Package 'gslnls'

September 28, 2025

Type Package

Title GSL Multi-Start Nonlinear Least-Squares Fitting

Version 1.4.2

Date 2025-09-27

Description An R interface to weighted 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. Multi-start optimization based on quasi-random samples is implemented using a modified version of the algorithm in Hickernell and Yuan (1997, OR Transactions). Robust nonlinear regression can be performed using various robust loss functions, in which case the optimization problem is solved by iterative reweighted least squares (IRLS). 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 LGPL-3

SystemRequirements GSL (>= 2.3)

Biarch true

RoxygenNote 7.3.2

NeedsCompilation yes

Author Joris Chau [aut, cre]

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

Repository CRAN

Date/Publication 2025-09-28 12:30:02 UTC

Contents

anova.gsl_nls . . . 2
coef.gsl_nls . . . 3
confint.gsl_nls . . . 4
confintd . . . 5
confintd.gsl_nls . . . 6
cooks.distance.gsl_nls . . . 7
deviance.gsl_nls . . . 8
df.residual.gsl_nls . . . 9
fitted.gsl_nls . . . 10
formula.gsl_nls . . . 11
gsl_nls . . . 12
gsl_nls_control . . . 20
gsl_nls_large . . . 24
gsl_nls_loss . . . 28
hatvalues.gsl_nls . . . 31
logLik.gsl_nls . . . 32
nls_test_list . . . 33
nls_test_problem . . . 34
nobs.gsl_nls . . . 35
predict.gsl_nls . . . 36
residuals.gsl_nls . . . 38
sigma.gsl_nls . . . 39
summary.gsl_nls . . . 40
vcov.gsl_nls . . . 41

Index **42**

`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 <- 25
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 <- 25
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 <- 25
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 <- 25
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"))
```

---

cooks.distance.gsl_nls

Calculate Cook's distance

Description

Returns Cook's distance values from a fitted "gsl_nls" object based on the estimated variance-covariance matrix of the model parameters.

Usage

```
## S3 method for class 'gsl_nls'
cooks.distance(model, ...)
```

Arguments

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

Value

Numeric vector of Cook's distance values similar to cooks.distance.

See Also

cooks.distance

Examples

```
## data
set.seed(1)
n <- 25
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))

