

# Package: cirls (via r-universe)

October 15, 2026

**Title** Constrained Iteratively Reweighted Least Squares

**Version** 0.4.0-11

**Description** Fitting and inference functions for generalized linear models with constrained coefficients.

**License** GPL (>= 3)

**Encoding** UTF-8

**LazyData** true

**Roxygen** list(markdown = TRUE, old\_usage = TRUE)

**Imports** quadprog, osqp, coneproj, TruncatedNormal, stats, lpSolve

**Depends** R (>= 4.1.0)

**Suggests** dlnm, splines, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**URL** <https://github.com/PierreMasselot/cirls>

**Config/roxygen2/version** 8.0.0

**Config/pak/sysreqs** make

**Repository** <https://pierremaasselot.r-universe.dev>

**Date/Publication** 2026-09-15 14:09:00 UTC

**RemoteUrl** <https://github.com/pierremaasselot/cirls>

**RemoteRef** HEAD

**RemoteSha** 279407d5e00fce7fb69ec6327c1eaa183b61349d

## Contents

|                                  |    |
|----------------------------------|----|
| cirls-package . . . . .          | 2  |
| boundConstr . . . . .            | 6  |
| boundConstr.bs . . . . .         | 9  |
| boundConstr.crossbasis . . . . . | 10 |
| boundConstr.onebasis . . . . .   | 15 |
| buildCmat . . . . .              | 16 |

|                         |    |
|-------------------------|----|
| cirls.control . . . . . | 20 |
| cirls.fit . . . . .     | 22 |
| edf . . . . .           | 26 |
| fgl . . . . .           | 28 |
| logLik.cirls . . . . .  | 29 |
| london . . . . .        | 30 |
| reduceCons . . . . .    | 32 |
| shapeConstr . . . . .   | 34 |
| simulCoef . . . . .     | 39 |
| uncons . . . . .        | 42 |
| warming . . . . .       | 43 |
| zerosumConstr . . . . . | 44 |

|              |           |
|--------------|-----------|
| <b>Index</b> | <b>46</b> |
|--------------|-----------|

---

|               |                                                                |
|---------------|----------------------------------------------------------------|
| cirls-package | <i>cirls: Constrained Iteratively Reweighted Least Squares</i> |
|---------------|----------------------------------------------------------------|

---

## Description

The package `cirls` provides routines to fit **generalised linear models (GLM) with linear constraints on the coefficients**. These routines implement a general framework to perform a range of constraint-based methods, including common applications such as shape-constrained regression or compositional regression. However, the package allows users to specify *any* custom linear constraint for more niche applications. Functions in the `cirls` package include fitting procedures as well as dedicated methods to perform inference and model selection.

The statistical framework is implemented by embedding constrained optimisation procedures in [glm](#), so that users can flexibly perform constraint-based analyses with simple calls to this standard R function. The user can use the standard `glm` ecosystem on constrained model, augmented with dedicated methods for `cirls` objects.

## Methodological framework

Applying linear constraints to GLMs entails restricting the fitted coefficients  $\beta$  to respect specific linear combinations. These constraints can represent assumptions on the coefficients or enforce identifiability. More formally, the fitted GLM is subject to:

$$\mathbf{l} \leq \mathbf{C}\beta \leq \mathbf{u}$$

where  $\mathbf{C}$  is a matrix encoding the constraints, with each row representing a single constraint and each column representing a regressor in the design matrix of the model.  $\mathbf{l}$  and  $\mathbf{u}$  are lower and upper bounds for the constraints, respectively.

The model is fitted by [constrained iteratively-reweighted least-squares](#) (CIRLS), which extends the classical IRLS algorithm to ensure the constraints specified by  $\mathbf{C}$ ,  $\mathbf{l}$ , and  $\mathbf{u}$  are respected.

Linearly constrained coefficients follow a **truncated multivariate Normal** (TMVN) distribution, *i.e.* a multivariate normal that respects the constraints specified by  $\mathbf{C}$ ,  $\mathbf{l}$  and  $\mathbf{u}$ . Inference is conducted by [simulating](#) from the TMVN distribution of fitted coefficients, and the output can then be used to compute useful summaries like the [variance-covariance matrix](#) and [confidence intervals](#).

## Usage

### Fitting the model:

The `cirls` package has been developed as a **plug-in** to `glm`. The main function is `cirls.fit`, designed to be used within `glm` through the argument `method = "cirls.fit"`. This will fit the GLM using specific CIRLS algorithms instead of the default IRLS algorithm. All arguments of `glm` such as `formula`, `data` and `family` retain the same usage. Additional arguments specific to `cirls` are passed through the ellipsis `...` or the `control` argument (*but not both at the same time*). If any parameter is passed through `control`, then `...` will be ignored. See `cirls.control` for the full list of additional arguments and examples below for how to use additional functions in the package.

### Specifying constraints:

Constraints can be specified through arguments `constr` and/or `Cmat/lb/ub`. The `constr` argument provides a user-friendly way to pass common built-in constraints through a formula and is the recommended way for new users. See the list of available built-in constraints below. Alternatively, the arguments `Cmat/lb/ub` allow more flexibility in passing constraints. These arguments take either a full constraint matrix (`Cmat`) and bound vectors (`lb` and `ub`), or named lists to specify a constrained matrix and bound vectors for specific terms in the model. See the help for `buildCmat` for more specific details.

## Functions included in the package

`cirls.fit` is the workhorse performing the fitting, and is controlled by `cirls.control`. The function returns an object of class `cirls` which inherits from class `glm`. This means that all methods available for usual `glm` objects can also be used on a `cirls` object. This typically includes `coef` and `summary`, for instance. However, some methods are supplanted by new methods that are specific to `cirls` objects. The sections below provides a comprehensive illustration of such specific methods, including an index of the functions.

### Inference:

Inference for `cirls` objects includes computing the **variance-covariance matrix** and **confidence intervals** for constrained (and unconstrained) coefficients. These functions build on `simulCoef`, which generates coefficient vectors based on the TMVN distribution of fitted coefficients, and can be used to compute other summaries.

### Model selection:

Degrees of freedom of a fitted `cirls` object can be computed through the `edf` function. A specific `logLik` method extracts the log-likelihood of the model with the right degrees of freedom in its `df` attribute. This can be used by `AIC` and `BIC` with appropriate degrees of freedom.

### Other functions:

`uncons` refits the model removing some or all of the constraints, acting similarly to the `update` function although focusing on the constraints.

`buildCmat` and `reduceCons` are convenience functions that are mainly used internally in `cirls.fit`. They are not meant to be called directly, but can be useful for advanced users to build and/or check constraint matrices. Specifically, `buildCmat` builds a full constraint matrix from the `constr` and/or `Cmat/lb/ub` arguments and a model frame. `reduceCons` checks whether a constraint matrix can be reduced by detecting redundant and underlying equality constraints. The help pages of both functions also provide technical details on constraints and constraint matrices.

## Index

### Built-in constraints:

These constraints can be used in the `constr` formula as functions applied to specific terms:

- [shape](#): constraining the shape of an association. Typically used with spline bases or factors.
- [zerosum](#): specify that a group of coefficients should sum to zero. Typically used for compositional regression.
- [bound](#): boundary constraint for an association. Typically used with spline bases or factors.

### Methods specific to `cirls` objects:

The following supplant `glm` methods:

- [vcov](#): compute the variance-covariance matrix of coefficients.
- [confint](#): compute confidence intervals for constrained coefficients.
- [logLik](#): extract the log-likelihood of a fitted object, assigning the correct [degrees of freedom](#).

## Datasets

- [fgl](#): Measurement of forensic glass fragments
- [london](#): Daily mortality, temperature, and pollution data in London
- [warming](#): Global temperature anomaly

## Author(s)

**Maintainer:** Pierre Masselot <pierre.masselot@lshtm.ac.uk> ([ORCID](#)) [copyright holder]

Authors:

- Pierre Masselot <pierre.masselot@lshtm.ac.uk> ([ORCID](#)) [copyright holder]
- Antonio Gasparriani <antonio.gasparrini@lshtm.ac.uk> ([ORCID](#))

## References

Masselot, P., Nenon, D., Vanoli, J., Chalabi, Z., Gasparriani, A., 2025. Estimation and inference in generalised linear models with constrained iteratively-reweighted least squares. *ArXiv preprint*. DOI:10.48550/arXiv.2509.18406

## See Also

Useful links:

- <https://github.com/PierreMasselot/cirls>

## Examples

```
#####
# Non-negative coefficient example with the london dataset

library(splines)

### Association between Ozone and mortality
```

```

# Nonnegative constraint on Ozone
model <- glm(death ~ o3, data = london, family = "quasipoisson",
  method = "cirls.fit", cons = ~ shape(o3, "pos"))

# Coefficient and confidence interval
coef(model)[2]
confint(model)[2,]

# Comparing to an unconstrained model
umodel <- uncons(model)
coef(umodel)[2]
confint(umodel)[2,]
#####
# Monotone strata levels with the warming dataset

### Fit the model

# Non-decreasing constraint on decadal strata
model <- glm(anomaly ~ decade, data = warming, method = "cirls.fit",
  cons = ~ shape(decade, "inc"))

# Plot result
plot(anomaly ~ year, data = warming, xlab = "", ylab = "Temperature anomaly")
lines(warming$year, predict(model), col = 2, lwd = 2)

### Extract results

# Coefficients and confidence intervals
betas <- coef(model)
v <- vcov(model)
cis <- confint(model)

# Degrees of freedom: represent number of strata change
# ?edf
edf(model)

### Compare with an unconstrained model

# Refit the model without constraints
umodel <- uncons(model)

# Add result
lines(warming$year, predict(umodel), col = 3, lwd = 2, lty = 2)

#####
# Monotone splines with the warming dataset

library(splines)

# Define a natural spline basis
basis <- ns(warming$year, df = 10)

### Constrained model

```

```

# Non-decreasing splines
model <- glm(anomaly ~ basis, data = warming, method = "cirls.fit",
  cons = ~ shape(basis, "inc"))

# Plot result
plot(anomaly ~ year, data = warming, xlab = "", ylab = "Temperature anomaly")
lines(warming$year, predict(model), col = 2, lwd = 2)

### Unconstrained model
umodel <- uncons(model)
lines(warming$year, predict(umodel), col = 3, lwd = 2, lty = 2)

#####
# Compositional regression example with the fgl dataset

# Select the composition and use log
x <- as.matrix(fgl[,2:9])
z <- log(x)

# Fit model
model <- glm(RI ~ z, data = fgl, method = "cirls.fit",
  cons = ~ zerosum(z))

# Coefficients sum to 1
coef(model)
sum(coef(model)[-1])

# Confidence intervals
confint(model)

```

---

boundConstr

*Boundary constraints*


---

## Description

Generic function to build a constraint matrix and bound vectors for constraining the boundary of [B-splines](#) and other nonlinear functions to a given value. It is designed to be used in the [constr](#) formula interface. It allows methods for a wide range of regression terms.

