

# Package ‘adelie’

February 28, 2025

**Title** Group Lasso and Elastic Net Solver for Generalized Linear Models

**Version** 1.0.7

**Date** 2025-02-27

**Description** Extremely efficient procedures for fitting the entire group lasso and group elastic net regularization path for GLMs, multinomial, the Cox model and multi-task Gaussian models. Similar to the R package 'glmnet' in scope of models, and in computational speed. This package provides R bindings to the C++ code underlying the corresponding Python package 'adelie'. These bindings offer a general purpose group elastic net solver, a wide range of matrix classes that can exploit special structure to allow large-scale inputs, and an assortment of generalized linear model classes for fitting various types of data. The package is an implementation of Yang, J. and Hastie, T. (2024) <[doi:10.48550/arXiv.2405.08631](https://doi.org/10.48550/arXiv.2405.08631)>.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**LinkingTo** Rcpp, RcppEigen

**SystemRequirements** C++17

**Imports** Matrix, r2r, Rcpp, methods, stringr, utils

**Suggests** ggplot2, gridExtra, testthat (>= 3.0.0), knitr, rmarkdown

**Config/testthat/edition** 3

**VignetteBuilder** knitr

**URL** <https://github.com/JamesYang007/adelie-r>

**BugReports** <https://github.com/JamesYang007/adelie-r/issues>

**NeedsCompilation** yes

**Author** James Yang [aut, cph],  
Trevor Hastie [aut, cph, cre],  
Balasubramanian Narasimhan [aut]

**Maintainer** Trevor Hastie <[hastie@stanford.edu](mailto:hastie@stanford.edu)>

**Repository** CRAN

**Date/Publication** 2025-02-28 17:50:02 UTC

## Contents

constraint.box . . . . .	3
cv.glintnet . . . . .	4
cv.grpnet . . . . .	6
gaussian_cov . . . . .	9
glintnet . . . . .	11
glm.binomial . . . . .	13
glm.cox . . . . .	14
glm.gaussian . . . . .	15
glm.multigaussian . . . . .	16
glm.multinomial . . . . .	17
glm.poisson . . . . .	18
grpnet . . . . .	19
io.snp_phased_ancestry . . . . .	22
io.snp_unphased . . . . .	23
matrix.block_diag . . . . .	24
matrix.concatenate . . . . .	25
matrix.convex_relu . . . . .	26
matrix.dense . . . . .	27
matrix.eager_cov . . . . .	28
matrix.interaction . . . . .	28
matrix.kronecker_eye . . . . .	29
matrix.lazy_cov . . . . .	30
matrix.one_hot . . . . .	31
matrix.snp_phased_ancestry . . . . .	32
matrix.snp_unphased . . . . .	33
matrix.sparse . . . . .	34
matrix.standardize . . . . .	35
matrix.subset . . . . .	36
plot.cv.glintnet . . . . .	37
plot.grpnet . . . . .	38
predict.cv.glintnet . . . . .	40
predict.cv.grpnet . . . . .	41
predict.glintnet . . . . .	42
predict.grpnet . . . . .	44
print.cv.grpnet . . . . .	46
print.glintnet . . . . .	47
print.grpnet . . . . .	48
set_configs . . . . .	49

---

constraint.box	<i>Create a box constraint for a group.</i>
----------------	---

---

### Description

A box constraint sets upper and lower bounds for coefficients in a model. This is done per group, and this function is used separately to set the bounds for each group in the model. The constraints are returned as a list, with number of elements the number of groups. List entries can be NULL, which means no constraints for that group. Currently works with single-response models (so `glm.multinomial` and `glm.multigaussian` are excluded). Note that for each group for which non-null constraints are provide, a separate call to `constraint.box()` must be made (i.e. the constraint object cannot be replicated). See the second example below.

### Usage

```
constraint.box(lower, upper, max_iters = 100, tol = 1e-09)
```

### Arguments

lower	lower bound for each coefficient in the group. If the group has $m$ variables, this should be a vector of length $m$ . Values can be <code>-Inf</code> .
upper	upper bound for each coefficient in the group. If the group has $m$ variables, this should be a vector of length $m$ . Values can be <code>Inf</code> .
max_iters	maximum number of proximal Newton iterations; default is 100.
tol	convergence tolerance for proximal Newton; default is <code>1e-9</code> .

### Value

Box constraint object.

### Author(s)

Trevor Hastie and James Yang  
Maintainer: Trevor Hastie [hastie@stanford.edu](mailto:hastie@stanford.edu)

### Examples

```
# Group of length 10, with positivity constraint on all the coefficients.
lower <- rep(0,10)
upper <- rep(Inf,10)
cont <- constraint.box(lower = lower, upper = upper)

# 10 singleton groups, and non-negativity constraints on all parameters.
cont <- lapply(1:10, function(i)constraint.box(lower = 0, upper = Inf))

# Same as above, but non-negativity constraints only on first 5 parameters.
cont <- lapply(1:5, function(i)constraint.box(lower = 0, upper = Inf))
cont <- c(cont, rep(list(NULL), 5)) # rep rule does not apply to NULL
```

cv.glintnet

*Cross-validation for glintnet***Description**

Does k-fold cross-validation for glintnet

**Usage**

```
cv.glintnet(
  X,
  glm,
  offsets = NULL,
  intr_keys = NULL,
  intr_values,
  levels = NULL,
  n_folds = 10,
  foldid = NULL,
  n_threads = 1,
  ...
)
```

**Arguments**

X	Feature matrix. Either a regular R matrix, or else an <code>adelle</code> custom matrix class, or a concatenation of such.
glm	GLM family/response object. This is an expression that represents the family, the response and other arguments such as weights, if present. The choices are <code>glm.gaussian()</code> , <code>glm.binomial()</code> , <code>glm.poisson()</code> , <code>glm.multinomial()</code> , <code>glm.cox()</code> , <code>glm.multinomial()</code> , and <code>glm.multigaussian()</code> . This is a required argument, and there is no default. In the simple example below, we use <code>glm.gaussian(y)</code> .
offsets	Offsets, default is <code>NULL</code> . If present, this is a fixed vector or matrix corresponding to the shape of the natural parameter, and is added to the fit.
intr_keys	List of feature indices. This is a list of all features with which interactions can be formed. Default is <code>1:p</code> where <code>p</code> is the number of columns in <code>X</code> .
intr_values	List of integer vectors of feature indices. For each of the $m \leq p$ indices listed in <code>intr_keys</code> , there is a vector of indices indicating which columns are candidates for interaction with that feature. If a vector is <code>NULL</code> , that means all other features are candidates for interactions. The default is a list of length <code>m</code> where each element is <code>NULL</code> ; that is <code>rep(list(NULL), m)</code> .
levels	Number of levels for each of the columns of <code>mat</code> , with 1 representing a quantitative feature. A factor with <code>K</code> levels should be represented by the numbers <code>0, 1, ..., K-1</code> .
n_folds	(default 10). Although <code>n_folds</code> can be as large as the sample size (leave-one-out CV), it is not recommended for large datasets. Smallest value allowable is <code>n_folds=3</code> .

foldid	An optional vector of values between 1 and n_folds identifying what fold each observation is in. If supplied, n_folds can be missing.
n_threads	Number of threads, default 1.
...	Additional named arguments to grpnet.

### Details

The function runs `glintnet`  $n\_folds+1$  times; the first to get the lambda sequence, and then the remainder to compute the fit with each of the folds omitted. The out-of-fold deviance is accumulated, and the average deviance and standard deviation over the folds is computed. Note that `cv.glintnet` does NOT search for values for alpha. A specific value should be supplied, else  $\alpha=1$  is assumed by default. If users would like to cross-validate alpha as well, they should call `cv.glintnet` with a pre-computed vector `foldid`, and then use this same `foldid` vector in separate calls to `cv.glintnet` with different values of alpha. Note also that the results of `cv.glintnet` are random, since the folds are selected at random. Users can reduce this randomness by running `cv.glintnet` many times, and averaging the error curves.

### Value

A list of class "glintnet", which inherits from class "grpnet". This has a few additional components such as `pairs`, `groups` and `levels`. Users typically use methods like `predict()`, `print()`, `plot()` etc to examine the object.

### Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan  
 Maintainer: Trevor Hastie <hastie@stanford.edu>

### References

- Lim, Michael and Hastie, Trevor (2015) *Learning interactions via hierarchical group-lasso regularization*, JCGS doi:10.1080/10618600.2014.938812
- Yang, James and Hastie, Trevor. (2024) A Fast and Scalable Pathwise-Solver for Group Lasso and Elastic Net Penalized Regression via Block-Coordinate Descent. arXiv doi:10.48550/arXiv.2405.08631.
- Friedman, J., Hastie, T. and Tibshirani, R. (2008) *Regularization Paths for Generalized Linear Models via Coordinate Descent (2010)*, *Journal of Statistical Software*, Vol. 33(1), 1-22, doi:10.18637/jss.v033.i01.
- Simon, N., Friedman, J., Hastie, T. and Tibshirani, R. (2011) *Regularization Paths for Cox's Proportional Hazards Model via Coordinate Descent*, *Journal of Statistical Software*, Vol. 39(5), 1-13, doi:10.18637/jss.v039.i05.
- Tibshirani, Robert, Bien, J., Friedman, J., Hastie, T., Simon, N., Taylor, J. and Tibshirani, Ryan. (2012) *Strong Rules for Discarding Predictors in Lasso-type Problems*, *JRSSB*, Vol. 74(2), 245-266, <https://arxiv.org/abs/1011.2234>.

### See Also

`cv.glintnet`, `predict.glintnet`, `plot.glintnet`, `print.glintnet`.

