Package: rpql (via r-universe)

October 7, 2026

Title Regularized PQL for Joint Selection in GLMMs

Version 0.8.4

Date 2026-09-06

Maintainer Francis K.C. Hui <fhui28@gmail.com>

Description Performs joint selection in Generalized Linear Mixed Models (GLMMs) using penalized likelihood methods. Specifically, the Penalized Quasi-Likelihood (PQL) is used as a loss function, and penalties are then augmented to perform simultaneous fixed and random effects selection. Regularized PQL avoids the need for integration (or approximations such as the Laplace's method) during the estimation process, and so the full solution path for model selection can be constructed relatively quickly.

License GPL-3

Imports lme4, Matrix, MASS, mvtnorm, Rcpp,

Suggests nlme

NeedsCompilation yes

LinkingTo Rcpp, RcppArmadillo

Author Francis K.C. Hui [aut, cre]

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

Date/Publication 2026-09-06 22:10:02 UTC

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

RemoteRef HEAD

RemoteSha 28eab3127556a169148c84f31f4d39d3de24ed2e

Contents

rpql-package 2
build.start.fit 3
calc.marglogL 6
gendat.glmm 8

lseq 11
nb2 13
rpql 14
rpqlseq 24
summary.rpql 26
ZIP 27

Index **29**

`rpql-package` *Joint effects selection in GLMMs using regularized PQL*

Description

`rpql` offers fast joint selection of fixed and random effects in Generalized Linear Mixed Model (GLMMs) via regularization. Specifically the penalized quasi-likelihood (PQL, Breslow and Clayton, 1993) is used as a loss function, and penalties are added on to perform fixed and random effects selection e.g., the lasso (Tibshirani, 1996) penalty. This method of joint selection in GLMMs, referred to regularized PQL, is very fast compared to information criterion and hypothesis testing, and has attractive large sample properties (Hui et al., 2016). Its performance however may not be great if the amount of data to estimate each random effect is not large, i.e. the cluster size is not large.

Details

Package:	rpql
Type:	Package
Version:	0.8.1
Date:	2023-08-01
License:	GPL-2

Author(s)

Francis K.C. Hui [aut, cre]

Maintainer: Francis K.C. Hui <fhui28@gmail.com>

References

- Breslow, N. E., and Clayton, D. G. (1993). Approximate inference in generalized linear mixed models. Journal of the American Statistical Association, 88, 9-25.
- Hui, F.K.C., Mueller, S., and Welsh, A.H. (2017). Joint Selection in Mixed Models using Regularized PQL. Journal of the American Statistical Association, 112, 1323-1333.
- Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological), 58, 267-288.

Examples

```
## Please see examples in help file for the rpql function
```

`build.start.fit` *Constructs a start fit for use in the* `rpql` *function*

Description

Takes a GLMM fitted using the `lme4` package i.e., using either the `lmer` or `glmer` functions, and construct a list containing starting values for use in the `start` argument in main fitting function `rpql`. It also constructs adaptive lasso weights, which can subsequently be used in the `pen.weights` arguments in the `rpql` function, if the adaptive lasso penalty is used for variable selection.

Usage

```
build.start.fit(lme4.fit, id = NULL, gamma = 0, cov.groups = NULL)
```

Arguments

`lme4.fit`	An object of class "lmerMod" or "glmerMod", obtained when fitting a (G)LMM using the `lmer` and `glmer` functions in the `lme4` package.
`id`	A optional list with each element being a vector of IDs that reference the model matrix in the corresponding element in the list `Z`. Each vector of IDs *must* be integers (but not factors). Note this is optional argument as it is only use for non-compulsory formatting purposes in the function.
`gamma`	A vector of power parameters, γ, for use in constructing adaptive lasso weights. Can be a vector of one or two elements. If two elements, then the first and second elements are the power parameter for the fixed and random effect weights respectively. If one element, the same power parameter is used for both fixed and random effect weights. Defaults to 0, in which case the weights are all equal to 1 i.e., it reverts to the unweighted lasso penalty.
`cov.groups`	A vector specifying if fixed effect coefficients (including the intercept) should be regarded and therefore penalized in groups. For example, if one or more of the fixed effect covariates are factors, then `lme4` will automatically create dummy variables in the model matrix and estimate coefficients for each level, using one level as the reference. `cov.groups` is then used to identify all the coefficients that corresponds to that factor, such that all of these coefficients are penalized collectively as a group. Defaults to NULL, in which case it is assumed all coefficients should be treated independently. Please see the details and examples for more details.

Details

This function is mainly used when: 1) you want to produce good starting values for the main fitting function `rpql`, and so you fit a saturated (full) GLMM using `lme4` and use the estimates from there as starting values, and/or 2) you want to obtain adaptive lasso weights of the form $weight_k = |\tilde{parameter}_k|^{-\gamma}$, where $\gamma > 0$ is the power parameter and $\tilde{parameter}_k$ is the parameter estimate from the saturated model fit. For regularized PQL specifically, this function will construct adaptive lasso weights from the `lme4` fit as follows: Let w^F and w^R denote fixed and random effect adaptive weights respectively. Then we have,

$$w_k^F = |\tilde{\beta}_k|^{-\gamma_1}$$
$$w_l^R = |\tilde{\Sigma}_{ll}|^{-\gamma_2},$$

where $\tilde{\beta}_k$ is the estimated coefficient for the k^{th} fixed effect, $\tilde{\Sigma}_{ll}$ is the l^{th} diagonal element from the estimated random effects covariance matrix, and γ is a vector of two power parameters; see Zou (2006) for the adaptive lasso, and Hui et al. (2016) for regularized PQL selection in GLMMs using on adaptive lasso type penalties.

If `cov.groups` is supplied, this means that some of the fixed effects coefficients should be treated and penalized collectively as a group. The most common cases where this is used is when you have factor or categorical variables with more than two levels, or when you have polynomial terms that should be dealt with together. For instance, suppose you have a model matrix consisting of six columns, where first three columns correspond to separate covariates (including the intercept) and the last three columns all correspond to dummy variables created for a factor variable with four levels , e.g. soil moisture with levels dry, moderately moist, very moist, wet. The coefficients from the last three columns should then be penalized together, and so we can set `cov.groups = c(1,2,3,4,4,4)`.

In doing so, the adaptive lasso weights for the grouped coefficients are then constructed differently. Following on from the example above, we have the fixed effect weight for soil moisture defined as

$$w^F = \|\tilde{\beta}\|^{-\gamma_1},$$

where $\|\cdot\|$ corresponds to the L2-norm and $\tilde{\beta}$ are the fixed effect coefficients belonging in the group (three in this case). When entered into the `rpql` function, an adaptive group lasso (Wang and Leng, 2008) is applied to these set of coefficients, such that they are all encouraged to be shrunk to zero at the same time.

Of course, after construction the adaptive lasso weights can be manually altered before entering into the main `rpql` function e.g., if one wants certain fixed and/or random effects to not be penalized.

Value

A list containing the following elements

`fixef`	Fixed effect coefficient estimates from `lme4.fit`.
`ranef`	A list of random effect predicted coefficients from `lme4.fit`.
`ran.cov`	A list of random effects covariance matrices from `lme4.fit`.
`cov.groups`	The argument `cov.groups`. Defaults to `NULL`.

`pen.weights` A list of adaptive lasso weights constructed from `lme4.fit`. Contains elements `pen.weight$fixed` and `pen.weights$random`, which are the weights for the fixed and random effects respectively. Please see details above as to their construction.

