Package ‘regress’

July 23, 2025

Version 1.3-22

Date 2025-05-09

Title Gaussian Linear Models with Linear Covariance Structure

Author David Clifford [aut],
Peter McCullagh [aut],
HJ Auinger [ctb],
Karl W Broman [ctb, cre] (ORCID:
<https://orcid.org/0000-0002-4914-6671>)

Maintainer Karl W Broman <`broman@wisc.edu`>

Description Functions to fit Gaussian linear model by maximising the residual log likelihood where the covariance structure can be written as a linear combination of known matrices. Can be used for multivariate models and random effects models. Easy straight forward manner to specify random effects models, including random interactions. Code now optimised to use Sherman Morrison Woodbury identities for matrix inversion in random effects models. We've added the ability to fit models using any kernel as well as a function to return the mean and covariance of random effects conditional on the data (best linear unbiased predictors, BLUPs).
Clifford and McCullagh (2006)
<https://www.r-project.org/doc/Rnews/Rnews_2006-2.pdf>.

License GPL-2

URL https://github.com/kbroman/regress

BugReports https://github.com/kbroman/regress/issues

Suggests nlme, MASS

Encoding UTF-8

ByteCompile true

NeedsCompilation no

Repository CRAN

Date/Publication 2025-05-09 15:50:02 UTC

Contents

`regress` *Fit a Gaussian Linear Model with Linear Covariance Structure*

Description

Fits Gaussian linear models in which the covariance structure can be expressed as a linear combination of known matrices. For example, random effects, block effects models and spatial models that include a nugget effect. Fits model by maximising the log-likelihood of the model. The choice of kernel affects which likelihood and by default it is the REML log likelihood or restricted log likelihood but the ordinary log-likelihood is also possible. The regress algorithm uses a Newton-Raphson algorithm to locate the maximum of the log-likelihood surface. Some computational efficiencies are achieved when all variance components are associated with factors. In such a random effects model the matrix inversion is computed using the Sherman-Morrison-Woodbury identities.

Usage

```
regress(formula, Vformula, identity=TRUE, kernel=NULL,
                    start=NULL, taper=NULL, pos, verbose=0, gamVals=NULL,
                    maxcyc=50, tol=1e-4, data)
```

Arguments

`formula`	a symbolic description of the model to be fitted. The details of model specification are the same as for `lm`
`Vformula`	Specifies the matrices to include in the covariance structure. Each term is either a symmetric matrix, or a factor. Independent Gaussian random effects are included by passing the corresponding block factor.
`identity`	Logical variable, includes the identity as the final matrix of the covariance structure. Default is TRUE
`kernel`	Compute the log likelihood based on a reduced observation TY where T has this kernel. Default value of NULL assumes that the kernal matches the fixed effects model matrix X corresponding to REML. Setting kernel=0 gives the ordinary likelihood and kernel=1 gives the one dimensional subspace of constant vectors. See examples for more details.
`start`	Specify the variance components at which the Newton-Raphson algorithm starts. Default value is `rep(var(y),k)`.
`taper`	The proportion of each step to take. A vector of values from 0 to 1 of length maxcyc. Default value takes smaller steps initially.
`pos`	logical vector of length k, where k is the number of matrices in the covariance structure. Indicates which variance components are positive (TRUE) and which are real (FALSE). Important for multivariate problems.

`verbose`	Controls level of time output, takes values 0, 1 or 2, Default is 0, level 1 gives parameter estimates and value of log likelihood at each stage.
`gamVals`	When `k=2`, the marginal log likelihood based on the residual configuration statistic (see Tunnicliffe Wilson(1989)), is evaluated first at `(1-gam) V1 + gam V2` for each value of `gam` in `gamVals`, a set of values from the unit interval. Subsequently the Newton-Raphson algorithm is started at variance components corresponding the the value of `gam` that has the highest marginal log likelihood. This is overridden if `start` is specified.
`maxcyc`	Maximum number of cycles allowed. Default value is 50. A warning is output to the screen if this is reached before convergence.
`tol`	Convergence criteria. If the change in residual log likelihood for one cycle is less than `10 x tol` the algorithm finishes. If each component of the change proposed by the Newton-Raphson is lower in magnitude than `tol` the algorithm finishes. Default value is `1e-4`.
`data`	an optional data frame containing the variables in the model. By default the variables are taken from 'environment(formula)', typically the environment from which 'regress' is called.