cooks.distance(obj)
```

`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 <- 25
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 <- 25
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 <- 25
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 <- 25
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"),
 loss = c("default", "huber", "barron", "bisquare", "welsh", "optimal", "hampel", "ggw",
    "lqq"),
  control = gsl_nls_control(),
  lower,
  upper,
  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"),
 loss = c("default", "huber", "barron", "bisquare", "welsh", "optimal", "hampel", "ggw",
    "lqq"),
  control = gsl_nls_control(),
  lower,
  upper,
  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 vector, list or matrix of initial parameter values or parameter ranges. `start` is only allowed to be missing if `fn` is a `selfStart` model. The following choices are supported: • a named list or named vector of numeric starting values. If `start` has no missing values, a standard single-start optimization is performed. If `start` contains missing values for one or more parameters, a multi-start algorithm (see 'Details') with dynamic starting ranges for the undefined parameters and fixed starting values for the remaining parameters is executed. If `start` is a named list or vector containing *only* missing values, the multi-start algorithm considers dynamically changing starting ranges for all parameters. Note that there is no guarantee that the optimizing solution is a global minimum of the least-squares objective. • a named list with starting parameter ranges in the form of length-2 numeric vectors. Can also be a (`2` by `p`) named matrix with as columns the numeric starting ranges for the parameters. If `start` contains no missing values, a multi-start algorithm with fixed starting ranges for the parameters is executed. Otherwise, if `start` contains infinities or missing values (e.g. `c(0, Inf)` or `c(NA, NA)`), the multi-start algorithm considers dynamically changing starting ranges for the parameters with infinite and/or missing ranges.
`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. 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.

`loss` character string specifying the loss function to optimize. The following choices are supported:

- `"default"` default squared loss function.
- `"huber"` Huber loss function.
- `"barron"` Barron's smooth family of loss functions.
- `"biweight"` Tukey's biweight/bisquare loss function.
- `"welsh"` Welsh/Leclerc loss function.
- `"optimal"` Optimal loss function (Maronna et al. (2006), Section 5.9.1).
- `"hampel"` Hampel loss function.
- `"ggw"` Generalized Gauss-Weight loss function.
- `"lqq"` Linear Quadratic Quadratic loss function.

If a character string, the default tuning parameters as specified by `gsl_nls_loss` are used. Instead, a list as returned by `gsl_nls_loss` with non-default tuning parameters is also accepted. For all choices other than `rho = "default"`, iterative reweighted least squares (IRLS) is used to solve the MM-estimation problem.

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

`lower` a named list or named numeric vector of parameter lower bounds, or an unnamed numeric scalar to be replicated for all parameters. If missing (default), the parameters are unconstrained from below.

`upper` a named list or named numeric vector of parameter upper bounds, or an unnamed numeric scalar to be replicated for all parameters. If missing (default), the parameters are unconstrained from above.

`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 of length n or an n-by-n symmetric positive definite weight matrix. If weights is a vector or a diagonal matrix, the objective function is weighted least squares. If weights is a general matrix, the objective function is generalized 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)

- gsl_nls(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, hatvalues, cooks.distance and weights are also applicable to the returned list object. In addition, a method confintd is available for inference of derived parameters.
- gsl_nls(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, hatvalues, cooks.distance and weights. In addition, a method confintd is available for inference of derived parameters.

Multi-start algorithm

If start is a list or matrix of parameter ranges, or contains any missing values, a modified version of the multi-start algorithm described in Hickernell and Yuan (1997) is applied. Note that the start parameter ranges are only used to bound the domain for the starting values, i.e. the resulting parameter estimates are not constrained to lie within these bounds, use lower and/or upper for this purpose

instead. Quasi-random starting values are sampled in the unit hypercube from a Sobol sequence if `p < 41` or from a Halton sequence (up to `p = 1229`) otherwise. The initial starting values are scaled to the specified parameter ranges using an inverse (scaled) logistic function favoring starting values near the center of the (scaled) domain. The trust region method as specified by `algorithm` used for the inexpensive and expensive local search (see Algorithm 2.1 of Hickernell and Yuan (1997)) are the same, only differing in the number of search iterations `mstart_p` versus `mstart_maxiter`, where `mstart_p` is typically much smaller than `mstart_maxiter`. When a new stationary point is detected, the scaling step from the unit hypercube to the starting value domain is updated using the diagonal of the estimated trust method's scaling matrix `D`, which improves optimization performance especially when the parameters live on very different scales. The multi-start algorithm terminates when NSP (number of stationary points) is larger than or equal to `mstart_minsp` and NWSP (number of worse stationary points) is larger than `mstart_r + sqrt(mstart_r) * NSP`, or when the maximum number of major iterations `mstart_maxstart` is reached. After termination of the multi-start algorithm, a full single-start optimization is executed starting from the best multi-start solution.

Missing starting values

If `start` contains missing (or infinite) values, the multi-start algorithm is executed without fixed parameter ranges for the missing parameters. The ranges for the missing parameters are initialized to the unit interval and dynamically increased or decreased in each major iteration of the multi-start algorithm. The decision to increase or decrease a parameter range is driven by the minimum and maximum parameter values obtained by the first `mstart_q` inexpensive local searches ordered by their squared loss, which typically provide a decent indication of the order of magnitude of the parameter range in which to search for the optimal solution. Note that this procedure is not expected to always return a global minimum of the nonlinear least-squares objective. Especially when the objective function contains many local optima, the algorithm may be unable to select parameter ranges that include the global minimizing solution. In this case, it may help to increase the values of `mstart_n`, `mstart_r` or `mstart_minsp` to avoid early termination of the algorithm at the cost of increased computational effort.

References

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

Hickernell, F.J. and Yuan, Y. (1997) *"A simple multistart algorithm for global optimization"*, OR Transactions, Vol. 1 (2).

See Also

`nls`

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

https://CRAN.R-project.org/package=robustbase/vignettes/psi_functions.pdf

Examples

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

## data
```

```
set.seed(1)
n <- 25
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

## multi-start
gsl_nls(
  fn = y ~ A * exp(-lam * x) + b,                               ## model formula
  data = data.frame(x = x, y = y),                              ## model fit data
  start = list(A = c(0, 100), lam = c(0, 10), b = c(-10, 10)) ## fixed starting ranges
)
## missing starting values
gsl_nls(
  fn = y ~ A * exp(-lam * x) + b,                          ## model formula
  data = data.frame(x = x, y = y),                         ## model fit data
  start = c(A = NA, lam = NA, b = NA)                      ## dynamic starting ranges
)

## robust regression
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
  loss = "barron"                                        ## L1-L2 loss
)

## 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 <- 100
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
)

# Available example problems
nls_test_list()

## BOD regression
## (https://www.itl.nist.gov/div898/strd/nls/nls_main.shtml)
(boxbod <- nls_test_problem(name = "BoxBOD"))
with(boxbod,
     gsl_nls(
       fn = fn,
       data = data,
       start = list(b1 = NA, b2 = NA)
     )
)

## Rosenbrock function
(rosenbrock <- nls_test_problem(name = "Rosenbrock"))
with(rosenbrock,
     gsl_nls(
       fn = fn,
```

```
      y = y,
      start = c(x1 = NA, x2 = NA),
      jac = jac
    )
)
```