**Examples**

```

set.seed(0)
n=500
d_cont = 5      # number of continuous features
d_disc = 5      # number of categorical features
Z_cont = matrix(rnorm(n*d_cont), n, d_cont)
levels = sample(2:5,d_disc, replace = TRUE)
Z_disc = matrix(0,n,d_disc)
for(i in seq(d_disc))Z_disc[,i] = sample(0:(levels[i]-1),n,replace=TRUE)
Z = cbind(Z_cont,Z_disc)
levels = c(rep(1,d_cont),levels)

xmat = model.matrix(~Z_cont[,1]*factor(Z_disc[,2]))
nc=ncol(xmat)
beta = rnorm(nc)
y = xmat%%beta+rnorm(n)*1.5

cvfit <- cv.glintnet(Z, glm.gaussian(y), levels=levels, intr_keys = 1)
plot(cvfit)
predict(cvfit, newx=Z[1:5,])

```

---

cv.grpnet

*Cross-validation for grpnet*


---

**Description**

Does k-fold cross-validation for grpnet

**Usage**

```

cv.grpnet(
  X,
  glm,
  n_folds = 10,
  foldid = NULL,
  min_ratio = 0.01,
  lmda_path_size = 100,
  offsets = NULL,
  progress_bar = FALSE,
  n_threads = 1,
  ...
)

```

**Arguments**

**X** Feature matrix. Either a regular R matrix, or else an adelic custom matrix class, or a concatenation of such.

glm	GLM family/response object. This is an expression that represents the family, the response and other arguments such as weights, if present. The choices are <code>glm.gaussian()</code> , <code>glm.binomial()</code> , <code>glm.poisson()</code> , <code>glm.multinomial()</code> , <code>glm.cox()</code> , <code>glm.multinomial()</code> , and <code>glm.multigaussian()</code> . This is a required argument, and there is no default. In the simple example below, we use <code>glm.gaussian(y)</code> .
n_folds	(default 10). Although <code>n_folds</code> can be as large as the sample size (leave-one-out CV), it is not recommended for large datasets. Smallest value allowable is <code>n_folds=3</code> .
foldid	An optional vector of values between 1 and <code>n_folds</code> identifying what fold each observation is in. If supplied, <code>n_folds</code> can be missing.
min_ratio	Ratio between smallest and largest value of lambda. Default is 1e-2.
lmda_path_size	Number of values for lambda, if generated automatically. Default is 100.
offsets	Offsets, default is NULL. If present, this is a fixed vector or matrix corresponding to the shape of the natural parameter, and is added to the fit.
progress_bar	Progress bar. Default is FALSE.
n_threads	Number of threads, default 1.
...	Other arguments that can be passed to <code>grpnet</code>

### Details

The function runs `grpnet` `n_folds+1` times; the first to get the lambda sequence, and then the remainder to compute the fit with each of the folds omitted. The out-of-fold deviance is accumulated, and the average deviance and standard deviation over the folds is computed. Note that `cv.grpnet` does NOT search for values for alpha. A specific value should be supplied, else `alpha = 1` is assumed by default. If users would like to cross-validate alpha as well, they should call `cv.grpnet` with a pre-computed vector `foldid`, and then use this same `foldid` vector in separate calls to `cv.grpnet` with different values of alpha. Note also that the results of `cv.grpnet` are random, since the folds are selected at random (unless supplied via `foldid`). Users can reduce this randomness by running `cv.grpnet` many times, and averaging the error curves.

### Value

an object of class "`cv.grpnet`" is returned, which is a list with the ingredients of the cross-validation fit.

lambda	the values of lambda used in the fits.
cvm	The mean cross-validated deviance - a vector of length <code>length(lambda)</code> .
cvsd	estimate of standard error of <code>cvm</code> .
cvup	upper curve = <code>cvm+cvsd</code> .
cvlo	lower curve = <code>cvm-cvsd</code> .
nzero	number of non-zero coefficients at each lambda.
name	a text string indicating type of measure (for plotting purposes). Currently this is "deviance"
grpnet.fit	a fitted <code>grpnet</code> object for the full data.

lambda.min	value of lambda that gives minimum cvm.
lambda.1se	largest value of lambda such that mean deviance is within 1 standard error of the minimum.
index	a one column matrix with the indices of lambda.min and lambda.1se in the sequence of coefficients, fits etc.

### Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan  
 Maintainer: Trevor Hastie <hastie@stanford.edu>

### References

- Yang, James and Hastie, Trevor. (2024) A Fast and Scalable Pathwise-Solver for Group Lasso and Elastic Net Penalized Regression via Block-Coordinate Descent. arXiv [doi:10.48550/arXiv.2405.08631](https://arxiv.org/abs/2405.08631).
- Friedman, J., Hastie, T. and Tibshirani, R. (2008) *Regularization Paths for Generalized Linear Models via Coordinate Descent (2010)*, *Journal of Statistical Software*, Vol. 33(1), 1-22, [doi:10.18637/jss.v033.i01](https://doi.org/10.18637/jss.v033.i01).
- Simon, N., Friedman, J., Hastie, T. and Tibshirani, R. (2011) *Regularization Paths for Cox's Proportional Hazards Model via Coordinate Descent*, *Journal of Statistical Software*, Vol. 39(5), 1-13, [doi:10.18637/jss.v039.i05](https://doi.org/10.18637/jss.v039.i05).
- Tibshirani, Robert, Bien, J., Friedman, J., Hastie, T., Simon, N., Taylor, J. and Tibshirani, Ryan. (2012) *Strong Rules for Discarding Predictors in Lasso-type Problems*, *JRSSB*, Vol. 74(2), 245-266, <https://arxiv.org/abs/1011.2234>.

### See Also

print.cv.grpnet, predict.cv.grpnet, coef.cv.grpnet, plot.cv.grpnet.

### Examples

```
set.seed(0)
n <- 100
p <- 200
X <- matrix(rnorm(n * p), n, p)
y <- X[,1:25] %*% rnorm(25)/4 + rnorm(n)
groups <- c(1, sample(2:199, 60, replace = FALSE))
groups <- sort(groups)
cvfit <- cv.grpnet(X, glm.gaussian(y), groups = groups)
print(cvfit)
plot(cvfit)
predict(cvfit, newx = X[1:5,])
predict(cvfit, type = "nonzero")
```



---

gaussian_cov	<i>Solves group elastic net via covariance method.</i>
--------------	--

---

### Description

Solves group elastic net via covariance method.

### Usage

```
gaussian_cov(  
  A,  
  v,  
  constraints = NULL,  
  groups = NULL,  
  alpha = 1,  
  penalty = NULL,  
  lmda_path = NULL,  
  max_iters = as.integer(1e+05),  
  tol = 1e-07,  
  rdev_tol = 0.001,  
  newton_tol = 1e-12,  
  newton_max_iters = 1000,  
  n_threads = 1,  
  early_exit = TRUE,  
  screen_rule = "pivot",  
  min_ratio = 0.01,  
  lmda_path_size = 100,  
  max_screen_size = NULL,  
  max_active_size = NULL,  
  pivot_subset_ratio = 0.1,  
  pivot_subset_min = 1,  
  pivot_slack_ratio = 1.25,  
  check_state = FALSE,  
  progress_bar = FALSE,  
  warm_start = NULL  
)
```

### Arguments

A	Positive semi-definite matrix.
v	Linear term.
constraints	Constraints.
groups	Groups.
alpha	Elastic net parameter.
penalty	Penalty factor.

<code>lmda_path</code>	The regularization path.
<code>max_iters</code>	Maximum number of coordinate descents.
<code>tol</code>	Coordinate descent convergence tolerance.
<code>rdev_tol</code>	Relative percent deviance explained tolerance.
<code>newton_tol</code>	Convergence tolerance for the BCD update.
<code>newton_max_iters</code>	Maximum number of iterations for the BCD update.
<code>n_threads</code>	Number of threads.
<code>early_exit</code>	TRUE if the function should exit early.
<code>screen_rule</code>	Screen rule (currently the only value is the default "pivot").
<code>min_ratio</code>	Ratio between largest and smallest regularization parameter, default is 0.01.
<code>lmda_path_size</code>	Number of regularization steps in the path, default is 100.
<code>max_screen_size</code>	Maximum number of screen groups, default is NULL for no maximum.
<code>max_active_size</code>	Maximum number of active groups, default is NULL for no maximum.
<code>pivot_subset_ratio</code>	Subset ratio of pivot rule, default is 0.1.
<code>pivot_subset_min</code>	Minimum subset of pivot rule, default is 1.
<code>pivot_slack_ratio</code>	Slack ratio of pivot rule, default is 1.25.
<code>check_state</code>	Check state, default is FALSE.
<code>progress_bar</code>	Progress bar, default is FALSE.
<code>warm_start</code>	Warm start, default is NULL (no warm start).

**Value**

State of the solver.

**Examples**

```

set.seed(0)
n <- 100
p <- 200
X <- matrix(rnorm(n * p), n, p)
y <- X[,1] * rnorm(1) + rnorm(n)
A <- t(X) %*% X / n
v <- t(X) %*% y / n
state <- gaussian_cov(A, v)

```

---

glintnet	<i>fit a GLM interaction model with group lasso or group elastic-net regularization</i>
----------	---

---

## Description

This function is an implementation of the `glinternet` model of Lim and Hastie, for fitting interactions between pairs of variables in a model. The method creates *interaction matrices* and enforces hierarchy using the *overlap group lasso*. Once the augmented model matrix is set up, `glintnet` uses `grpnet` to fit the overlap group lasso path. It hence inherits all the capabilities of `grpnet`, and in particular can fit interaction models for all the GLM families.