Warnings

- In order to construct sensible starting values and weights, this function should really only be used when `lme4.fit` is a fit of the saturated GLMM, i.e. all fixed and random effects included.

Author(s)

Francis K.C. Hui [aut, cre]

Maintainer: Francis K.C. Hui <fhui28@gmail.com>

References

- Hui, F.K.C., Mueller, S., and Welsh, A.H. (2016). Joint Selection in Mixed Models using Regularized PQL. Journal of the American Statistical Association: accepted for publication.
- Wang, H., and Leng, C. (2008). A note on adaptive group lasso. Computational Statistics & Data Analysis, 52, 5277-5286.
- Zou, H. (2006). The adaptive lasso and its oracle properties. Journal of the American statistical association, 101, 1418-1429.

See Also

`rpql` for fitting and performing model selection in GLMMs using regularized PQL, which may use the values obtained from `build.start.fit` for starting values and adaptive lasso weights.

Examples

```
##-------------------------
## Example 1: Bernoulli GLMM with grouped covariates.
## Independent cluster model with 50 clusters and equal cluster sizes of 10
## Nine covariates where the last covariate (soil type) is a factor with four levels
##-------------------------
n <- 50; p <- 8; m <- 10
set.seed(123)
X <- data.frame(matrix(rnorm(n*m*p),n*m,p), soil=sample(1:4,size=m*n,replace=TRUE))
X$soil <- factor(X$soil)
X <- model.matrix(~ ., data = X)
colnames(X) <- paste("X",1:ncol(X),sep="")

Z <- X[,1:5] ## Random effects model matrix taken as first five columns
true_betas <- c(-0.1,1,-1,1,-1,1,-1,0,0,0,0,0)
true_D <- matrix(0,ncol(Z),ncol(Z))
true_D[1:3,1:3] <- matrix(c(9,4.8,0.6,4.8,4,1,0.6,1,1),
 3,3,byrow=TRUE) ## 3 important random effects

simy <- gendat.glmm(id = list(cluster = rep(1:n,each=m)), X = X, beta = true_betas,
```

```
 Z = list(cluster = Z), D = list(cluster = true_D), family = binomial())


## Not run:
library(lme4)
dat <- data.frame(y = simy$y, simy$X, simy$Z$cluster, simy$id)
fit_satlme4 <- glmer(y ~ X - 1 + (Z - 1 | cluster), data = dat,
 family = "binomial")
fit_sat <- build.start.fit(fit_satlme4, id = simy$id, gamma = 2,
 cov.groups = c(1:9,10,10,10))

new.fit <- rpql(y = simy$y, X = simy$X, Z = simy$Z, id = simy$id, lambda = 0.01,
 pen.type = "adl", pen.weights = fit_sat$pen.weights,
 cov.groups = fit_sat$cov.groups, start = fit_sat, family = binomial())

## End(Not run)
```

`calc.marglogL` *Calculate the marginal log-likelihood for a GLMM fitted using* `rpql`

Description

After fitting and performing joint (fixed and random effects) using regularized PQL, one may then (for one reason or another) want to calculate the marginal likelihood for the (sub)model, possibly on a test dataset for prediction. This is the main purpose of `calc.marglogL`.

Usage

```
calc.marglogL(new.data, fit, B = 1000)
```

Arguments

`new.data`	A list containing the elements `new.data$y`, `new.data$X`, and `new.data$Z`. These correspond respectively to the responses, fixed effects model matrix, and random effects model matrices that the marginal log-likelihood is be calculated on. No check is made against the elements in `fit` to ensure that these are of the correct dimensions compared, and furthermore it is assumed that `new.data$Z` is a list in the same order as the `Z` used when fitting the original model via `rpql`.
`fit`	An object of class `pqrl`. In the least, `fit` should be a list containing the elements `fit$family` for the family, e.g. gaussian(), poisson(), `fit$fixef` for the estimated vector of fixed effects, `fit$ran.cov` which is a list of estimated random effects covariance matrices. If appropriate, `fit` may also contain the elements `fit$phi` for the estimated variance parameter in normal, and negative binomial GLMMs, `fit$shape` for the estimated shape parameter used in Gamma GLMMs, `fit$trial.size` for the trial size(s) for binomial GLMMs, and `fit$zeroprob` for the estimated probability of zero inflation in ZIP GLMMs.

B — A positive integer for the number of random effects examples to generate, when performing Monte-Carlo integration. Defaults to 1000.

Details

Regularized PQL performs penalized joint (fixed and random effects) selection for GLMMs, where the penalized quasi-likelihood (PQL, Breslow and Clayton, 1993) is used the loss function. After fitting, one may then wish to calculate the marginal log-likelihood for the (sub)model, defined as

$$\ell = \log\left(\int f(\boldsymbol{y};\boldsymbol{\beta},\boldsymbol{b},\phi)f(\boldsymbol{b};\boldsymbol{\Sigma})d\boldsymbol{b}\right),$$

where $f(\boldsymbol{y};\boldsymbol{\beta},\boldsymbol{b},\phi)$ denotes the conditional likelihood of the responses $\boldsymbol{y}$ given the fixed effects $\boldsymbol{\beta}$, random effects $\boldsymbol{b}$, and nuisance parameters ϕ if appropriate, and $f(\boldsymbol{b};\boldsymbol{\Sigma})$ is the multivariate normal distribution for the random effects, with covariance matrix $\boldsymbol{\Sigma}$. `calc.marglogL` calculates the above marginal likelihood using Monte-Carlo integration.

Admittedly, this function is not really useful for fitting the GLMM *per-se*: it is never called by the main function `rpql`, and the marginal likelihood is (approximately) calculated anyway if `hybrid.est = TRUE` and the final submodel is refitted using `lme4`. Where the function comes in handy is if you have a validation or test dataset, and you want to calculated the predicted (log) likelihood of the test data given the regularized PQL fit.

Value

The marginal log-likelihood of `new.data` given the GLMM in `fit`.

Warnings

- No check is made to see if the dimensions of the elements `new.data` and `fit` match, e.g. the number of columns in `new.data$X` is equal to the number of elements in `fit$fixef`. Please ensure they are!
- Monte-Carlo integration is computationally intensive especially if $\boldsymbol{y}$ is long!

Author(s)

Francis K.C. Hui [aut, cre]

Maintainer: Francis K.C. Hui <fhui28@gmail.com>

References

- Breslow, N. E., & Clayton, D. G. (1993). Approximate inference in generalized linear mixed models. Journal of the American Statistical Association, 88, 9-25.

See Also

`rpql` for fitting and performing model selection in GLMMs using regularized PQL. `lme4` also approximately calculates the marginal log-likelihood when fitting a GLMM.

Examples

```
## Not given
```

`gendat.glmm` *Simulates datasets based on a Generalized Linear Mixed Model (GLMM).*

Description

Datasets are simulated from a GLMM given a series of inputs including: model matrices `X` and `Z` for the fixed and random effects respectively, a set of true fixed effect coefficients `beta`, a list of true random effect covariance matrices `D`, the family of response, and some other nusiance parameters if appropriate.