---

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 = 100,
  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$double.eps),
  gtol = sqrt(.Machine$double.eps),
  mstart_n = 30,
  mstart_p = 5,
  mstart_q = mstart_n%/%10,
  mstart_r = 4,
  mstart_s = 2,
  mstart_tol = 0.25,
  mstart_maxiter = 10,
  mstart_maxstart = 250,
  mstart_minsp = 1,
  irls_maxiter = 50,
  irls_xtol = .Machine$double.eps^0.25,
  ...
)
```

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 sub-problem 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 `sqrt(.Machine$double.eps)`
`mstart_n`	positive integer, number of quasi-random points drawn in each major iteration, parameter `N` in Hickernell and Yuan (1997). Default is 30.
`mstart_p`	positive integer, number of iterations of inexpensive local search to concentrate the sample, parameter `p` in Hickernell and Yuan (1997). Default is 5.
`mstart_q`	positive integer, number of points retained in the concentrated sample, parameter `q` in Hickernell and Yuan (1997). Default is `mstart_n %/% 10`..
`mstart_r`	positive integer, scaling factor of number of stationary points determining when the multi-start algorithm terminates, parameter `r` in Hickernell and Yuan (1997). Default is 4. If the starting ranges for one or more parameters are unbounded and updated dynamically, `mstart_r` is multiplied by a factor 10 to avoid early termination.
`mstart_s`	positive integer, minimum number of iterations a point needs to be retained before starting an efficient local search, parameter `s` in Hickernell and Yuan (1997). Default is 2.
`mstart_tol`	numeric value, multiplicative tolerance `(1 + mstart_tol)` used as criterion to start an efficient local search (epsilon in Algorithm 2.1, Hickernell and Yuan (1997)).
`mstart_maxiter`	positive integer, maximum number of iterations in the efficient local search algorithm (Algorithm B, Hickernell and Yuan (1997)), defaults to 10.
`mstart_maxstart`	positive integer, minimum number of major iterations (Algorithm 2.1, Hickernell and Yuan (1997)) before the multi-start algorithm terminates, defaults to 250.
`mstart_minsp`	positive integer, minimum number of detected stationary points before the multi-start algorithm terminates, defaults to 1.
`irls_maxiter`	positive integer, maximum number of IRLS iterations, defaults to 50. Only used in case of a non-default loss function (`loss != "default"`) optimized by IRLS.
`irls_xtol`	numeric value, termination of the IRLS procedure occurs when the relative change in parameters between IRLS iterations is <= `irls_xtol`, defaults to `.Machine$double.eps^(1/4)`. Only used in case of a non-default loss function (`loss != "default"`) optimized by IRLS.
`...`	any additional arguments (currently not used).

Value

A `list` with exactly twenty-three components:

- maxiter
- scale
- solver
- fdtype
- factor_up
- factor_down
- avmax
- h_df
- h_fvv
- xtol
- ftol
- gtol
- mstart_n
- mstart_p
- mstart_q
- mstart_r
- mstart_s
- mstart_tol
- mstart_maxiter
- mstart_maxstart
- mstart_minsp
- irls_maxiter
- irls_xtol

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.

Hickernell, F.J. and Yuan, Y. (1997) *"A simple multistart algorithm for global optimization"*, OR Transactions, Vol. 1 (2).

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 and multistart algorithm. 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)

- `gsl_nls_large(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`, `hatvalues`, `cooks.distance` and `weights` are also applicable to the returned list object. In addition, a method `confintd` is available for inference of derived parameters.
- `gsl_nls_large(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`, `hatvalues`, `cooks.distance` 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
)
```

