# Package 'MatrixModels'

October 12, 2022

**Version** 0.5-1

**Date** 2022-09-03

**Title** Modelling with Sparse and Dense Matrices

**Author** Douglas Bates <bates@stat.wisc.edu> and Martin Maechler <maechler@stat.math.ethz.ch>

**Maintainer** Martin Maechler <mmaechler+Matrix@gmail.com>

**Contact** Doug and Martin <Matrix-authors@R-project.org>

**Description** Modelling with sparse and dense 'Matrix' matrices, using modular prediction and response module classes.

**Depends** R (>= 3.6.0)

**Imports** stats, methods, Matrix (>= 1.4-2)

**ImportsNote** _not_yet_stats4

**Encoding** UTF-8

**LazyLoad** yes

**License** GPL (>= 2)

**URL** http://Matrix.R-forge.R-project.org/

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2022-09-11 06:02:55 UTC

## R topics documented:

glm4 . . . . . . . . . . 2
glpModel-class . . . . . . . . . . 4
lm.fit.sparse . . . . . . . . . . 5
mkRespMod . . . . . . . . . . 7
Model-class . . . . . . . . . . 8
model.Matrix . . . . . . . . . . 9
modelMatrix-class . . . . . . . . . . 10
predModule-class . . . . . . . . . . 11

resid-et-al . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
respModule-class . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
reweightPred . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
solveCoef . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
updateMu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
updateWts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

**Index** **18**

---

`glm4` *Fitting Generalized Linear Models (using S4)*

---

## Description

`glm4`, very similarly as standard R's `glm()` is used to fit generalized linear models, specified by giving a symbolic description of the linear predictor and a description of the error distribution.

It is more general, as it fits linear, generalized linear, non-linear and generalized nonlinear models.

## Usage

```
glm4(formula, family, data, weights, subset, na.action,
     start = NULL, etastart, mustart, offset,
     sparse = FALSE, drop.unused.levels = FALSE, doFit = TRUE,
     control = list(...),
     model = TRUE, x = FALSE, y = TRUE, contrasts = NULL, ...)
```

## Arguments

| | |
|---|---|
| `formula` | an object of class `"formula"` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'. |
| `family` | a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See `family` for details of family functions.) |
| `data` | an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `glm` is called. |
| `weights` | an optional vector of 'prior weights' to be used in the fitting process. Should be `NULL` or a numeric vector. |
| `subset` | an optional vector specifying a subset of observations to be used in the fitting process. |
| `na.action` | a function which indicates what should happen when the data contain `NA`s. The default is set by the `na.action` setting of `options`, and is `na.fail` if that is unset. The 'factory-fresh' default is `na.omit`. Another possible value is `NULL`, no action. Value `na.exclude` can be useful. |

`start, etastart, mustart`
: starting values for the parameters in the linear predictor, the predictor itself and for the vector of means.

`offset`
: this can be used to specify an *a priori* known component to be included in the linear predictor during fitting. This should be `NULL` or a numeric vector of length equal to the number of cases. One or more `offset` terms can be included in the formula instead or as well, and if more than one is specified their sum is used. See `model.offset`.

`sparse`
: logical indicating if the model matrix should be sparse or not.

`drop.unused.levels`
: used only when `sparse` is TRUE: Should factors have unused levels dropped? (This used to be true, *implicitly* in the first versions up to July 2010; the default has been changed for compatibility with R's standard (dense) `model.matrix()`.

`doFit`
: logical indicating if the model should be fitted (or just returned unfitted).

`control`
: a list with options on fitting; currently passed unchanged to (hidden) function `IRLS()`.

`model, x, y`
: currently ignored; here for back compatibility with `glm`.

`contrasts`
: currently ignored

`...`
: potentially arguments passed on to fitter functions; not used currently.

## Value

an object of class `glpModel`.

## See Also

`glm()` the standard R function;
`lm.fit.sparse()` a sparse least squares fitter.

The resulting class `glpModel` documentation.

