Package ‘gslnls’

January 17, 2023

Type Package

Title GSL Nonlinear Least-Squares Fitting

Version 1.1.2

Date 2023-01-17

Description An R interface to nonlinear least-squares optimization with the GNU Scientific Library (GSL), see M. Galassi et al. (2009, ISBN:0954612078). The available trust region methods include the Levenberg-Marquardt algorithm with and without geodesic acceleration, the Steihaug-Toint conjugate gradient algorithm for large systems and several variants of Powell's dogleg algorithm. Bindings are provided to tune a number of parameters affecting the low-level aspects of the trust region algorithms. The interface mimics R's nls() function and returns model objects inheriting from the same class.

BugReports https://github.com/JorisChau/gslnls/issues

URL https://github.com/JorisChau/gslnls

Depends R (>= 3.5)

Imports stats, Matrix

Encoding UTF-8

Language en-US

License GPL-3

SystemRequirements GSL (>= 2.2)

RoxygenNote 7.2.0

NeedsCompilation yes

Author Joris Chau [aut, cre]

Maintainer Joris Chau <joris.chau@openanalytics.eu>

Repository CRAN

Date/Publication 2023-01-17 15:20:05 UTC

R topics documented:

`anova.gsl_nls` *Anova tables*

Description

Returns the analysis of variance (or deviance) tables for two or more fitted "gsl_nls" objects.

Usage

```
## S3 method for class 'gsl_nls'
anova(object, ...)
```

Arguments

object	An object inheriting from class "gsl_nls".
...	Additional objects inheriting from class "gsl_nls".

Value

A data.frame object of class "anova" similar to anova representing the analysis-of-variance table of the fitted model objects when printed.

See Also

anova

Examples

```
## data
set.seed(1)
n <- 50
xy <- data.frame(
  x = (1:n) / n,
  y = 2.5 * exp(-1.5 * (1:n) / n) + 1 + rnorm(n, sd = 0.1)
)
## model
obj1 <- gsl_nls(fn = y ~ A * exp(-lam * x), data = xy, start = c(A = 1, lam = 1))
obj2 <- gsl_nls(fn = y ~ A * exp(-lam * x) + b, data = xy,
    start = c(A = 1, lam = 1, b = 0))

anova(obj1, obj2)
```

---

coef.gsl_nls *Extract model coefficients*

Description

Returns the fitted model coefficients from a "gsl_nls" object. `coefficients` can also be used as an alias.

Usage

```
## S3 method for class 'gsl_nls'
coef(object, ...)
```

Arguments

`object`	An object inheriting from class "gsl_nls".
`...`	At present no optional arguments are used.

Value

Named numeric vector of fitted coefficients similar to `coef`

See Also

`coef`

Examples

```
## data
set.seed(1)
n <- 50
xy <- data.frame(
  x = (1:n) / n,
```

```
  y = 2.5 * exp(-1.5 * (1:n) / n) + rnorm(n, sd = 0.1)
)
## model
obj <- gsl_nls(fn = y ~ A * exp(-lam * x), data = xy, start = c(A = 1, lam = 1))

coef(obj)
```

---

confint.gsl_nls *Confidence interval for model parameters*

Description

Returns asymptotic or profile likelihood confidence intervals for the parameters in a fitted "gsl_nls" object.

Usage

```
## S3 method for class 'gsl_nls'
confint(object, parm, level = 0.95, method = c("asymptotic", "profile"), ...)
```

Arguments

object	An object inheriting from class "gsl_nls".
parm	A character vector of parameter names for which to evaluate confidence intervals, defaults to all parameters.
level	A numeric scalar between 0 and 1 giving the level of the parameter confidence intervals.
method	Method to be used, either "asymptotic" for asymptotic confidence intervals or "profile" for profile likelihood confidence intervals. The latter is only available for "gsl_nls" objects that are also of class "nls".
...	At present no optional arguments are used.

Details

Method "asymptotic" assumes (approximate) normality of the errors in the model and calculates standard asymptotic confidence intervals based on the quantiles of a t-distribution. Method "profile" calculates profile likelihood confidence intervals using the confint.nls method in the **MASS** package and for this reason is only available for "gsl_nls" objects that are *also* of class "nls".

Value

A matrix with columns giving the lower and upper confidence limits for each parameter.

See Also

confint, confint.nls in package **MASS**.

Examples

```
## data
set.seed(1)
n <- 50
xy <- data.frame(
  x = (1:n) / n,
  y = 2.5 * exp(-1.5 * (1:n) / n) + rnorm(n, sd = 0.1)
)
## model
obj <- gsl_nls(fn = y ~ A * exp(-lam * x), data = xy, start = c(A = 1, lam = 1))

## asymptotic ci's
confint(obj)
## Not run:
## profile ci's (requires MASS)
confint(obj, method = "profile")

## End(Not run)
```

---

confintd *Confidence intervals for derived parameters*

Description

`confintd` is a generic function to compute confidence intervals for continuous functions of the parameters in a fitted model. The function invokes particular *methods* which depend on the `class` of the first argument.