## Usage

```
boundConstr(x, ...)
```

```
## Default S3 method:
```

```
boundConstr(x, value = 0, side = c("right", "left",
  "both"), sm = 1, intercept = FALSE, ...)
```

```
## S3 method for class 'factor'
boundConstr(x, value = 0, side = "right", thr = NULL,
            sm = 1, intercept = FALSE, ...)
```

### Arguments

|           |                                                                                                                                                                                                                                 |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x         | An object representing a design matrix of predictor variables, typically basis functions. See Details for supported objects.                                                                                                    |
| ...       | Additional parameters passed to or from other methods. Includes value and side for most methods.                                                                                                                                |
| value     | A numerical indicating the boundary constraint value. Default to 0.                                                                                                                                                             |
| side      | Character indicating the side on which the constraint applies. One of "right" (the default), "left" or "both".                                                                                                                  |
| sm        | A positive integer indicating the degree of smoothness for the constraint. In the default method indicates the number of coefficients to constrain to value, but can have a slightly different interpretation in other methods. |
| intercept | For the default and factor method, a logical value indicating if the design matrix includes an intercept. In most cases, it will be automatically extracted from x, but this argument can be used to override it.               |
| thr       | For the factor method, an integer indicating a level from which the association should be equal to value. Cannot be used with side = both. This parameter has a slightly different interpretation for other methods.            |

### Details

Enforcing boundary constraints amounts to imposing an equality constraint on the `sm` first (if `side = left`), last (if `side = right`), or both (if `side = both`) coefficients of the basis matrix `x`. The equality constraint sets the bounds to `lb = ub = value`.

#### Usage:

The recommended usage is to use this function through a call to `bound` on a term in the `constr` formula interface. This method is then called internally to create the constraint matrix and bound vectors. However, `boundConstr` can also be called directly on a matrix-like object to manually build or inspect the constraint matrix.

All methods internally rely on the default method for general matrices. Unless specified, all methods use the same parameters as the default one, which are passed through `...`. The only exception is `intercept` which is often inferred from the `x` object itself. In a typical usage in which `boundConstr` is called from the `constr` argument, `intercept` is automatically determined from the `glm` formula.

#### The `sm` parameter:

The `sm` parameter indicates the number of coefficients on the left/right that are constrained to be equal to `value` and can be interpreted as a smoothness degree for the boundary constraint. The default is to constraint only one coefficient. Note that `sm` has a slightly different interpretation when `x` represents `spline bases`.

**Available methods:**

In addition to the default method, boundConstr currently supports methods for several classes. The full list can also be consulted through `methods(boundConstr)`.

*Categorical variables:*

- **factor**: for factor objects. Extract the **contrasts** to define the constraint matrix. Here the **intercept** argument has the same interpretation as in the default method, i.e. if set to **TRUE** it means the **glm** model does not include an intercept externally to the factor. Note that, in this case, a simple dummy coding is done in R.
- **strata**: Indicator variables defining strata from the **dlm** package.

*Splines:*

- **bs, ns, ps**: Specify boundary constraints on various flavors of B-splines.

*Distributed-lag linear and nonlinear models (DLNM):*

- **onebasis**: General method for basis functions generated in the package. Internally calls the relevant method for the specified basis.
- **crossbasis**: Most exhaustive method to constrain DLNMs. See its dedicated [help page](#).

**Value**

A list containing the constraint matrix **Cmat**, and lower/upper bound vectors (**lb** and **ub**, respectively).

**See Also**

[buildCmat](#) detailing the **constr** interface.

**Examples**

```
#####
# Bound constraints example: warming dataset

library(splines)

# Force the fit to reach the last value
# Remove the intercept, so that it is included in the constraint
model <- glm(anomaly ~ 0 + decade, data = warming, method = "cirls.fit",
  constr = ~ bound(decade, value = 0.75))

# Same but with splines
basis <- ns(warming$year, df = 10)
splmodel <- glm(anomaly ~ 0 + basis, data = warming, method = "cirls.fit",
  constr = ~ bound(basis, value = 0.75))

# Plot result
plot(anomaly ~ year, data = warming, xlab = "", ylab = "Temperature anomaly")
lines(warming$year, predict(model), col = 2, lwd = 2)
lines(warming$year, predict(splmodel), col = 3, lwd = 2)
```

boundConstr.bs

*Boundary constraints for spline bases***Description**

Method for constraining a smooth functions estimated using B-Spline bases to reach a given value at its boundary.

**Usage**

```
## S3 method for class 'bs'
boundConstr(x, value = 0, side = "right", thr = NULL,
            sm = NULL, ...)

## S3 method for class 'ns'
boundConstr(x, ...)

## S3 method for class 'ps'
boundConstr(x, ...)
```

**Arguments**

|       |                                                                                                                                                                                         |
|-------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x     | A B-spline basis object.                                                                                                                                                                |
| value | A numerical indicating the boundary constraint value. Default to 0.                                                                                                                     |
| side  | Character indicating the side on which the constraint applies. One of "right" (the default), "left" or "both".                                                                          |
| thr   | A numeric value indicating a threshold from which the smooth should be equal to value. Default to the boundary knots of the spline basis. Cannot be used with side = both. See details. |
| sm    | A positive integer controlling the smoothness of the constraint. Ignored when thr is provided. See details.                                                                             |
| ...   | Additional parameters passed to or from other methods.                                                                                                                                  |

**Details**

See the [generic](#) method for an overview of bound constraints.

**Boundary constraints with B-splines:**

By construction, B-spline basis functions sum to one at any point. Therefore, boundary constraints can essentially be enforced through equality constraints on the first (if side = left) or last (if side = right) few coefficients of the B-spline basis, with small variations for [natural](#) or [penalised](#) splines.

**The thr and sm arguments:**

The sm argument can control how smoothly the resulting function reaches the constraint value. If only one basis is non-null at the boundary and need its coefficient to be constrained for the

boundary constraint, the number of bases that reach zero exactly at the boundary is equal to the degree of the spline. Forcing their coefficients to equal value results in a smoother convergence at the boundary. Note that increasing `sm` to be higher than the degree of the spline will propagate the equality constraint towards interior knots.

Alternatively, the user can provide a threshold through `thr` above which (if `side = right`) or below which (if `side = left`) the resulting function is guaranteed to be equal to `value`. This allows propagating the boundary constraint towards the interior of the range. Note that where the constraint starts depends on the knots, as using `thr` will actually enforce the constraint from the maximum (minimum) interior knot such as `thr` is included in the constrained range. This does not allow `side = both` so if thresholds are needed on both sides of the range, it needs to be done with two calls to `boundConstr`.

### Value

A list containing the constraint matrix `Cmat`, and lower/upper bound vectors (`lb` and `ub`, respectively).

### See Also

The generic `boundConstr` functions.

### Examples

```
#####
# Bound constraints for bs: warming dataset

library(splines)

# Force the fit to reach the last value
# Remove the intercept, so that it is included in the constraint
basis <- bs(warming$year, df = 10)
mod <- glm(anomaly ~ 0 + basis, data = warming, method = "cirls.fit",
  constr = ~ bound(basis, value = 0.75))

# Plot result
plot(anomaly ~ year, data = warming, xlab = "", ylab = "Temperature anomaly")
lines(warming$year, predict(mod), col = 3, lwd = 2)
```

---

`boundConstr.crossbasis`

*Constraints for distributed lag linear and non-linear models*

---

### Description

Methods to generate constraint matrices associated with `crossbasis` objects, allowing the fitting of constrained distributed lag linear and non-linear models (DLMs and DLNMs). Designed for use within the `constr` interface in `cirls`.

**Usage**

```
## S3 method for class 'crossbasis'
boundConstr(x, dim = "var", overall = FALSE,
            odrng = NULL, ...)

## S3 method for class 'crossbasis'
shapeConstr(x, dim = "var", overall = FALSE,
            odrng = NULL, ...)
```

**Arguments**

|         |                                                                                                                                                                                                                                    |
|---------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x       | A crossbasis object.                                                                                                                                                                                                               |
| dim     | Either "var" (the default) or "lag". This is the dimension on which the constraint will be applied, constraining either the exposure-response or lag-response relationship, respectively.                                          |
| overall | Only when dim = "var", logical indicating whether the constraint should be applied only on the overall cumulative association or for each specific lag.                                                                            |
| odrng   | A numeric vector of length 2 restricting the constraint on a specific range of the <i>other</i> dimension, namely the lag (for dim = "var") or exposure (dim = "lag") space. By default, no restriction is performed. See Details. |
| ...     | Parameters specific to the type of constraint to be applied. Includes for instance shape for <a href="#">shapeConstr</a> , or value for <a href="#">boundConstr</a> . See the main help page of the relevant generic method.       |

**Details**

Constraints for DLMs and DLNMs apply to one of two dimensions of an exposure-lag-response association, specifically the exposure space (for dim = "var") or the lag space (for dim = "lag"). Operationally, the functions generate a constraint matrix for the requested dimension of a crossbasis object, which is then expanded consistently in the other dimension. This allows the user to apply specific constraints to either the exposure-response association or the lag-response association, relying on the same [shape](#) or [bound](#) methods used for other functions.