## Usage

```
glintnet(
  X,
  glm,
  offsets = NULL,
  intr_keys = NULL,
  intr_values,
  levels = NULL,
  n_threads = 1,
  save.X = FALSE,
  ...
)
```

## Arguments

<code>X</code>	A dense matrix, which can include factors with levels coded as non-negative integers starting at 0.
<code>glm</code>	GLM family/response object. This is an expression that represents the family, the response and other arguments such as weights, if present. The choices are <code>glm.gaussian()</code> , <code>glm.binomial()</code> , <code>glm.poisson()</code> , <code>glm.multinomial()</code> , <code>glm.cox()</code> , <code>glm.multinomial()</code> , and <code>glm.multigaussian()</code> . This is a required argument, and there is no default. In the simple example below, we use <code>glm.gaussian(y)</code> .
<code>offsets</code>	Offsets, default is <code>NULL</code> . If present, this is a fixed vector or matrix corresponding to the shape of the natural parameter, and is added to the fit.
<code>intr_keys</code>	List of feature indices. This is a list of all features with which interactions can be formed. Default is <code>1:p</code> where <code>p</code> is the number of columns in <code>X</code> .
<code>intr_values</code>	List of integer vectors of feature indices. For each of the $m \leq p$ indices listed in <code>intr_keys</code> , there is a vector of indices indicating which columns are candidates for interaction with that feature. If a vector is <code>NULL</code> , that means all other features are candidates for interactions. The default is a list of length <code>m</code> where each element is <code>NULL</code> ; that is <code>rep(list(NULL), m)</code> .

levels	Number of levels for each of the columns of mat, with 1 representing a quantitative feature. A factor with K levels should be represented by the numbers 0, 1, . . . , K-1.
n_threads	Number of threads, default 1.
save.X	Logical flag, default FALSE. If TRUE, the internally constructed X matrix is returned.
. . .	Additional named arguments to grpnet.

### Details

The input matrix can be composed of quantitative variables or columns representing factors. The argument `levels` indicates which are quantitative, and which are factors. The later are represented by numbers starting at 0, up to one less than the number of levels (sorry!) Each of the factors are converted to "one-hot" matrices, and hence a group of columns are created for each of these. This is done using the matrix utility function `matrix.one_hot()`. In addition interaction matrices are created. For each pair of variables for which an interaction is considered, a matrix is created consisting of the cross-product of each of the constituent matrices, as described in the "glinternet" reference. Once this much bigger matrix is established, the model is handed to `grpnet` to produce the fit.

### Value

A list of class "glintnet", which inherits from class "grpnet". This has a few additional components such as `pairs`, `groups` and `levels`. Users typically use methods like `predict()`, `print()`, `plot()` etc to examine the object.

### Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan  
 Maintainer: Trevor Hastie <hastie@stanford.edu>

### References

- Lim, Michael and Hastie, Trevor (2015) *Learning interactions via hierarchical group-lasso regularization*, JCGS doi:10.1080/10618600.2014.938812
- Yang, James and Hastie, Trevor. (2024) A Fast and Scalable Pathwise-Solver for Group Lasso and Elastic Net Penalized Regression via Block-Coordinate Descent. arXiv doi:10.48550/arXiv.2405.08631.
- Friedman, J., Hastie, T. and Tibshirani, R. (2008) *Regularization Paths for Generalized Linear Models via Coordinate Descent (2010)*, *Journal of Statistical Software*, Vol. 33(1), 1-22, doi:10.18637/jss.v033.i01.
- Simon, N., Friedman, J., Hastie, T. and Tibshirani, R. (2011) *Regularization Paths for Cox's Proportional Hazards Model via Coordinate Descent*, *Journal of Statistical Software*, Vol. 39(5), 1-13, doi:10.18637/jss.v039.i05.
- Tibshirani, Robert, Bien, J., Friedman, J., Hastie, T., Simon, N., Taylor, J. and Tibshirani, Ryan. (2012) *Strong Rules for Discarding Predictors in Lasso-type Problems*, *JRSSB*, Vol. 74(2), 245-266, <https://arxiv.org/abs/1011.2234>.

**See Also**

cv.glintnet, predict.glintnet, plot.glintnet, print.glintnet.

**Examples**

```
set.seed(0)
n=500
d_cont = 5      # number of continuous features
d_disc = 5      # number of categorical features
Z_cont = matrix(rnorm(n*d_cont), n, d_cont)
levels = sample(2:5,d_disc, replace = TRUE)
Z_disc = matrix(0,n,d_disc)
for(i in seq(d_disc))Z_disc[,i] = sample(0:(levels[i]-1),n,replace=TRUE)
Z = cbind(Z_cont,Z_disc)
levels = c(rep(1,d_cont),levels)

xmat = model.matrix(~Z_cont[,1]*factor(Z_disc[,2]))
nc=ncol(xmat)
beta = rnorm(nc)
y = xmat%%beta+rnorm(n)*1.5

fit <- glintnet(Z, glm.gaussian(y), levels=levels, intr_keys = 1)
print(fit)
```

---

glm.binomial

*Creates a Binomial GLM family object.*

---

**Description**

A GLM family object specifies the type of model fit, provides the appropriate response object and makes sure it is represented in the right form for the model family, and allows for optional parameters such as a weight vector.

**Usage**

```
glm.binomial(y, weights = NULL, link = "logit")
```

**Arguments**

**y** Binary response vector, with values 0 or 1, or a logical vector. Alternatively, if data are represented by a two-column matrix of proportions (with row-sums = 1), then one can provide one of the columns as the response. This is useful for grouped binomial data, where each observation represents the result of  $m[i]$  successes out of  $n[i]$  trials. Then the response is provided as  $y[i] = m[i]/n[i]$  and the corresponding element of the weight vector as  $w[i]=n[i]$ . Alternatively can use `glm.multinomial()` instead.

weights	Observation weight vector, with default NULL, which results in weight 1/n for each observation.
link	The link function type, with choice "logit" (default) or "probit").

**Value**

Binomial GLM object.

**Author(s)**

Trevor Hastie and James Yang

Maintainer: Trevor Hastie [hastie@stanford.edu](mailto:hastie@stanford.edu)

**See Also**

glm.gaussian, glm.binomial, glm.poisson, glm.multinomial, glm.multigaussian, glm.cox.

**Examples**

```
n <- 100
y <- rbinom(n, 1, 0.5)
obj <- glm.binomial(y)
```

---

glm.cox

*Creates a Cox GLM family object.*

---

**Description**

A GLM family object specifies the type of model fit, provides the appropriate response object and makes sure it is represented in the right form for the model family, and allows for optional parameters such as a weight vector.

**Usage**

```
glm.cox(  
  stop,  
  status,  
  start = -Inf,  
  strata = NULL,  
  weights = NULL,  
  tie_method = c("efron", "breslow")  
)
```

**Arguments**

stop	Stop time vector.
status	Binary status vector of same length as stop, with 1 a "death", and 0 censored.
start	Start time vector. Default is a vector of -Inf of same length as stop.
strata	Observations can belong in strata, labeled 1,2, .... If strata = NULL then all observations are in a single stratum.
weights	Observation weights, with default NULL.
tie_method	The tie-breaking method - one of "efron" (default) or "breslow".

**Value**

Cox GLM object.

**Author(s)**

James Yang, Trevor Hastie, and Balasubramanian Narasimhan  
Maintainer: Trevor Hastie [hastie@stanford.edu](mailto:hastie@stanford.edu)

**See Also**

glm.gaussian, glm.binomial, glm.poisson, glm.multinomial, glm.multigaussian, glm.cox.

**Examples**

```
n <- 100
start <- sample.int(20, size = n, replace = TRUE)
stop <- start + 1 + sample.int(5, size = n, replace = TRUE)
status <- rbinom(n, 1, 0.5)
strata <- sample(c(1,2), n, replace = TRUE)
obj1 <- glm.cox(stop, status)
obj2 <- glm.cox(stop, status, start = start)
obj3 <- glm.cox(stop, status, start = start, strata = strata)
```

---

glm.gaussian

*Creates a Gaussian GLM family object.*

---

**Description**

A GLM family object specifies the type of model fit, provides the appropriate response object and makes sure it is represented in the right form for the model family, and allows for optional parameters such as a weight vector.

**Usage**

```
glm.gaussian(y, weights = NULL, opt = TRUE)
```

**Arguments**

y	Response vector.
weights	Observation weight vector, with default NULL.
opt	If TRUE (default), an optimized routine is run.

**Value**

Gaussian GLM

**Author(s)**

James Yang, Trevor Hastie, and Balasubramanian Narasimhan  
Maintainer: Trevor Hastie [hastie@stanford.edu](mailto:hastie@stanford.edu)

**See Also**

glm.gaussian, glm.binomial, glm.poisson, glm.multinomial, glm.multigaussian, glm.cox.

**Examples**

```
n <- 100
y <- rnorm(n)
obj <- glm.gaussian(y)
```

---

glm.multigaussian      *Creates a MultiGaussian GLM family object.*

---

**Description**

A GLM family object specifies the type of model fit, provides the appropriate response object and makes sure it is represented in the right form for the model family, and allows for optional parameters such as a weight vector.

**Usage**

```
glm.multigaussian(y, weights = NULL, opt = TRUE)
```

**Arguments**

y	Response matrix, with two or more columns.
weights	Observation weight vector, with default NULL.
opt	If TRUE (default), an optimized routine is run.

**Value**

MultiGaussian GLM object.



**Author(s)**

James Yang, Trevor Hastie, and Balasubramanian Narasimhan  
Maintainer: Trevor Hastie [hastie@stanford.edu](mailto:hastie@stanford.edu)

**See Also**

glm.gaussian, glm.binomial, glm.poisson, glm.multinomial, glm.multigaussian, glm.cox.

**Examples**

```
n <- 100
K <- 5
y <- matrix(rnorm(n*K), n, K)
obj <- glm.multigaussian(y)
```

---

glm.multinomial      *Creates a Multinomial GLM family object.*

---

**Description**

A GLM family object specifies the type of model fit, provides the appropriate response object and makes sure it is represented in the right form for the model family, and allows for optional parameters such as a weight vector.

