

# Package ‘parglm’

May 22, 2026

**Title** Parallel GLM

**Version** 0.1.10

**Description** Provides a parallel estimation method for generalized linear models without compiling with a multithreaded LAPACK or BLAS.

**License** GPL-2

**URL** <https://github.com/remlapmot/parglm>,  
<https://remlapmot.github.io/parglm/>

**BugReports** <https://github.com/remlapmot/parglm/issues>

**Imports** Matrix, parallelly, Rcpp

**Suggests** biglm, broom, broom.helpers, fastglm, glm2, gtsummary, knitr, lmtree, microbenchmark, rmarkdown, sandwich, speedglm, SuppDists, testthat

**LinkingTo** Rcpp, RcppArmadillo

**VignetteBuilder** knitr

**Config/roxygen2/version** 8.0.0

**Encoding** UTF-8

**NeedsCompilation** yes

**Author** Benjamin Christoffersen [aut] (ORCID:  
<<https://orcid.org/0000-0002-7182-1346>>),  
Anthony Williams [cph],  
Boost developers [cph],  
Tom Palmer [aut, cre] (ORCID: <<https://orcid.org/0000-0003-4655-4511>>)

**Maintainer** Tom Palmer <remlapmot@hotmail.com>

**Repository** CRAN

**Date/Publication** 2026-05-22 09:40:02 UTC

## Contents

parglm . . . . .	2
parglm.control . . . . .	4
tidy_parglm_robust . . . . .	5

---

parglm

*Fitting Generalized Linear Models in Parallel*

---

### Description

A function like `glm` which can make the computation in parallel. The function supports most families listed in `family`. See `"vignette("parglm", "parglm")"` for run time examples.

### Usage

```
parglm(  
  formula,  
  family = gaussian,  
  data,  
  weights,  
  subset,  
  na.action,  
  start = NULL,  
  offset,  
  control = list(...),  
  contrasts = NULL,  
  model = TRUE,  
  x = FALSE,  
  y = TRUE,  
  ...  
)  
  
parglm.fit(  
  x,  
  y,  
  weights = rep(1, NROW(x)),  
  start = NULL,  
  etastart = NULL,  
  mustart = NULL,  
  offset = rep(0, NROW(x)),  
  family = gaussian(),  
  control = list(),  
  intercept = TRUE,  
  ...  
)
```

### Arguments

`formula` an object of class `formula`.  
`family` a `family` object.

data	an optional data frame, list or environment containing the variables in the model.
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 NAs.
start	starting values for the parameters in the linear predictor.
offset	this can be used to specify an a priori known component to be included in the linear predictor during fitting.
control	a list of parameters for controlling the fitting process. For <code>parglm.fit</code> this is passed to <a href="#">parglm.control</a> .
contrasts	an optional list. See the <code>contrasts.arg</code> of <a href="#">model.matrix.default</a> .
model	a logical value indicating whether model frame should be included as a component of the returned value.
x, y	For <code>parglm</code> : logical values indicating whether the response vector and model matrix used in the fitting process should be returned as components of the returned value. For <code>parglm.fit</code> : <code>x</code> is a design matrix of dimension $n * p$ , and <code>y</code> is a vector of observations of length <code>n</code> .
...	For <code>parglm</code> : arguments to be used to form the default <code>control</code> argument if it is not supplied directly. For <code>parglm.fit</code> : unused.
etastart	starting values for the linear predictor. Not supported.
mustart	starting values for the vector of means. Not supported.
intercept	logical. Should an intercept be included in the null model?

### Details

The current implementation uses `min(as.integer(n / p), nthreads)` threads where `n` is the number of observations, `p` is the number of covariates, and `nthreads` is the `nthreads` element of the list returned by [parglm.control](#). Thus, there is likely little (if any) reduction in computation time if `p` is almost equal to `n`. The current implementation cannot handle `p > n`.

Since `parglm` returns a standard `glm` object, it is compatible with the **sandwich** package for heteroskedasticity-consistent (HC) and cluster-robust standard errors via [vcovHC](#) and [vcovCL](#). This requires `model = TRUE` (the default). See `vignette("sandwich", "parglm")` for examples.

### Value

`glm` object as returned by [glm](#) but differs mainly by the `qr` element. The `qr` element in the object returned by `parglm(.fit)` only has the `R` matrix from the QR decomposition.

**Examples**

```
# mtcars has 32 rows, sufficient for 2 threads (>= 16 rows per thread)
f1 <- glm (mpg ~ wt + hp, data = mtcars, family = Gamma(link = "log"))
f2 <- parglm(mpg ~ wt + hp, data = mtcars, family = Gamma(link = "log"),
             control = parglm.control(nthreads = 2L))
all.equal(coef(f1), coef(f2))
```

---

parglm.control

*Auxiliary for Controlling GLM Fitting in Parallel*

---

**Description**

Auxiliary function for `parglm` fitting.