**Overall vs full surface:**

By default, when dim = "var", the constraint is applied across the whole lag space. This means that the exposure-response will be constrained at every lag. When overall is switched to TRUE (allowed only when dim = "var"), the constraint will only hold for the overall cumulative association (namely the net effect summed across lags), while it can be violated for exposure-responses defined at specific lags. See [crossreduce](#) for more information.

**Restricting the range on the other dimension:**

The odrng argument can be used to restrict the constraint on a specific range of the other dimension. For instance, when dim = "var", setting odrng = c(0, 5) will enforce the constraint on the exposure-response relationship only over lags 0 to 5, meaning it could be violated at other lags (> 5 in this example). Note that the actual range depends on the specific basis functions used for the opposite dimension (see the range argument in [shapeConstr](#) for additional details). When overall = TRUE, it will only constraint the cumulative association over the range specified by odrng (see above).

**Note:**

These methods for [crossbasis](#) objects only allow to specify a single constraint to one dimension of the exposure-lag-response association. However, the user can apply multiple constraints, possibly to both dimensions, by adding multiple terms for the same object in the [constr](#) formula (see Examples below).

**Warnings:**

In the current implementation, constraints applied to the lag dimension (when `dim = "lag"`) can be straightforwardly defined when the exposure-response function is linear, namely for simpler DLMS. For non-linear exposure-responses, the centering performed by [crosspred](#) when predicting the exposure-lag-response relationship can break the constraint definition and return unpredictable results. Users are advised not to apply constraints in the lag dimension of complex DLNMs, or to do so with particular care.

**Value**

A list containing the constraint matrix `Cmat`, and lower/upper bound vectors (`lb` and `ub`, respectively).

**References**

- Gasparrini, A., Armstrong, B., Kenward, M.G., 2010. Distributed lag non-linear models. *Statistics in Medicine* **29**, 2224–2234. DOI:10.1002/sim.3940
- Gasparrini, A., Armstrong, B., 2013. Reducing and meta-analysing estimates from distributed lag non-linear models. *BMC Medical Research Methodology* **13**, 1. DOI:10.1186/1471-2288-13-1

**See Also**

[crossbasis](#) for the definition of DLMS and DLNMs as well as their dimensions. [shapeConstr](#) and [boundConstr](#) for generic constraint methods. [buildCmat](#) for how to specify constraints.

**Examples**

```
#####
# Constrained distributed lag models (DLM)

library(dlnm)
library(splines)

#-----
# Association between PM10 and mortality
#-----

# Number of years and dow
ny <- length(unique(format(london$date, "%Y")))

# Create a flexible crossbasis
dlnm <- crossbasis(london$pm10, lag = 15, argvar = list(fun = "lin"),
  arglag = list(fun = "ns", knots = logknots(15, df = 4)))
```

```

#----- Fit models

# We always have to set `dim = "lag"` because it is set to "var" by default

# Fit with bound constraint, value = 0 by default
bm <- glm(age0_64 ~ dlm + ns(date, df = 7 * ny) + dow, data = london,
  family = "quasipoisson", method = "cirls.fit",
  constr = ~ bound(dlm, dim = "lag"))

# Get a decaying lag-response function with a decreasing constraint
dm <- glm(age0_64 ~ dlm + ns(date, df = 7 * ny) + dow, data = london,
  family = "quasipoisson", method = "cirls.fit",
  constr = ~ shape(dlm, dim = "lag", shape = "dec"))

# Both can be used at the same time
dbm <- glm(age0_64 ~ dlm + ns(date, df = 7 * ny) + dow, data = london,
  family = "quasipoisson", method = "cirls.fit",
  constr = ~ shape(dlm, dim = "lag", shape = "dec") + bound(dlm, dim = "lag"))

#----- Plot to compare

# Compare to unconstrained model
um <- glm(age0_64 ~ dlm + ns(date, df = 7 * ny) + dow, data = london,
  family = "quasipoisson")

# Use crosspred to get the lag-response function
ucp <- crosspred(dlm, um, cen = 0, at = 10)
bcp <- crosspred(dlm, bm, cen = 0, at = 10)
dcp <- crosspred(dlm, dm, cen = 0, at = 10)
dbcp <- crosspred(dlm, dbm, cen = 0, at = 10)

# Now plot
par(mfrow = c(2,2))
plot(ucp, ptype = "slices", lwd = 2, var = 10, main = "Unconstrained")
plot(bcp, ptype = "slices", lwd = 2, var = 10, col = 2,
  main = "Bound")
plot(dcp, ptype = "slices", lwd = 2, var = 10, col = 3,
  main = "Decreasing")
plot(dbcp, ptype = "slices", lwd = 2, var = 10, col = 4,
  main = "Bound & decreasing")

#####
# Constrained DLNM: var dimension

## Not run:

library(dlnm)
library(splines)

#-----
# Association between temperature and mortality
#-----

```

```

# Number of years and dow
ny <- length(unique(format(london$date, "%Y")))

# Create a flexible crossbasis
cb <- crossbasis(london$tmean, lag = 21,
  argvar = list(fun = "bs", degree = 2, df = 10),
  arglag = list(fun = "ns", knots = logknots(21, df = 5)))

#----- Fit several models

# In the following, `dim` is not shown as it is set to "var" by default

# A nondecreasing model
icm <- glm(age0_64 ~ cb + ns(date, df = 7 * ny) + dow, data = london,
  family = "quasipoisson", method = "cirls.fit",
  constr = ~ shape(cb, shape = "inc"))

# A convex model
ccm <- glm(age0_64 ~ cb + ns(date, df = 7 * ny) + dow, data = london,
  family = "quasipoisson", method = "cirls.fit",
  constr = ~ shape(cb, shape = "cvx"))

# A convex model with a constraint on overall only
ocm <- glm(age0_64 ~ cb + ns(date, df = 7 * ny) + dow, data = london,
  family = "quasipoisson", method = "cirls.fit",
  constr = ~ shape(cb, shape = "cvx", overall = TRUE))

# A convex model effectively constraining only lags 0 and 1
subcm <- glm(age0_64 ~ cb + ns(date, df = 7 * ny) + dow, data = london,
  family = "quasipoisson", method = "cirls.fit",
  constr = ~ shape(cb, shape = "cvx", slice = c(0, 1)))

#----- Plot results

# Plot non-decreasing
icp <- crosspred(cb, icm, cen = 20)
par(mfrow = c(1, 2))
plot(icp, main = "Non-decreasing")
plot(icp, ptype = "overall")

# Plot convex
ccp <- crosspred(cb, ccm, cen = 20)
par(mfrow = c(1, 2))
plot(ccp, main = "Convex")
plot(ccp, ptype = "overall")

# This one is restricted to overall but not for every single lag
ocp <- crosspred(cb, ocm, cen = 20)
par(mfrow = c(1, 2))
plot(ocp, main = "Overall convex")
plot(ocp, ptype = "overall")

```

```
# Restricted to lags 0 and 1 only
subcp <- crosspred(cb, subcm, cen = 20)
par(mfrow = c(1, 2))
plot(subcp, main = "Convex")
plot(subcp, ptype = "overall")

## End(Not run)
```

---

boundConstr.onebasis    *Boundary constraints for onebasis objects*

---

## Description

Method to set up boundary constraints for basis objects constructed through [onebasis](#) and other specifications from the [dlnm](#) package.

## Usage

```
## S3 method for class 'onebasis'
boundConstr(x, ...)

## S3 method for class 'strata'
boundConstr(x, value = 0, side = "right", thr = NULL,
            sm = 1, ...)
```

## Arguments

|       |                                                                                                                                                  |
|-------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| x     | A onebasis object or any other objects from the dlnm package.                                                                                    |
| ...   | Additional parameters passed to or from other methods.                                                                                           |
| value | A numerical indicating the boundary constraint value. Default to 0.                                                                              |
| side  | Character indicating the side on which the constraint applies. One of "right" (the default), "left" or "both".                                   |
| thr   | An integer indicating a threshold from which the association should be equal to value. Using thr overrides sm. Cannot be used with side = both.  |
| sm    | A positive integer indicating the degree of smoothness for the constraint. Typically indicates the number of coefficients to constrain to value. |

## Details

The onebasis method simply check there is an available [boundConstr](#) method available for the underlying basis type (for instance [strata](#) or [P-splines](#)) and dispatches x to the appropriate method. The ellipsis ... is used to pass the necessary arguments to the method, including value and side. See the help for the specific methods.

The strata method allows to specify boundary constraints for [strata](#) objects, which are factors expanded through dummy parametrisation. In this context, boundary constraints essentially add

an equality constraint to value on the sm extreme coefficients (depending on side). A numeric threshold thr can be specified instead of sm, which will add the equality constraint from the strata that contains thr. This allows to propagate boundary constraints towards the interior of the range.

### Value

A list containing the constraint matrix Cmat, and lower/upper bound vectors (lb and ub, respectively).

### See Also

The generic [boundConstr](#) functions. The [crossbasis](#) is used to specify the constraint on distributed lag (non-linear) models.

### Examples

```
#####
# Bound constraints for onebasis: warming dataset

library(dlnm)

# Force the fit to reach the last value
# Remove the intercept, so that it is included in the constraint
basis <- onebasis(warming$year, fun = "strata", df = 10, intercept = TRUE)
mod <- glm(anomaly ~ 0 + basis, data = warming, method = "cirls.fit",
  constr = ~ bound(basis, value = 0.75))

# Plot result
plot(anomaly ~ year, data = warming, xlab = "", ylab = "Temperature anomaly")
lines(warming$year, predict(mod), col = 2, lwd = 2)
```

---

buildCmat

*Build a constraint matrix*

---

### Description

Internal function building a full constraint matrix from a list of constraint matrices and/or a formula providing specific constraints. In-depth technical details are provided below.