Usage

```
gendat.glmm(id, X, beta, Z, D, trial.size = 1, family = gaussian(),
  phi = NULL, shape = NULL, zeroprob = NULL, upper.count = Inf)
```

Arguments

`id`	A list with each element being a vector of IDs that reference the model matrix in the corresponding element in the list `Z`. Each vector of IDs *must* be integers (but not factors).
`X`	A model matrix of corresponding to the fixed effects. A column of ones should be included if a fixed intercept is to be included in the model.
`beta`	A vector of true fixed effect parameters, with the same length as the number of columns in `X`.
`Z`	A list with each element being a model matrix for a set of random effects. Each element of `Z` is referenced by a vector of IDs given by the corresponding element in the list `id`. Each model matrix (element of `Z`) should have the same number of rows as the length of `y`.
`D`	A list with each element being a symmetric random effects covariance matrix which is used to generate random effects. These random effects are then applied to the corresponding element in the list `Z`, and are referenced by the corresponding element in the list `id`.
`trial.size`	The trial size if `family = binomial()`. Either takes a single non-zero value or a vector of non-zero values with length the same as the number of rows in `X`. The latter allows for differing trial sizes across responses. Defaults to 1.
`family`	The distribution for the responses in GLMM. The argument must be applied as a object of class "family". Currently supported arguments include: `gaussian()`, `poisson()`, `binomial()`, `Gamma()`, `nb2()` for negative binomial, and `ZIP()` for zero-inflated Poisson.

`phi` A non-zero value for the true variance parameter σ^2 if `family = gaussian()`, or the overdispersion parameter if `family = nb2()`, where the negative binomial variance is parameterized as $V = \mu + \phi\mu^2$. Defaults to NULL.

`shape` A non-zero value for the shape parameter a if `family = Gamma()`, where the variance is parameterized as $V = \mu^2/a$. Defaults to NULL.

`zeroprob` A value between 0 and 1 for the probability of zero inflation if `family = ZIP()` for zero-inflated Poisson. Defaults to NULL.

`upper.count` A non-zero integer which allows the user to control the maximum value of the counts generates for datasets when `family = poisson()` or `nb2()`. When the responses are simulated, a `while` loop is run to ensure that all responses generated are less than or equal to `upper.count`. Default to `Inf`.

Details

The relationship between the mean of the responses and covariates in a GLMM is given as follows: For $i = 1, \ldots, n$, where n is, equivalently, the number of rows in `X`, the length of each element in `id`, and the number of rows in each element of `Z`, we have

$$g(\mu_i) = \boldsymbol{x}_i^T\boldsymbol{\beta} + \boldsymbol{z}_{i1}^T\boldsymbol{b}_{i1} + \boldsymbol{z}_{i2}^T\boldsymbol{b}_{i2} + \ldots,$$

where $g(\cdot)$ is the link function, μ_i is the mean of the distribution for observation i, $\boldsymbol{x}_i$ is row i of the fixed effects model matrix `X`, and $\boldsymbol{\beta}$ is the fixed effects coefficients. For the random effects, $\boldsymbol{z}_{i1}$ is row i of the random effects model matrix in the first element of `Z`, while $\boldsymbol{b}_{i1}$ is the vector of random effects generated for observation i based on the first element of `D`. The remaining parameters $\boldsymbol{z}_{i2}$, $\boldsymbol{b}_{i2}$ and so on, are defined similarly.

Having lists for `id`, `Z`, and `D` allows for multiple sets of random effects to be included in the true GLMM. This is analogous to the `lme4` package, where multiple random effects are permitted in the formula, e.g., `(1|creek) + (1|creek:sample)`. If the true GLMM contains only one set of random effects, e.g., in longitudinal data, then the three lists will all contain only one element. Cases with multiple sets of random effects include nested and crossed designs, in which case `id`, `Z`, and `D` will have two or more elements.

It is recommended that the user think through and design these lists carefully to ensure that they are actually constructing a true GLMM that they want to simulated data from. Yes it takes some getting use too, and we apologize for this =(Please see examples below for some ideas.

Finally, note that some of the elements of `beta` can be zero, i.e. truly unimportant fixed effects. Likewise, each element of `D` can be a random effects covariance matrix containing zero rows and columns, i.e. truly unimportant random effects.

Value

A list containing the following elements

`y` The vector simulated responses.

`b` A list with each element being a matrix of random effects simulated from a multivariate normal distribution with mean zero and covariance matrix equal to the corresponding element in the list `D`. For each element in `b`, the number of columns of the matrix equals the dimension of corresponding covariance matrix

element in D, while the number of rows equals to the number of unique IDs in the corresponding element of the list id.

id, X, Z, beta, D, phi, shape, zeroprob, trial.size, family
: Some of the arguments entered into gendat.glmm.

nonzero.beta
: A vector indexing the non-zero values of beta, i.e. the truly important fixed effects.

nonzero.b
: A list with each element being a vector indexing the non-zero diagonal variances in the corresponding element of the list D, i.e. the truly important random effects.

Author(s)

Francis K.C. Hui [aut, cre]

Maintainer: Francis K.C. Hui <fhui28@gmail.com>

References

- Schielzeth, H., & Nakagawa, S. (2013). Nested by design: model fitting and interpretation in a mixed model era. Methods in Ecology and Evolution, 4, 14-24.

See Also

rpql for fitting and performing model selection in GLMMs using regularized PQL.

Examples

```
##-------------------------
## Example 1: Linear Mixed Models
## Independent cluster model with 50 clusters
## Nine covariates including a fixed and random intercept
## Please note the rpql is currently not optimized for LMMs!
##-------------------------
library(mvtnorm)
library(lme4)

n <- 50; m <- 10; p <- 8;
## Generate rows of a model matrix from a multivariate normal distribution with
## AR1 covariance structure.

H <- abs(outer(1:p, 1:p, "-"))
X <- cbind(1,rmvnorm(n*m,rep(0,p),sigma=0.5^H));

Z <- X
true_betas <- c(1,3,2,1.5,-1,0,0,0,0) ## 5 important fixed effects
true_D <- matrix(0,p+1,p+1) ## 3 important random effects
true_D[1:3,1:3] <- matrix(c(9,4.8,0.6,4.8,4,1,0.6,1,1),3,3,byrow=TRUE)

simy <- gendat.glmm(id = list(cluster = rep(1:n,each=m)), X = X, beta = true_betas,
 Z = list(cluster = Z), D = list(cluster = true_D), phi = 1, family = gaussian())
## Notice how id, Z, and D all are lists with one element, and that
## the name of the first element (a generic name "cluster") is the
```

```
## same for all three lists.
## id is where the action takes place. In particular, id$cluster is
## designed so that the first m elements correspond to cluster 1,
## the second m elements correspond to cluster 2, and so forth.
## In turn, the first m rows of X and Z$cluster correspond
## to cluster 1, and so on.

## Not run:
dat <- data.frame(y = simy$y, simy$X, simy$Z$cluster, simy$id)
fit_satlme4 <- lmer(y ~ X - 1 + (Z - 1 | cluster), data = dat,
 REML = FALSE)
fit_sat <- build.start.fit(fit_satlme4, gamma = 2)


lambda_seq <- lseq(1e-4,1,length=100)
fit <- rpqlseq(y = simy$y, X = simy$X, Z = simy$Z, id = simy$id,
 family = gaussian(), lambda = lambda_seq, pen.type = "adl",
 pen.weights = fit_sat$pen.weights, start = fit_sat)

summary(fit$best.fit[[5]])  ## Second of the hybrid ICs
# apply(fit$collect.ics, 2, which.min) ## Look at best fit chosen by different ICs

## End(Not run)

## Please see other examples in help file for the \code{rpql} function.
```

`lseq` *Generates a sequence of tuning parameters on the log scale*

Description

Generates a sequence of tuning parameters λ that are equally spaced on the log-scale. It may be used as part of constructing a solution path for the main fitting function `rpql`.