Usage

```
confintd(object, expr, level = 0.95, ...)
```

Arguments

`object`	A fitted model object.
`expr`	An expression or character vector that can be transformed to an `expression` giving the function(s) of the parameters to be evaluated. Each expression should evaluate to a numeric scalar.
`level`	A numeric scalar between 0 and 1 giving the level of the derived parameter confidence intervals.
`...`	Additional argument(s) for methods

Value

A matrix with columns giving the fitted values and lower and upper confidence limits for each derived parameter. The row names list the individual derived parameter expressions.

See Also

confint

---

confintd.gsl_nls *Confidence intervals for derived parameters*

Description

Returns fitted values and confidence intervals for continuous functions of parameters from a fitted "gsl_nls" object.

Usage

```
## S3 method for class 'gsl_nls'
confintd(object, expr, level = 0.95, dtype = "symbolic", ...)
```

Arguments

object	A fitted model object.
expr	An expression or character vector that can be transformed to an expression giving the function(s) of the parameters to be evaluated. Each expression should evaluate to a numeric scalar.
level	A numeric scalar between 0 and 1 giving the level of the derived parameter confidence intervals.
dtype	A character string equal to "symbolic" for *symbolic* differentiation of expr with deriv, or "numeric" for *numeric* differentiation of expr with numericDeriv using forward finite differencing.
...	Additional argument(s) for methods

Details

This method assumes (approximate) normality of the errors in the model and confidence intervals are calculated using the *delta method*, i.e. a first-order Taylor approximation of the (continuous) function of the parameters. If dtype = "symbolic" (the default), expr is differentiated with respect to the parameters using symbolic differentiation with deriv. As such, each expression in expr must contain only operators that are known to deriv. If dtype = "numeric", expr is differentiated using numeric differentiation with numericDeriv, which should be used if expr cannot be derived symbolically with deriv.

Value

A matrix with columns giving the fitted values and lower and upper confidence limits for each derived parameter. The row names list the individual derived parameter expressions.

See Also

confint

Examples

```
## data
set.seed(1)
n <- 50
xy <- data.frame(
  x = (1:n) / n,
  y = 2.5 * exp(-1.5 * (1:n) / n) + rnorm(n, sd = 0.1)
)
## model
obj <- gsl_nls(fn = y ~ A * exp(-lam * x), data = xy, start = c(A = 1, lam = 1))

## delta method ci's
confintd(obj, expr = c("log(lam)", "A / lam"))
```

`deviance.gsl_nls` *Model deviance*

Description

Returns the deviance of a fitted `"gsl_nls"` object.

Usage

```
## S3 method for class 'gsl_nls'
deviance(object, ...)
```

Arguments

`object`	An object inheriting from class `"gsl_nls"`.
`...`	At present no optional arguments are used.

Value

Numeric deviance value similar to `deviance`

See Also

`deviance`

Examples

```
## data
set.seed(1)
n <- 50
xy <- data.frame(
  x = (1:n) / n,
  y = 2.5 * exp(-1.5 * (1:n) / n) + rnorm(n, sd = 0.1)
)
## model
```

```
obj <- gsl_nls(fn = y ~ A * exp(-lam * x), data = xy, start = c(A = 1, lam = 1))

deviance(obj)
```

`df.residual.gsl_nls` *Residual degrees-of-freedom*

Description

Returns the residual degrees-of-freedom from a fitted "`gsl_nls`" object.

Usage

```
## S3 method for class 'gsl_nls'
df.residual(object, ...)
```

Arguments

`object`	An object inheriting from class "`gsl_nls`".
`...`	At present no optional arguments are used.

Value

Integer residual degrees-of-freedom similar to `df.residual`.

See Also

`df.residual`

Examples

```
## data
set.seed(1)
n <- 50
xy <- data.frame(
  x = (1:n) / n,
  y = 2.5 * exp(-1.5 * (1:n) / n) + rnorm(n, sd = 0.1)
)
## model
obj <- gsl_nls(fn = y ~ A * exp(-lam * x), data = xy, start = c(A = 1, lam = 1))

df.residual(obj)
```

fitted.gsl_nls *Extract model fitted values*

Description

Returns the fitted responses from a "gsl_nls" object. fitted.values can also be used as an alias.

Usage

```
## S3 method for class 'gsl_nls'
fitted(object, ...)
```

Arguments

object	An object inheriting from class "gsl_nls".
...	At present no optional arguments are used.

Value

Numeric vector of fitted responses similar to fitted.

See Also

fitted

Examples

```
## data
set.seed(1)
n <- 50
xy <- data.frame(
  x = (1:n) / n,
  y = 2.5 * exp(-1.5 * (1:n) / n) + rnorm(n, sd = 0.1)
)
## model
obj <- gsl_nls(fn = y ~ A * exp(-lam * x), data = xy, start = c(A = 1, lam = 1))

fitted(obj)
```

formula.gsl_nls *Extract model formula*

Description

Returns the model formula from a fitted "gsl_nls" object.

Usage

```
## S3 method for class 'gsl_nls'
formula(x, ...)
```

Arguments

x	An object inheriting from class "gsl_nls".
...	At present no optional arguments are used.

Value

If the object inherits from class "nls" returns the fitted model as a formula similar to formula. Otherwise returns the fitted model as a function.

