Package 'ceres'

November 27, 2026

Type Package

Title An Interface to 'Ceres Solver' for Non-Linear Least Squares and Optimization

Version 0.1.0

Date 2026-11-06

Description Provides an interface to the 'Ceres Solver' C++ library <https://github.com/ceres-solver/ceres-solver>. 'Ceres Solver' is an open source library for modeling and solving large, complicated optimization problems. It can solve non-linear least squares problems with bounds constraints and general unconstrained optimization problems. It is a mature, feature rich and high-performance library that has been used in production at Google since 2010. Non-linear least squares problems with residuals written in R can be solved with ceres_nls(), with optional bounds and robust loss functions. The bundled library is built from source and can also be used from C++ through 'Rcpp', either with Rcpp::sourceCpp() or from other packages.

License GPL (>= 3)

Copyright See the file inst/COPYRIGHTS for the copyright holders and licenses of the bundled 'Ceres Solver' sources.

SystemRequirements cmake (>= 3.16), GNU make

Biarch true

Encoding UTF-8

Imports Rcpp, stats

Suggests RcppEigen, testthat (>= 3.0.0)

Config/testthat/edition 3

LinkingTo Rcpp, RcppEigen

RoxygenNote 7.2.0

NeedsCompilation yes

Author Satyaprakash Nayak [aut, cre], Google Inc. [cph] (Ceres Solver, bundled in src/ceres-solver-2.2.0.tar.gz),

Sameer Agarwal [ctb] (Ceres Solver),
Keir Mierle [ctb] (Ceres Solver),
The Ceres Solver authors [ctb] (Ceres Solver; see inst/COPYRIGHTS),
The Abseil Authors [cph] (fixed_array.h and memory.h in Ceres Solver (Apache-2.0)),
Sergiu Deitsch [cph] (FindMETIS.cmake in Ceres Solver (MIT)),
Bogdan Cristea [cph] (iOS.cmake in Ceres Solver (BSD-3-Clause)),
LTE Engineering Software [cph] (iOS.cmake in Ceres Solver (BSD-3-Clause)),
Kitware, Inc. [cph] (iOS.cmake in Ceres Solver (BSD-3-Clause)),
Insight Software Consortium [cph] (iOS.cmake in Ceres Solver (BSD-3-Clause)),
Ken Martin, Will Schroeder and Bill Lorensen [cph] (FindTBB.cmake in Ceres Solver, from VTK (BSD-3-Clause))

Maintainer Satyaprakash Nayak <satyaprakash.nayak@gmail.com>

Repository CRAN

Date/Publication 2026-11-27 18:20:16 UTC

Contents

ceres-package 2
ceres_control 3
ceres_helloworld 4
ceres_nls 5
CxxFlags 8

Index **9**

`ceres-package` *An Interface to 'Ceres Solver'*

Description

Bundles the 'Ceres Solver' C++ library for non-linear least squares and general unconstrained optimization, built from source with Eigen from **RcppEigen**, and makes it available to `C++` code compiled with **Rcpp**.

Details

A `.cpp` file containing `// [[Rcpp::depends(ceres)]]` and `#include <ceres.h>` can be compiled with `sourceCpp`; see the files in `system.file("examples", package = "ceres")`. Other packages can link to Ceres with the flags from `CxxFlags` and `LdFlags`.

Ceres is built with its own minimal logging library ('miniglog') in place of glog and gflags. Log messages are printed with `REprintf()`, and fatal errors (e.g. a failed `CHECK`) throw a `C++` exception that **Rcpp** turns into an R error.

Author(s)

Satyaprakash Nayak.

Maintainer: Satyaprakash Nayak <satyaprakash.nayak@gmail.com>

References

Sameer Agarwal, Keir Mierle and The Ceres Solver Team. Ceres Solver. https://github.com/ceres-solver/ceres-solver

See Also

ceres_nls, ceres_helloworld, CxxFlags

Examples

```
ceres_helloworld(0.5)
```

---

ceres_control *Solver options for ceres_nls*

Description

Options for ceres_nls. The defaults of function_tol and parameter_tol are tighter than those of Ceres Solver (1e-6 and 1e-8), which can stop noticeably short of the solution. The other defaults are those of Ceres Solver.

Usage

```
ceres_control(max_iter = 50, function_tol = 1e-10, gradient_tol = 1e-10,
              parameter_tol = 1e-10, method = c("lm", "dogleg"),
              diff = c("central", "forward"), trace = FALSE)
```

Arguments

max_iter	the maximum number of iterations of each solve. A fit with parameters at a bound can have several, see ceres_nls.
function_tol	stop when the relative change in the cost is below this.
gradient_tol	stop when the largest gradient element is below this.
parameter_tol	stop when the relative step size is below this.
method	the trust region strategy: Levenberg-Marquardt or dogleg.
diff	the finite differences used when ceres_nls() has no jac. "central" is more accurate and "forward" needs fewer calls of fn. Next to a bound, one-sided differences are used in either case.
trace	if TRUE, print the progress of the solver.