**Usage**

```
glm.multinomial(y, weights = NULL)
```

**Arguments**

y	Response matrix with $K > 1$ columns, and row sums equal to 1. This can either be a "one-hot" encoded version of a $K$ -category factor variable, or else a matrix of proportions. This is useful for grouped multinomial data, where column $y[i, k]$ represents the proportion of outcomes in category $k$ in $n[i]$ trials. Then the corresponding element of the weight vector is $w[i] = n[i]$ .
weights	Observation weights.

**Value**

Multinomial GLM object.

**Author(s)**

James Yang, Trevor Hastie, and Balasubramanian Narasimhan  
Maintainer: Trevor Hastie [hastie@stanford.edu](mailto:hastie@stanford.edu)

**See Also**

glm.gaussian, glm.binomial, glm.poisson, glm.multinomial, glm.multigaussian, glm.cox.

**Examples**

```
n <- 100
K <- 5
y <- t(rmultinom(n, 1, rep(1/K, K)))
obj <- glm.multinomial(y)
```

---

glm.poisson

*Creates a Poisson GLM family object.*

---

**Description**

A GLM family object specifies the type of model fit, provides the appropriate response object and makes sure it is represented in the right form for the model family, and allows for optional parameters such as a weight vector.

**Usage**

```
glm.poisson(y, weights = NULL)
```

**Arguments**

y	Response vector of non-negative counts.
weights	Observation weight vector, with default NULL.

**Value**

Poisson GLM object.

**Author(s)**

James Yang, Trevor Hastie, and Balasubramanian Narasimhan  
Maintainer: Trevor Hastie [hastie@stanford.edu](mailto:hastie@stanford.edu)

**See Also**

glm.gaussian, glm.binomial, glm.poisson, glm.multinomial, glm.multigaussian, glm.cox.

**Examples**

```
n <- 100
y <- rpois(n, 1)
obj <- glm.poisson(y)
```

---

`grpnet`*fit a GLM with group lasso or group elastic-net regularization*

---

## Description

Computes a group elastic-net regularization path for a variety of GLM and other families, including the Cox model. This function extends the abilities of the `glmnet` package to allow for grouped regularization. The code is very efficient (core routines are written in C++), and allows for specialized matrix classes.

## Usage

```
grpnet(  
  X,  
  glm,  
  constraints = NULL,  
  groups = NULL,  
  alpha = 1,  
  penalty = NULL,  
  offsets = NULL,  
  lambda = NULL,  
  standardize = TRUE,  
  irls_max_iters = as.integer(10000),  
  irls_tol = 1e-07,  
  max_iters = as.integer(1e+05),  
  tol = 1e-07,  
  adev_tol = 0.9,  
  ddev_tol = 0,  
  newton_tol = 1e-12,  
  newton_max_iters = 1000,  
  n_threads = 1,  
  early_exit = TRUE,  
  intercept = TRUE,  
  screen_rule = c("pivot", "strong"),  
  min_ratio = 0.01,  
  lmda_path_size = 100,  
  max_screen_size = NULL,  
  max_active_size = NULL,  
  pivot_subset_ratio = 0.1,  
  pivot_subset_min = 1,  
  pivot_slack_ratio = 1.25,  
  check_state = FALSE,  
  progress_bar = FALSE,  
  warm_start = NULL  
)
```

**Arguments**

<code>X</code>	Feature matrix. Either a regular R matrix, or else an <code>adelie</code> custom matrix class, or a concatenation of such.
<code>glm</code>	GLM family/response object. This is an expression that represents the family, the response and other arguments such as weights, if present. The choices are <code>glm.gaussian()</code> , <code>glm.binomial()</code> , <code>glm.poisson()</code> , <code>glm.multinomial()</code> , <code>glm.cox()</code> , <code>glm.multinomial()</code> , and <code>glm.multigaussian()</code> . This is a required argument, and there is no default. In the simple example below, we use <code>glm.gaussian(y)</code> .
<code>constraints</code>	Group-wise constraints on the parameters, supplied as a list with an element for each group. Default is <code>NULL</code> , which means no constraints. List elements can be <code>NULL</code> as well. Currently only 'box constraints' are supported, which means upper and lower limits. The function <code>constraint.box()</code> must be used to set the constraints for each group that has constraints. Details are given in the documentation for <code>constraint.box</code> .
<code>groups</code>	This is an ordered vector of integers that represents the groupings, with each entry indicating where a group begins. The entries refer to column numbers in the feature matrix, and hence the members of a group have to be contiguous. If there are $p$ features, the default is <code>1:p</code> (no groups; i.e. $p$ groups each of size 1). So the length of <code>groups</code> is the number of groups. (Note that in the state output of <code>grpnet</code> this vector might be shifted to start from 0, since internally <code>adelie</code> uses zero-based indexing.)
<code>alpha</code>	The elasticnet mixing parameter, with $0 \leq \alpha \leq 1$ . The penalty is defined as $(1 - \alpha)/2 \sum_j \ \beta_j\ _2^2 + \alpha \sum_j \ \beta_j\ _2,$ where the sum is over groups. <code>alpha=1</code> is pure group lasso penalty, and <code>alpha=0</code> the pure ridge penalty.
<code>penalty</code>	Separate penalty factors can be applied to each group of coefficients. This is a number that multiplies <code>lambda</code> to allow differential shrinkage for groups. Can be 0 for some groups, which implies no shrinkage, and that group is always included in the model. Default is square-root of group sizes for each group.
<code>offsets</code>	Offsets, default is <code>NULL</code> . If present, this is a fixed vector or matrix corresponding to the shape of the natural parameter, and is added to the fit.
<code>lambda</code>	A user supplied <code>lambda</code> sequence. Typical usage is to have the program compute its own <code>lambda</code> sequence based on <code>lmda_path_size</code> and <code>min_ratio</code> . This is returned with the fit.
<code>standardize</code>	If <code>TRUE</code> (the default), the columns of <code>X</code> are standardized before the fit is computed. This is good practice if the features are on different scales, because it has an impact on the penalty. The regularization path is computed using the standardized features, and the standardization information is saved on the object for making future predictions. The different matrix classes have their own methods for standardization. For example, for a sparse matrix the standardization information will be computed, but not actually applied (eg centering would destroy the sparsity). Rather, the methods for matrix multiply will be aware, and incorporate the standardization information.

<code>irls_max_iters</code>	Maximum number of IRLS iterations, default is 1e4.
<code>irls_tol</code>	IRLS convergence tolerance, default is 1e-7.
<code>max_iters</code>	Maximum total number of coordinate descent iterations, default is 1e5.
<code>tol</code>	Coordinate descent convergence tolerance, default 1e-7.
<code>adev_tol</code>	Fraction deviance explained tolerance, default 0.9. This can be seen as a limit on overfitting the training data.
<code>ddev_tol</code>	Difference in fraction deviance explained tolerance, default 0. If a step in the path changes the deviance by this amount or less, the algorithm truncates the path.
<code>newton_tol</code>	Convergence tolerance for the BCD update, default 1e-12. This parameter controls the iterations in each block-coordinate step to establish the block solution.
<code>newton_max_iters</code>	Maximum number of iterations for the BCD update, default 1000.
<code>n_threads</code>	Number of threads, default 1.
<code>early_exit</code>	TRUE if the function should be allowed to exit early.
<code>intercept</code>	Default TRUE to include an unpenalized intercept.
<code>screen_rule</code>	Screen rule, with default "pivot". Other option is "strong". (an empirical improvement over "strong", the other option.)
<code>min_ratio</code>	Ratio between smallest and largest value of lambda. Default is 1e-2.
<code>lmda_path_size</code>	Number of values for lambda, if generated automatically. Default is 100.
<code>max_screen_size</code>	Maximum number of screen groups. Default is NULL.
<code>max_active_size</code>	Maximum number of active groups. Default is NULL.
<code>pivot_subset_ratio</code>	Subset ratio of pivot rule. Default is 0.1. Users not expected to fiddle with this.
<code>pivot_subset_min</code>	Minimum subset of pivot rule. Defaults is 1. Users not expected to fiddle with this.
<code>pivot_slack_ratio</code>	Slack ratio of pivot rule, default is 1.25. Users not expected to fiddle with this. See reference for details.
<code>check_state</code>	Check state. Internal parameter, with default FALSE.
<code>progress_bar</code>	Progress bar. Default is FALSE.
<code>warm_start</code>	Warm start (default is NULL). Internal parameter.

### Value

A list of class "grpnet". This has a main component called `state` which represents the fitted path, and a few extra useful components such as the `call`, the `family` name, `groups` and `group_sizes`. Users are encouraged to use methods like `predict()`, `coef()`, `print()`, `plot()` etc to examine the object.

**Author(s)**

James Yang, Trevor Hastie, and Balasubramanian Narasimhan  
 Maintainer: Trevor Hastie <hastie@stanford.edu>

**References**

- Yang, James and Hastie, Trevor. (2024) A Fast and Scalable Pathwise-Solver for Group Lasso and Elastic Net Penalized Regression via Block-Coordinate Descent. arXiv doi:10.48550/arXiv.2405.08631.
- Friedman, J., Hastie, T. and Tibshirani, R. (2008) *Regularization Paths for Generalized Linear Models via Coordinate Descent (2010)*, *Journal of Statistical Software*, Vol. 33(1), 1-22, doi:10.18637/jss.v033.i01.
- Simon, N., Friedman, J., Hastie, T. and Tibshirani, R. (2011) *Regularization Paths for Cox's Proportional Hazards Model via Coordinate Descent*, *Journal of Statistical Software*, Vol. 39(5), 1-13, doi:10.18637/jss.v039.i05.
- Tibshirani, Robert, Bien, J., Friedman, J., Hastie, T., Simon, N., Taylor, J. and Tibshirani, Ryan. (2012) *Strong Rules for Discarding Predictors in Lasso-type Problems*, *JRSSB*, Vol. 74(2), 245-266, <https://arxiv.org/abs/1011.2234>.