### Usage

```
buildCmat(mf, assign = NULL, constr = NULL, Cmat = NULL, lb = NULL,
  ub = NULL, redundant = TRUE, equality = TRUE, warn = FALSE)
```

## Arguments

|           |                                                                                                                                         |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------|
| mf        | A <a href="#">model.frame</a> or a list of variables. Defines the model for which the constraint matrix is built.                       |
| assign    | A vector of indices mapping columns of the design matrix to model terms.                                                                |
| constr    | A formula specifying constraints.                                                                                                       |
| Cmat      | Either a matrix or a named list of constraint matrices. In the latter case, names should correspond to terms in mf (See details).       |
| lb, ub    | Vector or named list of vectors containing constraint bounds. In the latter case, names should correspond to terms in mf (See details). |
| redundant | Logical indicating if potential redundant constraints found should be removed from the constraint matrix.                               |
| equality  | Logical indicating if any underlying equality constraint found should be reduced in the constraint matrix.                              |
| warn      | Logical indicating if the user should be warned when constraint reduction happens.                                                      |

## Details

This function is called internally by [cirls.fit](#) and is not meant to be used directly by the user. It prepares the full Cmat, lb and ub for the model, providing a way to specify constraints without having to build a full constraint matrix beforehand. It uses the model frame in mf to match specific constraints to the right columns in the design matrix.

This function also checks that any constraint matrix provided to [cirls.fit](#) is irreducible, which can be controlled by arguments redundant, equality and warn. See [reduceCons](#) for details.

There are three ways to specify constraints in cirls:

1. Through the constr formula. This provides a simple interface for many commonly encountered constraints and is the recommended way for new users.
2. Through named lists of constraint matrices and bounds can be provided to Cmat/lb/ub arguments. This is useful for constraints that are not (yet) implemented for the constr formula.
3. Through a fully specified Cmat/lb/ub, provided directly.

Each one is detailed in a subsection below. Note that options 1 and 2 can be used simultaneously.

### The constr formula:

The easiest way to specify constraints is as a formula of the form `constr = ~ cons(x, ...)` where cons represents a constraint to be applied to term x of the cirls model and ... represents additional arguments that depend on the cons function. Several constraints can be applied to the same or to different terms, for example `constr = ~ cons1(x) + cons1(y) + cons2(x)`. All terms appearing in constr can either be vectors or matrices and should be found in the model.frame mf, otherwise the constraint is dropped with a warning.

Internally, buildCmat will look for a function called consConstr that returns a list of Cmat, lb, and ub built specifically for the term provided to cons() in the formula. This provides a simple way to add new constraints to the interface, by creating a function with the Constr suffix, taking a term as an input and outputting a list with Cmat, lb, and ub.

The list of available cons functions can be found on the main help page of the [cirls](#) package (subsection 'Built-in constraints'). Each function has its own help page detailing the implemented constraint with available parameters.

#### **Term-specific Cmat/lb/ub:**

Constraints for specific terms of the model can be provided as named lists to one or several of arguments Cmat, lb, and ub, which represent the constraint matrix, lower bound, and upper bound, respectively. This will take the form `Cmat = list(x = cm1, y = cm2)` where x and y are terms in the model.frame mf, while cm1 and cm2 are constraint matrices (or bound vectors for lb and ub). Note that these objects *must* be consistent with the dimensions of their respective terms.

When terms are found in lb and/or ub, but not in Cmat, it is assumed that *simple* bound constraints are to be applied to the terms. In that case, Cmat will be internally created as a simple identity matrix matching the dimensions of the terms in question.

Names in Cmat/lb/ub can include several terms, which should be separated by a semicolon ;, for instance `Cmat = list("x;y" = cm)`. This allows specifying constraints that span several terms in the model.

#### **Fully specified Cmat/lb/ub:**

If one of Cmat/lb/ub is neither NULL nor a list, it is assumed it is provided as fully specified, i.e. should be returned as it is. In that case, constr and any list provided to other arguments are ignored. When not all of Cmat/lb/ub are fully specified, the other ones will be filled with default values that match the dimension of the model matrix, i.e. an identity matrix for Cmat, a vector of zeros for lb and a vector of Inf for ub.

#### **Unconstrained model:**

When all of constr/Cmat/lb/ub are NULL, a list of empty Cmat/lb/ub is returned and an unconstrained model is fit.

### **Value**

A list with elements Cmat, lb, and ub containing the fully specified constraint matrix, lower and upper bounds for the model specified in argument mf. Cmat additionally includes an attribute called terms that maps constraints represented in the matrix to individual terms in the model.

### **See Also**

The main [help page](#) for the list of cons functions implemented and examples. [reduceCons](#) for details on irreducibility.

### **Examples**

```
#####
# Examples of constraint specifications

#---- constr interface

# Fit an initial model and extract model.frame
mf <- model.frame(glm(death ~ pm10 + o3 + co, data = london))

# Simple non-negative constraint
```

```

buildCmat(mf, constr = ~ shape(pm10, "pos"))

# Several constraints
buildCmat(mf, constr = ~ shape(pm10, "pos") + shape(o3, "pos"))

# Different constraints
buildCmat(mf, constr = ~ shape(pm10, "pos") + zerosum(o3, co))

### In practice, this is done directly when calling glm
model <- glm(death ~ pm10 + o3 + co, data = london, family = "quasipoisson",
  method = "cirls.fit", constr = ~ shape(pm10, "pos"))
model[c("Cmat", "lb", "ub")]
buildCmat(mf, constr = ~ shape(pm10, "pos"))

#----- Specific terms

# Simple bound constraint
buildCmat(mf, lb = list("pm10" = 0))

# Equivalent of zerosum
buildCmat(mf, Cmat = list("o3;co" = matrix(1, 1, 2)), ub = list("o3;co" = 0))

### In practice, this is done directly when calling glm
model <- glm(death ~ pm10 + o3 + co, data = london, family = "quasipoisson",
  method = "cirls.fit", lb = list("pm10" = 0))
model[c("Cmat", "lb", "ub")]
buildCmat(mf, lb = list("pm10" = 0))

#----- Both options simultaneously

# Bound constraint and zerosum
buildCmat(mf, lb = list("pm10" = 0), constr = ~ zerosum(o3, co))

# Same as before, should be done in glm
model <- glm(death ~ pm10 + o3 + co, data = london, family = "quasipoisson",
  method = "cirls.fit",
  lb = list("pm10" = 0), constr = ~ zerosum(o3, co))
model[c("Cmat", "lb", "ub")]

#----- Fully specified matrix

# Simple bound constraints
Cmat = cbind(0, diag(2), 0)
buildCmat(mf, Cmat = Cmat)

# Zerosum constraint
Cmat <- t(c(0, rep(1, 3)))
buildCmat(mf, Cmat = Cmat, lb = 0, ub = 0)

#----- Unconstrained model
buildCmat(mf)

```

cirls.control

*Parameters controlling CIRLS fitting***Description**

Function controlling the [cirls.fit](#) algorithm. Typically only used internally with arguments passed to ... in [glm](#), but may be used to construct a control argument to either function.

**Usage**

```
cirls.control(constr = NULL, Cmat = NULL, lb = NULL, ub = NULL,
  epsilon = 1e-08, maxit = 25, trace = FALSE, redundant = TRUE,
  equality = TRUE, warn = FALSE, qp_solver = "quadprog",
  qp_pars = list(), qrtol = 1e-07)
```

**Arguments**

|           |                                                                                                                                                                                                                    |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| constr    | A formula specifying constraints to be applied to specific terms in the model. See details in <a href="#">buildCmat</a> for constraint specification.                                                              |
| Cmat      | Constraint matrix specifying the linear constraints applied to coefficients. Can also be provided as a list of matrices for specific terms. See details in <a href="#">buildCmat</a> for constraint specification. |
| lb, ub    | Lower and upper bound vectors for the linear constraints. If not provided, defaults to 0 and Inf, respectively.                                                                                                    |
| 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.                                                                                                                                                |
| redundant | Logical indicating if potential redundant constraints found should be removed from the constraint matrix.                                                                                                          |
| equality  | Logical indicating if any underlying equality constraint found should be reduced in the constraint matrix.                                                                                                         |
| warn      | Logical indicating if warning related to constraints should be produced.                                                                                                                                           |
| qp_solver | The quadratic programming solver. One of "quadprog" (the default), "osqp", or "coneproj".                                                                                                                          |
| qp_pars   | List of parameters specific to the quadratic programming solver. See the help pages in the respective packages (links below).                                                                                      |
| qrtol     | Tolerance parameter for detecting linear dependencies through QR decomposition. Passed to tol in <a href="#">qr</a> .                                                                                              |

## 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`.

### Constraint specification:

Constraint specification through the `constr`, `Cmat`, `lb` and `ub` argument is fully detailed in the help of the `buildCmat` functions.

### Reducing constraints:

The default behaviour is to silently reduce the constraint matrix and bound vectors when redundant or underlying equality constraint are found. This mean that the `Cmat`, `lb` and `ub` returned by `cirls` can be different than those provided. Switching the arguments `redundant` and `equality` to `FALSE` will avoid checking for redundant and underlying equality constraints respectively. **Switching these arguments off can cause problems for inference.** See `reduceCons` for more details on reducing constraints.

### Quadratic programming solvers:

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

- "quadprog" (the default) solves the quadratic program via a dual algorithm. It relies on the function `solve.QP`.
- "osqp" solves the quadratic program via the Alternating Direction Method of Multipliers (ADMM). It relies on the function `solve_osqp`.
- "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. "quadprog" is set as the default being generally more reliable than the other solvers. "osqp" is faster but can be less accurate, in which case it is recommended to increase convergence tolerance at the cost of speed. "coneproj" is efficient for inequality constrained models but less recommended when there are equality constraints.

## Value

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

## See Also

Specification of a `cirls` model and constraint specification in `buildCmat`.

