

# Package ‘parglms’

May 9, 2026

**Title** support for parallelized estimation of GLMs/GEEs

**Version** 1.45.0

**Author** VJ Carey <stvjc@channing.harvard.edu>

**Description** This package provides support for parallelized estimation of GLMs/GEEs, catering for dispersed data.

**Suggests** RUnit, sandwich, MASS, knitr, GenomeInfoDb, GenomicRanges, gwascat, BiocStyle, rmarkdown

**VignetteBuilder** knitr

**Depends** methods

**Imports** BiocGenerics, BatchJobs, foreach, doParallel

**Maintainer** VJ Carey <stvjc@channing.harvard.edu>

**License** Artistic-2.0

**LazyLoad** yes

**BiocViews** statistics, genetics

**ByteCompile** TRUE

**git\_url** <https://git.bioconductor.org/packages/parglms>

**git\_branch** devel

**git\_last\_commit** cb1af4d

**git\_last\_commit\_date** 2026-04-28

**Repository** Bioconductor 3.24

**Date/Publication** 2026-05-08

## Contents

parglms-package	2
parGLM-methods	2
<b>Index</b>	<b>4</b>

---

parGLMs-package

*support for parallelized estimation of GLMs/GEEs*

---

### Description

This package provides support for parallelized estimation of GLMs/GEEs, catering for dispersed data.

### Details

The DESCRIPTION file: This package was not yet installed at build time.

Index: This package was not yet installed at build time.

In version 0.0.0 we established an approach to fitting GLM from data that have been persistently dispersed and managed by a [Registry](#).

### Author(s)

VJ Carey <stvjc@channing.harvard.edu>

Maintainer: VJ Carey <stvjc@channing.harvard.edu>

### References

This package shares an objective with the `bigglm` methods of `biglm`. In `bigglm`, a small-RAM-footprint algorithm is employed, with sequential chunking to update statistics in each iteration. In `parGLM` the footprint is likewise controllable, but statistics in each iteration are evaluated in parallel over chunks.

### Examples

```
showMethods("parGLM")
```

---

parGLM-methods

*fit GLM-like models with parallelized contributions to sufficient statistics*

---

### Description

This package addresses the problem of fitting GLM-like models in a scalable way, recognizing that data may be dispersed, with chunks processed in parallel, to create low-dimensional summaries from which model fits may be constructed.

## Methods

`signature(formula = "formula", store = "Registry")` The model data are assumed to lie in the `file.dir/jobs/*` folders, with `file.dir` defined in the store, which is an instance of [Registry](#).

Additional arguments must be supplied:

**family** a function that serves as a family for `stats::glm`

**binit** a vector of initial values for regression parameter estimation, must conform to expectations of `formula`

**maxit** an integer giving the maximum number of iterations allowed

**tol** a numeric giving the tolerance criterion

Failure to specify these triggers a fatal error.

The Registry instance can be modified to include a list element 'extractor'. This must be a function with arguments `store`, and `codei`. The standard extraction function is

```
function(store, i) loadResult(store, i)
```

It must return a data frame, conformant with the expectations of `formula`. Limited checking is performed.

The `predict` method computes the linear predictor on data identified by `jobid` in a BatchJobs registry. Results are returned as output of `foreach` over the `jobids` specified in the `predict` call.

Note that setting option `parGLM.showiter` to `TRUE` will provide a message tracing progress of the optimization.

## Examples

```
if (require(MASS) & require(BatchJobs)) {
  # here is the 'sharding' of a small dataset
  data(anorexia) # N = 72
  # in .BatchJobs.R:
  # best setting for sharding a small dataset on a small machine:
  # cluster.functions = BatchJobs::makeClusterFunctionsInteractive()
  myr = makeRegistry("abc", file.dir=tempfile())
  chs = chunk(1:nrow(anorexia), n.chunks=18) # 4 recs/chunk
  f = function(x) {library(MASS); data(anorexia); anorexia[x,]}
  batchMap(myr, f, chs)
  submitJobs(myr) # now getResult(myr,1) gives back a data.frame
  waitForJobs(myr) # simple dispersal
  # now myr is populated
  oldopt = options()$parGLM.showiter
  options(parGLM.showiter=TRUE)
  pp = parGLM( Postwt ~ Treat + Prewt, myr,
    family=gaussian, binit = c(0,0,0,0), maxit=10, tol=.001 )
  print(summary(theLM <- lm(Postwt~Treat+Prewt, data=anorexia)))
  print(pp$coefficients - coef(theLM))
  if (require(sandwich)) {
    hc0 <- vcovHC(theLM, type="HC0")
    print(pp$robust.variance - hc0)
  }
}
predict(pp, store=myr, jobids=2:3)
options(parGLM.showiter=oldopt)
```

# Index

- \* **methods**
  - parGLM-methods, [2](#)
- \* **modeling**
  - parGLM-methods, [2](#)
- \* **package**
  - parGLMs-package, [2](#)

parGLM (parGLM-methods), [2](#)  
parGLM, formula, Registry-method  
(parGLM-methods), [2](#)  
parGLM-methods, [2](#)  
parGLMs (parGLMs-package), [2](#)  
parGLMs-package, [2](#)  
predict (parGLM-methods), [2](#)  
print (parGLM-methods), [2](#)

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

summary (parGLM-methods), [2](#)