefficients, along with inference on the coefficients. Designed to be used in conjunction with the base glm() function.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.1

Imports quadprog, osqp, coneproj, TruncatedNormal, stats

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

URL https://github.com/PierreMasselot/cirls

NeedsCompilation no

Author Pierre Masselot [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-7326-1290>), Antonio Gasparrini [aut] (ORCID: <https://orcid.org/0000-0002-2271-3568>)

Maintainer Pierre Masselot <pierre.masselot@lshtm.ac.uk>

Repository CRAN

Date/Publication 2024-09-12 11:20:06 UTC

Contents

check_cmat *Check constraint matrix irreducibility*

Description

Checks a constraint matrix does not contains redundant rows

Usage

```
check_cmat(Cmat)
```

Arguments

`Cmat`	A constraint matrix as passed to `cirls.fit()`

Details

The user typically doesn't need to use `check_cmat` as it is internally called by `cirls.control()`. However, it might be useful to undertsand if `Cmat` can be reduced for inference purpose. See the note in `confint.cirls()`.

A constraint matrix is irreducible if no row can be expressed as a *positive* linear combination of the other rows. When it happens, it means the constraint is actually implicitly included in other constraints in the matrix and can be dropped. Note that this a less restrictive condition than the constraint matrix having full row rank (see some examples).

The function starts by checking if some constraints are redundant and, if so, checks if they underline equality constraints. In the latter case, the constraint matrix can be reduced by expressing these constraints as a single equality constraint with identical lower and upper bounds (see `cirls.fit()`).

Value

A list with two elements:

`redundant`	Vector of indices of redundant constraints
`equality`	Indicates which constraints are part of an underlying equality constraint

References

Meyer, M.C., 1999. An extension of the mixed primal–dual bases algorithm to the case of more constraints than dimensions. *Journal of Statistical Planning and Inference* **81**, 13–31. doi:10.1016/S03783758(99)000257

See Also

`confint.cirls()`

Examples

```
###################################################
# Example of reducible matrix

# Constraints: successive coefficients should increase and be convex
p <- 5
cmatic <- rbind(diff(diag(p)), diff(diag(p), diff = 2))

# Checking indicates that constraints 2 to 4 are redundant.
# Intuitively, if the first two coefficients increase,
# then convexity forces the rest to increase
check_cmat(cmatic)

# Check without contraints
check_cmat(cmatic[-(2:4),])

###################################################
# Example of irreducible matrix

# Constraints: coefficients form an S-shape
p <- 4
cmats <- rbind(
  diag(p)[1,], # positive
  diff(diag(p))[c(1, p - 1),], # Increasing at both end
  diff(diag(p), diff = 2)[1:(p/2 - 1),], # First half convex
  -diff(diag(p), diff = 2)[(p/2):(p-2),] # second half concave
)

# Note, this matrix is not of full row rank
qr(t(cmats))$rank
all.equal(cmats[2,] + cmats[4,] - cmats[5,], cmats[3,])

# However, it is irreducible: all constraints are necessary
check_cmat(cmats)

###################################################
# Example of underlying equality constraint

# Contraint: Parameters sum is >= 0 and sum is <= 0
cmateq <- rbind(rep(1, 3), rep(-1, 3))

# Checking indicates that both constraints imply equality constraint (sum == 0)
check_cmat(cmateq)
```

---

cirls.control *Parameters controlling CIRLS fitting*

Description

Internal function controlling the glm fit with linear constraints. Typically only used internally by cirls.fit, but may be used to construct a control argument.

Usage

```
cirls.control(epsilon = 1e-08, maxit = 25, trace = FALSE, Cmat = NULL,
  lb = 0L, ub = Inf, qp_solver = "osqp", qp_pars = list())
```

Arguments

`epsilon`	Positive convergence tolerance. The algorithm converges when the relative change in deviance is smaller than `epsilon`.
`maxit`	Integer giving the maximal number of CIRLS iterations.
`trace`	Logical indicating if output should be produced for each iteration.
`Cmat`	Constraint matrix specifying the linear constraints applied to coefficients. Can also be provided as a list of matrices for specific terms.
`lb`, `ub`	Lower and upper bound vectors for the linear constraints. Identical values in `lb` and `ub` identify equality constraints. Recycled if length is different than the number of constraints defined by `Cmat`.
`qp_solver`	The quadratic programming solver. One of `"osqp"`, `"quadprog"` or `"coneproj"`.
`qp_pars`	List of parameters specific to the quadratic programming solver. See respective packages help.