## Examples

```
### All the following is very experimental -- and probably will change: -------

data(CO2, package="datasets")
## dense linear model
str(glm4(uptake ~ 0 + Type*Treatment, data=CO2, doFit = FALSE), 4)
## sparse linear model
str(glm4(uptake ~ 0 + Type*Treatment, data=CO2, doFit = FALSE,
                 sparse = TRUE), 4)

## From example(glm): -----------------

## Dobson (1990) Page 93: Randomized Controlled Trial :
str(trial <- data.frame(counts=c(18,17,15,20,10,20,25,13,12),
                        outcome=gl(3,1,9,labels=LETTERS[1:3]),
                        treatment=gl(3,3,labels=letters[1:3])))
glm.D93 <- glm(counts ~ outcome + treatment, family=poisson, data=trial)
```

```
summary(glm.D93)
c.glm <- unname(coef(glm.D93))
glmM  <- glm4(counts ~ outcome + treatment, family = poisson, data=trial)
glmM2 <- update(glmM, quick = FALSE) # slightly more accurate
glmM3 <- update(glmM, quick = FALSE, finalUpdate = TRUE)
                 # finalUpdate has no effect on 'coef'
stopifnot( identical(glmM2@pred@coef, glmM3@pred@coef),
           all.equal(glmM @pred@coef, c.glm, tolerance=1e-7),
           all.equal(glmM2@pred@coef, c.glm, tolerance=1e-12))


## Watch the iterations --- and use no intercept --> more sparse X
## 1) dense generalized linear model
glmM <- glm4(counts ~ 0+outcome + treatment, poisson, trial,
                      verbose = TRUE)
## 2) sparse generalized linear model
glmS <- glm4(counts ~ 0+outcome + treatment, poisson, trial,
                      verbose = TRUE, sparse = TRUE)
str(glmS, max.lev = 4)
stopifnot( all.equal(glmM@pred@coef, glmS@pred@coef),
           all.equal(glmM@pred@Vtr,  glmS@pred@Vtr) )


## A Gamma example, from McCullagh & Nelder (1989, pp. 300-2)
clotting <- data.frame(u = c(5,10,15,20,30,40,60,80,100),
                       lot1 = c(118,58,42,35,27,25,21,19,18),
                       lot2 = c(69,35,26,21,18,16,13,12,12))
str(gMN <- glm4(lot1 ~ log(u), data=clotting, family=Gamma, verbose=TRUE))
glm. <- glm(lot1 ~ log(u), data=clotting, family=Gamma)
stopifnot( all.equal(gMN@pred@coef, unname(coef(glm.)), tolerance=1e-7) )
```

---

`glpModel-class` *Class "glpModel" of General Linear Prediction Models*

---

## Description

The class "`glpModel`" conceptually contains a very large class of *"General Linear Prediction Models"*.

Its `resp` slot (of class "`respModule`") may model linear, non-linear, generalized linear and non-linear generalized response models.

## Objects from the Class

Objects can be created by calls of the form `new("glpModel", ...)`, but typically rather are returned by our modeling functions, e.g., the (experimental, hence currently hidden) `glm4()`.

## Slots

**`resp`:** a "`respModule`" object.

**`pred`:** a "`predModule`" object.

## Extends

Class "Model", directly.

## Methods

**coef** signature(object = "glpModel"): extract the coefficient vector $\beta$ from the object.

**fitted** signature(object = "glpModel"): fitted values; there may be several types, corresponding to the residuals, see there (below).

**residuals** signature(object = "glpModel"): residuals, depending on the type of the model, there are several types of residuals and correspondingly residuals, see residuals.glm from the **stats** package.

## See Also

glm4() returns fitted glpModel objects.

The constituents of this class are respModule and predModule, both of which have several sub classes.

## Examples

```
showClass("glpModel")

## Use   example(glm4)  or see  help(glm4) for many more examples.
```

---

lm.fit.sparse *Fitter Function for Sparse Linear Models*

---

## Description

A basic computing engine for sparse linear least squares regression.