**See Also**

cv.grpnet, predict.grpnet, coef.grpnet, plot.grpnet, print.grpnet.

**Examples**

```
set.seed(0)
n <- 100
p <- 200
X <- matrix(rnorm(n * p), n, p)
y <- X[,1] * rnorm(1) + rnorm(n)
## Here we create 60 groups randomly. Groups need to be contiguous, and the `groups` variable
## indicates the beginning position of each group.
groups <- c(1, sample(2:199, 60, replace = FALSE))
groups <- sort(groups)
print(groups)
fit <- grpnet(X, glm.gaussian(y), groups = groups)
print(fit)
plot(fit)
coef(fit)
cvfit <- cv.grpnet(X, glm.gaussian(y), groups = groups)
print(cvfit)
plot(cvfit)
predict(cvfit, newx=X[1:5,], lambda="lambda.min")
```

**Description**

IO handler for SNP phased, ancestry matrix.

**Usage**

```
io.snp_phased_ancestry(filename, read_mode = "file")
```

**Arguments**

filename	File name.
read_mode	Reading mode.

**Value**

IO handler for SNP phased, ancestry data.

**Examples**

```
n <- 123
s <- 423
A <- 8
filename <- paste(tempdir(), "snp_phased_ancestry_dummy.snpdat", sep="/")
handle <- io.snp_phased_ancestry(filename)
calldata <- matrix(
  as.integer(sample.int(
    2, n * s * 2,
    replace=TRUE,
    prob=c(0.7, 0.3)
  ) - 1),
  n, s * 2
)
ancestries <- matrix(
  as.integer(sample.int(
    A, n * s * 2,
    replace=TRUE,
    prob=rep_len(1/A, A)
  ) - 1),
  n, s * 2
)
handle$write(calldata, ancestries, A, 1)
handle$read()
file.remove(filename)
```

---

io.snp\_unphased

*IO handler for SNP unphased matrix.*

---

**Description**

IO handler for SNP unphased matrix.

**Usage**

```
io.snp_unphased(filename, read_mode = "file")
```

**Arguments**

filename	File name.
read_mode	Reading mode.

**Value**

IO handler for SNP unphased data.

**Examples**

```
n <- 123
s <- 423
filename <- paste(tempdir(), "snp_unphased_dummy.snpdat", sep="/")
handle <- io.snp_unphased(filename)
mat <- matrix(
  as.integer(sample.int(
    3, n * s,
    replace=TRUE,
    prob=c(0.7, 0.2, 0.1)
  ) - 1),
  n, s
)
impute <- double(s)
handle$write(mat, "mean", impute, 1)
handle$read()
file.remove(filename)
```

---

matrix.block_diag	<i>Creates a block-diagonal matrix.</i>
-------------------	---

---

**Description**

Creates a block-diagonal matrix.

**Usage**

```
matrix.block_diag(mats, method = c("naive", "cov"), n_threads = 1)
```

**Arguments**

mats	List of matrices.
method	Method type, with default method="naive".
n_threads	Number of threads.



**Value**

Block-diagonal matrix.

**Author(s)**

Trevor Hastie and James Yang

Maintainer: Trevor Hastie [hastie@stanford.edu](mailto:hastie@stanford.edu)

**Examples**

```
n <- 100
ps <- c(10, 20, 30)
mats <- lapply(ps, function(p) {
  X <- matrix(rnorm(n * p), n, p)
  matrix.dense(t(X) %*% X, method="cov")
})
out <- matrix.block_diag(mats, method="cov")
mats <- lapply(ps, function(p) {
  X <- matrix(rnorm(n * p), n, p)
  matrix.dense(X, method="naive")
})
out <- matrix.block_diag(mats, method="naive")
```

---

`matrix.concatenate`      *Creates a concatenation of the matrices.*

---

**Description**

Creates a concatenation of the matrices.

**Usage**

```
matrix.concatenate(mats, axis = 2, n_threads = 1)
```

**Arguments**

<code>mats</code>	List of matrices.
<code>axis</code>	The axis along which the matrices will be joined. With <code>axis = 2</code> (default) this function is equivalent to <code>cbind()</code> and <code>axis = 1</code> is equivalent to <code>rbind()</code> .
<code>n_threads</code>	Number of threads.

**Value**

Concatenation of matrices. The object is an S4 class with methods for efficient computation in C++ by `adelle`. Note that for the object itself `axis` is represented with base 0 (so 1 less than the argument here).

**Author(s)**

Trevor Hastie and James Yang  
Maintainer: Trevor Hastie [hastie@stanford.edu](mailto:hastie@stanford.edu)

**Examples**

```
n <- 100
ps <- c(10, 20, 30)
n <- 100
mats <- lapply(ps, function(p) {
  matrix.dense(matrix(rnorm(n * p), n, p))
})
out <- matrix.concatenate(mats, axis=2)
```

---

`matrix.convex_relu`      *Creates a feature matrix for the convex relu problem.*

---

**Description**

Creates a feature matrix for the convex relu problem.

**Usage**

```
matrix.convex_relu(mat, mask, gated = FALSE, n_threads = 1)
```

**Arguments**

<code>mat</code>	Base feature matrix. It is either a dense or sparse matrix.
<code>mask</code>	Boolean mask matrix.
<code>gated</code>	Flag to indicate whether to use the convex gated relu feature matrix.
<code>n_threads</code>	Number of threads.

**Value**

Convex relu feature matrix. The object is an S4 class with methods for efficient computation in C++ by *adelie*.

**Author(s)**

Trevor Hastie and James Yang  
Maintainer: Trevor Hastie [hastie@stanford.edu](mailto:hastie@stanford.edu)

**Examples**

```
n <- 100
p <- 20
m <- 10
Z_dense <- matrix(rnorm(n * p), n, p)
mask <- matrix(rbinom(n * m, 1, 0.5), n, m)
out <- matrix.convex_relu(Z_dense, mask)
Z_sparse <- as(Z_dense, "dgCMatrix")
out <- matrix.convex_relu(Z_sparse, mask)
```

---

matrix.dense                      *Creates a dense matrix object.*

---

**Description**

Creates a dense matrix object.

**Usage**

```
matrix.dense(mat, method = c("naive", "cov", "constraint"), n_threads = 1)
```

**Arguments**

mat	The dense matrix.
method	Method type, with default method="naive". If method="cov", the matrix is used with the solver <code>gaussian_cov()</code> . Used for <code>glm.gaussian()</code> and <code>glm.multigaussian()</code> families. Generally "naive" is used for wide matrices, and "cov" for tall matrices. If method="constraint", the matrix is used as input to the constraint objects.
n_threads	Number of threads.

**Value**

Dense matrix. The object is an S4 class with methods for efficient computation by `adelle`.

**Author(s)**

Trevor Hastie and James Yang  
 Maintainer: Trevor Hastie [hastie@stanford.edu](mailto:hastie@stanford.edu)

**Examples**

```
n <- 100
p <- 20
X_dense <- matrix(rnorm(n * p), n, p)
out <- matrix.dense(X_dense, method="naive")
A_dense <- t(X_dense) %*% X_dense
out <- matrix.dense(A_dense, method="cov")
out <- matrix.dense(X_dense, method="constraint")
```

---

`matrix.eager_cov`      *Creates an eager covariance matrix.*

---

**Description**

Creates an eager covariance matrix.

**Usage**

```
matrix.eager_cov(mat, n_threads = 1)
```

**Arguments**

`mat`                    A dense matrix to be used with the `gaussian_cov()` solver.  
`n_threads`              Number of threads.

**Value**

The dense covariance matrix. This matrix is exactly `t(mat)%*%mat`, computed with some efficiency.

**Examples**

```
n <- 100  
p <- 20  
mat <- matrix(rnorm(n * p), n, p)  
out <- matrix.eager_cov(mat)
```

---

`matrix.interaction`      *Creates a matrix with pairwise interactions.*

---

**Description**

Creates a matrix with pairwise interactions.

**Usage**

```
matrix.interaction(  
  mat,  
  intr_keys = NULL,  
  intr_values,  
  levels = NULL,  
  n_threads = 1  
)
```

**Arguments**

mat	The dense matrix, which can include factors with levels coded as non-negative integers.
intr_keys	List of feature indices. This is a list of all features with which interactions can be formed. Default is 1:p where p is the number of columns in mat.
intr_values	List of integer vectors of feature indices. For each of the $m \leq p$ indices listed in intr_keys, there is a vector of indices indicating which columns are candidates for interaction with that feature. If a list is <code>list(NULL)</code> , that means all other features are candidates for interactions. The default is a list of length m where each element is <code>list(NULL)</code> ; that is <code>rep(list(NULL), m)</code> .
levels	Number of levels for each of the columns of mat, with 1 representing a quantitative feature. A factor with K levels should be represented by the numbers 0, 1, ..., K-1.
n_threads	Number of threads.

**Value**

Pairwise interaction matrix. Logic is used to avoid repetitions. For each factor variable, the column is one-hot-encoded to form a basis for that feature. The object is an S4 class with methods for efficient computation by `adelle`. Note that some of the arguments are transformed to C++ base 0 for internal use, and if the object is examined, it will reflect that.

**Author(s)**

Trevor Hastie and James Yang  
 Maintainer: Trevor Hastie [hastie@stanford.edu](mailto:hastie@stanford.edu)

**Examples**

```
n <- 10
p <- 20
X_dense <- matrix(rnorm(n * p), n, p)
X_dense[,1] <- rbinom(n, 4, 0.5)
intr_keys <- c(1, 2)
intr_values <- list(NULL, c(1, 3))
levels <- c(c(5), rep(1, p-1))
out <- matrix.interaction(X_dense, intr_keys, intr_values, levels)
```

---

`matrix.kronecker_eye` *Creates a Kronecker product with an identity matrix.*

---

**Description**

Creates a Kronecker product with an identity matrix.