Details

As the code is running it can output the variance components, and the residual log likelihood at each iteration when `verbose` is non-zero.

To avoid confusion over terminology. I define variance components to be the multipliers of the matrices and variance parameters to the parameter space over which the Newton-Raphson algorithm is run. I can force a component to be positive be defining the corresponding variance parameter on the log scale.

All output to the screen is for variance components (i.e. the multiples of the matrices). Values for `start` are on the variance component scale. Use `pos` to force certain variance components to be positive.

NOTE: The final stage of the algorithm converts the estimates of the variance components and the Fisher Information to the usual linear scale, i.e. as if `pos` were a vector of zeroes.

NOTE: No `predict` functionality is provided with regress due to some ambiguity. Are we predicting conditional on the observed data. Are we predicting observations from the fitted model itself? It is all normal anyway so it is straightforward, see our paper on regress.

When you fit a Gaussian regression model using fit <- regress(y~X, ~V, kernel=K) the function computes the log likelihood based on the reduced observation $TY ~ N(TX, T V T')$, where T is a linear transformation with kernel K. Only $n-k$ degrees of freedom are available. Ordinary likelihood corresponds to $K=0$, and REML to $K=X$, but these are not the only options.

When you fit two nested Gaussian models ($X0 subset of X1$ and $V0 subset of V1$) using the commands:

fit0 <- regress(y~X0, ~V0, kernel=K)

fit1 <- regress(y~X1, ~V1, kernel=K)

then the likelihood ratio statistic fit1$llik - fit0$llik is the ordinary likelihood ratio based on the Gaussian observation TY where the kernel of T is K. So if you set kernel=0, you get the ordinary likelihood ratio based on the complete observation y; And if you set kernel=1, you get the likelihood ratio based on simple contrasts $y_i - y_j$ only. In the latter case, you have only $n-1$ degrees of freedom to work with. And if you set kernel=X0, you get the likelihood ratio based on contrasts Ty with kernel $X0$, which for fit0 is the REML likelihood.

We recommend fitting the models with the "largest" kernel possible. For example, with models fit0 and fit1 above, we could choose K=0, or K=X0 to get the desired result. Our experience though is that the model with K=X0 may be easier to fit with regress compared with a model where K=0.

Value

`trace`	Matrix with one row for each iteration of algorithm. Each row contains the residual log likelihood, marginal log likelihood, variance parameters and increments.
`llik`	Value of the marginal log likelihood at the point of convergence.
`cycle`	Number of cycles to convergence.
`rdf`	Residual degrees of freedom.
`beta`	Estimate of the linear effects.
`beta.cov`	Estimate of the covariance structure for terms in beta.
`beta.se`	Standard errors for terms in beta.
`sigma`	Variance component estimates, interpretation does not depend on value of `pos`
`sigma.cov`	Covariance matrix for the variance component estimates based on the Fisher Information at the point of convergence.
`W`	Inverse of covariance matrix at point of convergence.
`Q`	$I - X^T (X^T W X)^-1 X^T W$ at point of convergence.
`fitted`	$X beta$, the fitted values.
`predicted`	If `identity=TRUE`, decompose y into the part associated with the identity and that assosicated with the rest of the variance structure, this second part is the predicted values. If $Sigma = V1 + V2$ at point of convergence then y = V1 W y + V2 W y is the decomposition. This is the conditional expectation for new observations conditional on the observed data.
`predictedVariance`	Variance of new observations conditional on the observed data
`predictedVariance2`	Additional variance associated with the uncertainty of beta. Can be be added to predictedVariance
`pos`	Indicator for the scale for each variance parameter.
`Vnames`	Names associated with each variance component, used in `print.regress`.
`formula`	Copy of formula
`Vformula`	Updated version of Vformula to include identity if necessary
`Kcolnames`	Names associated with the kernel
`model`	Response, covariates and matrices/factors to be used for model fitting
`Z`	Design matrices associated with the random effects, used for computation of BLUPs

Author(s)

David Clifford, Peter McCullagh

References

G. Tunnicliffe Wilson (1989), "On the use of marginal likelihood in time series model estimation." *JRSS B*, Vol 51, No 1, 15-27.