Note that the exact interface (arguments, return value) currently is **experimental**, and is bound to change. Use at your own risk!

## Usage

```
lm.fit.sparse(x, y, w = NULL, offset = NULL,
              method = c("qr", "cholesky"),
              tol = 1e-7, singular.ok = TRUE, order = NULL,
              transpose = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | *sparse* design matrix of dimension `n * p`, i.e., an R object of a `class` extending `dsparseMatrix`; typically the result of `sparse.model.matrix`. |
| `y` | vector of observations of length `n`, or a matrix with `n` rows. |
| `w` | vector of weights (length `n`) to be used in the fitting process. Weighted least squares is used with weights `w`, i.e., `sum(w * e^2)` is minimized. **Not yet implemented !** |
| `offset` | numeric of length `n`). This can be used to specify an *a priori* known component to be included in the linear predictor during fitting. |
| `method` | a character string specifying the (factorization) method. Currently, `"qr"` or `"cholesky"`. |
| `tol` | [for back-compatibility only; unused:] tolerance for the `qr` decomposition. Default is 1e-7. |
| `singular.ok` | [for back-compatibility only; unused:] logical. If `FALSE`, a singular model is an error. |
| `order` | integer or `NULL`, for `method == "qr"`, will determine how the fill-reducing ordering (aka permutation) for the "symbolic" part is determined (in `cs_amd()`), with the options 0: natural, 1: Chol, 2: LU, and 3: QR, where `3` is the default. |
| `transpose` | logical; if true, use the transposed matrix `t(x)` instead of `x`. |

## Value

Either a single numeric vector or a list of four numeric vectors.

## See Also

`glm4` is an alternative (much) more general fitting function.

`sparse.model.matrix` from the **Matrix** package; the non-sparse function in standard R's package **stats**: `lm.fit()`.

## Examples

```
dd <- expand.grid(a = as.factor(1:3),
                  b = as.factor(1:4),
                  c = as.factor(1:2),
                  d= as.factor(1:8))
n <- nrow(dd <- dd[rep(seq_len(nrow(dd)), each = 10), ])
set.seed(17)
dM <- cbind(dd, x = round(rnorm(n), 1))
## randomly drop some
n <- nrow(dM <- dM[- sample(n, 50),])
dM <- within(dM, { A <- c(2,5,10)[a]
                   B <- c(-10,-1, 3:4)[b]
                   C <- c(-8,8)[c]
                   D <- c(10*(-5:-2), 20*c(0, 3:5))[d]
   Y <- A + B + A*B + C + D + A*D + C*x + rnorm(n)/10
   wts <- sample(1:10, n, replace=TRUE)
```

```
  rm(A,B,C,D)
})
str(dM) # 1870 x 7

X <- Matrix::sparse.model.matrix( ~ (a+b+c+d)^2 + c*x, data = dM)
dim(X) # 1870 x 69
X[1:10, 1:20]

## For now, use  'MatrixModels:::'  --- TODO : export once interface is clear!

Xd <- as(X,"matrix")
system.time(fmDense <- lm.fit(Xd, y = dM[,"Y"]))
system.time( r1 <- MatrixModels:::lm.fit.sparse(X, y = dM[,"Y"]) ) # *is* faster
stopifnot(all.equal(r1, unname(fmDense$coeff), tolerance = 1e-12))
system.time(
     r2 <- MatrixModels:::lm.fit.sparse(X, y = dM[,"Y"], method = "chol") )
stopifnot(all.equal(r1, r2$coef, tolerance = 1e-12),
          all.equal(fmDense$residuals, r2$residuals, tolerance=1e-9)
         )
## with weights:
system.time(fmD.w <- with(dM, lm.wfit(Xd, Y, w = wts)))
system.time(fm.w1 <- with(dM, MatrixModels:::lm.fit.sparse(X, Y, w = wts)))
system.time(fm.w2 <- with(dM, MatrixModels:::lm.fit.sparse(X, Y, w = wts,
                                                     method = "chol") ))
stopifnot(all.equal(fm.w1, unname(fmD.w$coeff), tolerance = 1e-12),
          all.equal(fm.w2$coef, fm.w1, tolerance = 1e-12),
          all.equal(fmD.w$residuals, fm.w2$residuals, tolerance=1e-9)
          )
```

---

`mkRespMod` *Create a respModule object*

---

## Description

Create a `respModule` object, which could be from a derived class such as `glmRespMod` or `nlsRespMod`.

## Usage