Details

The `control` argument of glm is by default passed to the `control` argument of cirls.fit, which uses its elements as arguments for cirls.control: the latter provides defaults and sanity checking. The control parameters can alternatively be passed through the `...` argument of glm. See glm.control for details on general GLM fitting control, and cirls.fit for details on arguments specific to constrained GLMs.

Value

A named list containing arguments to be used in cirls.fit.

See Also

the main function cirls.fit, and glm.control.

Examples

```
# Simulate predictors and response with some negative coefficients
set.seed(111)
n <- 100
p <- 10
betas <- rep_len(c(1, -1), p)
x <- matrix(rnorm(n * p), nrow = n)
y <- x %*% betas + rnorm(n)

# Define constraint matrix (includes intercept)
# By default, bounds are 0 and +Inf
Cmat <- cbind(0, diag(p))
```

```
# Fit GLM by CIRLS
res1 <- glm(y ~ x, method = cirls.fit, Cmat = Cmat)
coef(res1)

# Same as passing Cmat through the control argument
res2 <- glm(y ~ x, method = cirls.fit, control = list(Cmat = Cmat))
identical(coef(res1), coef(res2))
```

`cirls.fit` *Constrained Iteratively Reweighted Least-Squares*

Description

Fits a generalized linear model with linear constraints on the coefficients through a Constrained Iteratively Reweighted Least-Squares (CIRLS) algorithm. This function is the constrained counterpart to glm.fit and is meant to be called by glm through its `method` argument. See details for the main differences.

Usage

```
cirls.fit(x, y, weights = rep.int(1, nobs), start = NULL,
  etastart = NULL, mustart = NULL, offset = rep.int(0, nobs),
  family = stats::gaussian(), control = list(), intercept = TRUE,
  singular.ok = TRUE)
```

Arguments

`x, y`	`x` is a design matrix and `y` is a vector of response observations. Usually internally computed by glm.
`weights`	An optional vector of observation weights.
`start`	Starting values for the parameters in the linear predictor.
`etastart`	Starting values for the linear predictor.
`mustart`	Starting values for the vector or means.
`offset`	An optional vector specifying a known component in the model. See model.offset.
`family`	The result of a call to a family function, describing the error distribution and link function of the model. See family for details of available family functions.
`control`	A list of parameters controlling the fitting process. See details and cirls.control.
`intercept`	Logical. Should an intercept be included in the null model?
`singular.ok`	Logical. If `FALSE`, the function returns an error for singular fits.

Details

This function is a plug-in for glm and works similarly to glm.fit. In addition to the parameters already available in glm.fit, `cirls.fit` allows the specification of a constraint matrix `Cmat` with bound vectors `lb` and `ub` on the regression coefficients. These additional parameters can be passed through the `control` list or through `...` in glm.

The CIRLS algorithm is a modification of the classical IRLS algorithm in which each update of the regression coefficients is performed by a quadratic program (QP), ensuring the update stays within the feasible region defined by `Cmat`, `lb` and `ub`. More specifically, this feasible region is defined as

`lb <= Cmat %*% coefficients <= ub`

where `coefficients` is the coefficient vector returned by the model. This specification allows for any linear constraint, including equality ones.

Specifying `Cmat`, `lb` and `ub`:

`Cmat` is a matrix that defines the linear constraints. If provided directly as a matrix, the number of columns in `Cmat` must match the number of coefficients estimated by glm. This includes all variables that are not involved in any constraint potential expansion such as factors or splines for instance, as well as the intercept. Columns not involved in any constraint will be filled by 0s.

Alternatively, it may be more convenient to pass `Cmat` as a list of constraint matrices for specific terms. This is advantageous if a single term should be constrained in a model containing many terms. If provided as a list, `Cmat` is internally expanded to create the full constraint matrix. See examples of constraint matrices below.

`lb` and `ub` are vectors defining the bounds of the constraints. By default they are set to `0` and `Inf`, meaning that the linear combinations defined by `Cmat` should be positive, but any bounds are possible. When some elements of `lb` and `ub` are identical, they define equality constraints. Setting `lb = -Inf` and `ub = Inf` disable the constraints.

Quadratic programming solvers:

The function cirls.fit relies on a quadratic programming solver. Several solver are currently available.