See Also

formula

Examples

```
## data
set.seed(1)
n <- 50
xy <- data.frame(
  x = (1:n) / n,
  y = 2.5 * exp(-1.5 * (1:n) / n) + rnorm(n, sd = 0.1)
)
## model
obj <- gsl_nls(fn = y ~ A * exp(-lam * x), data = xy, start = c(A = 1, lam = 1))

formula(obj)
```

gsl_nls *GSL Nonlinear Least Squares fitting*

Description

Determine the nonlinear least-squares estimates of the parameters of a nonlinear model using the `gsl_multifit_nlinear` module present in the GNU Scientific Library (GSL).

Usage

```
gsl_nls(fn, ...)

## S3 method for class 'formula'
gsl_nls(
  fn,
  data = parent.frame(),
  start,
  algorithm = c("lm", "lmaccel", "dogleg", "ddogleg", "subspace2D"),
  control = gsl_nls_control(),
  jac = NULL,
  fvv = NULL,
  trace = FALSE,
  subset,
  weights,
  na.action,
  model = FALSE,
  ...
)

## S3 method for class 'function'
gsl_nls(
  fn,
  y,
  start,
  algorithm = c("lm", "lmaccel", "dogleg", "ddogleg", "subspace2D"),
  control = gsl_nls_control(),
  jac = NULL,
  fvv = NULL,
  trace = FALSE,
  weights,
  ...
)
```

Arguments

`fn` a nonlinear model defined either as a two-sided formula including variables and parameters, or as a function returning a numeric vector, with first argument the vector of parameters to be estimated. See the individual method descriptions below.

`data`
: an optional data frame in which to evaluate the variables in `fn` if defined as a formula. Can also be a list or an environment, but not a matrix.

`y`
: numeric response vector if `fn` is defined as a function, equal in length to the vector returned by evaluation of the function `fn`.

`start`
: a named list or named numeric vector of starting estimates. `start` is only allowed to be missing if `fn` is a `selfStart` model. If `fn` is a `formula`, a naive guess for `start` is tried, but this should not be relied on.

`algorithm`
: character string specifying the algorithm to use. The following choices are supported:

  - `"lm"` Levenberg-Marquardt algorithm (default)
  - `"lmaccel"` Levenberg-Marquardt algorithm with geodesic acceleration. Can be faster than `"lm"` but less stable. Stability is controlled by the `avmax` parameter in `control`, setting `avmax` to zero is analogous to not using geodesic acceleration.
  - `"dogleg"` Powell's dogleg algorithm
  - `"ddogleg"` Double dogleg algorithm, an improvement over `"dogleg"` by including information about the Gauss-Newton step while the iteration is still far from the minimum.
  - `"subspace2D"` 2D generalization of the dogleg algorithm. This method searches a larger subspace for a solution, it can converge more quickly than `"dogleg"` on some problems.

`control`
: an optional list of control parameters to tune the least squares iterations. See `gsl_nls_control` for the available control parameters and their default values.

`jac`
: either `NULL` (default) or a function returning the `n` by `p` dimensional Jacobian matrix of the nonlinear model `fn`, where `n` is the number of observations and `p` the number of parameters. If a function, the first argument must be the vector of parameters of length `p`. If `NULL`, the Jacobian is computed internally using a finite difference approximations. Can also be `TRUE`, in which case `jac` is derived symbolically with `deriv`, this only works if `fn` is defined as a (non-selfstarting) formula. If `fn` is a `selfStart` model, the Jacobian specified in the `"gradient"` attribute of the self-start model is used instead.

`fvv`
: either `NULL` (default) or a function returning an `n` dimensional vector containing the second directional derivatives of the nonlinear model `fn`, with `n` the number of observations. This argument is only used if geodesic acceleration is enabled (`algorithm = "lmaccel"`). If a function, the first argument must be the vector of parameters of length `p` and the second argument must be the velocity vector also of length `p`. If `NULL`, the second directional derivative vector is computed internal using a finite difference approximation. Can also be `TRUE`, in which case `fvv` is derived symbolically with `deriv`, this only works if `fn` is defined as a (non-selfstarting) formula. If the model function in `fn` also returns a `"hessian"` attribute (similar to the `"gradient"` attribute in a `selfStart` model), this Hessian matrix is used to evaluate the second directional derivatives instead.

trace	logical value indicating if a trace of the iteration progress should be printed. Default is FALSE. If TRUE, the residual (weighted) sum-of-squares and the current parameter estimates are printed after each iteration.
subset	an optional vector specifying a subset of observations to be used in the fitting process. This argument is only used if fn is defined as a formula.
weights	an optional numeric vector of (fixed) weights. When present, the objective function is weighted least squares.
na.action	a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting of options, and is na.fail if that is unset. The 'factory-fresh' default is na.omit. Value na.exclude can be useful. This argument is only used if fn is defined as a formula.
model	a logical value. If TRUE, the model frame is returned as part of the object. Defaults to FALSE. This argument is only used if fn is defined as a formula.
...	additional arguments passed to the calls of fn, jac and fvv if defined as functions.

Value

If fn is a formula returns a list object of class nls. If fn is a function returns a list object of class gsl_nls. See the individual method descriptions for the structures of the returned lists and the generic functions applicable to objects of both classes.