D. Clifford and P. McCullagh (2006), "The regress function" *R News* 6(2):6-10

Weisstein, Eric W. "Woodbury Formula." From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/Wood

Weisstein, Eric W. "Sherman-Morrison Formula." From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/Sherman-MorrisonFormula.html

Examples

```
######################
## Comparison with lme
######################

## Example of Random Effects model from Venables and Ripley, page 205
library(nlme)
library(regress)

citation("regress")

names(Oats) <- c("B","V","N","Y")
Oats$N <- as.factor(Oats$N)

## Using regress
oats.reg <- regress(Y~N+V,~B+I(B:V),identity=TRUE,verbose=1,data=Oats)
summary(oats.reg)

## Using lme
oats.lme <- lme(Y~N+V,random=~1|B/V,data=Oats,method="REML")
summary(oats.lme)

## print and summary
oats.reg
print(oats.reg)
summary(oats.reg)

ranef(oats.lme)
BLUP(oats.reg)

rm(oats.reg, oats.lme, Oats)

#######################
## Computation of BLUPs
#######################

ex2 <- list()
```

```
ex2 <- within(ex2,{

  ## Set up example
  set.seed(1001)
  n <- 101
  x1 <- runif(n)
  x2 <- seq(0,1,l=n)
  z1 <- gl(4,10,n)
  z2 <- gl(6,1,n)

  X <- model.matrix(~1 + x1 + x2)
  Z1 <- model.matrix(~z1-1)
  Z2 <- model.matrix(~z2-1)

  ## Create the individual random and fixed effects
  beta <- c(1,2,3)
  eta1 <- rnorm(ncol(Z1),0,10)
  eta2 <- rnorm(ncol(Z2),0,10)
  eps <- rnorm(n,0,3)

  ## Combine them into a response
  y <- X %*% beta + Z1 %*% eta1 + Z2 %*% eta2 + eps
})

## Data frame containing all we need for model fitting
regressDF <- with(ex2,data.frame(y,x1,x2,z1,z2))
rm(ex2)

## Fit the model using regress
regress.output <- regress(y~1 + x1 + x2,~z1 + z2,data=regressDF)

summary(regress.output)

blup1 <- BLUP(regress.output,RE="z1")
blup1$Mean
blup1$Variance
blup1$Covariance
cov2cor(blup1$Covariance) ## Large correlation terms

blup2 <- BLUP(regress.output) ## Joint BLUP of z1 and z2 by default
blup2$Mean
blup2$Variance
cov2cor(blup2$Covariance)  ## Strong negative correlation between BLUPs
                           ## for z1 and z2

rm(blup1,blup2)

############################
## Examples of use of kernel
############################

## REML LRT for x2 which will be 0 as x2 lies in the kernel
with(regressDF,{
```

```
     K <- model.matrix(~1+x1+x2)
     model1 <- regress(y~1+x1,~z1,kernel=K)
     model2 <- regress(y~1+x1+x2,~z1,kernel=K)
     2*(model2$llik - model1$llik)
})

## LRT for x2 using ordinary likelihood
with(regressDF,{
     K <- 0
     model1 <- regress(y~1+x1,~z1,kernel=K)
     model2 <- regress(y~1+x1+x2,~z1,kernel=K)
     2*(model2$llik - model1$llik)
})

## LRT for x2 based on a reduced observation TY with kernel K. This
## LRT is approximately equal to the last one, and numerically this
## turns out to be the case also.
with(regressDF,{
     K <- model.matrix(~1+x1)
     model1 <- regress(y~1+x1,~z1,kernel=K)
     model2 <- regress(y~1+x1+x2,~z1,kernel=K)
     2*(model2$llik - model1$llik)
})

## Two ways to drop out the 17th and 19th observations.
with(regressDF,{
     n <- length(y)
     K <- matrix(0,n,2)
     K[17,1] <- K[19,2] <- 1
     model1 <- regress(y~1+x1,~z1,kernel=K,tol=1e-8)
     drop <- c(17,19)
     model2 <- regress(y[-drop]~1+x1[-drop],~z1[-drop],kernel=0,tol=1e-8)
     print(model1)
     print(model2)
})

rm(regressDF, regress.output)
```

Index