Usage

```
lseq(from, to, length, decreasing = FALSE)
```

Arguments

`from`	The minimum tuning parameter to start the sequence from.
`to`	The maximum tuning parameter to go to.
`length`	The length of the sequence.
`decreasing`	Should the sequence be in ascending or descending order?

Details

For joint selection of fixed and random effects in GLMMs, regularized PQL (Hui et al., 2016) works taking the penalized quasi-likelihood (PQL, Breslow and Clayton, 1993) as a loss function, and then sticking on some penalties in order to model variable. The penalties will depend upon one or more tuning parameters $\lambda > 0$, and the typical way this is chosen is to construct a sequence of λ values, fit the regularized PQL to each one value, and then use a method like information criterion to select the best λ and hence the best model. Please see the help file for `rpql` for more details, and `glmnet` (Friedman et al., 2010) and `ncvreg` (Breheny, and Huang, 2011) as examples of other packages that do penalized regression and involve tuning parameter selection.

The idea of equally spacing the sequence of λ's on the log (base 10) scale may not necessary be what you want to do, and one is free to use the standard `seq()` function for constructing sequences. By equaling spacing them on log-scale, it means that there will be a large concentration of small tuning parameter values, with less large tuning parameter values (analogous to a right skewed distribution). This may be useful if you believe the that most of the penalization/variable selection action takes place on smaller values of λ.

It is somewhat of an art form to construct a good sequence of tuning parameter values: the smallest λ should produce the saturated model if possible, and the largest λ should shrink most if not all covariates to zero i.e., the null model. Good luck!

Value

A sequence of tuning parameter values of length equal to `length`.

Author(s)

Francis K.C. Hui [aut, cre]

Maintainer: Francis K.C. Hui <fhui28@gmail.com>

References

- Breheny, P. and Huang, J. (2011) Coordinate descent algorithms fof nonconvex penalized regression, with applications to biological feature selection. The Annals of Appliedv Statistics, 5, 232-253.
- Breslow, N. E., and Clayton, D. G. (1993). Approximate inference in generalized linear mixed models. Journal of the American Statistical Association, 88, 9-25.
- Friedman, J., Hastie T., and Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. Journal of Statistical Software, 33, 1-22. URL: http://www.jstatsoft.org/v33/i01/.
- Hui, F.K.C., Mueller, S., and Welsh, A.H. (2016). Joint Selection in Mixed Models using Regularized PQL. Journal of the American Statistical Association: accepted for publication.

See Also

`rpql` for fitting and performing model selection in GLMMs using regularized PQL.

Examples

```
## Please see examples in help file for the rpql function
```

nb2 *A negative binomial family*

Description

Since the negative binomial is not a family in base R, an nb2() family has been created which establishes the negative binomial as a family for use in the main rpql function. Only the log link is available at the moment, with the variance parameterized as $V = \mu + \phi\mu^2$ where ϕ is the overdispersion parameter.

Usage

```
nb2()
```

Details

Used in the form rpql(y, ..., family = nb2(), ...).

Value

An object of class "family"

Author(s)

Francis K.C. Hui [aut, cre]

Maintainer: Francis K.C. Hui <fhui28@gmail.com>

Examples

```
## Not run:
## The function is currently defined as follows
nb2 <- function () {
    link <- "log"
    linkfun <- function(mu) log(mu)
    linkinv <- function(eta) pmax(exp(eta), .Machine$double.eps)
    mu.eta <- function(eta) pmax(exp(eta), .Machine$double.eps)
    variance <- function(mu, phi) mu + phi * mu^2
    valideta <- function(eta) TRUE
    validmu <- function(mu) all(mu > 0)
    structure(list(family = "negative.binomial", link = "log",
        linkfun = linkfun, linkinv = linkinv, mu.eta = mu.eta,
        variance = variance, valideta = valideta, validmu = validmu,
        name = link), class = "family")
  }

## End(Not run)
```

`rpql` *Joint effects selection in GLMMs using regularized PQL.*

Description

`rpql` offers fast joint selection of fixed and random effects in Generalized Linear Mixed Model (GLMMs) via regularization. The penalized quasi-likelihood (PQL) is used as a loss function, and penalties are added on to perform fixed and random effects selection. This method of joint selection in GLMMs, referred to regularized PQL, is fast compared to information criterion and hypothesis testing (Hui et al., 2016).

Please note `rpql` is the core workshops function that performed regularized PQL on a single set of tuning parameters. `rpqlseq` is a wrapper to permit a sequence of tuning parameter values. The latter is often what users want to use.

Usage

```
rpql(y, ...)

## Default S3 method:
rpql(y, X, Z, id, family = gaussian(), trial.size = 1, lambda,
  pen.type = "lasso", start = NULL, cov.groups = NULL, pen.weights = NULL,
  hybrid.est = FALSE, offset = NULL, intercept = TRUE, save.data = FALSE,
  control = list(tol = 1e-4, maxit = 100, trace = FALSE, restarts = 5,
  scad.a = 3.7, mcp.gamma = 2, lasso.lambda.scale = TRUE, seed = NULL), ...)

## S3 method for class 'rpql'
print(x, ...)
```

Arguments

`y`	A vector of responses
`X`	A model matrix corresponding to the fixed effects. It should have the same number of rows as the length of `y`. An intercept column must be included if a fixed intercept is desired.
`Z`	A list with each element being a model matrix for a set of random effects. Each element of `Z` is referenced by a vector of IDs given by the corresponding element in the list `id`. Each model matrix (element of `Z`) should have the same number of rows as the length of `y`.
`id`	A list with each element being a vector of IDs that reference the model matrix in the corresponding element in the list `Z`. Each vector of IDs *must* be integers (but not factors).
`x`	An object for class "rpql".

`family` The distribution for the responses in GLMM. The argument must be applied as a object of class "family". Currently supported arguments include: `gaussian()`, `poisson()`, `binomial()`, `Gamma()`, `nb2()` for negative binomial, and `ZIP()` for zero-inflated Poisson.

`trial.size` The trial size if `family = binomial()`. Either takes a single non-zero value or a vector of non-zero values with length the same as the number of rows in `X`. The latter allows for differing trial sizes across responses. Defaults to 1.

`lambda` A vector of length one or two specifying the tuning parameters used in regularized PQL. If two elements are supplied, then first and second elements are for the fixed and random effects penalty respectively. If one element, then it is applied to both penalties.

`pen.type` A vector of one or two strings, specifying the penalty used for variable selection. If two elements are supplied, then first and second strings are the fixed and random effects penalty respectively. If one element, the same type of penalty is used. Currently supported argument include: "`lasso`" for standard lasso (Tibshirani, 1996), "`scad`" for SCAD penalty with a controlled by `control$scad.a` (Fan and Li, 2001), "`adl`" for adaptive lasso (Zou, 06), "`mcp`" for MC+ penalty with γ controlled by controlled by `control$mcp.gamma` (Zhang, 2010). If the adaptive lasso is used, then `pen.weights` must also be supplied. Defaults to standard lasso penalty for both fixed and random effects.

`start` A list of starting values. It must contain the following elements: `start$fixef` as starting values for the fixed effect coefficients, `start$ranef` which is a list containing matrices of starting values for the random effects coefficients. It may also contain `start$D` which is a list of matrices to act as starting values for random effects covariance matrices.

`cov.groups` A vector specifying if the columns of `X` (including the intercept) should be regarded and therefore penalized in groups. For example, if one or more of the fixed effect covariates are factors, then `lme4` will automatically create dummy variables in the model matrix and estimate coefficients for each level, using one level as the reference. `cov.groups` is then used to identify all the coefficients that corresponds to that factor, such that all of these coefficients are penalized collectively as a group. Defaults to NULL, in which case it is assumed all coefficients should be treated independently. Please see the details and examples for more details.