Methods (by class)

- formula: If fn is a formula, the returned list object is of classes gsl_nls and nls. Therefore, all generic functions applicable to objects of class nls, such as anova, coef, confint, deviance, df.residual, fitted, formula, logLik, nobs, predict, print, profile, residuals, summary, vcov and weights are also applicable to the returned list object. In addition, a method confintd is available for inference of derived parameters.
- function: If fn is a function, the first argument must be the vector of parameters and the function should return a numeric vector containing the nonlinear model evaluations at the provided parameter and predictor or covariate vectors. In addition, the argument y needs to contain the numeric vector of observed responses, equal in length to the numeric vector returned by fn. The returned list object is (only) of class gsl_nls. Although the returned object is not of class nls, the following generic functions remain applicable for an object of class gsl_nls: anova, coef, confint, deviance, df.residual, fitted, formula, logLik, nobs, predict, print, residuals, summary, vcov and weights. In addition, a method confintd is available for inference of derived parameters.

References

M. Galassi et al., *GNU Scientific Library Reference Manual (3rd Ed.)*, ISBN 0954612078.

See Also

nls

https://www.gnu.org/software/gsl/doc/html/nls.html

Examples

```
# Example 1: exponential model
# (https://www.gnu.org/software/gsl/doc/html/nls.html#exponential-fitting-example)

## data
set.seed(1)
n <- 50
x <- (seq_len(n) - 1) * 3 / (n - 1)
f <- function(A, lam, b, x) A * exp(-lam * x) + b
y <- f(A = 5, lam = 1.5, b = 1, x) + rnorm(n, sd = 0.25)

## model fit
ex1_fit <- gsl_nls(
  fn = y ~ A * exp(-lam * x) + b,                        ## model formula
  data = data.frame(x = x, y = y),                       ## model fit data
  start = c(A = 0, lam = 0, b = 0)                       ## starting values
)
summary(ex1_fit)                                         ## model summary
predict(ex1_fit, interval = "prediction")                ## prediction intervals

## analytic Jacobian 1
gsl_nls(
  fn = y ~ A * exp(-lam * x) + b,                        ## model formula
  data = data.frame(x = x, y = y),                       ## model fit data
  start = c(A = 0, lam = 0, b = 0),                      ## starting values
  jac = function(par) with(as.list(par),                 ## jacobian
    cbind(A = exp(-lam * x), lam = -A * x * exp(-lam * x), b = 1)
  )
)

## analytic Jacobian 2
gsl_nls(
  fn = y ~ A * exp(-lam * x) + b,                        ## model formula
  data = data.frame(x = x, y = y),                       ## model fit data
  start = c(A = 0, lam = 0, b = 0),                      ## starting values
  jac = TRUE                                             ## automatic derivation
)

## self-starting model
gsl_nls(
  fn =  y ~ SSasymp(x, Asym, R0, lrc),                   ## model formula
  data = data.frame(x = x, y = y)                        ## model fit data
)

# Example 2: Gaussian function
# (https://www.gnu.org/software/gsl/doc/html/nls.html#geodesic-acceleration-example-2)

## data
set.seed(1)
n <- 300
x <- seq_len(n) / n
f <- function(a, b, c, x) a * exp(-(x - b)^2 / (2 * c^2))
```

```
y <- f(a = 5, b = 0.4, c = 0.15, x) * rnorm(n, mean = 1, sd = 0.1)

## Levenberg-Marquardt (default)
gsl_nls(
  fn = y ~ a * exp(-(x - b)^2 / (2 * c^2)),            ## model formula
  data = data.frame(x = x, y = y),                     ## model fit data
  start = c(a = 1, b = 0, c = 1),                      ## starting values
  trace = TRUE                                         ## verbose output
)

## Levenberg-Marquardt w/ geodesic acceleration 1
gsl_nls(
  fn = y ~ a * exp(-(x - b)^2 / (2 * c^2)),            ## model formula
  data = data.frame(x = x, y = y),                     ## model fit data
  start = c(a = 1, b = 0, c = 1),                      ## starting values
  algorithm = "lmaccel",                               ## algorithm
  trace = TRUE                                         ## verbose output
)

## Levenberg-Marquardt w/ geodesic acceleration 2
## second directional derivative
fvv <- function(par, v, x) {
  with(as.list(par), {
    zi <- (x - b) / c
    ei <- exp(-zi^2 / 2)
    2 * v[["a"]] * v[["b"]] * zi / c * ei + 2 * v[["a"]] * v[["c"]] * zi^2 / c * ei -
      v[["b"]]^2 * a / c^2 * (1 - zi^2) * ei -
      2 * v[["b"]] * v[["c"]] * a / c^2 * zi * (2 - zi^2) * ei -
      v[["c"]]^2 * a / c^2 * zi^2 * (3 - zi^2) * ei
  })
}

## analytic fvv 1
gsl_nls(
  fn = y ~ a * exp(-(x - b)^2 / (2 * c^2)),            ## model formula
  data = data.frame(x = x, y = y),                     ## model fit data
  start = c(a = 1, b = 0, c = 1),                      ## starting values
  algorithm = "lmaccel",                               ## algorithm
  trace = TRUE,                                        ## verbose output
  fvv = fvv,                                           ## analytic fvv
  x = x                                                ## argument passed to fvv
)

## analytic fvv 2
gsl_nls(
  fn = y ~ a * exp(-(x - b)^2 / (2 * c^2)),            ## model formula
  data = data.frame(x = x, y = y),                     ## model fit data
  start = c(a = 1, b = 0, c = 1),                      ## starting values
  algorithm = "lmaccel",                               ## algorithm
  trace = TRUE,                                        ## verbose output
  fvv = TRUE                                           ## automatic derivation
)
```

```
# Example 3: Branin function
# (https://www.gnu.org/software/gsl/doc/html/nls.html#comparing-trs-methods-example)

## Branin model function
branin <- function(x) {
  a <- c(-5.1 / (4 * pi^2), 5 / pi, -6, 10, 1 / (8 * pi))
  f1 <- x[2] + a[1] * x[1]^2 + a[2] * x[1] + a[3]
  f2 <- sqrt(a[4] * (1 + (1 - a[5]) * cos(x[1])))
  c(f1, f2)
}

## Dogleg minimization w/ model as function
gsl_nls(
  fn = branin,                  ## model function
  y = c(0, 0),                  ## response vector
  start = c(x1 = 6, x2 = 14.5), ## starting values
  algorithm = "dogleg"          ## algorithm
)
```