## Examples

```
#####
# Examples of control parameters

# The simplest way to specify parameters is through ...
model <- glm(death ~ o3, data = london, family = "quasipoisson",
```

```

method = "cirls.fit", cons = ~ shape(o3, "pos"))
osqpmode1 <- glm(death ~ o3, data = london, family = "quasipoisson",
  method = "cirls.fit", cons = ~ shape(o3, "pos"), qp_solver = "osqp")

# Comparing the control
model$control
osqpmode1$control

# Alternatively through the control argument
osqpmode2 <- glm(death ~ o3, data = london, family = "quasipoisson",
  method = "cirls.fit", control = list(
    cons = ~ shape(o3, "pos"), qp_solver = "osqp"))
all.equal(osqpmode1$control, osqpmode2$control)

# However, both cannot be used at the same time
osqpmode3 <- glm(death ~ o3, data = london, family = "quasipoisson",
  method = "cirls.fit", control = list(cons = ~ shape(o3, "pos")),
  qp_solver = "osqp")
all.equal(osqpmode1$control, osqpmode3$control)

# The control argument takes precedence
model2 <- glm(death ~ o3, data = london, family = "quasipoisson",
  method = "cirls.fit", control = list(cons = ~ shape(o3, "pos")),
  cons = ~ shape(o3, "neg"))
model2$control$cons

```

cirls.fit

*Constrained Iteratively Reweighted Least-Squares algorithm*

## Description

Routine implementing the constrained iteratively-reweighted least-squares (CIRLS) algorithm to fit generalised linear models (GLM) with linear constraints on the coefficients. This function is designed to replace the `glm.fit` function passed through the `method` argument of [glm](#).

## 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

|                      |                                                                                                                                                 |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>x, y</code>    | <code>x</code> is a design matrix and <code>y</code> is a vector of response observations. Usually internally computed by <a href="#">glm</a> . |
| <code>weights</code> | An optional vector of observation weights.                                                                                                      |
| <code>start</code>   | 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 <a href="#">model.offset</a> .                                                                                   |
| family      | The result of a call to a family function, describing the error distribution and link function of the model. See <a href="#">family</a> for details of available family functions. |
| control     | A list of parameters controlling the fitting process. See details and <a href="#">cirls.control</a> .                                                                              |
| 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 constraints through different arguments (see [buildCmat](#)). These additional parameters can be passed through the control list or through `...` in [glm](#) but not both. If any parameter is passed through control, then `...` will be ignored. See the full list of parameters in [cirls.control](#).

#### Algorithm:

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

$$lb \leq Cmat \% \% coefficients \leq ub$$

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

### Value

A object of class `cirls` inheriting from `glm`. The object of class `cirls` includes all components from [glm](#) objects, with the addition of:

|              |                                                                                                                                    |
|--------------|------------------------------------------------------------------------------------------------------------------------------------|
| Cmat, lb, ub | the constraint matrix, and lower and upper bound vectors. If provided as lists, the full expanded matrix and vectors are returned. |
| 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.                                                                  |
| etastart     | the initialisation of the linear predictor eta. The same as etastart when provided.                                                |
| singular.ok  | the value of the singular.ok argument.                                                                                             |

Any method for `glm` objects can be used on `cirls` objects.

### References

Masselot, P., Nenon, D., Vanoli, J., Chalabi, Z., Gasparrini, A., 2025. Estimation and inference in generalised linear models with constrained iteratively-reweighted least squares. *ArXiv preprint*. DOI:10.48550/arXiv.2509.18406

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/s12532-020-00179-2

## See Also

[buildCmat](#) for details on constraint specification and [cirls.control](#) for parameters controlling the algorithm. [glm](#) provides details on glm objects.

## Examples

```
#####
# Non-negative coefficient example with the london dataset

library(splines)

### Association between Ozone and mortality
# Nonnegative constraint on Ozone
model <- glm(death ~ o3, data = london, family = "quasipoisson",
  method = "cirls.fit", cons = ~ shape(o3, "pos"))

# Coefficient and confidence interval
coef(model)[2]
confint(model)[2,]

# Comparing to an unconstrained model
umodel <- uncons(model)
coef(umodel)[2]
confint(umodel)[2,]
#####
# Monotone strata levels with the warming dataset

### Fit the model

# Non-decreasing constraint on decadal strata
model <- glm(anomaly ~ decade, data = warming, method = "cirls.fit",
  cons = ~ shape(decade, "inc"))

# Plot result
plot(anomaly ~ year, data = warming, xlab = "", ylab = "Temperature anomaly")
lines(warming$year, predict(model), col = 2, lwd = 2)

### Extract results

# Coefficients and confidence intervals
betas <- coef(model)
v <- vcov(model)
```

```

cis <- confint(model)

# Degrees of freedom: represent number of strata change
# ?edf
edf(model)

### Compare with an unconstrained model

# Refit the model without constraints
umodel <- uncons(model)

# Add result
lines(warming$year, predict(umodel), col = 3, lwd = 2, lty = 2)

#####
# Monotone splines with the warming dataset

library(splines)

# Define a natural spline basis
basis <- ns(warming$year, df = 10)

### Constrained model

# Non-decreasing splines
model <- glm(anomaly ~ basis, data = warming, method = "cirls.fit",
  cons = ~ shape(basis, "inc"))

# Plot result
plot(anomaly ~ year, data = warming, xlab = "", ylab = "Temperature anomaly")
lines(warming$year, predict(model), col = 2, lwd = 2)

### Unconstrained model
umodel <- uncons(model)
lines(warming$year, predict(umodel), col = 3, lwd = 2, lty = 2)

#####
# Compositional regression example with the fgl dataset

# Select the composition and use log
x <- as.matrix(fgl[,2:9])
z <- log(x)

# Fit model
model <- glm(RI ~ z, data = fgl, method = "cirls.fit",
  cons = ~ zerosum(z))

# Coefficients sum to 1
coef(model)
sum(coef(model)[-1])

# Confidence intervals
confint(model)

```

---

|     |                                    |
|-----|------------------------------------|
| edf | <i>Expected degrees of freedom</i> |
|-----|------------------------------------|

---

## Description

Estimates expected degrees of freedom (df) of a [cirls](#) object through simulations.

## Usage

```
edf(object, nsim = 10000, seed = NULL)
```

## Arguments

|        |                                                                                             |
|--------|---------------------------------------------------------------------------------------------|
| object | A <a href="#">cirls</a> object or any object inheriting from <code>lm</code> . See details. |
| nsim   | The number of simulations.                                                                  |
| seed   | An optional seed for the random number generator. See <a href="#">set.seed</a> .            |

## Details

Computes **unconstrained**, **observed**, and **expected** df for [cirls](#) objects. The function also works for most objects inheriting from `lm` although, in this case, only the unconstrained (udf) part makes sense.

### Unconstrained df:

**Unconstrained** degrees of freedom (udf) refer to the usual degrees of freedom, i.e. the number of estimated parameters  $p$ . In GLMs, this corresponds to the number of coefficients, plus one degree of freedom for the dispersion parameter, when relevant.

### Observed df:

**Observed** degrees of freedom (odf) correspond to the constrained df of a fitted [cirls](#) model. This corresponds to  $p - m_a$ , where  $m_a$  is *the number of active constraint* in the fitted [cirls](#) model. Intuitively, the more restricted the final fit, the more df are decreased.

### Expected df:

**Expected** degrees of freedom (edf) account for the sampling variation in the number of active constraints. A different sample might result in a fit with another number of active constraints  $m_a$  which means odf represents an inaccurate estimation of the reduction in degrees of freedom induced by the constraints. edf is defined as  $p - \bar{m}_a$  where  $\bar{m}_a$  is the expected number of active constraints. Here,  $\bar{m}_a$  is estimated by sampling from the [unconstrained](#) distribution of coefficients (see [simulCoef](#)).

**Value**

A vector of length three containing the three types of degrees of freedom:

|     |                                                                                                           |
|-----|-----------------------------------------------------------------------------------------------------------|
| udf | The <i>unconstrained</i> degrees of freedom, i.e. the rank plus any dispersion parameter for glm objects. |
| odf | The <i>observed</i> degrees of freedom, that is udf minus the number of active constraints.               |
| edf | The <i>expected</i> degrees of freedom estimated by simulation, as described in the Details section.      |

For `cirls` objects, the vector includes the simulated distribution of the number of active constraints as an `actfreq` attribute.

**References**

Meyer, M.C., 2013. Semi-parametric additive constrained regression. *Journal of Nonparametric Statistics* **25**, 715–730. DOI:10.1080/10485252.2013.797577

**See Also**

[logLik.cirls](#) which internally calls `edf` to compute degrees of freedom. [simulCoef](#) for coefficient simulation.

**Examples**

```
#####
# Monotone strata levels with the warming dataset

### Fit the model

# Non-decreasing constraint on decadal strata
model <- glm(anomaly ~ decade, data = warming, method = "cirls.fit",
  cons = ~ shape(decade, "inc"))

# Plot result
plot(anomaly ~ year, data = warming, xlab = "", ylab = "Temperature anomaly")
lines(warming$year, predict(model), col = 2, lwd = 2)

### Extract results

# Coefficients and confidence intervals
betas <- coef(model)
v <- vcov(model)
cis <- confint(model)

# Degrees of freedom: represent number of strata change
# ?edf
edf(model)

### Compare with an unconstrained model
```

```
# Refit the model without constraints
umodel <- uncons(model)

# Add result
lines(warming$year, predict(umodel), col = 3, lwd = 2, lty = 2)
```

fgl

*Measurement of forensic glass fragments***Description**

Dataset adapted from the [fgl](#) dataset. Chemical composition of fragments of glass collected in forensic work.

**Usage**

fgl

**Format**

A data frame of 214 observations and 10 variables. Apart from RI and type, all other variables represent a ratio of the given chemical component.

**RI** Refractive index; more precisely the refractive index is 1.518xxxx.

**Na** Sodium

**Mg** Manganese

**Al** Aluminium

**Si** Silicon

**K** Potassium

**Ca** Calcium

**Ba** Barium

**Fe** Iron

**type** Class type (see Details)

**Details**

The fragments were originally classed into seven types, one of which was absent in this dataset. The categories which occur are window float glass (WinF: 70), window non-float glass (WinNF: 76), vehicle window glass (Veh: 17), containers (Con: 13), tableware (Tabl: 9) and vehicle headlamps (Head: 29).

**Source**

Venables, W. N. and Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth edition. Springer.

## Examples

```
#####
# Compositional regression example with the fgl dataset

# Select the composition and use log
x <- as.matrix(fgl[,2:9])
z <- log(x)

# Fit model
model <- glm(RI ~ z, data = fgl, method = "cirls.fit",
  cons = ~ zerosum(z))

# Coefficients sum to 1
coef(model)
sum(coef(model)[-1])

# Confidence intervals
confint(model)
```

---

logLik.cirls

---

*Extract Log-Likelihood*


---

## Description

Extracts the log-likelihood of a fitted `cirls` object with associated degrees of freedom. Typically used with [AIC](#).