```
mkRespMod(fr, family = NULL, nlenv = NULL, nlmod = NULL)
```

## Arguments

| | |
|---|---|
| `fr` | a model frame, usually created by a call to `model.frame`. |
| `family` | an optional glm `family` object (`glmRespMod` objects only). |
| `nlenv` | an environment for evaluation of the nonlinear model, `nlmod`. (`nlsRespMod` objects only). |
| `nlmod` | the nonlinear model function, as a function call (`nlsRespMod` objects only). |

### Details

The internal representation of a statistical model based on a linear predictor expression is derived from a `formula` expression and a `data` argument, possibly supplemented with a `family` object and/or a nonlinear model expression. The steps to obtain this representation usually involve calls to `model.frame` and to `model.matrix` or `model.Matrix`, which encapsulate important parts of this process. This function encapsulates other operations related to weights and offsets and to the model family to create a `respModule` object.

### Value

an object of a class inheriting from `respModule`.

### See Also

The `respModule` class description.

### Examples

```
## see  help("glpModel-class")
```

---

`Model-class` *Mother Class "Model" of all S4 Models*

---

### Description

Class `"Model"` is meant to be the mother class of all (S4) model classes. As some useful methods are already defined for `"Model"` objects, derived classes inherit those “for free”.

### Objects from the Class

A virtual Class: No objects may be created from it.

### Slots

`call`**:** the `call` which generated the model.

`fitProps`**:** a `list`; must be named, i.e., have unique `names`, but can be empty.
When the main object is a *fitted* model, the list will typically have components such as `iter` (non-negative integer) and `convergenece` (`logical` typically).

### Methods

**formula** `signature(x = "Model")`: extract the model formula - if there is one, or `NULL`.

**update** `signature(object = "Model")`: Update the model with a new formula, new data, ...... etc. This semantically equivalent (and as R function almost identical) to the standard `update` (package **stats**).

### See Also

the `glpModel` class in package **MatrixModels** which extends this class.

### Examples

```
showClass("Model")
```

---

`model.Matrix` *Construct Possibly Sparse Design or Model Matrices*

---

### Description

`model.Matrix` creates design matrix, very much like the standard R function `model.matrix`, however returning a dense or sparse object of class `modelMatrix`.

### Usage

```
model.Matrix(object, data = environment(object),
             contrasts.arg = NULL, xlev = NULL,
             sparse = FALSE, drop.unused.levels = FALSE, ...)
```

### Arguments

`object` an object of an appropriate class. For the default method, a model formula or a `terms` object.

`data` a data frame created with `model.frame`. If another sort of object, `model.frame` is called first.

`contrasts.arg` A list, whose entries are values (numeric matrices or character strings naming functions) to be used as replacement values for the `contrasts` replacement function and whose names are the names of columns of `data` containing `factor`s.

`xlev` to be used as argument of `model.frame` if `data` has no `"terms"` attribute.

`sparse` logical indicating if the result should be sparse (of class `sparseModelMatrix`), using `sparse.model.matrix()` (package **Matrix**).

`drop.unused.levels` used only when `sparse` is TRUE: Should factors have unused levels dropped? (This used to be true, *implicitly* in the first versions up to July 2010; the default has been changed for compatibility with R's standard (dense) `model.matrix()`.

`...` further arguments passed to or from other methods.

### Details

model.Matrix() is a simple wrapper either (sparse = FALSE) around the traditional model.matrix() returning a "ddenseModelMatrix", or (sparse = TRUE) around sparse.model.matrix(), returning a "dsparseModelMatrix" object.

model.Matrix creates a design matrix from the description given in terms(object), using the data in data which must supply variables with the same names as would be created by a call to model.frame(object) or, more precisely, by evaluating attr(terms(object), "variables").

For more details, see model.matrix.

### Value

an object inheriting from class modelMatrix, by default, ddenseModelMatrix.

### See Also

model.matrix, sparse.model.matrix.

### Examples