`pen.weights` A list containing up to two elements for additional (adaptive lasso) weights to be included for penalization. This must be supplied if `pen.type` has one or both elements set to "adl", otherwise it is optional. A weights equal to zero implies no penalization is applied to the parameter. The two elements in the list are as follows: for fixed effects, `pen.type$fixed` should be a vector with length equal to the number of columns in `X`. For random effects, `pen.weights$ran` should be a list of the same length as the list `Z`, where each element in that list is a vector with length equal to the number of columns in the corresponding element of the list `Z` (recall that each element of `Z` is a model matrix). Defaults to NULL, in which case there are no weights involved in the penalization.

`hybrid.est` Should a hybrid estimation approach be used? That is, once model selection is performed using regularized PQL, should the submodel be re-estimated using the `lme4` package, if possible? Defaults to FALSE.

`offset` This can be used to specify an *a priori* known component to be included in the linear predictor during fitting. It should be numeric vector of length equal to `y`. Defaults to NULL.

`intercept` Is one of the columns of `X` an intercept term? This is used to indicate the presence of a fixed intercept in the model, which subsequently will NOT be penalized. Defaults to TRUE.

`save.data` Should `y`, `X`, and `Z`, be saved as part of the output? Defaults to FALSE. The data is not saved by default in order to save memory.

`control` A list controlling the finer details of the rPQL algorithm. These include:

`tol`: Tolerance value for convergence in the regularized PQL to be declared, where convergence is measured as the difference between the estimated parameters in successive iterations. Defaults to a value of 1e-4.

`maxit`: The maximum number of update iterations for regularized PQL. Defaults to 100.

`trace`: Should the update estimates of the fixed effect coefficients and that random effect covariance matrices be printed at each iteration? Defaults to FALSE.

`restarts`: The number of restarts to try in case the algorithm diverges, i.e. the fixed effect coefficients and /or random effects covariance matrices "blow up". Defaults to a value of 5. Divergence is mostly likely to occur when you have count responses with some extremely large counts in there, in which regularized PQL can throw a hissy fit.

`scad.a`, `mcp.gamma`: Controls the a and γ parameters in the SCAD and MC+ penalty respectively. Defaults to $a = 3.7$ (Fan and Li, 2001) and $\gamma = 2$ (Zhang, 2010) respectively. Please note these parameters are only in use when `pen.type` involves these penalties.

`seed`: A seed that can be used if results need to be replicated. Defaults to NULL, in which case a random seed is used.

`...` Not used.

Details

Intro

Generalized Linear Mixed Models (GLMMs) are an extension of Generalized Linear Models (GLM, see the `glm` function) to include one or more sets of random effects. For $i = 1, \ldots, n$, where n is the length of `y`, we have

$$g(\mu_i) = \boldsymbol{x}_i^T \boldsymbol{\beta} + \boldsymbol{z}_{i1}^T \boldsymbol{b}_{i1} + \boldsymbol{z}_{i2}^T \boldsymbol{b}_{i2} + \ldots,$$

where $g(\cdot)$ is the link function, μ_i is the mean of the distribution for observation i, $\boldsymbol{x}_i$ is row i of the fixed effects model matrix `X`, and $\boldsymbol{\beta}$ is the fixed effects coefficients. For the random effects, $\boldsymbol{z}_{i1}$ is row i of the random effects model matrix in the first element of `Z`, $\boldsymbol{z}_{i2}$ is from the second element of `Z` and so forth. The random effects $\boldsymbol{b}_{i1}, \boldsymbol{b}_{i2}, \ldots$ are drawn from a multivariate normal distribution with mean zero and differing covariance matrices $\boldsymbol{D}_1, \boldsymbol{D}_2, \ldots$.

Note that having lists for `id`, `Z`, allows for multiple sets of random effects to be included in the GLMM. This is analogous to the `lme4` package, where multiple random effects are permitted in

the formula e.g., `(1|creek) + (1|creek:sample)`. If the GLMM contains only one set of random effects, e.g., in longitudinal data, then the two lists will all contain only one element. Cases where multiple sets of random effects may be used include nested and crossed designs, in which case `id, Z`, will have two or more elements. It is recommended that the user think through and design these lists carefully to ensure that they are actually constructing the appropriate GLMM of interest. Yes it takes some getting use too, and we apologize for this =(Please see examples below for some ideas.

Regularized PQL

Regularized PQL is designed as a fast approach to joint selection to GLMMs (Hui et al., 2016). It works by taking the penalized quasi-likelihood (PQL, Breslow and Clayton, 1993) and adding on penalties to perform selection of the fixed and random effects. That is, maximize the regularized PQL function

$$\ell = \sum_{i=1}^{n} \log(f(y_i|\boldsymbol{\beta}, \boldsymbol{b}_{i1}, \boldsymbol{b}_{i2}, \ldots)) - \frac{1}{2}\sum_{i=1}^{n} \boldsymbol{b}_{i1}^T \boldsymbol{D}_1^{-1} \boldsymbol{b}_{i1} - \frac{1}{2}\sum_{i=1}^{n} \boldsymbol{b}_{i2}^T \boldsymbol{D}_2^{-1} \boldsymbol{b}_{i2} - \ldots - P_\lambda$$

where P_λ denotes penalties to shrink the fixed effect $\boldsymbol{\beta}$ and random effect $\boldsymbol{b}_{i1}, \boldsymbol{b}_{i2}, \ldots$ coefficients, which depend on one or more tuning parameters λ. Like the PQL itself, regularized PQL is a fast approach for estimating GLMMs because it treats the random effects as "fixed" coefficients, and therefore no integration is required. Penalties are then used to shrunk one or more $\boldsymbol{\beta}$'s and $\boldsymbol{b}$'s to zero, the latter done so in a group-based manner, in order to perform joint selection (see Hui et al., 2016, for details). In short, regularized PQL is able to fit many GLMMs in a relatively short period of time, which in turn facilitates the construction of a solution or regularization path ranging from the null (intercept-only) to the full (saturated) model. A tuning parameter selection method such as information criterion can then be used to pick the select the final subset of fixed and random effects. A few penalty types are available in the package, from which we prefer to use the adaptive LASSO (with weights based on the full model, Zou, 2006) mainly because by having weights, we can avoids have to search through a two-dimensional grid of tuning parameter values.

Note that if one only wanted to penalize the fixed effects and leave the random effects unpenalized, this can be achieved by setting the second element/s of lambda equal to to e.g., `lambda = c(1,0)`. Note though that in longitudinal studies, for covariates included as both fixed and random effects, if the random effects is not penalized then neither should the fixed effect. This ensures that no covariates end up being selected in the model as a purely random effects (non-hierarchical shrinkage, Hui et al., 2016). This can be accounted for also setting the corresponding elements of `pen.weights$fixed` to zero.

AN IMPORTANT NOTE

While regularized PQL is relatively fast, it will produce biased estimates of the fixed and random effects parameters for non-normal responses, especially if the amount of data to estimate each random effect is not large e.g., if the number of time points or cluster size is not large. We envision regularized PQL as a method of joint variable selection ONLY, and strongly encourage the user to adopt a hybrid estimation approach (using `hybrid.est = TRUE`, for instance). That is, once model selection is performed using regularized PQL, the final submodel should be re-estimated using more exact methods like quadrature or MCMC.