---

gsl_nls_control *Tunable Nonlinear Least Squares iteration parameters*

Description

Allow the user to tune the characteristics of the gsl_nls and gsl_nls_large nonlinear least squares algorithms.

Usage

```
gsl_nls_control(
  maxiter = 50,
  scale = "more",
  solver = "qr",
  fdtype = "forward",
  factor_up = 2,
  factor_down = 3,
  avmax = 0.75,
  h_df = sqrt(.Machine$double.eps),
  h_fvv = 0.02,
  xtol = sqrt(.Machine$double.eps),
  ftol = sqrt(.Machine$double.eps),
  gtol = .Machine$double.eps^(1/3)
)
```

Arguments

maxiter	positive integer, termination occurs when the number of iterations reaches maxiter.
scale	character, scaling method or damping strategy determining the diagonal scaling matrix D. The following options are supported:

- "more" Moré rescaling (default). This method makes the problem scale-invariant and has been proven effective on a large class of problems.
- "levenberg" Levenberg rescaling. This method has also proven effective on a large class of problems, but is not scale-invariant. It may perform better for problems susceptible to *parameter evaporation* (parameters going to infinity).
- "marquardt" Marquardt rescaling. This method is scale-invariant, but it is generally considered inferior to both the Levenberg and Moré strategies.

solver
: character, method used to solve the linear least squares system resulting as a subproblem in each iteration. For large-scale problems fitted with gsl_nls_large, the Cholesky solver ("cholesky") is always selected and this parameter is not used. For least squares problems fitted with gsl_nls the following choices are supported:
  - "qr" QR decomposition of the Jacobian (default). This method will produce reliable solutions in cases where the Jacobian is rank deficient or near-singular but does require more operations than the Cholesky method.
  - "cholesky" Cholesky decomposition of the Jacobian. This method is faster than the QR approach, however it is susceptible to numerical instabilities if the Jacobian matrix is rank deficient or near-singular.
  - "svd" SVD decomposition of the Jacobian. This method will produce the most reliable solutions for ill-conditioned Jacobians but is also the slowest.

fdtype
: character, method used to numerically approximate the Jacobian and/or second-order derivatives when geodesic acceleration is used. Either "forward" for forward finite differencing or "center" for centered finite differencing. For least squares problems solved with gsl_nls_large, numerical approximation of the Jacobian matrix is not available and this parameter is only used to numerically approximate the second-order derivatives (if geodesic acceleration is used).

factor_up
: numeric factor by which to increase the trust region radius when a search step is accepted. Too large values may destabilize the search, too small values slow down the search, defaults to 2.

factor_down
: numeric factor by which to decrease the trust region radius when a search step is rejected. Too large values may destabilize the search, too small values slow down the search, defaults to 3.

avmax
: numeric value, the ratio of the acceleration term to the velocity term when using geodesic acceleration to solve the nonlinear least squares problem. Any steps with a ratio larger than avmax are rejected, defaults to 0.75. For problems which experience difficulty converging, this threshold could be lowered.

h_df
: numeric value, the step size for approximating the Jacobian matrix with finite differences, defaults to sqrt(.Machine$double.eps).

h_fvv
: numeric value, the step size for approximating the second directional derivative when geodesic acceleration is used to solve the nonlinear least squares problem, defaults to 0.02. This is only used if no analytic second directional derivative (fvv) is specified in gsl_nls or gsl_nls_large.

xtol
: numeric value, termination occurs when the relative change in parameters between iterations is <= xtol. A general guideline for selecting the step tolerance

is to choose `xtol = 10^(-d)` where `d` is the number of accurate decimal digits desired in the parameters, defaults to `sqrt(.Machine$double.eps)`.

`ftol` numeric value, termination occurs when the relative change in sum of squared residuals between iterations is <= `ftol`, defaults to `sqrt(.Machine$double.eps)`.