```
data(CO2, package="datasets")
class(sm <- model.Matrix(~ 0+Type*Treatment, data=CO2, sparse=TRUE))
class(dm <- model.Matrix(~ 0+Type*Treatment, data=CO2, sparse=FALSE))
stopifnot(dim(sm) == c(84,4), dim(sm) == dim(dm), all(sm == dm))
```

---

## modelMatrix-class *Class "modelMatrix" and SubClasses*

---

### Description

The class "modelMatrix" and notably its subclass "dsparseModelMatrix" are used to encode additional information, analogously to what the standard R function model.matrix() returns.

### Objects from the Classes

Only "dsparseModelMatrix" and "ddenseModelMatrix" are "actual" (aka non-virtual) classes. For these, objects can be created by calls of the form new("dsparseModelMatrix", x, assign, contrast), where x is a dgCMatrix classed object.

### Slots

The "modelMatrix" mother class contains Matrix plus two extra slots,

**assign:** "integer" vector of length ncol(.), coding the variables which make up the matrix columns, see model.matrix.

**contrasts:** a named list of contrasts, as in model.matrix().

**Dim:** integer vector of length two with the matrix dimensions.

**Dimnames:** list of length two, the dimnames(.) of the matrix.

whereas the (current only) actual classes "d*ModelMatrix", have an at least an additional (numeric slot "x". E.g., "dsparseModelMatrix" has the additional slots

i,p: row number and “pointer” integer vectors, see class "dgCMatrix".

x: "numeric" vector of non-zero entries.

factors: a (possibly empty) list of factorizations.

## Extends

"dsparseModelMatrix" extends class "dgCMatrix" directly,
"ddenseModelMatrix" extends class "dgeMatrix" directly.

## Methods

**show** signature(object = "modelMatrix"): show(.) the matrix, but also the assign and contrasts slots.

**print** signature(x = "modelMatrix"): as show(), however (via ...) allowing to pass further arguments for printing the matrix.

## Author(s)

Martin Maechler

## See Also

sparse.model.matrix will return a "dsparseModelMatrix" object. model.Matrix which is a simple wrapper around the traditional model.matrix and returns a "ddenseModelMatrix" object.

## Examples

```
showClass("modelMatrix")
showClass("dsparseModelMatrix")

## see   example(model.Matrix)
```

---

predModule-class *Class "predModule" and SubClasses*

---

## Description

The class "predModule" and notably its subclasses "dPredModule" and "sPredModule" encapsulate information about linear predictors in statistical models. They incorporate a modelMatrix, the corresponding coefficients and a representation of a triangular factor from the, possibly weighted or otherwise modified, model matrix.

## Objects from the Classes

Objects are typically created by coercion from objects of class ddenseModelMatrix or dsparseModelMatrix.

## Slots

The virtual class "predModule" and its two subclasses all have slots

X: a modelMatrix.

coef: "numeric" coefficient vector of length ncol(.) $:= p$.

Vtr: "numeric" vector of length $p$, to contain $V'r$ ("**V** **t**ransposed **r**").

fac: a representation of a triangular factor, the Cholesky decomposition of $V'V$.

The actual classes "dPredModule" and "sPredModule" specify specific (sub) classes for the two non-trivial slots,

X: a "ddenseModelMatrix" or "dsparseModelMatrix", respectively.

fac: For the "dpredModule" class this factor is a Cholesky object. For the "spredModule" class it is of class CHMfactor.

## Methods

**coerce** signature(from = "ddenseModelMatrix", to = "predModule"): Creates a "dPredModule" object.

**coerce** signature(from = "dsparseModelMatrix", to = "predModule"): Creates an "sPredModule" object.

## Author(s)

Douglas Bates

## See Also

model.Matrix() which returns a "ddenseModelMatrix" or "dsparseModelMatrix" object, depending if its sparse argument is false or true. In both cases, the resulting "modelMatrix" can then be coerced to a sparse or dense "predModule".

## Examples