Because regularized PQL treats the random effects as “fixed" coefficients and therefore penalizes these, then the random effects covariance matrices $\boldsymbol{D}_1, \boldsymbol{D}_2, \ldots$ are regarded more as nuisance parameters. This is in contrast to traditional maximum likelihood estimation where the random effect coefficients $\boldsymbol{b}_{i1}, \boldsymbol{b}_{i2}, \ldots$ are integrated over. As nuisance parameters, regularized PQL employs an

iterative estimator based on maximizing the Laplace-approximated marginal log-likelihood, assuming all other parameters are fixed, for estimating the covariance matrix $\boldsymbol{D}_1, \boldsymbol{D}_2, \ldots$. This iterative estimator was used in Hui et al., (2016) for independent clustered data specifically. When they are multiple sets of random effects, each covariance matrix is estimated conditionally on all others i.e., the random effect coefficients corresponding to all other random effects are held constant. This can be thought of as employing a series of conditional Laplace approximations to obtain updates for $\boldsymbol{D}_1, \boldsymbol{D}_2, \ldots$.

A not so short discussion about information criterion

How to choose the tuning parameters for penalized regression is an active area of area of research in statistics (see for instance Zhang et al., 2010, Hui et al., 2014), with the most popular solutions being cross validation and information criteria. That is, a solution path is constructed and the best submodel is then chosen by minimizing the value of the information criterion. Anyway, `rpql` offers the following information criteria for tuning parameter selection, as available in `ics` in the output. Please note all of the criteria below use only the first part of the PQL function as the loss function i.e., $IC = -2\sum_{i=1}^{n} \log(f(y_i|\boldsymbol{\beta}, \boldsymbol{b}_{i1}, \boldsymbol{b}_{i2}, \ldots))+$ model complexity terms.

1. A AIC-type criterion that penalizes a values of 2 for every non-zero fixed effect coefficient, and, for each set of random effects, penalizes a value of 2 for every non-zero random effect coefficient in that set.
2. A BIC-type criterion that penalizes a value of $\log(n)$ for every non-zero fixed effect coefficient, and, for each set of random effects, penalizes a value of $\log(n_c)$ for every non-zero, unique element in covariance matrix for that set, where `n_c` denotes the number of clusters corresponding to that random effect.
3. A BIC-type criterion that penalizes a value of $\log(n)$ for every non-zero fixed effect coefficient, and, for each set of random effects, penalizes a value of $\log(n)$ for every non-zero, unique element in covariance matrix for that set. This combination of penalties is the one used in the package `lme4`.
4. Three hybrid information criteria that penalizes a value $\log(n)$ for every non-zero fixed effect coefficient, and, for each set of random effects, penalizes a value of 2/1/0.5 for every non-zero random effect coefficient in that set.

Selection consistency for all but the first AIC criteria have been established, although empirically performance may differ. We generally prefer the three hybrid criterion, although it is recommended that the user tries several of them and see how results differ!

Value

An object of class "rpql" containing the following elements:

`call`	The matched call.
`fixef`	A vector of estimated fixed effect coefficients, β.
`ranef`	A list with each element being a matrix of estimated (predicted) random effect coefficients, $\boldsymbol{b}_{i1}$, $\boldsymbol{b}_{i2}$, and so on.
`ran.cov`	A list with each element being an estimated random effect covariance matrices, $\boldsymbol{D}_1, \boldsymbol{D}_2, \ldots$.
`logLik`	The (unpenalized) PQL likelihood value at convergence.

phi, shape, zeroprob
: Estimates of nuisance parameters (if appropriate), including the variance and overdispersion parameter for normal and negative binomial families, the shape parameter for the Gamma family, and the probability of a structural zero for zero-inflated Poisson family.

family
: The family fitted.

n
: The length of y.

id
: The id argument.

lambda, pen.type
: The tuning parameters and penalties used.

ics
: A vector containing the number of estimated parameters in the GLMM (note regularized PQL treats the random effects as "fixed"), and some information criteria. Please see details above for more information.

nonzero.fixef
: A vector indexing which of the estimated fixed effect coefficients are non-zero.

nonzero.ranef
: A list with each element being a vector indexing which of the estimated random effects are non-zero, i.e. which of the diagonal elements in the corresponding element of ran.cov are non-zero.

hybrid
: The estimated fit from lme4, if hybrid.est = TRUE.

y, X, Z
: The data the GLMM is fitted to, if save.data = TRUE.

Warnings

- We strongly recommend you scale your responses (if normally distributed) and any continuous covariates, otherwise rpql like all penalized likelihood methods, may not make much sense!
- Like its standard unpenalized counterpart, regularized PQL can produce very bias parameter estimates in finite samples, especially if you do not have a lot of data to estimate each random effect. We therefore envision regularized PQL as a tool for fast model selection in GLMMs, and strongly recommend you re-estimate the final submodel using more accurate estimation methods i.e., use a hybrid estimation approach, in order to obtain better final parameter estimates and predictions of the random effects.
- If save.data = TRUE, the data you fitted the GLMM is also saved as part of the output, and this can potentially take up a lot of memory.
- If you are constantly suffering convergence issues with regularized PQL, even after multiple restarts, consider increasing lambda[2] to penalized the random effects more and stabilize the estimation algorithm. You may also want to consider better starting values, in particular, smaller values of start$ranef. Good luck!

Author(s)

Francis K.C. Hui [aut, cre]

Maintainer: Francis K.C. Hui <fhui28@gmail.com>

References

- Breslow, N. E., and Clayton, D. G. (1993). Approximate inference in generalized linear mixed models. Journal of the American Statistical Association, 88, 9-25.
- Fan, J., and Li, R. (2001). Variable selection via nonconcave penalized likelihood and its oracle properties. Journal of the American statistical Association, 96, 1348-1360.
- Hui, F. K. C., Mueller, S., and Welsh, A.H. (2017). Joint Selection in Mixed Models using Regularized PQL. Journal of the American Statistical Association, 112, 1323-1333.
- Hui, F. K. C., Mueller, S., and Welsh, A.H. (2017). Hierarchical Selection of Fixed and Random Effects in Generalized Linear Mixed Models. Statistica Sinica, 27, 501-518.
- Hui, F. K. C., Warton, D. I., and Foster, S. D. (2014). Tuning parameter selection for the adaptive lasso using ERIC. Journal of the American Statistical Association, 110, 262-269.
- Lin, X., and Breslow, N. E. (1996). Bias correction in generalized linear mixed models with multiple components of dispersion. Journal of the American Statistical Association, 91, 1007-1016.
- Mueller, S., Scealy, J. L., and Welsh, A. H. (2013). Model selection in linear mixed models. Statistical Science, 28, 135-167.
- Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological), 58, 267-288.
- Zhang, Y., Li, R., and Tsai, C. L. (2010). Regularization parameter selections via generalized information criterion. Journal of the American Statistical Association, 105, 312-323.
- Zhang, C. H. (2010). Nearly unbiased variable selection under minimax concave penalty. The Annals of Statistics, 38, 894-942.
- Zou, H. (2006). The adaptive lasso and its oracle properties. Journal of the American statistical association, 101, 1418-1429.

See Also

`rpqlseq` for the wrapper function that runs `rpql` multiple times on a sequence of tuning parameter values, `build.start.fit` for building `start` lists from a GLMM fitted using the `lme4` package, `summary` for a summary of the regularized PQL fit. For alternative methods of fitting GLMMs, you may also want be check out the packages `lme4`, `nlme`, `MCMCglmm` and `glmmADMB`.