`gtol` numeric value, termination occurs when the relative size of the gradient of the sum of squared residuals is <= `gtol`, indicating a local minimum, defaults to `.Machine$double.eps^(1/3)`

Value

A `list` with exactly twelve components:

- maxiter
- scale
- solver
- fdtype
- factor_up
- factor_down
- avmax
- h_df
- h_fvv
- xtol
- ftol
- gtol

with meanings as explained under 'Arguments'.

Note

`ftol` is disabled in some versions of the GSL library.

References

M. Galassi et al., *GNU Scientific Library Reference Manual (3rd Ed.)*, ISBN 0954612078.

See Also

`nls.control`

https://www.gnu.org/software/gsl/doc/html/nls.html#tunable-parameters

Examples

```
## default tuning parameters
gsl_nls_control()
```

gsl_nls_large *GSL Large-scale Nonlinear Least Squares fitting*

Description

Determine the nonlinear least-squares estimates of the parameters of a large nonlinear model system using the `gsl_multilarge_nlinear` module present in the GNU Scientific Library (GSL).

Usage

```
gsl_nls_large(fn, ...)

## S3 method for class 'formula'
gsl_nls_large(
  fn,
  data = parent.frame(),
  start,
  algorithm = c("lm", "lmaccel", "dogleg", "ddogleg", "subspace2D", "cgst"),
  control = gsl_nls_control(),
  jac,
  fvv,
  trace = FALSE,
  subset,
  weights,
  na.action,
  model = FALSE,
  ...
)

## S3 method for class 'function'
gsl_nls_large(
  fn,
  y,
  start,
  algorithm = c("lm", "lmaccel", "dogleg", "ddogleg", "subspace2D", "cgst"),
  control = gsl_nls_control(),
  jac,
  fvv,
  trace = FALSE,
  weights,
  ...
)
```

Arguments

`fn` a nonlinear model defined either as a two-sided formula including variables and parameters, or as a function returning a numeric vector, with first argument the vector of parameters to be estimated. See the individual method descriptions below.

data
: an optional data frame in which to evaluate the variables in fn if defined as a formula. Can also be a list or an environment, but not a matrix.

y
: numeric response vector if fn is defined as a function, equal in length to the vector returned by evaluation of the function fn.

start
: a named list or named numeric vector of starting estimates. start is only allowed to be missing if fn is a selfStart model. If fn is a formula, a naive guess for start is tried, but this should not be relied on.

algorithm
: character string specifying the algorithm to use. The following choices are supported:

- "lm" Levenberg-Marquardt algorithm (default)
- "lmaccel" Levenberg-Marquardt algorithm with geodesic acceleration. Can be faster than "lm" but less stable. Stability is controlled by the avmax parameter in control, setting avmax to zero is analogous to not using geodesic acceleration.
- "dogleg" Powell's dogleg algorithm
- "ddogleg" Double dogleg algorithm, an improvement over "dogleg" by including information about the Gauss-Newton step while the iteration is still far from the minimum.
- "subspace2D" 2D generalization of the dogleg algorithm. This method searches a larger subspace for a solution, it can converge more quickly than "dogleg" on some problems.
- "cgst" Steihaug-Toint Conjugate Gradient algorithm, a generalization of the dogleg algorithm that avoids solving for the Gauss-Newton step directly, instead using an iterative conjugate gradient algorithm. The method performs well at points where the Jacobian is singular, and is also suitable for large-scale problems where factoring the Jacobian matrix is prohibitively expensive.

control
: an optional list of control parameters to tune the least squares iterations. See gsl_nls_control for the available control parameters and their default values.

jac
: a function returning the n by p dimensional Jacobian matrix of the nonlinear model fn, where n is the number of observations and p the number of parameters. The first argument must be the vector of parameters of length p. Can also be TRUE, in which case jac is derived symbolically with deriv, this only works if fn is defined as a (non-selfstarting) formula. If fn is a selfStart model, the Jacobian specified in the "gradient" attribute of the self-start model is used instead.

fvv
: a function returning an n dimensional vector containing the second directional derivatives of the nonlinear model fn, with n the number of observations. This argument is only used if geodesic acceleration is enabled (algorithm = "lmaccel"). The first argument must be the vector of parameters of length p and the second argument must be the velocity vector also of length p. Can also be TRUE, in which case fvv is derived symbolically with deriv, this only works if fn is defined as a (non-selfstarting) formula. If the model function in fn also returns a "hessian" attribute (similar to the "gradient" attribute in a selfStart

	model), this Hessian matrix is used to evaluate the second directional derivatives instead.
`trace`	logical value indicating if a trace of the iteration progress should be printed. Default is `FALSE`. If `TRUE`, the residual (weighted) sum-of-squares, the squared (Euclidean) norm of the current parameter estimates and the condition number of the Jacobian are printed after each iteration.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process. This argument is only used if `fn` is defined as a formula.
`weights`	an optional numeric vector of (fixed) weights. When present, the objective function is weighted least squares.
`na.action`	a function which indicates what should happen when the data contain `NA`s. The default is set by the `na.action` setting of `options`, and is `na.fail` if that is unset. The 'factory-fresh' default is `na.omit`. Value `na.exclude` can be useful. This argument is only used if `fn` is defined as a formula.
`model`	a logical value. If `TRUE`, the model frame is returned as part of the object. Defaults to `FALSE`. This argument is only used if `fn` is defined as a formula.
`...`	additional arguments passed to the calls of `fn`, `jac` and `fvv` if defined as functions.