## Usage

```
## S3 method for class 'cirls'
logLik(object, df = "edf", ...)
```

## Arguments

|                     |                                                                                                                                                                                                            |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>object</code> | A <code>cirls</code> object.                                                                                                                                                                               |
| <code>df</code>     | Character. The type of degrees of freedom to assign to the log-Likelihood. Default to expected degrees of freedom "edf". See <a href="#">edf</a> for a description of various types of degrees of freedom. |
| <code>...</code>    | Additional arguments to be passed to <a href="#">edf</a> to compute the degrees of freedom.                                                                                                                |

## Details

The argument `df` provide the type of degrees of freedom attributed to the returned log-likelihood value. This is typically used in the computation of [AIC](#) and [BIC](#), and changing the degrees of freedom can ultimately change the values of the information criteria. By default, the expected number of freedom given the constraints is used. See [edf](#) for details on the computation and for the various types of degrees of freedom.

**Value**

A numeric value of class `logLik` with attributes `df` (degrees of freedom, see details) and `nobs` (number of observations used in the estimation).

**See Also**

[edf](#) to compute expected degrees of freedom.

**Examples**

```
#####
# Example of AIC and logLik

#----- Fit warming model

# Non-decreasing constraint on decadal strata
model <- glm(anomaly ~ decade, data = warming, method = "cirls.fit",
  cons = ~ shape(decade, "inc"))

#----- Log-Likelihood and information criteria

# Extract log-likelihood with different degrees of freedom
(lle <- logLik(model))
(llo <- logLik(model, df = "o"))
(llu <- logLik(model, df = "u"))

# The AIC function can be used directly on the model: uses default df
# NB: the small differences come from the simulations for edf computation
AIC(model)
AIC(lle)

# To compute AIC with different df, need to go through logLik
AIC(llo)
AIC(llu)
```

---

london

---

*Daily mortality, temperature and pollution data in London*


---

**Description**

This dataset contains daily time series of all-cause mortality for several age groups, temperature (maximum, mean, and minimum), and air pollutants (PM10, ozone, and carbon monoxide) for the period of 1996-2003.

**Usage**

```
london
```

**Format**

A data frame with 2922 observations and 12 variables:

**date** Date in the period 1996-2003  
**doy** Day of year as decimal number  
**dow** Day of week  
**death** Counts of all-cause and all-age mortality  
**age0\_64** Counts of mortality for age group 0-64  
**age65\_74** Counts of mortality for age group 65-74  
**age75\_84** Counts of mortality for age group 75-84  
**age85plus** Counts of mortality for age group 85 and older  
**tmax** Daily maximum temperature (in Celsius degree)  
**tmean** Daily mean temperature (in Celsius degree)  
**tmin** Daily minimum temperature (in Celsius degree)  
**pm10** Daily mean coarse particulate matter concentration ( $\mu g/m^3$ )  
**o3** Daily mean ozone concentration ( $\mu g/m^3$ )  
**co** Daily mean carbon monoxide concentration ( $\mu g/m^3$ )

**Source**

Mortality data were provided by the Office for National Statistics (ONS). Environmental data were collected and provided by the British Atmospheric Data Centre (BADC).

**Examples**

```
#####
# Non-negative coefficient example with the london dataset

library(splines)

### Association between Ozone and mortality
# Nonnegative constraint on Ozone
model <- glm(death ~ o3, data = london, family = "quasipoisson",
  method = "cirls.fit", cons = ~ shape(o3, "pos"))

# Coefficient and confidence interval
coef(model)[2]
confint(model)[2,]

# Comparing to an unconstrained model
umodel <- uncons(model)
coef(umodel)[2]
confint(umodel)[2,]
```

---

|            |                                           |
|------------|-------------------------------------------|
| reduceCons | <i>Reduce a set of linear constraints</i> |
|------------|-------------------------------------------|

---

### Description

Internal function checking whether a set of Cmat, lb and ub contains redundant or underlying equality constraints, and attempting to reduce the set accordingly. The objective is to obtain the smallest possible set of constraints without affecting the original problem.

### Usage

```
reduceCons(Cmat, lb = NULL, ub = NULL, redundant = TRUE,
           equality = TRUE, warn = FALSE)
```

### Arguments

|           |                                                                                                                     |
|-----------|---------------------------------------------------------------------------------------------------------------------|
| Cmat      | A constraint matrix.                                                                                                |
| lb, ub    | Bound vectors. If provided, should be consistent with Cmat. if not, default to a vector of 0 and Inf, respectively. |
| redundant | Logical switch indicating whether to check for redundant constraints.                                               |
| equality  | Logical switch indicating whether to check for underlying equality constraints.                                     |
| warn      | Logical indicating if the user should be warned when constraint reduction happens.                                  |

### Details

This function is mostly for internal used, being silently called by [buildCmat](#) and in some Constr functions used to create Cmat/lb/ub. Redundant or underlying equality constraints can result from putting together several sets of constraints together (for example non-decreasing and convex regression) and reducing the constraint matrix can be crucial to allow for inference. Therefore, if this check can be turned off when using `cirls` (see [cirls.control](#)), it is not recommended to do so.

#### Redundant constraint:

In a set of linear constraints  $\mathbf{C}\beta \geq \mathbf{l}$ , the constraint  $k$  is **redundant** if it can be removed without affecting the feasible region of the regression. Redundancy can be checked by finding the minimum possible value for the constraint  $\mathbf{c}_k^T \beta$  subject to all other constraints, i.e. by solving the linear optimisation problem

$$\begin{aligned} \min \quad & \mathbf{c}_k^T \beta \\ \text{s.t.} \quad & \mathbf{C}_{-k} \beta \geq \mathbf{l}_{-k} \end{aligned}$$

where  $\mathbf{C}_{-k}$  and  $\mathbf{l}_{-k}$  are the constraint matrix and bound vector to which the  $k$ th row has been removed.

If the minimum possible value is greater than or equal to  $l_k$ , then the corresponding constraint is redundant. This is easily extended for upper bounds, replacing the min by max in the problem above, and checking the maximum against the upper bound  $u_k$ .

**Underlying equality:**

There is an underlying equality constraint when a constraint is forced to be equal to its lower bound  $l_k$  by other constraints. If the maximum possible value for  $\mathbf{c}_k^T \beta$  subject to all other constraints is equal to  $l_k$ , then it means  $\mathbf{c}_k^T \beta$  is forced to be exactly  $l_k$  and is therefore an underlying equality constraint. This is checked by changing the min to max in the problem above.

**Algorithm:**

This function implements a simple algorithm to reduce the set of constraints. There is an initial check for row of Cmat that only contain zeros, as well as for constraints with infinite bounds only. Then it goes through every constraint with the following steps:

1. Find the minimum and maximum possible values of  $\mathbf{c}_k^T \beta$  using the optimisation problem above.
2. If the minimum is higher than  $l_k$ , remove the bound, and same if the maximum is lower than  $u_k$ .
3. If no bound remains, this constraint is discarded.
4. If bounds remain, check for underlying equality constraint by comparing the minimum/maximum to  $u_k/l_k$ .

This algorithm is similar to the "naive" algorithm of Caron et al. (1989).

**Value**

A list with the following elements:

|            |                                                                                                 |
|------------|-------------------------------------------------------------------------------------------------|
| Cmat/lb/ub | The reduced set of constraints.                                                                 |
| redundant  | Vector of indices indicating the redundant constraints that have been removed.                  |
| equality   | Vector of indices indicating the constraints that were part of underlying equality constraints. |

**Note**

The result can depend on the ordering of Cmat because constraints are checked in the order provided. However, the results will not impact the feasible region in `cirls`, only what the matrix looks like.

**References**

Caron, R.J., McDonald, J.F., Ponc, C.M., 1989. A degenerate extreme point strategy for the classification of linear constraints as redundant or necessary. *J Optim Theory Appl* **62**, 225–237. DOI:10.1007/BF00941055

Telgen, J., 1983. Identifying Redundant Constraints and Implicit Equalities in Systems of Linear Constraints. *Management Science* **29**, 1209–1222. DOI:10.1287/mnsc.29.10.1209

**See Also**

[buildCmat](#) for how constraint matrices are built in `cirls`.

## Examples

```
#####
# Simple example

# Two constraints: one >=0 and one >= 1
Cmat <- as.matrix(rep(1, 2))
lb <- 0:1

# The first one is removed because redundant
reduceCons(Cmat, lb)

#####
# Example of underlying equality constraint

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

# Transformed into a single equality constraint
reduceCons(cmateq)

#####
# An example with shape constraints

library(dlnm)

# Constraints: successive coefficients should increase and be convex
# Intuitively, if the first two coefficients increase,
# then convexity forces the rest to increase which means there is redundancy
basis <- ps(1:10, df = 5)
cmatic <- rbind(
  shapeConstr(basis, shape = "inc")$Cmat, # Increasing
  shapeConstr(basis, shape = "cvx")$Cmat # Convex
)

# Checking indicates that some constraints are redundant
# Returns reduced matrix and a warning
reduceCons(cmatic)

# Compare without removing the redundant constraints
reduceCons(cmatic, redundant = FALSE)

# Note that this is silently done when both "inc" and "cvx" are provided
shapeConstr(basis, shape = c("inc", "cvx"))
```

---

shapeConstr

*Shape constraints*


---

## Description

Builds a constraint matrix and bound vectors for defining shape constraints on a set of coefficients. This is a generic function designed to be used in the [constr](#) formula interface. It allows methods for

a wide range of regression terms (see Details).