Examples

```
## Please note all examples below use the \code{rpqlseq} wrapper function.

library(lme4)

##-------------------------
## Example 1: Poisson GLMM on simulated data
## Indepenent cluster model with 30 clusters and equal cluster sizes of 10
## 9 fixed and random effect covariates including a fixed and random intercept
##-------------------------
library(mvtnorm)
set.seed(1)
n <- 30; m <- 10; p <- 8;
```

```
## Generate rows of a model matrix from a multivariate normal distribution
## with AR1 covariance structure.

H <- abs(outer(1:p, 1:p, "-"))
X <- cbind(1,rmvnorm(n*m,rep(0,p),sigma=0.5^H));
Z <- X
true_betas <- c(0.1,1,-1,-1,1,rep(0,p-4)) ## 5 truly important fixed effects
true_D <- matrix(0,ncol(Z),ncol(Z))
true_D[1:3,1:3] <- matrix(c(1,0.6,0.6,0.6,1,0.4,0.6,0.4,1),3,3,byrow=TRUE)
## 3 important random effects

simy <- gendat.glmm(id = list(cluster=rep(1:n,each=m)), X = X, beta = true_betas,
 Z = list(cluster=Z), D = list(cluster=true_D), family = poisson())


## Not run:
## Construct a solution path using adaptive LASSO for selection
dat <- data.frame(y = simy$y, simy$X, simy$Z$cluster, simy$id)
fit_satlme4 <- glmer(y ~ X - 1 + (Z - 1 | cluster), data = dat,
 family = "poisson")
fit_sat <- build.start.fit(fit_satlme4, gamma = 2)
## Please see example 3 for another way of constructing the adaptive weights

lambda_seq <- lseq(1e-6,1,length=100)
fit <- rpqlseq(y = simy$y, X = simy$X, Z = simy$Z, id = simy$id,
 family = poisson(), lambda = lambda_seq, pen.type = "adl",
 pen.weights = fit_sat$pen.weights, start = fit_sat)

summary(fit$best.fit[[5]]) ## Second of the hybrid ICs
# apply(fit$collect.ics, 2, which.min) ## Look at best fit chosen by different ICs

## Note, if you wanted to penalized the fixed effects only, this can achieved
## by setting fit_sat$pen.weights$random$cluster <- rep(0,ncol(simy$Z$cluster))


## An alternative way to construct the X and Z matrices for input into rpqlseq is as follows:
## Big thanks for Andrew Olney for this suggestion!
XMM <- unname(model.matrix(fit_satlme4))
ZMM <- getME(fit_satlme4,"mmList"); names(ZMM) <- "cluster"
lambda_seq <- lseq(1e-6,1,length=100)
fit <- rpqlseq(y = simy$y, X = XMM, Z = ZMM, id = simy$id,
  family = poisson(), lambda = lambda_seq, pen.type = "adl",
 pen.weights = fit_sat$pen.weights, start = fit_sat)
summary(fit$best.fit[[5]]) ## Second of the hybrid ICs

## End(Not run)


##-------------------------
## Example 2: Similar to example 1 but with Bernoulli GLMMs
## 30 clusters, cluster size of 20
##-------------------------
library(mvtnorm)
```

```
set.seed(1)
n <- 30; m <- 20; p <- 8;
## Generate rows of a model matrix from a multivariate normal distribution
## with AR1 covariance structure.

H <- abs(outer(1:p, 1:p, "-"))
X <- cbind(1,rmvnorm(n*m,rep(0,p),sigma=0.5^H));
Z <- X
true_betas <- c(-0.1,1,-1,1,-1,rep(0,p-4)) ## 5 truly important fixed effects
true_D <- matrix(0,ncol(Z),ncol(Z))
true_D[1:3,1:3] <- diag(c(3,2,1), nrow = 3)
## 3 important random effects

simy <- gendat.glmm(id = list(cluster=rep(1:n,each=m)), X = X,
  beta = true_betas, Z = list(cluster=Z), D = list(cluster=true_D), family = binomial())


## Not run:
## Construct a solution path using adaptive LASSO for selection
dat <- data.frame(y = simy$y, simy$X, simy$Z$cluster, simy$id)
fit_satlme4 <- glmer(y ~ X - 1 + (Z - 1 | cluster), data = dat,
 family = "binomial")
fit_sat <- build.start.fit(fit_satlme4, gamma = 2)

lambda_seq <- lseq(1e-6,1,length=100)
best.fit <- list(ics = rep(Inf,6))
fit <- rpqlseq(y = simy$y, X = simy$X, Z = simy$Z, id = simy$id,
 family = binomial(), lambda = lambda_seq, pen.type = "adl",
 pen.weights = fit_sat$pen.weights, start = fit_sat)

summary(fit$best.fit[[5]]) ## Second of the hybrid ICs
# apply(fit$collect.ics, 2, which.min) ## Look at best fit chosen by different ICs

## An alternative way to construct the X and Z matrices for input into rpqlseq is as follows:
XMM <- unname(model.matrix(fit_satlme4))
ZMM <- getME(fit_satlme4,"mmList"); names(ZMM) <- "cluster"
lambda_seq <- lseq(1e-6,1,length=100)
fit <- rpqlseq(y = simy$y, X = XMM, Z = ZMM, id = simy$id,
  family = binomial(), lambda = lambda_seq, pen.type = "adl",
 pen.weights = fit_sat$pen.weights, start = fit_sat)
summary(fit$best.fit[[5]]) ## Second of the hybrid ICs

## End(Not run)


##-------------------------
## Example 3: Bernoulli GLMMs on simulated data
## Nested data with 200 observations in total: split into 10 creeks,
## 5 samples nested within each creek
##-------------------------
mn <- 100;
X <- matrix(1,mn,1);
ids <- list(samples = rep(1:50,each=2), creek = rep(1:10,each=10))
```

```
## We have two sets of random intercepts only, one for creek and one for
## samples nested within creek.
Zs <- list(samples = X, creek = X)

true_betas <- 0.25
true_D <- list(samples = as.matrix(1e-5), creek = as.matrix(0.5))
## Please ensure each element of true_D is a matrix

simy <- gendat.glmm(id = ids, X = X, beta = true_betas, Z = Zs,
 D = true_D, trial.size = 1, family = binomial())

## Not run:
fit <- rpqlseq(y = simy$y, X = simy$X, Z = simy$Z, id = simy$id,
 family = binomial(), lambda = lambda_seq, pen.type = "scad")

summary(fit$best.fit[[5]]) ## Second of the hybrid ICs
# apply(fit$collect.ics, 2, which.min) ## Look at best fit chosen by different ICs

## End(Not run)


##-------------------------
## Example 4: Linear mixed models on Alfalfa split-plot data
##-------------------------
## Not run:

library(nlme)
data(Alfalfa)
Alfalfa$Yield <- scale(Alfalfa$Yield)
X <- as.matrix(model.matrix(~ Date, data = Alfalfa))
## Note Date is categorical variable!
colnames(X)[1] <- "x1"
Z <- list(BlockVariety = matrix(1,nrow(X),1), Block = matrix(1,nrow(X),1))
## Four samples of each Block*Variety
ids <- list(BlockVariety = rep(1:(nrow(X)/4),each=4),
 Block = as.numeric(Alfalfa$Block))

## How you would fit it in lme4
fit_satlme4 <- lmer(Yield ~ X - 1 + (1|Block/Variety), data = Alfalfa)
fit_sat <- build.start.fit(fit_satlme4, cov.groups = c(1,2,2,2), gamma = 2)

## Construct a solution path using adaptive LASSO for selection
lambda_seq <- lseq(1e-8,2,length=100)
fit <- rpqlseq(y = Alfalfa$Yield, X = X, Z = Z, id = ids,
 lambda = lambda_seq, cov.groups = c(1,2,2,2), pen.type = "adl",
 pen.weights = fit_sat$pen.weights, start = fit_sat)

summary(fit$best.fit[[5]]) ## Second of the hybrid ICs
# apply(fit$collect.ics, 2, which.min) ## Look at best fit chosen by different ICs

## End(Not run)
```

```
##-------------------------
## Example 5: Linear mixed models on sleep study dataset
##-------------------------

## Not run:

data(sleepstudy)

## How you fit it in lme4
## Response is scaled so as to avoid large variances and easier intepretation
sleepstudy$Reaction <- scale(sleepstudy$Reaction)
sleepstudy$Days <- scale(sleepstudy$Days)
fm1 <- lmer(Reaction ~ Days + (Days|Subject), sleepstudy)

## How you fit it using rpql
## Construct a solution path using adaptive LASSO for selection
X <- cbind(1, sleepstudy$Days)
Z <- list(subject = X)
ids <- list(subject = as.numeric(sleepstudy$Subject))
fit_sat <- build.start.fit(fm1, gamma = 2)

lambda_seq <- lseq(1e-8,1e-1,length=100)
fit <- rpqlseq(y = sleepstudy$Reaction, X = X, Z = Z, id = ids,
 lambda = lambda_seq, pen.type = "adl",
 pen.weights = fit_sat$pen.weights, start = fit_sat)

summary(fit$best.fit[[5]]) ## Second of the hybrid ICs
# apply(fit$collect.ics, 2, which.min) ## Look at best fit chosen by different ICs
## Best fit might well be the saturated fit!
## This is at least consistent with confint(fm1)

## End(Not run)
```