Value

If `fn` is a `formula` returns a list object of class `nls`. If `fn` is a `function` returns a list object of class `gsl_nls`. See the individual method descriptions for the structures of the returned lists and the generic functions applicable to objects of both classes.

Methods (by class)

- `formula`: If `fn` is a `formula`, the returned list object is of classes `gsl_nls` and `nls`. Therefore, all generic functions applicable to objects of class `nls`, such as `anova`, `coef`, `confint`, `deviance`, `df.residual`, `fitted`, `formula`, `logLik`, `nobs`, `predict`, `print`, `profile`, `residuals`, `summary`, `vcov` and `weights` are also applicable to the returned list object. In addition, a method `confintd` is available for inference of derived parameters.
- `function`: If `fn` is a `function`, the first argument must be the vector of parameters and the function should return a numeric vector containing the nonlinear model evaluations at the provided parameter and predictor or covariate vectors. In addition, the argument `y` needs to contain the numeric vector of observed responses, equal in length to the numeric vector returned by `fn`. The returned list object is (only) of class `gsl_nls`. Although the returned object is not of class `nls`, the following generic functions remain applicable for an object of class `gsl_nls`: `anova`, `coef`, `confint`, `deviance`, `df.residual`, `fitted`, `formula`, `logLik`, `nobs`, `predict`, `print`, `residuals`, `summary`, `vcov` and `weights`. In addition, a method `confintd` is available for inference of derived parameters.

References

M. Galassi et al., *GNU Scientific Library Reference Manual (3rd Ed.)*, ISBN 0954612078.

See Also

gsl_nls

https://www.gnu.org/software/gsl/doc/html/nls.html

Examples

```
# Large NLS example
# (https://www.gnu.org/software/gsl/doc/html/nls.html#large-nonlinear-least-squares-example)

## number of parameters
p <- 250

## model function
f <- function(theta) {
  c(sqrt(1e-5) * (theta - 1), sum(theta^2) - 0.25)
}

## jacobian function
jac <- function(theta) {
  rbind(diag(sqrt(1e-5), nrow = length(theta)), 2 * t(theta))
}

## dense Levenberg-Marquardt
gsl_nls_large(
  fn = f,                       ## model
  y = rep(0, p + 1),            ## (dummy) responses
  start = 1:p,                  ## start values
  algorithm = "lm",             ## algorithm
  jac = jac,                    ## jacobian
  control = list(maxiter = 250)
)

## dense Steihaug-Toint conjugate gradient
gsl_nls_large(
  fn = f,                       ## model
  y = rep(0, p + 1),            ## (dummy) responses
  start = 1:p,                  ## start values
  jac = jac,                    ## jacobian
  algorithm = "cgst"            ## algorithm
)

## sparse Jacobian function
jacsp <- function(theta) {
  rbind(Matrix::Diagonal(x = sqrt(1e-5), n = length(theta)), 2 * t(theta))
}

## sparse Levenberg-Marquardt
gsl_nls_large(
  fn = f,                       ## model
  y = rep(0, p + 1),            ## (dummy) responses
  start = 1:p,                  ## start values
  algorithm = "lm",             ## algorithm
```

```
  jac = jacsp,                   ## sparse jacobian
  control = list(maxiter = 250)
)

## sparse Steihaug-Toint conjugate gradient
gsl_nls_large(
  fn = f,                        ## model
  y = rep(0, p + 1),             ## (dummy) responses
  start = 1:p,                   ## start values
  jac = jacsp,                   ## sparse jacobian
  algorithm = "cgst"             ## algorithm
)
```

---

logLik.gsl_nls *Extract model log-likelihood*

Description

Returns the model log-likelihood of a fitted "gsl_nls" object.

Usage

```
## S3 method for class 'gsl_nls'
logLik(object, REML = FALSE, ...)
```

Arguments

object	An object inheriting from class "gsl_nls".
REML	logical value; included for compatibility reasons only, should not be used.
...	At present no optional arguments are used.

Value

Numeric object of class "logLik" identical to logLik.

See Also

logLik

Examples

```
## data
set.seed(1)
n <- 50
xy <- data.frame(
  x = (1:n) / n,
  y = 2.5 * exp(-1.5 * (1:n) / n) + rnorm(n, sd = 0.1)
)
```

```
## model
obj <- gsl_nls(fn = y ~ A * exp(-lam * x), data = xy, start = c(A = 1, lam = 1))

logLik(obj)
```

---

nobs.gsl_nls *Extract the number of observations*

Description

Returns the number of *observations* from a "gsl_nls" object.

Usage

```
## S3 method for class 'gsl_nls'
nobs(object, ...)
```

Arguments

object	An object inheriting from class "gsl_nls".
...	At present no optional arguments are used.