### Usage

```
shapeConstr(x, ...)

## Default S3 method:
shapeConstr(x, shape, range = NULL, intercept = FALSE,
  ...)

## S3 method for class 'factor'
shapeConstr(x, shape, range = NULL, intercept = FALSE,
  ...)

## S3 method for class 'bs'
shapeConstr(x, shape, range = NULL, ...)

## S3 method for class 'ns'
shapeConstr(x, shape, ...)

## S3 method for class 'ps'
shapeConstr(x, shape, ...)

## S3 method for class 'onebasis'
shapeConstr(x, shape, ...)

## S3 method for class 'strata'
shapeConstr(x, shape, range = NULL, ...)

## S3 method for class 'lin'
shapeConstr(x, shape, ...)
```

### Arguments

|           |                                                                                                                                                                                                        |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x         | An object representing a design matrix of predictor variables, typically basis functions. See Details for supported objects.                                                                           |
| ...       | Additional parameters passed to or from other methods.                                                                                                                                                 |
| shape     | A character vector indicating one or several shape-constraints. See Details for supported shapes.                                                                                                      |
| range     | A numeric vector of length 2. When specified, only this range is constrained. The default is to not restrict the range.                                                                                |
| intercept | For the default method, a logical value indicating if the design matrix includes an intercept. In most cases, it will be automatically extracted from x, but this argument can be used to override it. |

### Details

This function is used to specify shape constraints on terms in a `cirls` model. Shapes can be enforced through the relation between the coefficients of several variables, for instance dummy in-

dicators to impose shapes on the levels of a factor, or splines for smooth shapes. Note that this function can also be used to easily specify non-negativity or non-positivity constraints. See below for the list of implemented shapes and the examples section for several use cases.

### Usage:

The recommended usage is to use this function through a call to `shape` on a term in the `constr` formula interface. This method is then called internally to create the constraint matrix and bound vectors. However, `shapeConstr` can also be called directly on a matrix-like object to manually build or inspect the constraint matrix.

The parameters necessary to build the constraint matrix (e.g. `knots` and `ord` for splines) are typically extracted from the `x` object. This is also true for the `intercept` for most of the object, except for the default method for which it can be useful to explicitly provide it. In a typical usage in which `shapeConstr` is called from the `constr` argument, `intercept` is automatically determined from the `glm` formula.

### Allowed shapes:

The `shape` argument allows to define a specific shape for the association between the expanded term in `x` and the response of the regression model. This shape can describe the relation between coefficients for the default method, or the shape of the smooth term for spline bases. At the moment, six different shapes are supported, with up to three allowed simultaneously (one from each category):

- `"pos"` or `"neg"`: Positive/Negative. Applies to the full association.
- `"inc"` or `"dec"`: Monotonically Increasing/Decreasing.
- `"cvx"` or `"ccv"`: Convex/Concave.

### Constraining a sub-range:

The `range` argument restricts the constraint over the specified range, leaving the association outside of this range unconstrained. Note that the exact range being constrained depends on the `x` and its specifications. For instance, when `x` represents bases that depend on knots or breakpoints (e.g. `bs`, `ns` or `strata`), the function will constrain between the knots/breakpoints closest to the specified range. In that sense `range` specifies a *minimum* range for the constraint, but most of the time a larger range will be constrained.

### Available methods:

In addition to the default method, `shapeConstr` currently supports methods for several classes, creating an appropriate shape-constraint matrix depending on the object. The full list (also provided by `methods(shapeConstr)`) is:

#### *Categorical variables:*

- `factor()`: for factor objects. Extract the `contrasts` to define the constraint matrix. Here the `intercept` argument has the same interpretation as in the default method, i.e. if set to `TRUE` it means the `glm` model does not include an intercept externally to the factor. Note that, in this case, a simple dummy coding is done in R. Here `range` specifies the levels of constraining, having values between 1 and `nlevel(x)`.
- `strata`: Indicator variables defining strata from the `dlnm` package. Here the shape is applied to the coefficients of the strata, treating it like a categorical variable. Here `range` is specified on the original scale of the variable, treating the strata similarly to spline bases.

#### *B-spline bases:*

- **bs**: B-splines bases from the [splines](#) package.
- **ns**: Natural splines bases from the [splines](#) package.
- **ps**: Penalised splines (P-Splines) from the [dlnm](#) package.

*Distributed-lag linear and nonlinear models (DLNM):*

- **onebasis**: General method for basis functions generated in the package.
- **lin**: Mostly for compatibility with the [dlnm](#) package. Here "pos" and "inc" ("neg" and "dec") have the same interpretation, a non-negative (non-positive) coefficient associated to the linear term.
- **crossbasis**: Allows shape-constrained DLNM. See its dedicated [help page](#).

## Value

A list containing the constraint matrix Cmat, and lower/upper bound vectors (lb and ub, respectively).

## References

Zhou, S. & Wolfe, D. A., 2000. On derivative estimation in spline regression. *Statistica Sinica* **10**, 93–108.

## See Also

[buildCmat](#) detailing the constr interface.

## Examples

```
#####
# Non-negative coefficient example with the london dataset

library(splines)

### Association between Ozone and mortality
# Nonnegative constraint on Ozone
model <- glm(death ~ o3, data = london, family = "quasipoisson",
  method = "cirls.fit", cons = ~ shape(o3, "pos"))

# Coefficient and confidence interval
coef(model)[2]
confint(model)[2,]

# Comparing to an unconstrained model
umodel <- unconstrained(model)
coef(umodel)[2]
confint(umodel)[2,]
#####
# Monotone strata levels with the warming dataset

### Fit the model

# Non-decreasing constraint on decadal strata
```

```

model <- glm(anomaly ~ decade, data = warming, method = "cirls.fit",
  cons = ~ shape(decade, "inc"))

# Plot result
plot(anomaly ~ year, data = warming, xlab = "", ylab = "Temperature anomaly")
lines(warming$year, predict(model), col = 2, lwd = 2)

### Extract results

# Coefficients and confidence intervals
betas <- coef(model)
v <- vcov(model)
cis <- confint(model)

# Degrees of freedom: represent number of strata change
# ?edf
edf(model)

### Compare with an unconstrained model

# Refit the model without constraints
umodel <- uncons(model)

# Add result
lines(warming$year, predict(umodel), col = 3, lwd = 2, lty = 2)

#####
# Monotone splines with the warming dataset

library(splines)

# Define a natural spline basis
basis <- ns(warming$year, df = 10)

### Constrained model

# Non-decreasing splines
model <- glm(anomaly ~ basis, data = warming, method = "cirls.fit",
  cons = ~ shape(basis, "inc"))

# Plot result
plot(anomaly ~ year, data = warming, xlab = "", ylab = "Temperature anomaly")
lines(warming$year, predict(model), col = 2, lwd = 2)

### Unconstrained model
umodel <- uncons(model)
lines(warming$year, predict(umodel), col = 3, lwd = 2, lty = 2)

#####
# Monotone splines with the warming dataset
# Specifying a subrange

library(splines)

```

```

# Define a natural spline basis
basis <- ns(warming$year, df = 10)

### Constrained model

# Non-decreasing splines only for 1950 and later
model <- glm(anomaly ~ basis, data = warming, method = "cirls.fit",
  cons = ~ shape(basis, "inc", range = c(1950, 2015)))

# Plot result
plot(anomaly ~ year, data = warming, xlab = "", ylab = "Temperature anomaly")
lines(warming$year, predict(model), col = 2, lwd = 2)

### Unconstrained model
umodel <- uncons(model)
lines(warming$year, predict(umodel), col = 3, lwd = 2, lty = 2)

```

---

simulCoef

---

*Methods for inference on the coefficients of a cirls object.*


---

## Description

simulCoef simulates coefficients for a fitted cirls object and uses these simulations for inference. confint and vcov directly compute confidence intervals and the variance-covariance matrix for coefficients from a fitted cirls object. These methods for cirls objects supersede the default glm methods.

## Usage

```

simulCoef(object, nsim = 1, seed = NULL, complete = TRUE,
  constrained = TRUE)

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

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

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

## S3 method for class 'sim.cirls'
vcov(object, ...)

```

## Arguments

|             |                                                                                                                                                                                                                                                                          |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object      | An object of class <code>cirls</code> or <code>sim.cirls</code> .                                                                                                                                                                                                        |
| nsim        | The number of simulations to perform.                                                                                                                                                                                                                                    |
| seed        | Either NULL or an integer that will be used in a call to <code>set.seed()</code> before simulating the coefficients.                                                                                                                                                     |
| complete    | If FALSE, it does not return inference for undetermined coefficients in case of an over-determined model.                                                                                                                                                                |
| constrained | A logical switch indicating whether to simulate from the constrained (the default) or unconstrained (see <code>uncons</code> ) coefficients distribution. When set to FALSE in <code>vcov</code> , returns the untruncated covariance matrix as in an unconstrained GLM. |
| parm        | A specification of which parameters to compute the confidence intervals for. Either a vector of index 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. For <code>vcov</code> and <code>confint</code> , it can be used to provide a seed for the internal coefficient simulation.                                                                                            |

## Details

### Coefficient inference:

To perform inference for coefficients the `simulCoef` function simulates from the distribution of  $\mathbf{C}\beta$  which follows a **Truncated Multivariate Normal** distribution  $TMVN(\mathbf{C}\beta^*, \mathbf{C}\Sigma^*\mathbf{C}^T, \mathbf{l}, \mathbf{u})$  where  $\mathbf{C}$  is the constraint matrix with bound vectors  $\mathbf{l}$  and  $\mathbf{u}$ , and  $\beta^*$  and  $\Sigma^*$  are the unconstrained coefficient vector and variance matrix. The TMVN simulations are then back-transformed to the domain of  $\beta$  to allow for inference.

### Functions:

`simulCoef` is the workhorse of the inference, performing the simulations described above. It is called internally by `confint.cirls` and `vcov.cirls` to compute confidence intervals and the variance-covariance matrix, respectively. These are custom methods for `cirls` objects that supersede the default methods used for `glm` objects. Alternatively `simulCoef` can be used directly to simulate `nsim` coefficient to then be passed to `vcov.sim.cirls` and `confint.sim.cirls` methods. This avoids simulating several times when both confidence intervals and variance-covariance are needed for instance.

## Value

For `simulCoef`: a `sim.cirls` object, that is a matrix of `nsim` rows containing simulated coefficients with the attributes `seed`, `complete` and `constrained`.

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

For `vcov` methods: a matrix of the estimated covariance between the parameter estimates of the model.

## Note

These methods only work when there are less constraints than variables in cirls model, i.e. when Cmat has less rows than columns.