**Usage**

```
matrix.kronecker_eye(mat, K = 1, n_threads = 1)
```

**Arguments**

mat	The matrix to view as a Kronecker product.
K	Dimension of the identity matrix (default is 1, which does essentially nothing).
n_threads	Number of threads.

**Value**

Kronecker product with identity matrix. If mat is  $n \times p$ , the the resulting matrix will be  $nK \times np$ . The object is an S4 class with methods for efficient computation by *adelle*.

**Author(s)**

James Yang, Trevor Hastie, and Balasubramanian Narasimhan  
Maintainer: Trevor Hastie [hastie@stanford.edu](mailto:hastie@stanford.edu)

**Examples**

```
n <- 100
p <- 20
K <- 2
mat <- matrix(rnorm(n * p), n, p)
out <- matrix.kronecker_eye(mat, K)
mat <- matrix.dense(mat)
out <- matrix.kronecker_eye(mat, K)
```

---

matrix.lazy_cov	<i>Creates a lazy covariance matrix.</i>
-----------------	--

---

**Description**

Creates a lazy covariance matrix.

**Usage**

```
matrix.lazy_cov(mat, n_threads = 1)
```

**Arguments**

mat	A dense data matrix to be used with the <code>gaussian_cov()</code> solver.
n_threads	Number of threads.

**Value**

Lazy covariance matrix. This is essentially the same matrix, but with a setup to create covariance terms as needed on the fly. The object is an S4 class with methods for efficient computation by *adelie*.

**Author(s)**

James Yang, Trevor Hastie, and Balasubramanian Narasimhan  
Maintainer: Trevor Hastie [hastie@stanford.edu](mailto:hastie@stanford.edu)

**Examples**

```
n <- 100
p <- 20
mat <- matrix(rnorm(n * p), n, p)
out <- matrix.lazy_cov(mat)
```

---

matrix.one_hot	<i>Creates a one-hot encoded matrix.</i>
----------------	--

---

**Description**

Creates a one-hot encoded matrix.

**Usage**

```
matrix.one_hot(mat, levels = NULL, n_threads = 1)
```

**Arguments**

mat	A dense matrix, which can include factors with levels coded as non-negative integers.
levels	Number of levels for each of the columns of mat, with 1 representing a quantitative feature. A factor with K levels should be represented by the numbers 0, 1, ..., K-1.
n_threads	Number of threads.

**Value**

One-hot encoded matrix. All the factor columns, with levels > 1, are replaced by a collection of one-hot encoded versions (dummy matrices). The resulting matrix has `sum(levels)` columns. The object is an S4 class with methods for efficient computation by *adelie*. Note that some of the arguments are transformed to C++ base 0 for internal use, and if the object is examined, it will reflect that.

**Author(s)**

James Yang, Trevor Hastie, and Balasubramanian Narasimhan  
Maintainer: Trevor Hastie [hastie@stanford.edu](mailto:hastie@stanford.edu)

**Examples**

```
n <- 100
p <- 20
mat <- matrix(rnorm(n * p), n, p)
fac <- sample(0:5, n, replace = TRUE)
mat=cbind(fac,mat)
levels <- c(6, rep(1,p))
out <- matrix.one_hot(mat, levels = levels)
```

---

matrix.snp\_phased\_ancestry

*Creates a SNP phased, ancestry matrix.*

---

**Description**

Creates a SNP phased, ancestry matrix.

**Usage**

```
matrix.snp_phased_ancestry(io, n_threads = 1)
```

**Arguments**

io	IO handler.
n_threads	Number of threads.

**Value**

SNP phased, ancestry matrix.

**Author(s)**

James Yang, Trevor Hastie, and Balasubramanian Narasimhan  
Maintainer: Trevor Hastie [hastie@stanford.edu](mailto:hastie@stanford.edu)

**Examples**

```
n <- 123
s <- 423
A <- 8
filename <- paste(tempdir(), "snp_phased_ancestry_dummy.snpdat", sep="/")
handle <- io.snp_phased_ancestry(filename)
calldata <- matrix(
```



```

    as.integer(sample.int(
      2, n * s * 2,
      replace=TRUE,
      prob=c(0.7, 0.3)
    ) - 1),
    n, s * 2
  )
  ancestries <- matrix(
    as.integer(sample.int(
      A, n * s * 2,
      replace=TRUE,
      prob=rep_len(1/A, A)
    ) - 1),
    n, s * 2
  )
  handle$write(calldata, ancestries, A, 1)
  out <- matrix.snp_phased_ancestry(handle)
  file.remove(filename)

```

---

matrix.snp\_unphased     *Creates a SNP unphased matrix.*

---

### Description

Creates a SNP unphased matrix.

### Usage

```
matrix.snp_unphased(io, n_threads = 1)
```

### Arguments

io	IO handler.
n_threads	Number of threads.

### Value

SNP unphased matrix.

### Examples

```

n <- 123
s <- 423
filename <- paste(tempdir(), "snp_unphased_dummy.snpdat", sep="/")
handle <- io.snp_unphased(filename)
mat <- matrix(
  as.integer(sample.int(
    3, n * s,
    replace=TRUE,

```

```

        prob=c(0.7, 0.2, 0.1)
      ) - 1),
      n, s
    )
  impute <- double(s)
  handle$write(mat, "mean", impute, 1)
  out <- matrix.snp_unphased(handle)
  file.remove(filename)

```

---

matrix.sparse	<i>Creates a sparse matrix object.</i>
---------------	--

---

### Description

Creates a sparse matrix object.

### Usage

```
matrix.sparse(mat, method = c("naive", "cov", "constraint"), n_threads = 1)
```

### Arguments

mat	A sparse matrix.
method	Method type, with default method="naive". If method="cov", the matrix is used with the solver <code>gaussian_cov()</code> . Used for <code>glm.gaussian()</code> and <code>glm.multigaussian()</code> families. Generally "naive" is used for wide matrices, and "cov" for tall matrices. If method="constraint", the matrix is used as input to the constraint objects.
n_threads	Number of threads.

### Value

Sparse matrix object. The object is an S4 class with methods for efficient computation by *adelie*.

### Examples

```

n <- 100
p <- 20
X_dense <- matrix(rnorm(n * p), n, p)
X_sp <- as(X_dense, "dgCMatrix")
out <- matrix.sparse(X_sp, method="naive")
A_dense <- t(X_dense) %*% X_dense
A_sp <- as(A_dense, "dgCMatrix")
out <- matrix.sparse(A_sp, method="cov")
out <- matrix.sparse(X_sp, method="constraint")

```

---

matrix.standardize      *Creates a standardized matrix.*

---

### Description

Creates a standardized matrix.

### Usage

```
matrix.standardize(  
  mat,  
  centers = NULL,  
  scales = NULL,  
  weights = NULL,  
  ddof = 0,  
  n_threads = 1  
)
```

### Arguments

mat	An <code>adelle</code> matrix.
centers	The center values. Default is to use the column means.
scales	The scale values. Default is to use the sample standard deviations.
weights	Observation weight vector, which defaults to $1/n$ per observation.
ddof	Degrees of freedom for standard deviations, with default 0 ( $1/n$ ). The alternative is 1 leading to $1/(n-1)$ .
n_threads	Number of threads.

### Value

Standardized matrix. The object is an S4 class with methods for efficient computation by `adelle`. Conventions depend on the matrix class. For example, if a matrix is constructed using `matrix.onehot()`, only the quantitative variables are standardized.

### Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan  
Maintainer: Trevor Hastie [hastie@stanford.edu](mailto:hastie@stanford.edu)

### Examples

```
n <- 100  
p <- 20  
X <- matrix(rnorm(n * p), n, p)  
out <- matrix.standardize(matrix.dense(X))
```

---

matrix.subset	<i>Creates a subset of the matrix along an axis.</i>
---------------	--

---

## Description

Creates a subset of the matrix along an axis.

## Usage

```
matrix.subset(mat, indices, axis = 1, n_threads = 1)
```

## Arguments

mat	The adelic matrix to subset.
indices	Vector of indices to subset the matrix.
axis	The axis along which to subset (2 is columns, 1 is rows).
n_threads	Number of threads.

## Value

Matrix subsetted along the appropriate axis. The object is an S4 class with methods for efficient computation by adelic.

## Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan  
Maintainer: Trevor Hastie [hastie@stanford.edu](mailto:hastie@stanford.edu)

## Examples

```
n <- 100
p <- 20
X <- matrix.dense(matrix(rnorm(n * p), n, p))
indices <- c(1, 3, 10)
out <- matrix.subset(X, indices, axis=1)
out <- matrix.subset(X, indices, axis=2)
```

---

plot.cv.glintnet      *plot the cross-validation curve produced by cv.glintnet*

---

### Description

Plots the cross-validation curve, and upper and lower standard deviation curves, as a function of the lambda values used.

Plots the cross-validation curve, and upper and lower standard deviation curves, as a function of the lambda values used.

### Usage

```
## S3 method for class 'cv.glintnet'  
plot(x, sign.lambda = -1, ...)
```

```
## S3 method for class 'cv.grpnet'  
plot(x, sign.lambda = -1, ...)
```

### Arguments

x	fitted "cv.grpnet" object
sign.lambda	Either plot against log(lambda) or its negative (default) if sign.lambda=-1
...	Other graphical parameters

### Details

A plot is produced, and nothing is returned.

A plot is produced, and nothing is returned.

### Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan

Maintainer: Trevor Hastie [hastie@stanford.edu](mailto:hastie@stanford.edu)

### References

Yang, James and Hastie, Trevor. (2024) A Fast and Scalable Pathwise-Solver for Group Lasso and Elastic Net Penalized Regression via Block-Coordinate Descent. arXiv doi:10.48550/arXiv.2405.08631.  
Adelie Python user guide <https://jamesyang007.github.io/adelie/>

### See Also

grpnet and cv.grpnet.