- `"osqp"` (the default) solves the quadratic program via the Alternating Direction Method of Multipliers (ADMM). Internally it calls the function solve_osqp.
- `"quadprog"` performs a dual algorithm to solve the quadratic program. It relies on the function solve.QP.
- `"coneproj"` solves the quadratic program by a cone projection method. It relies on the function qprog.

Each solver has specific parameters that can be controlled through the argument `qp_pars`. Sensible defaults are set within cirls.control and the user typically doesn't need to provide custom parameters.

Value

A `cirls` object inheriting from the class `glm`. At the moment, two non-standard methods specific to `cirls` objects are available: vcov.cirls to obtain the coefficients variance-covariance matrix and confint.cirls to obtain confidence intervals. These custom methods account for the reduced degrees of freedom resulting from the constraints, see vcov.cirls and confint.cirls. Any method for `glm` objects can be used, including the generic coef or summary for instance.

An object of class `cirls` includes all components from glm objects, with the addition of:

`active.cons` vector of indices of the active constraints in the fitted model.

`inner.iter` number of iterations performed by the last call to the QP solver.

`Cmat, lb, ub` the (expanded) constraint matrix, and lower and upper bound vectors.

References

Goldfarb, D., Idnani, A., 1983. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming* **27**, 1–33. doi:10.1007/BF02591962

Meyer, M.C., 2013. A Simple New Algorithm for Quadratic Programming with Applications in Statistics. *Communications in Statistics - Simulation and Computation* **42**, 1126–1139. doi:10.1080/03610918.2012.659820

Stellato, B., Banjac, G., Goulart, P., Bemporad, A., Boyd, S., 2020. OSQP: an operator splitting solver for quadratic programs. *Math. Prog. Comp.* **12**, 637–672. doi:10.1007/s12532020001792

See Also

vcov.cirls, confint.cirls for methods specific to `cirls` objects. cirls.control for fitting parameters specific to cirls.fit. glm for details on `glm` objects.

Examples

```
####################################################
# Simple non-negative least squares

# Simulate predictors and response with some negative coefficients
set.seed(111)
n <- 100
p <- 10
betas <- rep_len(c(1, -1), p)
x <- matrix(rnorm(n * p), nrow = n)
y <- x %*% betas + rnorm(n)

# Define constraint matrix (includes intercept)
# By default, bounds are 0 and +Inf
Cmat <- cbind(0, diag(p))

# Fit GLM by CIRLS
res1 <- glm(y ~ x, method = cirls.fit, Cmat = Cmat)
coef(res1)

# Same as passing Cmat through the control argument
res2 <- glm(y ~ x, method = cirls.fit, control = list(Cmat = Cmat))
identical(coef(res1), coef(res2))

####################################################
# Increasing coefficients

# Generate two group of variables: an isotonic one and an unconstrained one
set.seed(222)
p1 <- 5; p2 <- 3
```

```
x1 <- matrix(rnorm(100 * p1), 100, p1)
x2 <- matrix(rnorm(100 * p2), 100, p2)

# Generate coefficients: those in b1 should be increasing
b1 <- runif(p1) |> sort()
b2 <- runif(p2)

# Generate full data
y <- x1 %*% b1 + x2 %*% b2 + rnorm(100, sd = 2)

#----- Fit model

# Create constraint matrix and expand for intercept and unconstrained variables
Ciso <- diff(diag(p1))
Cmat <- cbind(0, Ciso, matrix(0, nrow(Ciso), p2))

# Fit model
resiso <- glm(y ~ x1 + x2, method = cirls.fit, Cmat = Cmat)
coef(resiso)

# Compare with unconstrained
plot(c(0, b1, b2), pch = 16)
points(coef(resiso), pch = 16, col = 3)
points(coef(glm(y ~ x1 + x2)), col = 2)

#----- More convenient specification

# Cmat can be provided as a list
resiso2 <- glm(y ~ x1 + x2, method = cirls.fit, Cmat = list(x1 = Ciso))

# Internally Cmat is expanded and we obtain the same result
identical(resiso$Cmat, resiso2$Cmat)
identical(coef(resiso), coef(resiso2))

#----- Adding bounds to the constraints
# Difference between coefficients must be above a lower bound and below 1
lb <- 1 / (p1 * 2)
ub <- 1

# Re-fit the model
resiso3 <- glm(y ~ x1 + x2, method = cirls.fit, Cmat = list(x1 = Ciso),
  lb = lb, ub = ub)

# Compare the fit
plot(c(0, b1, b2), pch = 16)
points(coef(resiso), pch = 16, col = 3)
points(coef(glm(y ~ x1 + x2)), col = 2)
points(coef(resiso3), pch = 16, col = 4)
```

coef_simu — *Simulate coefficients, calculate Confidence Intervals and Variance-Covariance Matrix for a* `cirls` *object.*