```
showClass("dPredModule")
showClass("sPredModule")

## see   example(model.Matrix)
```

## resid-et-al *Aliases for Model Extractors*

### Description

Aliases for model extractors; it is an old S and R tradition to have aliases for these three model extractor functions:

`resid()` equivalent to `residuals()`.

`fitted.values()` equivalent to `fitted()`.

`coefficients()` equivalent to `coef()`.

We provide S4 generics and methods for these.

### Methods

**resid** `signature(object = "ANY")`: return the residuals; this is a rarely used *alias* for `residuals()`.

**fitted.values** `signature(object = "ANY")`: return the fitted values; this is a rarely used *alias* for `fitted()`.

**coefficients** `signature(object = "ANY")`: return the coefficients of a model; this is a rarely used *alias* for `coef()`.

### See Also

`residuals`; `Methods` for general information about formal (S4) methods.

## respModule-class *"respModule" and derived classes*

### Description

The `"respModule"` class is the virtual base class of response modules for `glpModel` model objects. Classes that inherit from `"respModule"` include `glmRespMod`, for generalized linear models, `nlsRespMod`, for nonlinear models and `nglmRespMod` for generalized nonlinear models.

### Objects from the Class

Objects from these classes are usually created with `mkRespMod` as part of an `glpModel` object returned by model-fitting functions such as the hidden function `glm4`.

## Slots

`mu`: Fitted mean response.

`offset`: offset in the linear predictor – always present even if it is a vector of zeros. In an `nlsRespMod` object the length of the offset can be a multiple of the length of the response.

`sqrtXwt`: the matrix of weights for the model matrices, derived from the `sqrtrwt` slot.

`sqrtrwt`: Numeric vector of the square roots of the weights for the residuals. For `respModule` and `nlsRespMod` objects these are constant. For `glmRespMod` and `nglmRespMod` objects these are updated at each iteration of the iteratively reweighted least squares algorithm.

`weights`: Prior weights – always present even when it is a vector of ones.

`y`: Numeric response vector.

`family`: a glm family, see `family` for details - `glmRespMod` objects only.

`eta`: numeric vector, the linear predictor that is transformed to the conditional mean via the link function - `glmRespMod` objects only.

`n`: a numeric vector used for calculation of the aic family function (it is really only used with the binomial family but we need to include it everywhere) - `glmRespMod` objects only.

`nlenv`: an environment in which to evaluate the nonlinear model function - `nlsRespMod` objects only.

`nlmod`: an unevaluated call to the nonlinear model function - `nlsRespMod` objects only.

`pnames`: a character vector of parameter names - `nlsRespMod` objects only.

## Methods

**fitted** `signature(object = "respModule")`: fitted values; there may be several types, corresponding to the residuals, see there (below).

**residuals** `signature(object = "respModule")`: residuals, depending on the type of the model, there are several types of residuals and correspondingly residuals, see `residuals.glm` from the **stats** package. Because many of these types of residuals are identical except for objects that inherit from "glmRespMod", a separate method is defined for this subclass.

## See Also

`mkRespMod`

## Examples

```
showClass("respModule")
showClass("glmRespMod")
showClass("nlsRespMod")
```

`reweightPred` *Reweight Prediction Module Structure Internals*

## Description

Update any internal structures associated with sqrtXwt and the weighted residuals. The "V" matrix is evaluated from X using the sqrtXwt matrix and a Vtr vector is calculated.

## Usage