`rpqlseq` *Wrapper function for joint effects selection in GLMMs using regularized PQL.*

Description

`rpql` offers fast joint selection of fixed and random effects in Generalized Linear Mixed Model (GLMMs) via regularization. The penalized quasi-likelihood (PQL) is used as a loss function, and penalties are added on to perform fixed and random effects selection. This method of joint selection in GLMMs, referred to regularized PQL, is fast compared to information criterion and hypothesis testing (Hui et al., 2016).

`rpqlseq` is a wrapper function to permit a sequence of tuning parameter values, which wraps around the code workhorse function `rpql`.

Usage

```
rpqlseq(y, X, Z, id, family = gaussian(), trial.size = 1, lambda,
  pen.type = "lasso", start = NULL, cov.groups = NULL, pen.weights = NULL,
  offset = NULL, intercept = TRUE, save.data = FALSE,
  control = list(tol = 1e-4, maxit = 100, trace = FALSE, restarts = 5,
  scad.a = 3.7, mcp.gamma = 2, lasso.lambda.scale = TRUE, seed = NULL), ...)
```

Arguments

`y`, `X`, `Z`, `id`, `family`, `trial.size`
: As per the `rpql` function. Please see the help file for `rpql` for details on the arguments.

`lambda`
: Either a vector containing **sequence** of tuning parameter values, which is applied to both penalties, or two-column matrix containing a **sequence** of tuning parameter values for the fixed and random effects penalty respectively.

`pen.type`, `start`, `cov.groups`, `pen.weights`, `offset`, `intercept`, `save.data`, `control`
: As per the `rpql` function. Please see the help file for `rpql` for details on the arguments.

`...`
: Not used.

Details

Please see the help file for `rpql` for details on how regularized PQL works. `rpqlseq` is simply a wrapper function to run the core `rpql` function multiple times, on a sequence of tuning parameter values, in order to construct a regularization path. The best models, based on different information criteria for selecting the best tuning parameter (degree of sparsity) are then returned.

Value

An object of class "rpql" containing the following elements:

`best.fits`
: A list containing the best fitted models as based on different information criteria used to select the tuning parameter. Each element in this list has the same structure as the output from the `rpql` function. Please see the `rpql` function for details on the information criteria available as well as the nature of the output.

`collect.ics`
: A matrix containing the values of various information criteria calculated for the sequence of `lambda` values supplied. The best fitted models found in `best.fits` is based off this matrix i.e., each element in `best.fits` corresponds to a model that was chosen based on minimizing the corresponding information criterion in `collect.ics`. Please see the `rpql` function for details on the information criteria available.

`lambda`
: The sequence of tuning parameters considered.

Author(s)

Francis K.C. Hui [aut, cre]

Maintainer: Francis K.C. Hui <fhui28@gmail.com>

See Also

rpql, which is the core workhorse function that performed regularized PQL for a single set of tuning parameter values.

Examples

```
## Please see examples in help file for the \code{rpql} function for usage.
```

---

summary.rpql *Summary of GLMM fitted using regularized PQL.*

Description

A summary of the results from applying rpql.

Usage

```
## S3 method for class 'rpql'
summary(object, ...)

## S3 method for class 'summary.rpql'
print(x,...)
```

Arguments

object	An object of class "rpql".
x	An object of class "rpql".
...	Not used.

Value

A list (some of which is printed) containing the following elements:

Call	The matched call.
fixed	Estimated fixed effects coefficients.
ranef	A list with each element being a matrix of estimated random effects coefficients.
ran.cov	A list with each element being a estimated random effects covariance matrix.
logLik	PQL log-likelihood value at convergence.
family	The family argument, i.e. response type.

`pen.type`, `lambda`
: Penalties used for selection and the corresponding tuning parameter values.

`ics`
: A vector containing the number of estimated, non-zero parameters, and three information criterion. Please see the help file for `rpql` for details on these criteria.

`id`
: The `id` argument, i.e. list of IDs.

`nonzero.fixef`
: A vector indexing which of the estimated fixed effect coefficients are non-zero.

`nonzero.ranef`
: A list with each element being a vector indexing which of the estimated random effects are non-zero, i.e. which of the diagonal elements in the corresponding element of `ran.cov` are non-zero.

Author(s)

Francis K.C. Hui [aut, cre]

Maintainer: Francis K.C. Hui <fhui28@gmail.com>

See Also

`rpql` for fitting and performing model selection in GLMMs using regularized PQL.

Examples

```
## Please see other examples in help file for the \code{rpql} function.
```

ZIP *A zero-inflated Poisson family*

Description

This is basically a carbon-copy of `gamlss.dist::ZIP()`, which establishes the zero-inflated Poisson distribution as a family for use in the main `rpql` function. The distribution is parametrized in terms of the mean of the Poisson distribution, and the probability of zero inflation.

Full credit goes to the GAMLSS team for the original`gamlss.dist::ZIP()` family class, which was copied into this package due to issues with the `gamlss.dist` package being archived at some point (but might have since been resolved?!)

Usage

```
ZIP()
```

Details

Used in the form `rpql(y, ..., family = ZIP(), ...)`.

Value

An object of class "family"

Author(s)

Francis K.C. Hui [aut, cre]
Maintainer: Francis K.C. Hui <fhui28@gmail.com>

Index

build.start.fit, 3, *20*

calc.marglogL, 6

gendat.glmm, 8

lseq, 11

nb2, 13

print.rpql *(*rpql*)*, 14
print.summary.rpql *(*summary.rpql*)*, 26

rpql, *5*, *7*, *10*, *12*, 14, *26*, *27*
rpql-package, 2
rpqlseq, *20*, 24

summary, *20*
summary.rpql, 26

ZIP, 27