## 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:10.1007/978-1-4612-2414-3\_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. DOI:10.1111/rssb.12162

## See Also

[rtmvnorm](#) which is the routine used internally to simulate from a TMVN. [reduceCons](#) to reduce the set of constraints.

## Examples

```
#####
# Monotone strata levels with the warming dataset

### Typical usage

# Non-decreasing constraint on decadal strata
model <- glm(anomaly ~ decade, data = warming, method = "cirls.fit",
  cons = ~ shape(decade, "inc"))

# Extract covariance matrix and confidence intervals
v <- vcov(model)
ci <- confint(model)

### Using the sim.cirls objects

# Simulate coefficients directly
sims <- simulCoef(model, nsim = 1000, seed = 4)

# Get covariance matrix and confidence intervals
vs <- vcov(sims)
cis <- confint(sims)

# Check that we get the same results as the typical usage
v2 <- vcov(model, seed = 4)
ci2 <- confint(model, seed = 4)
identical(vs, v2)
identical(cis, ci2)
```

uncons

*Unconstrained model***Description**

Takes a fitted `cirls` object and fits the corresponding unconstrained model.

**Usage**

```
uncons(object)
```

**Arguments**

`object`            A `cirls` object.

**Details**

This function refits the original model that produced a `cirls` object, but removing all constraints. This is primarily used by [simulCoef](#) for inference, but it can also be used to easily compare a constrained and an unconstrained models.

The function still fits the model through [cirls.fit](#), but using an empty constraint matrix. Therefore, it returns a `cirls` object that can use all the facilities provided by the `cirls` package. In this instance, the CIRLS algorithm reduces to a classical IRLS and the results are identical to a usual [glm](#) fitted with `glm.fit`.

**Value**

A `cirls` object.

**Note**

If any starting values were provided to fit the `cirls` object, they are not transferred to the fitting of the unconstrained model.

**See Also**

[simulCoef](#) to perform inference.

**Examples**

```
#####
# Monotone strata levels with the warming dataset

### Fit the model

# Non-decreasing constraint on decadal strata
model <- glm(anomaly ~ decade, data = warming, method = "cirls.fit",
  cons = ~ shape(decade, "inc"))
```

```

# Plot result
plot(anomaly ~ year, data = warming, xlab = "", ylab = "Temperature anomaly")
lines(warming$year, predict(model), col = 2, lwd = 2)

### Extract results

# Coefficients and confidence intervals
betas <- coef(model)
v <- vcov(model)
cis <- confint(model)

# Degrees of freedom: represent number of strata change
# ?edf
edf(model)

### Compare with an unconstrained model

# Refit the model without constraints
umodel <- uncons(model)

# Add result
lines(warming$year, predict(umodel), col = 3, lwd = 2, lty = 2)

```

---

warming

*Global temperature anomaly*


---

## Description

Provides annual global temperature anomaly compared to the reference period 1961-1990 from 1850 to 2015.

## Usage

```
warming
```

## Format

A data frame of 166 observations (years) and three variables:

**year** The calendar year.

**decade** The related decade.

**anomaly** The difference between the global annual temperature and the average over 1961-1990.

## Source

Jones, P.D., Parker, D.E., Osborn, T.J., Briffa, K.R., 2000. Global and Hemispheric Temperature Anomalies: Land and Marine Instrumental Records (1850 - 2015). DOI:10.3334/CDIAC/CLI.002

## Examples

```
#####
# Monotone strata levels with the warming dataset

### Fit the model

# Non-decreasing constraint on decadal strata
model <- glm(anomaly ~ decade, data = warming, method = "cirls.fit",
  cons = ~ shape(decade, "inc"))

# Plot result
plot(anomaly ~ year, data = warming, xlab = "", ylab = "Temperature anomaly")
lines(warming$year, predict(model), col = 2, lwd = 2)

### Extract results

# Coefficients and confidence intervals
betas <- coef(model)
v <- vcov(model)
cis <- confint(model)

# Degrees of freedom: represent number of strata change
# ?edf
edf(model)

### Compare with an unconstrained model

# Refit the model without constraints
umodel <- uncons(model)

# Add result
lines(warming$year, predict(umodel), col = 3, lwd = 2, lty = 2)
```

---

zerosumConstr

*Specify Zero-sum constraints*

---

## Description

Builds a constraint matrix and bound vectors for defining zero-sum constraints on a set of coefficients. This is a generic function designed to be used in the [constr](#) formula interface.

## Usage

```
zerosumConstr(..., group = FALSE)
```

## Arguments

|       |                                                                                |
|-------|--------------------------------------------------------------------------------|
| ...   | Terms to be included in the constraint.                                        |
| group | If set to TRUE, the constraint is build independently for each variable in ... |

## Details

This function is used to force one or several set(s) of coefficients to sum to zero. The main use is for when (log) compositions are used as predictors in the model, but it can also be used more generally for variables that are relative to a reference.

### Usage:

The recommended usage is to use this function through a call to `zerosum` on a term in the `constr` formula interface. This method is then called internally to create the constraint matrix and bound vectors. However, `zerosumConstr` can also be called directly on a matrix-like object to manually build or inspect the constraint matrix.

## Value

A list containing the constraint matrix `Cmat`, and lower/upper bound vectors (`lb` and `ub`, respectively).

## References

Altenbuchinger, M., Rehberg, T., Zacharias, H.U., Stämmeler, F., Dettmer, K., Weber, D., Hiergeist, A., Gessner, A., Holler, E., Oefner, P.J., Spang, R., 2017. Reference point insensitive molecular data analysis. *Bioinformatics* **33**, 219–226. DOI:10.1093/bioinformatics/btw598

Aitchison, J., Bacon-Shone, J., 1984. Log contrast models for experiments with mixtures. *Biometrika* **71**, 323–330. DOI:10.1093/biomet/71.2.323

## See Also

`buildCmat` detailing the `constr` interface.

## Examples

```
#####
# Compositional regression example with the fgl dataset

# Select the composition and use log
x <- as.matrix(fgl[,2:9])
z <- log(x)

# Fit model
model <- glm(RI ~ z, data = fgl, method = "cirls.fit",
  cons = ~ zerosum(z))

# Coefficients sum to 1
coef(model)
sum(coef(model)[-1])

# Confidence intervals
confint(model)
```

# Index

## \* datasets

- fgl, [28](#)
- london, [30](#)
- warming, [43](#)

AIC, [3](#), [29](#)

B-splines, [6](#)

BIC, [3](#), [29](#)

bound, [4](#), [11](#)

bound (boundConstr), [6](#)

boundConstr, [6](#), [10–12](#), [15](#), [16](#)

boundConstr.bs, [9](#)

boundConstr.crossbasis, [10](#)

boundConstr.ns (boundConstr.bs), [9](#)

boundConstr.onebasis, [15](#)

boundConstr.ps (boundConstr.bs), [9](#)

boundConstr.strata

(boundConstr.onebasis), [15](#)

bs, [37](#)

bs, ns, ps, [8](#)

buildCmat, [3](#), [8](#), [12](#), [16](#), [20](#), [21](#), [23](#), [24](#), [32](#), [33](#),  
[37](#), [45](#)

cirls, [18](#), [21](#), [26](#), [40](#)

cirls (cirls-package), [2](#)

cirls-package, [2](#)

cirls.control, [3](#), [20](#), [21](#), [23](#), [24](#), [32](#)

cirls.fit, [3](#), [17](#), [20](#), [21](#), [22](#), [42](#)

Cmat/lb/ub, [3](#)

coef, [3](#)

confidence intervals, [2](#), [3](#)

confint, [4](#)

confint.cirls (simulCoef), [39](#)

confint.sim.cirls (simulCoef), [39](#)

constr, [3](#), [6](#), [7](#), [10](#), [12](#), [34](#), [36](#), [44](#), [45](#)

constrained iteratively-reweighted  
least-squares, [2](#)

contrasts, [8](#), [36](#)

crossbasis, [8](#), [10](#), [12](#), [16](#), [37](#)

crosspred, [12](#)

crossreduce, [11](#)

degrees of freedom, [4](#)

dlnm, [8](#), [15](#), [36](#), [37](#)

edf, [3](#), [26](#), [29](#), [30](#)

factor, [8](#)

factor(), [36](#)

family, [23](#)

fgl, [4](#), [28](#), [28](#)

generic, [9](#)

glm, [2](#), [3](#), [7](#), [20–24](#), [36](#), [40](#), [42](#)

glm.fit, [23](#)

help page, [8](#), [18](#), [37](#)

lin, [37](#)

logLik, [3](#), [4](#)

logLik.cirls, [27](#), [29](#)

london, [4](#), [30](#)

model.frame, [17](#)

model.offset, [23](#)

natural, [9](#)

ns, [37](#)

onebasis, [8](#), [15](#), [37](#)

P-splines, [15](#)

penalised, [9](#)

ps, [37](#)

qprog, [21](#)

qr, [20](#)

reduceCons, [3](#), [17](#), [18](#), [21](#), [32](#), [41](#)

rtmvnorm, [41](#)

set.seed, [26](#)  
set.seed(), [40](#)  
shape, [4](#), [11](#)  
shape(shapeConstr), [34](#)  
shapeConstr, [11](#), [12](#), [34](#)  
shapeConstr.crossbasis  
    (boundConstr.crossbasis), [10](#)  
simulating, [2](#)  
simulCoef, [3](#), [26](#), [27](#), [39](#), [42](#)  
solve.QP, [21](#)  
solve\_osqp, [21](#)  
spline bases, [7](#)  
splines, [37](#)  
strata, [8](#), [15](#), [36](#)  
summary, [3](#)  
  
uncons, [3](#), [40](#), [42](#)  
unconstrained, [26](#)  
update, [3](#)  
  
variance-covariance matrix, [2](#), [3](#)  
vcov, [4](#)  
vcov.cirls(simulCoef), [39](#)  
vcov.sim.cirls(simulCoef), [39](#)  
  
warming, [4](#), [43](#)  
  
zerosum, [4](#)  
zerosum(zerosumConstr), [44](#)  
zerosumConstr, [44](#)