```
reweightPred(predM, sqrtXwt, wtres, ...)
```

## Arguments

| | |
|---|---|
| `predM` | a predictor module |
| `sqrtXwt` | the sqrtXwt matrix |
| `wtres` | the vector of weighted residuals |
| `...` | potentially further arguments used in methods; not used currently. |

## Value

updated predM

## Methods

`signature(predM = "dPredModule", sqrtXwt = "matrix", wtres = "numeric")` ..

`signature(predM = "sPredModule", sqrtXwt = "matrix", wtres = "numeric")` ..

## Examples

```
## TODO
```

`solveCoef` *Solve for the Coefficients or Coefficient Increment*

## Description

The squared length of the intermediate solution is attached as an attribute of the returned value.

## Usage

```
solveCoef(predM, ...)
```

### Arguments

| | |
|---|---|
| `predM` | prediction module, i.e. from class `predModule`. |
| `...` | potentially further arguments used in methods; not used currently. |

### Value

coefficient vector or increment of coef.~vector.

### Methods

`signature(predM = "dPredModule")` ..

`signature(predM = "sPredModule")` ..

### Examples

```
## TODO
```

---

`updateMu` *Update 'mu', the Fitted Mean Response*

---

### Description

Updates the mean vector $\mu$ given the linear predictor $\gamma$. Evaluate the residuals and the weighted sum of squared residuals.

### Usage

```
updateMu(respM, gamma, ...)
```

### Arguments

| | |
|---|---|
| `respM` | a response module, see the `respModule` class. |
| `gamma` | the value of the linear predictor before adding the offset |
| `...` | potentially further arguments used in methods; not used currently. |

### Details

Note that the offset is added to the linear predictor before calculating mu.

The sqrtXwt matrix can be updated but the sqrtrwt should not be in that the weighted sum of squared residuals should be calculated relative to fixed weights. Reweighting is done in a separate call.

### Value

updated `respM`

## Methods

`signature(respM = "glmRespMod", gamma = "numeric")` ..

`signature(respM = "nglmRespMod", gamma = "numeric")` ..

`signature(respM = "nlsRespMod", gamma = "numeric")` ..

`signature(respM = "respModule", gamma = "numeric")` ..

## See Also

The `respModule` class (and specific subclasses); `glm4`.

## Examples

```
## TODO
```

---

`updateWts` *Update the Residual and X Weights - Generic and Methods*

---

## Description

Update the residual weights `sqrtrwt` and $X$ weights `sqrtXwt`.

## Usage

```
updateWts(respM, ...)
```

## Arguments

| | |
|---|---|
| `respM` | a response module, see the `respModule` class. |
| `...` | potentially further arguments used in methods; not used currently. |

## Value

updated response module.

## Methods

`signature(respM = "glmRespMod")` ..

`signature(respM = "respModule")` ..

## Examples