**Examples**

```

set.seed(0)
n=500
d_cont = 5      # number of continuous features
d_disc = 5      # number of categorical features
Z_cont = matrix(rnorm(n*d_cont), n, d_cont)
levels = sample(2:5,d_disc, replace = TRUE)
Z_disc = matrix(0,n,d_disc)
for(i in seq(d_disc))Z_disc[,i] = sample(0:(levels[i]-1),n,replace=TRUE)
Z = cbind(Z_cont,Z_disc)
levels = c(rep(1,d_cont),levels)

xmat = model.matrix(~Z_cont[,1]*factor(Z_disc[,2]))
nc=ncol(xmat)
beta = rnorm(nc)
y = xmat%*%beta+rnorm(n)*1.5

cvfit <- cv.glintnet(Z, glm.gaussian(y), levels=levels, intr_keys = 1)
plot(cvfit)

set.seed(1010)
n = 1000
p = 100
nzc = trunc(p/10)
x = matrix(rnorm(n * p), n, p)
beta = rnorm(nzc)
fx = (x[, seq(nzc)] %*% beta)
eps = rnorm(n) * 5
y = drop(fx + eps)
px = exp(fx)
px = px/(1 + px)
ly = rbinom(n = length(px), prob = px, size = 1)
cvob1 = cv.grpnet(x, glm.gaussian(y))
plot(cvob1)
title("Gaussian Family", line = 2.5)
frame()
set.seed(1011)
cvob2 = cv.grpnet(x, glm.binomial(ly))
plot(cvob2)
title("Binomial Family", line = 2.5)

```

---

plot.grpnet

*plot coefficients from a "grpnet" object*


---

**Description**

Produces a coefficient profile plot of the coefficient paths for a fitted "grpnet" object.

**Usage**

```
## S3 method for class 'grpnet'  
plot(x, sign.lambda = -1, glm.name = TRUE, ...)
```

**Arguments**

x	fitted "grpnet" model
sign.lambda	This determines whether we plot against $\log(\lambda)$ or its negative. values are -1(default) or 1
glm.name	This is a logical (default TRUE), and causes the glm name of the model to be included in the plot.
...	Other graphical parameters to plot

**Details**

A coefficient profile plot is produced. If x is a multinomial or multigaussian model, the 2norm of the vector of coefficients is plotted.

**Author(s)**

James Yang, Trevor Hastie, and Balasubramanian Narasimhan  
Maintainer: Trevor Hastie [hastie@stanford.edu](mailto:hastie@stanford.edu)

**References**

Yang, James and Hastie, Trevor. (2024) A Fast and Scalable Pathwise-Solver for Group Lasso and Elastic Net Penalized Regression via Block-Coordinate Descent. arXiv [doi:10.48550/arXiv.2405.08631](https://doi.org/10.48550/arXiv.2405.08631).

**See Also**

grpnet, and print, and coef methods, and cv.grpnet.

**Examples**

```
x=matrix(rnorm(100*20),100,20)  
y=rnorm(100)  
fit1=grpnet(x,glm.gaussian(y))  
plot(fit1)  
g4=diag(4)[sample(1:4,100,replace=TRUE),]  
fit2=grpnet(x,glm.multinomial(g4))  
plot(fit2,lwd=3)  
fit3=grpnet(x,glm.gaussian(y),groups=c(1,5,9,13,17))  
plot(fit3)
```

---

predict.cv.glintnet    *make predictions from a "cv.glintnet" object.*

---

### Description

This function makes predictions from a cross-validated glintnet model, using the stored "glintnet.fit" object, and the optimal value chosen for lambda.

### Usage

```
## S3 method for class 'cv.glintnet'
predict(object, newx, lambda = c("lambda.1se", "lambda.min"), ...)
```

### Arguments

object	Fitted "cv.glintnet".
newx	Matrix of new values for x at which predictions are to be made. This matrix is of the same form as in the call to glintnet.
lambda	Value(s) of the penalty parameter lambda at which predictions are required. Default is the entire sequence used to create the model. If values of lambda are supplied, the function uses linear interpolation to make predictions for values of lambda that do not coincide with those used in the fitting algorithm. Note: if newx is a vector (a single row which has lost its matrix dimensions), convert it to a 1-row matrix first, e.g. by supplying t(newx) instead.
...	Other arguments that can be passed to predict.glpnet

### Details

This function makes it easier to use the results of cross-validation to make a prediction.

### Examples

```
set.seed(0)
n=500
d_cont = 5    # number of continuous features
d_disc = 5    # number of categorical features
Z_cont = matrix(rnorm(n*d_cont), n, d_cont)
levels = sample(2:5,d_disc, replace = TRUE)
Z_disc = matrix(0,n,d_disc)
for(i in seq(d_disc))Z_disc[,i] = sample(0:(levels[i]-1),n,replace=TRUE)
Z = cbind(Z_cont,Z_disc)
levels = c(rep(1,d_cont),levels)

xmat = model.matrix(~Z_cont[,1]*factor(Z_disc[,2]))
nc=ncol(xmat)
beta = rnorm(nc)
y = xmat%*%beta+rnorm(n)*1.5
```



```
cvfit <- cv.glintnet(Z, glm.gaussian(y), levels=levels, intr_keys = 1)
plot(cvfit)
predict(cvfit, newx=Z[1:5,])
```

---

predict.cv.grpnet      *make predictions from a "cv.grpnet" object.*

---

## Description

This function makes predictions from a cross-validated grpnet model, using the stored "grpnet.fit" object, and the optimal value chosen for lambda.

## Usage

```
## S3 method for class 'cv.grpnet'
predict(object, newx, lambda = c("lambda.1se", "lambda.min"), ...)
```

## Arguments

object	Fitted "cv.grpnet".
newx	Matrix of new values for x at which predictions are to be made. Can be a matrix, a sparse matrix as in Matrix package, or else any of the matrix forms allowable in the adelig package. This argument is not used for type="coefficients".
lambda	Value(s) of the penalty parameter lambda at which predictions are required. Default is the value lambda="lambda.1se" stored on the CV object. Alternatively lambda="lambda.min" can be used. If lambda is numeric, it is taken as the value(s) of lambda to be used.
...	Other arguments to predict.grpnet, such as type.

## Details

This function makes it easier to use the results of cross-validation to make a prediction.

## Value

The object returned depends on the arguments.

## Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan  
 Maintainer: Trevor Hastie [hastie@stanford.edu](mailto:hastie@stanford.edu)

## References

Yang, James and Hastie, Trevor. (2024) A Fast and Scalable Pathwise-Solver for Group Lasso and Elastic Net Penalized Regression via Block-Coordinate Descent. arXiv [doi:10.48550/arXiv.2405.08631](https://doi.org/10.48550/arXiv.2405.08631).

**See Also**

grpnet, and print, and coef methods, and cv.grpnet.

**Examples**

```
x = matrix(rnorm(100 * 20), 100, 20)
y = rnorm(100)
cv.fit = cv.grpnet(x, glm.gaussian(y))
predict(cv.fit, newx = x[1:5, ])
coef(cv.fit)
coef(cv.fit, lambda = "lambda.min")
predict(cv.fit, newx = x[1:5, ], lambda = c(0.001, 0.002))
```

---

predict.glintnet	<i>make predictions from a "glintnet" object.</i>
------------------	---

---

**Description**

Similar to other predict methods, this functions predicts linear predictors, coefficients and more from a fitted "glintnet" object.

**Usage**

```
## S3 method for class 'glintnet'
predict(
  object,
  newx,
  lambda = NULL,
  type = c("link", "response", "coefficients", "nonzero"),
  newoffsets = NULL,
  n_threads = 1,
  ...
)
```

**Arguments**

object	Fitted "glintnet" model.
newx	Matrix of new values for x at which predictions are to be made. This matrix is of the same form as in the call to glintnet.
lambda	Value(s) of the penalty parameter lambda at which predictions are required. Default is the entire sequence used to create the model. If values of lambda are supplied, the function uses linear interpolation to make predictions for values of lambda that do not coincide with those used in the fitting algorithm. Note: if newx is a vector (a single row which has lost its matrix dimensions), convert it to a 1-row matrix first, e.g. by supplying t(newx) instead.

type	Type of prediction required. Type "link" is the default, and gives the linear predictors. Type "response" applies the inverse link to these predictions. Type "coefficients" extracts the coefficients, intercepts and the active-set sizes. Type "nonzero" returns a list of active groups along the path, indexed from 1 to number of groups.
newoffsets	If an offset is used in the fit, then one must be supplied for making predictions (except for type="coefficients").
n_threads	Number of threads, default 1.
...	Other arguments that can be passed to predict.glpnet

**Value**

The object returned depends on type.

**Author(s)**

James Yang, Trevor Hastie, and Balasubramanian Narasimhan  
 Maintainer: Trevor Hastie <hastie@stanford.edu>

**References**

Yang, James and Hastie, Trevor. (2024) A Fast and Scalable Pathwise-Solver for Group Lasso and Elastic Net Penalized Regression via Block-Coordinate Descent. arXiv doi:10.48550/arXiv.2405.08631.  
 Adelle Python user guide <https://jamesyang007.github.io/adelle/>

**See Also**

grpnet, and print, and coef methods, and cv.grpnet.