Value

Integer number of observations similar to nobs

See Also

nobs

Examples

```
## data
set.seed(1)
n <- 50
xy <- data.frame(
  x = (1:n) / n,
  y = 2.5 * exp(-1.5 * (1:n) / n) + rnorm(n, sd = 0.1)
)
## model
obj <- gsl_nls(fn = y ~ A * exp(-lam * x), data = xy, start = c(A = 1, lam = 1))

nobs(obj)
```

predict.gsl_nls *Calculate model predicted values*

Description

Returns predicted values for the expected response from a fitted "gsl_nls" object. Asymptotic confidence or prediction (tolerance) intervals at a given `level` can be evaluated by specifying the appropriate `interval` argument.

Usage

```
## S3 method for class 'gsl_nls'
predict(
  object,
  newdata,
  scale = NULL,
  interval = c("none", "confidence", "prediction"),
  level = 0.95,
  ...
)
```

Arguments

`object`	An object inheriting from class "gsl_nls".
`newdata`	A named list or data.frame in which to look for variables with which to predict. If `newdata` is missing, the predicted values at the original data points are returned.
`scale`	A numeric scalar or vector. If it is set, it is used as the residual standard deviation (or vector of residual standard deviations) in the computation of the standard errors, otherwise this information is extracted from the model fit.
`interval`	A character string indicating if confidence or prediction (tolerance) intervals at the specified level should be returned.
`level`	A numeric scalar between 0 and 1 giving the confidence level for the intervals (if any) to be calculated.
`...`	At present no optional arguments are used.

Value

If `interval = "none"` (default), a vector of predictions for the mean response. Otherwise, a matrix with columns `fit`, `lwr` and `upr`. The first column (`fit`) contains predictions for the mean response. The other two columns contain lower (`lwr`) and upper (`upr`) confidence or prediction bounds at the specified `level`.

See Also

`predict.nls`

Examples

```
## data
set.seed(1)
n <- 50
xy <- data.frame(
  x = (1:n) / n,
  y = 2.5 * exp(-1.5 * (1:n) / n) + rnorm(n, sd = 0.1)
)
## model
obj <- gsl_nls(fn = y ~ A * exp(-lam * x), data = xy, start = c(A = 1, lam = 1))

predict(obj)
predict(obj, newdata = data.frame(x = 1:(2 * n) / n))
predict(obj, interval = "confidence")
predict(obj, interval = "prediction", level = 0.99)
```

`residuals.gsl_nls` *Extract model residuals*

Description

Returns the model residuals from a fitted `"gsl_nls"` object. `resid` can also be used as an alias.

Usage

```
## S3 method for class 'gsl_nls'
residuals(object, type = c("response", "pearson"), ...)
```

Arguments

`object`	An object inheriting from class `"gsl_nls"`.
`type`	character; if `"response"` the raw residuals are returned, if `"pearson"` the Pearson are returned, i.e. the raw residuals divided by their standard error.
`...`	At present no optional arguments are used.

Value

Numeric vector of model residuals similar to `residuals`.

See Also

`residuals`

Examples

```
## data
set.seed(1)
n <- 50
xy <- data.frame(
  x = (1:n) / n,
  y = 2.5 * exp(-1.5 * (1:n) / n) + rnorm(n, sd = 0.1)
)
## model
obj <- gsl_nls(fn = y ~ A * exp(-lam * x), data = xy, start = c(A = 1, lam = 1))

residuals(obj)
```

---

summary.gsl_nls *Model summary*

Description

Returns the model summary for a fitted "gsl_nls" object.

Usage

```
## S3 method for class 'gsl_nls'
summary(object, correlation = FALSE, symbolic.cor = FALSE, ...)
```

Arguments

object	An object inheriting from class "gsl_nls".
correlation	logical; if TRUE, the correlation matrix of the estimated parameters is returned and printed.
symbolic.cor	logical; if TRUE, print the correlations in a symbolic form (see symnum) rather than as numbers.
...	At present no optional arguments are used.

Value

List object of class "summary.nls" identical to summary.nls

See Also

summary.nls

Examples

```
## data
set.seed(1)
n <- 50
xy <- data.frame(
  x = (1:n) / n,
  y = 2.5 * exp(-1.5 * (1:n) / n) + rnorm(n, sd = 0.1)
)
## model
obj <- gsl_nls(fn = y ~ A * exp(-lam * x), data = xy, start = c(A = 1, lam = 1))

summary(obj)
```

---

vcov.gsl_nls — *Calculate variance-covariance matrix*

Description

Returns the estimated variance-covariance matrix of the model parameters from a fitted "gsl_nls" object.

Usage

```
## S3 method for class 'gsl_nls'
vcov(object, ...)
```

Arguments

`object`	An object inheriting from class "gsl_nls".
`...`	At present no optional arguments are used.

Value

A matrix containing the estimated covariances between the parameter estimates similar to `vcov` with row and column names corresponding to the parameter names given by `coef.gsl_nls`.

See Also

`vcov`

Examples

```
## data
set.seed(1)
n <- 50
xy <- data.frame(
  x = (1:n) / n,
  y = 2.5 * exp(-1.5 * (1:n) / n) + rnorm(n, sd = 0.1)
```

```
)
## model
obj <- gsl_nls(fn = y ~ A * exp(-lam * x), data = xy, start = c(A = 1, lam = 1))

vcov(obj)
```

Index