```
## TODO
```

# Index


∗ **array**
lm.fit.sparse, 5
∗ **classes**
glpModel-class, 4
Model-class, 8
modelMatrix-class, 10
predModule-class, 11
respModule-class, 13
∗ **methods**
reweightPred, 15
solveCoef, 15
updateMu, 16
updateWts, 17
∗ **models**
glm4, 2
mkRespMod, 7
model.Matrix, 9
resid-et-al, 13
∗ **regression**
glm4, 2
lm.fit.sparse, 5
reweightPred, 15
solveCoef, 15
updateMu, 16
updateWts, 17

as.data.frame, *2*

call, *8*
CHMfactor, *12*
Cholesky, *12*
class, *6*
coef, *13*
coef,glpModel-method *(*glpModel-class*)*, 4
coefficients,ANY-method *(*resid-et-al*)*, 13
coerce,ddenseModelMatrix,predModule-method *(*predModule-class*)*, 11
coerce,dsparseModelMatrix,predModule-method *(*predModule-class*)*, 11
contrasts, *9*, *10*

ddenseModelMatrix, *10–12*
ddenseModelMatrix-class *(*modelMatrix-class*)*, 10
denseModelMatrix-class *(*modelMatrix-class*)*, 10
dgCMatrix, *10*, *11*
dgeMatrix, *11*
dimnames, *10*
dPredModule-class *(*predModule-class*)*, 11
dsparseMatrix, *6*
dsparseModelMatrix, *10–12*
dsparseModelMatrix-class *(*modelMatrix-class*)*, 10

factor, *9*
family, *2*, *7*, *8*, *14*
fitted, *13*
fitted,glpModel-method *(*glpModel-class*)*, 4
fitted,respModule-method *(*respModule-class*)*, 13
fitted.values,ANY-method *(*resid-et-al*)*, 13
formula, *2*, *8*, *9*
formula,Model-method *(*Model-class*)*, 8

glm, *2*, *3*
glm4, 2, *5*, *6*, *17*
glmRespMod, *7*, *13*, *14*
glmRespMod-class *(*respModule-class*)*, 13
glpModel, *3*, *9*, *13*
glpModel-class, 4

list, *8*, *10*, *11*
lm.fit, *6*
lm.fit.sparse, *3*, 5
logical, *8*

Matrix, *10*

Methods, *13*
mkRespMod, 7, *13, 14*
Model, *5*
Model-class, 8
model.frame, *7–9*
model.Matrix, *8*, 9, *11, 12*
model.matrix, *3*, *8–11*
model.offset, *3*
modelMatrix, *9–12*
modelMatrix-class, 10

na.exclude, *2*
na.fail, *2*
na.omit, *2*
names, *8*
nglmRespMod, *13, 14*
nglmRespMod-class *(*respModule-class*)*, 13
nlsRespMod, *7*, *13, 14*
nlsRespMod-class *(*respModule-class*)*, 13
NULL, *8*
numeric, *11*

offset, *3*
options, *2*

predModule, *4, 5*, *16*
predModule-class, 11
print,modelMatrix-method *(*modelMatrix-class*)*, 10

qr, *6*

resid,ANY-method *(*resid-et-al*)*, 13
resid-et-al, 13
residuals, *13*
residuals,glmRespMod-method *(*respModule-class*)*, 13
residuals,glpModel-method *(*glpModel-class*)*, 4
residuals,respModule-method *(*respModule-class*)*, 13
residuals.glm, *5*, *14*
respModule, *4, 5*, *7, 8*, *16, 17*
respModule-class, 13
reweightPred, 15
reweightPred,dPredModule,matrix,numeric-method *(*reweightPred*)*, 15
reweightPred,sPredModule,matrix,numeric-method *(*reweightPred*)*, 15
reweightPred-methods *(*reweightPred*)*, 15

show, *11*
show,modelMatrix-method *(*modelMatrix-class*)*, 10
solveCoef, 15
solveCoef,dPredModule-method *(*solveCoef*)*, 15
solveCoef,sPredModule-method *(*solveCoef*)*, 15
solveCoef-methods *(*solveCoef*)*, 15
sparse.model.matrix, *6*, *9–11*
sparseModelMatrix, *9*
sparseModelMatrix-class *(*modelMatrix-class*)*, 10
sPredModule-class *(*predModule-class*)*, 11

terms, *9*

update, *8*
update,Model-method *(*Model-class*)*, 8
updateMu, 16
updateMu,glmRespMod,numeric-method *(*updateMu*)*, 16
updateMu,nglmRespMod,numeric-method *(*updateMu*)*, 16
updateMu,nlsRespMod,numeric-method *(*updateMu*)*, 16
updateMu,respModule,numeric-method *(*updateMu*)*, 16
updateMu-methods *(*updateMu*)*, 16
updateWts, 17
updateWts,glmRespMod-method *(*updateWts*)*, 17
updateWts,respModule-method *(*updateWts*)*, 17
updateWts-methods *(*updateWts*)*, 17