**Examples**

```
set.seed(0)
n=500
d_cont = 5      # number of continuous features
d_disc = 5      # number of categorical features
Z_cont = matrix(rnorm(n*d_cont), n, d_cont)
levels = sample(2:5,d_disc, replace = TRUE)
Z_disc = matrix(0,n,d_disc)
for(i in seq(d_disc))Z_disc[,i] = sample(0:(levels[i]-1),n,replace=TRUE)
Z = cbind(Z_cont,Z_disc)
levels = c(rep(1,d_cont),levels)

xmat = model.matrix(~Z_cont[,1]*factor(Z_disc[,2]))
nc=ncol(xmat)
beta = rnorm(nc)
y = xmat%%beta+rnorm(n)*1.5

fit <- glintnet(Z, glm.gaussian(y), levels=levels, intr_keys = 1)
predict(fit, lambda = c(.1,.01), newx = Z[1:4,])
predict(fit, lambda = c(0.1,0.01), type="nonzero")
```

---

predict.grpnet            *make predictions from a "grpnet" object.*

---

### Description

Similar to other predict methods, this functions predicts linear predictors, coefficients and more from a fitted "grpnet" object. Note that if the default standardize=TRUE was used in fitting the grpnet object, the coefficients reported are for the standardized inputs. However, the predict function will apply the stored standardization to newx and give the correct predictions.

### Usage

```
## S3 method for class 'grpnet'
predict(
  object,
  newx,
  lambda = NULL,
  type = c("link", "response", "coefficients", "nonzero"),
  newoffsets = NULL,
  n_threads = 1,
  ...
)

## S3 method for class 'grpnet'
coef(object, lambda = NULL, ...)
```

### Arguments

object	Fitted "grpnet" model.
newx	Matrix of new values for x at which predictions are to be made. Can be a matrix, a sparse matrix as in Matrix package, or else any of the matrix forms allowable in the adeli.e package. The number of columns must match that of the input matrix used in fitting object. If the model object was fit with standardize=TRUE, the saved centers and scaling will be applied to this matrix. This argument is not used for type="coefficients"
lambda	Value(s) of the penalty parameter lambda at which predictions are required. Default is the entire sequence used to create the model. If values of lambda are supplied, the function uses linear interpolation to make predictions for values of lambda that do not coincide with those used in the fitting algorithm. Note: if newx is a vector (a single row which has lost its matrix dimensions), convert it to a 1-row matrix first, e.g. by supplying t(newx) instead.
type	Type of prediction required. Type "link" is the default, and gives the linear predictors. Type "response" applies the inverse link to these predictions. Type "coefficients" extracts the coefficients, intercepts and the active-set sizes. Type "nonzero" returns a list of active groups along the path, indexed from 1 to number of groups.

newoffsets	If an offset is used in the fit, then one must be supplied for making predictions (except for type="coefficients").
n_threads	Number of threads, default 1.
...	Currently ignored.

### Details

The shape of the objects returned are different for "multinomial" and "multigaussian" objects. `coef(...)` is equivalent to `predict(type="coefficients", ...)`

### Value

The object returned depends on type.

### Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan  
Maintainer: Trevor Hastie <hastie@stanford.edu>

### References

Yang, James and Hastie, Trevor. (2024) A Fast and Scalable Pathwise-Solver for Group Lasso and Elastic Net Penalized Regression via Block-Coordinate Descent. arXiv doi:10.48550/arXiv.2405.08631.  
Adelie Python user guide <https://jamesyang007.github.io/adelie/>

### See Also

grpnet, and print, and coef methods, and cv.grpnet.

### Examples

```
set.seed(0)
n <- 100
p <- 200
X <- matrix(rnorm(n * p), n, p)
y <- X[,1] * rnorm(1) + rnorm(n)
groups <- c(1, sample(2:199, 60, replace = FALSE))
groups <- sort(groups)
fit <- grpnet(X, glm.gaussian(y), groups = groups)
coef(fit)
predict(fit, newx = X[1:5,], lambda = c(0.1, 0.05))
predict(fit, type="nonzero", lambda = c(0.1, 0.05))
```

---

print.cv.grpnet      *print a cross-validated grpnet object*

---

### Description

Print a summary of the results of cross-validation for a grpnet model.

### Usage

```
## S3 method for class 'cv.grpnet'  
print(x, digits = max(3, getOption("digits") - 3), ...)
```

### Arguments

x	fitted 'cv.grpnet' object
digits	significant digits in printout
...	additional print arguments

### Author(s)

James Yang, Trevor Hastie, and Balasubramanian Narasimhan  
Maintainer: Trevor Hastie [hastie@stanford.edu](mailto:hastie@stanford.edu)

### References

Yang, James and Hastie, Trevor. (2024) A Fast and Scalable Pathwise-Solver for Group Lasso and Elastic Net Penalized Regression via Block-Coordinate Descent. arXiv [doi:10.48550/arXiv.2405.08631](https://doi.org/10.48550/arXiv.2405.08631).

### See Also

grpnet, predict and coef methods.

### Examples

```
x = matrix(rnorm(100 * 20), 100, 20)  
y = rnorm(100)  
fit1 = cv.grpnet(x, glm.gaussian(y))  
print(fit1)
```

---

print.glintnet	<i>Print a summary of the glintnet path at each step along the path.</i>
----------------	--

---

### Description

Print a summary of the grpnet path at each step along the path.

### Usage

```
## S3 method for class 'glintnet'  
print(x, digits = max(3, getOption("digits") - 3), ...)
```

### Arguments

x	fitted glintnet object
digits	significant digits in printout
...	additional print arguments

### Details

The call that produced the object x is printed, followed by a five-column matrix with columns N\_main, N\_int, Df, %Dev and Lambda. The N\_main column is the number of main-effect terms in the solution, and N\_int the number of interaction terms. Since an interaction term implies both main effects, the former is always at least as large as the latter. The Df column is the number of nonzero coefficients (Df is a reasonable name only for lasso fits). %Dev is the percent deviance explained (relative to the null deviance).

### Value

The matrix above is silently returned

### References

Yang, James and Hastie, Trevor. (2024) A Fast and Scalable Pathwise-Solver for Group Lasso and Elastic Net Penalized Regression via Block-Coordinate Descent. arXiv [doi:10.48550/arXiv.2405.08631](https://arxiv.org/abs/10.48550/arXiv.2405.08631).

### See Also

grpnet, predict, plot and coef methods.

### Examples

```
x = matrix(rnorm(100 * 20), 100, 20)  
y = rnorm(100)  
fit1 = grpnet(x, glm.gaussian(y), groups = c(1:5,7,9))  
print(fit1)
```

---

print.grpnet	<i>print a grpnet object</i>
--------------	------------------------------

---

### Description

Print a summary of the grpnet path at each step along the path.

### Usage

```
## S3 method for class 'grpnet'  
print(x, digits = max(3, getOption("digits") - 3), ...)
```

### Arguments

x	fitted grpnet object
digits	significant digits in printout
...	additional print arguments

### Details

The call that produced the object x is printed, followed by a four-column matrix with columns Groups, Df, %Dev and Lambda. The Groups column is the number of active groups in the solution. The Df column is the number of nonzero coefficients (Df is a reasonable name only for lasso fits). %Dev is the percent deviance explained (relative to the null deviance).

### Value

The matrix above is silently returned

### References

Yang, James and Hastie, Trevor. (2024) A Fast and Scalable Pathwise-Solver for Group Lasso and Elastic Net Penalized Regression via Block-Coordinate Descent. arXiv [doi:10.48550/arXiv.2405.08631](https://arxiv.org/abs/10.48550/arXiv.2405.08631).

### See Also

grpnet, predict, plot and coef methods.

### Examples

```
x = matrix(rnorm(100 * 20), 100, 20)  
y = rnorm(100)  
fit1 = grpnet(x, glm.gaussian(y), groups = c(1:5,7,9))  
print(fit1)
```



---

set_configs	<i>Set configuration settings.</i>
-------------	------------------------------------

---

**Description**

Set configuration settings.

**Usage**

```
set_configs(name, value = NULL)
```

**Arguments**

name	Configuration variable name.
value	Value to assign to the configuration variable.

**Value**

Assigned value.

**Examples**

```
set_configs("hessian_min", 1e-6)  
set_configs("hessian_min")
```

# Index

- \* **group**
  - plot.cv.glintnet, 37
  - print.cv.grpnet, 46
- \* **lasso**
  - plot.cv.glintnet, 37
  - print.cv.grpnet, 46
- \* **models**
  - plot.cv.glintnet, 37
  - plot.grpnet, 38
  - predict.cv.grpnet, 41
  - predict.glintnet, 42
  - predict.grpnet, 44
  - print.cv.grpnet, 46
  - print.grpnet, 48
- \* **regression**
  - plot.cv.glintnet, 37
  - plot.grpnet, 38
  - predict.cv.grpnet, 41
  - predict.glintnet, 42
  - predict.grpnet, 44
  - print.cv.grpnet, 46
  - print.grpnet, 48

coef.cv.glintnet (predict.cv.glintnet), 40

coef.cv.grpnet (predict.cv.grpnet), 41

coef.glintnet (predict.glintnet), 42

coef.grpnet (predict.grpnet), 44

constraint.box, 3

cv.glintnet, 4

cv.grpnet, 6

gaussian\_cov, 9

glintnet, 11

glm.binomial, 13

glm.cox, 14

glm.gaussian, 15

glm.multigaussian, 16

glm.multinomial, 17

glm.poisson, 18

grpnet, 19

io.snp\_phased\_ancestry, 22

io.snp\_unphased, 23

matrix.block\_diag, 24

matrix.concatenate, 25

matrix.convex\_relu, 26

matrix.dense, 27

matrix.eager\_cov, 28

matrix.interaction, 28

matrix.kronecker\_eye, 29

matrix.lazy\_cov, 30

matrix.one\_hot, 31

matrix.snp\_phased\_ancestry, 32

matrix.snp\_unphased, 33

matrix.sparse, 34

matrix.standardize, 35

matrix.subset, 36

plot.cv.glintnet, 37

plot.cv.grpnet (plot.cv.glintnet), 37

plot.grpnet, 38

predict.cv.glintnet, 40

predict.cv.grpnet, 41

predict.glintnet, 42

predict.grpnet, 44

print.cv.grpnet, 46

print.glintnet, 47

print.grpnet, 48

set\_configs, 49