Value

A list of class `"ceres_control"`.

See Also

`ceres_nls`

Examples

```
ceres_control(max_iter = 200, method = "dogleg")$max_iter
```

`ceres_helloworld` *Minimal Ceres Solver example*

Description

Solves the one-parameter problem of minimising $\frac{1}{2}(10 - x)^2$ with Ceres Solver, starting from `x`, and prints the solver progress and summary. It is the 'Hello World' example from the Ceres Solver tutorial and checks that the bundled library works.

Usage

```
ceres_helloworld(x)
```

Arguments

`x`	a number, the initial value of the parameter.

Value

The integer `0`; the function is called for its printed output.

References

https://github.com/ceres-solver/ceres-solver/blob/2.2.0/docs/source/nnls_tutorial.rst

Examples

```
ceres_helloworld(0.5)
```

ceres_nls *Non-linear least squares with Ceres Solver*

Description

Minimises $\frac{1}{2}\sum_i \rho(r_i(\theta)^2)$ over the parameters θ, optionally within bounds, where the residuals r_i are computed by an R function and ρ is an optional robust loss function.

Usage

```
ceres_nls(fn, par, ..., jac = NULL, lower = -Inf, upper = Inf,
          loss = c("linear", "huber", "soft_l1", "cauchy", "arctan", "tukey"),
          loss_scale = 1, vcov = TRUE, control = ceres_control())

## S3 method for class 'ceres_nls'
coef(object, ...)
## S3 method for class 'ceres_nls'
residuals(object, ...)
## S3 method for class 'ceres_nls'
vcov(object, ...)
## S3 method for class 'ceres_nls'
deviance(object, ...)
## S3 method for class 'ceres_nls'
df.residual(object, ...)
## S3 method for class 'ceres_nls'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
## S3 method for class 'ceres_nls'
summary(object, ...)
## S3 method for class 'summary.ceres_nls'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

fn	a function `fn(par, ...)` returning the numeric vector of residuals. Its length must not change between calls. Non-finite values during the solve make Ceres reject that step.
par	a numeric vector of starting values. Its names are passed on to `fn` and `jac` and used in the results.
...	further arguments passed to `fn` and `jac`. For the methods, ignored.
jac	an optional function `jac(par, ...)` returning the Jacobian of the residuals, a numeric matrix with one row per residual and one column per parameter. If `NULL`, finite differences are used, which never evaluate `fn` outside the bounds.
lower, upper	bounds on the parameters: a single value, a vector with one value per parameter, or a vector named by some of the parameters, in which case the others are unbounded. A parameter with equal lower and upper bounds is fixed at that value and is not counted as an estimate.

`loss`	the loss function ρ, applied to each residual. With `"linear"` this is ordinary least squares. The others reduce the influence of outliers, see 'Details'.
`loss_scale`	a positive number, the scale of the loss function: roughly the size of residual above which it starts to reduce the influence of an observation. It is in the units of the residuals, so the default of 1 is rarely right for your data. Ignored with `loss = "linear"`.
`vcov`	if `TRUE`, compute the covariance matrix of the estimates. It is not available with a robust loss, see 'Details'.
`control`	options for the solver, from `ceres_control`.
`object, x`	an object of class `"ceres_nls"`, or `"summary.ceres_nls"` for its print method.
`digits`	the number of significant digits to print.

Details

With scale a and $s = r^2$, the loss functions are those of Ceres Solver:

`"huber"`	s for $s \le a^2$, otherwise $2a\sqrt{s} - a^2$
`"soft_l1"`	$2a^2(\sqrt{1 + s/a^2} - 1)$
`"cauchy"`	$a^2 \log(1 + s/a^2)$
`"arctan"`	$a \arctan(s/a)$
`"tukey"`	$\frac{a^2}{3}(1 - (1 - s/a^2)^3)$ for $s \le a^2$, otherwise $a^2/3$

With a parameter at a bound, Ceres Solver alone can stop while the other parameters are far from their optimum, and report convergence. So parameters pushed against a bound are held there and the others solved again, in stages, releasing any that the gradient pulls back inside. `max_iter` applies to each stage. A bounded fit is reported as converged only if a Gauss-Newton step in the free parameters is predicted to reduce the objective by a negligible amount, relative to the objective. This takes more calls of `fn` than an unbounded fit.

The covariance matrix is $\hat{\sigma}^2(J^\top J)^{-1}$ with $\hat{\sigma}^2$ the residual sum of squares divided by the residual degrees of freedom, as in `nls`. That variance estimate assumes the linear loss, so with a robust loss `vcov` is `NULL` and a message says so. It is also `NULL`, with a warning, if the Jacobian is rank deficient, and has a warning if a parameter ends up at a bound, where the usual standard errors are not valid.

`fn` and `jac` are always called on the main R thread, one at a time. Each point costs one call of `fn`, plus one call of `jac` or up to `2 * length(par)` further calls of `fn` for the finite differences, when Ceres needs the Jacobian there. An error or interrupt in `fn` or `jac` stops the solve and is passed on to the caller, and Ceres may also print a line about the failed evaluation. If the solver does not converge, or an interrupt arrives between calls of `fn` and `jac`, there is a warning and the result has the best point found so far.

For more accurate estimates, or fewer calls of `fn`, change `function_tol` and `parameter_tol` in `ceres_control`.

Value

`ceres_nls()` returns a list of class `"ceres_nls"` with

`par`	the estimates.

`residuals`	the residuals at the estimates.
`jacobian`	the Jacobian of the residuals at the estimates.
`vcov`	the covariance matrix of the estimates, or `NULL`.
`cost`	the objective at the estimates, including the loss function.
`initial_cost`	the objective at `par`.
`rss`	the residual sum of squares, without the loss function.
`converged`	`TRUE` if Ceres reports convergence.
`status`, `message`	the termination type and message from Ceres.
`iterations`	the number of iterations.
`n_evals`	the number of calls of `fn` and `jac`.
`time`	the time spent in the solver, in seconds.
`trace`	a data frame with one row per iteration, with the stage it belongs to. Only bounded fits can have more than one stage.
`at_bound`	a logical vector, `TRUE` for estimated parameters at a bound.
`fixed`	a logical vector, `TRUE` for parameters with equal bounds.
`loss`, `loss_scale`, `n`, `call`	the loss function and its scale, the number of residuals and the call.

See Also

`ceres_control`, `nls`. For large problems, where calling R for every evaluation is too slow, write the residuals in C++: see `CxxFlags`.

Examples