**Usage**

```
parglm.control(
  epsilon = 1e-08,
  maxit = 25,
  trace = FALSE,
  nthreads = parallelly::availableCores(omit = 1L),
  block_size = NULL,
  method = "LINPACK",
  nthreads_auto = missing(nthreads)
)
```

**Arguments**

<code>epsilon</code>	positive convergence tolerance.
<code>maxit</code>	integer giving the maximal number of IWLS iterations.
<code>trace</code>	logical indicating if output should be produced during estimation.
<code>nthreads</code>	number of cores to use. Defaults to <code>parallelly::availableCores(omit = 1L)</code> , which leaves one core free. You may get the best performance by using all available physical cores if your data set is sufficiently large.
<code>block_size</code>	number of observations to include in each parallel block.
<code>method</code>	string specifying which method to use. Either "LINPACK", "LAPACK", or "FAST".
<code>nthreads_auto</code>	logical; for internal use only. Records whether <code>nthreads</code> was auto-detected (suppresses the thread-reduction warning when the dataset is small). Do not set this argument directly.

**Details**

The LINPACK method uses the same QR method as `glm.fit` for the final QR decomposition. This is the `dqrdc2` method described in [qr](#). All other QR decompositions except the last are made with DGEQP3 from LAPACK. See Wood, Goude, and Shaw (2015) for details on the QR method.

The FAST method computes the Fisher information and then solves the normal equation. This is faster but less numerically stable.

**Value**

A list with components named as the arguments.

**References**

Wood, S.N., Goude, Y. & Shaw, S. (2015) Generalized additive models for large datasets. *Journal of the Royal Statistical Society, Series C* 64(1): 139-155.

**Examples**

```
# use one core
f1 <- parglm(mpg ~ wt + hp, data = mtcars, family = Gamma(link = "log"),
            control = parglm.control(nthreads = 1L))

# use two cores (mtcars has 32 rows, sufficient for 2 threads)
f2 <- parglm(mpg ~ wt + hp, data = mtcars, family = Gamma(link = "log"),
            control = parglm.control(nthreads = 2L))
all.equal(coef(f1), coef(f2))
```

---

tidy\_parglm\_robust      *Tidy a parglm model with robust standard errors*

---

**Description**

A drop-in `tidy_fun` for [tbl\\_regression](#) that computes heteroskedasticity-consistent (HC) or cluster-robust confidence intervals via **sandwich** and **lmtest**.

**Usage**

```
tidy_parglm_robust(
  x,
  vcov. = "HC3",
  conf.int = TRUE,
  conf.level = 0.95,
  exponentiate = FALSE,
  ...
)
```

**Arguments**

<code>x</code>	a <code>parglm</code> (or <code>glm</code> ) model object.
<code>vcov.</code>	the robust variance-covariance estimator. A string is passed as the type argument to <code>vcovHC</code> (e.g. "HC3"). A function is called as <code>vcov.(x)</code> and should return a covariance matrix (use this for cluster-robust SEs via <code>vcovCL</code> ). A matrix is used directly. Defaults to "HC3".
<code>conf.int</code>	logical; whether to include confidence intervals.
<code>conf.level</code>	confidence level for the intervals.
<code>exponentiate</code>	logical; whether to exponentiate the estimate and confidence interval limits.
<code>...</code>	unused; present for compatibility with the <code>tidy_fun</code> interface of <code>tbl_regression</code> .

**Details**

Pass this function as `tidy_fun` to `tbl_regression`:

```
# HC3 (default)
tbl_regression(fit, tidy_fun = tidy_parglm_robust)

# HC1
tbl_regression(fit, tidy_fun = \(x, ...) tidy_parglm_robust(x, vcov. = "HC1", ...))

# Cluster-robust
tbl_regression(fit, tidy_fun = \(x, ...) tidy_parglm_robust(
  x, vcov. = \(m) sandwich::vcovCL(m, cluster = ~ cluster_var), ...))
```

**Value**

a data.frame with columns `term`, `estimate`, `std.error`, `statistic`, `p.value`, and (when `conf.int = TRUE`) `conf.low` and `conf.high`.

**Examples**

```
fp <- parglm(mpg ~ wt + hp, data = mtcars,
             control = parglm.control(nthreads = 1L))
if (requireNamespace("sandwich", quietly = TRUE) &&
    requireNamespace("lmtest", quietly = TRUE)) {
  tidy_parglm_robust(fp)
}
```

# Index

family, [2](#)

formula, [2](#)

glm, [2](#), [3](#)

glm.fit, [5](#)

model.matrix.default, [3](#)

parglm, [2](#), [4](#)

parglm.control, [3](#), [4](#)

qr, [5](#)

tbl\_regression, [5](#), [6](#)

tidy\_parglm\_robust, [5](#)

vcovCL, [3](#), [6](#)

vcovHC, [3](#), [6](#)