Description

`confint` computes confidence intervals for one of more parameters in a GLM fitted via cirls.fit. `vcov` compute the variance-covariance matrix of the parameters. Both methods are based on `coef_simu` that simulates coefficients from a Truncated Multivariate Normal distribution. These methods supersede the default confint and vcov methods for `cirls` objects.

Usage

```
coef_simu(object, nsim = 1000)

## S3 method for class 'cirls'
confint(object, parm, level = 0.95, nsim = 1000, ...)

## S3 method for class 'cirls'
vcov(object, nsim = 1000, ...)
```

Arguments

`object`	A fitted `cirls` object.
`nsim`	The number of simulations to consider. Corresponds to `n` in rtmvnorm. See details().
`parm`	A specification of which parameters to compute the confidence intervals for. Either a vector of numbers or a vector of names. If missing, all parameters are considered.
`level`	The confidence level required.
`...`	Further arguments passed to or from other methods. Currently ignored.

Details

These functions are custom methods for cirls objects to supersede the default methods used for glm objects.

Both methods rely on the fact that $C\hat{\beta}$ (with C the constraint matrix) follows a *Truncated Multivariate Normal* distribution

$$C\hat{\beta} \sim TMVN(C\beta, CVC^T), l, u$$

where TMVN represents a truncated Multivariate Normal distribution. C is the constraint matrix (`object$control$Cmat`) with bound l and u, while V is the unconstrained variance-covariance matrix (such as returned by `vcov.glm`).

`coef_simu` simulates from the TMVN above and transforms back the realisations into the coefficients space. These realisations are then used by the `confint` and `vcov` methods which compute empirical quantiles and variance-covariance matrix, respectively. `coef_simu` is called internally by `confint` and `vcov` and doesn't need to be used directly, but it can be used to check other summaries of the coefficients distribution.

Value

For `confint`, a two-column matrix with columns giving lower and upper confidence limits for each parameter.

For `vcov`, a matrix of the estimated covariances between the parameter estimates of the model.

For `coef_simu`, a matrix with `nsim` rows containing simulated coefficients.

Note

These methods only work when `Cmat` is of full row rank. If not the case, `Cmat` can be inspected through `check_cmat()`.

References

Geweke, J.F., 1996. Bayesian Inference for Linear Models Subject to Linear Inequality Constraints, in: Lee, J.C., Johnson, W.O., Zellner, A. (Eds.), Modelling and Prediction Honoring Seymour Geisser. *Springer, New York, NY*, pp. 248–263. doi:10.1007/9781461224143_15

Botev, Z.I., 2017, The normal law under linear restrictions: simulation and estimation via minimax tilting, *Journal of the Royal Statistical Society, Series B*, **79** (**1**), pp. 1–24.

See Also

rtmvnorm for the underlying routine to simulate from a TMVN. `check_cmat()` to check if the contraint matrix can be reduced.

Examples

```
####################################################
# Isotonic regression

#----- Perform isotonic regression

# Generate data
set.seed(222)
p1 <- 5; p2 <- 3
x1 <- matrix(rnorm(100 * p1), 100, p1)
x2 <- matrix(rnorm(100 * p2), 100, p2)
b1 <- runif(p1) |> sort()
b2 <- runif(p2)
y <- x1 %*% b1 + x2 %*% b2 + rnorm(100, sd = 2)

# Fit model
Ciso <- diff(diag(p1))
resiso <- glm(y ~ x1 + x2, method = cirls.fit, Cmat = list(x1 = Ciso))

#----- Extract uncertainty

# Extract variance covariance
vcov(resiso)

# Extract confidence intervals
```

```
confint(resiso)

# We can extract the usual unconstrained vcov
summary(resiso)$cov.scaled
all.equal(vcov(resiso), summary(resiso)$cov.scaled)

# Simulate from the distribution of coefficients
sims <- coef_simu(resiso, nsim = 10)

# Check that all simulated coefficient vectors are feasible
apply(resiso$Cmat %*% t(sims) >= resiso$lb, 2, all)
```

Index