```
set.seed(1)
x <- seq(0, 5, length.out = 100)
y <- exp(0.3 * x + 0.1) + rnorm(100, sd = 0.2)
y[c(20, 60)] <- y[c(20, 60)] + 10
resid_exp <- function(p, x, y) y - exp(p[["m"]] * x + p[["c"]])

## ordinary least squares is pulled away by the two outliers
fit <- ceres_nls(resid_exp, c(m = 0, c = 0), x = x, y = y)
summary(fit)

## a robust loss, with m constrained to be positive
robust <- ceres_nls(resid_exp, c(m = 0.2, c = 0), x = x, y = y,
                    lower = c(m = 0), loss = "cauchy", loss_scale = 0.5,
                    vcov = FALSE)
coef(robust)
```

CxxFlags *Compiler and linker flags for the bundled Ceres Solver*

Description

CxxFlags() and LdFlags() return the flags needed to compile and link C++ code against the Ceres Solver library that is built and installed with this package. inlineCxxPlugin() is the plugin used by sourceCpp when a source file contains // [[Rcpp::depends(ceres)]].

Usage

```
CxxFlags(print = TRUE)
LdFlags(print = TRUE)
inlineCxxPlugin(...)
```

Arguments

print	if TRUE, the flags are also printed with cat(), which is what a Makevars file needs.
...	ignored.

Details

In a package that uses Ceres, add ceres, RcppEigen and Rcpp to LinkingTo and use the following in src/Makevars:

```
CXX_STD = CXX17
PKG_LIBS = $(shell "${R_HOME}/bin/Rscript" -e "ceres::LdFlags()")
```

Value

CxxFlags() and LdFlags() invisibly return the flags as a character string. inlineCxxPlugin() returns a list of settings for sourceCpp.

Examples

```
CxxFlags()
LdFlags()

if (requireNamespace("RcppEigen", quietly = TRUE)) {
  Rcpp::sourceCpp(system.file("examples", "curve_fitting.cpp",
                              package = "ceres"),
                  env = new.env())
}
```

Index

∗ **nonlinear**
    ceres_nls, 5
∗ **optimize**
    ceres_control, 3
    ceres_helloworld, 4
    ceres_nls, 5
∗ **package**
    ceres-package, 2
∗ **programming**
    CxxFlags, 8

ceres *(*ceres-package*)*, 2
ceres-package, 2
ceres_control, 3, *6, 7*
ceres_helloworld, *3*, 4
ceres_nls, *3, 4*, 5
coef.ceres_nls *(*ceres_nls*)*, 5
CxxFlags, *2, 3*, *7*, 8

deviance.ceres_nls *(*ceres_nls*)*, 5
df.residual.ceres_nls *(*ceres_nls*)*, 5

inlineCxxPlugin *(*CxxFlags*)*, 8

LdFlags, *2*
LdFlags *(*CxxFlags*)*, 8

nls, *6, 7*

print.ceres_nls *(*ceres_nls*)*, 5
print.summary.ceres_nls *(*ceres_nls*)*, 5

residuals.ceres_nls *(*ceres_nls*)*, 5

sourceCpp, *2*, *8*
summary.ceres_nls *(*ceres_nls*)*, 5

vcov.ceres_nls *(*ceres_nls*)*, 5