`gsl_nls_loss` *Robust loss functions with tunable parameters*

Description

Allow the user to tune the coefficient(s) of the robust loss functions supported by `gsl_nls`. For all choices other than `rho = "default"`, the MM-estimation problem is optimized by means of iterative reweighted least squares (IRLS).

Usage

```
gsl_nls_loss(
  rho = c("default", "huber", "barron", "bisquare", "welsh", "optimal", "hampel", "ggw",
    "lqq"),
  cc = NULL
)
```

Arguments

`rho`	character loss function, one of `"default"`, `"huber"`, `"barron"`, `"bisquare"`, `"welsh"`, `"optimal"`, `"hampel"`, `"ggw"`, `"lqq"`.
`cc`	named or unnamed numeric vector of tuning parameters. The length of this argument depends on the selected `rho` function (see 'Details'). If `NULL`, the default tuning parameters are returned.

Value

A `list` with two components:

- rho
- cc

with meanings as explained under 'Arguments'.

Loss function details

`default` Default squared loss, no iterative reweighted least squares (IRLS) is required in this case.

$$\rho(x) = x^2$$

`huber` Huber loss function with scaling constant k, defaulting to $k = 1.345$ for 95% efficiency of the regression estimator.

$$\rho(x, k) = \begin{cases} \frac{1}{2}x^2 & \text{if } |x| \leq k \\ k(|x| - \frac{k}{2}) & \text{if } |x| > k \end{cases}$$

`barron` Barron's smooth family of loss functions with robustness parameter $\alpha \leq 2$ (default $\alpha = 1$) and scaling constant k (default $k = 1.345$). Special cases include: (scaled) squared loss for $\alpha = 2$; L1-L2 loss for $\alpha = 1$; Cauchy loss for $\alpha = 0$; Geman-McClure loss for $\alpha = -2$; and Welsch/Leclerc loss for $\alpha = -\infty$. See Barron (2019) for additional details.

$$\rho(x, \alpha, k) = \begin{cases} \frac{1}{2}(x/k)^2 & \text{if } \alpha = 2 \\ \log\left(\frac{1}{2}(x/k)^2 + 1\right) & \text{if } \alpha = 0 \\ 1 - \exp\left(-\frac{1}{2}(x/k)^2\right) & \text{if } \alpha = -\infty \\ \frac{|\alpha-2|}{\alpha}\left(\left(\frac{(x/k)^2}{|\alpha-2|} + 1\right)^{\alpha/2} - 1\right) & \text{otherwise} \end{cases}$$

`bisquare` Tukey's biweight/bisquare loss function with scaling constant k, defaulting to $k = 4.685$ for 95% efficiency of the regression estimator, ($k = 1.548$ gives a breakdown point of 0.5 of the S-estimator).

$$\rho(x, k) = \begin{cases} 1 - (1 - (x/k)^2)^3 & \text{if } |x| \leq k \\ 1 & \text{if } |x| > k \end{cases}$$

`welsh` Welsh/Leclerc loss function with scaling constant k, defaulting to $k = 2.11$ for 95% efficiency of the regression estimator, ($k = 0.577$ gives a breakdown point of 0.5 of the S-estimator). This is equivalent to the Barron loss function with robustness parameter $\alpha = -\infty$.

$$\rho(x, k) = 1 - \exp\left(-\frac{1}{2}(x/k)^2\right)$$

`optimal` Optimal loss function as in Section 5.9.1 of Maronna et al. (2006) with scaling constant k, defaulting to $k = 1.060$ for 95% efficiency of the regression estimator, ($k = 0.405$ gives a breakdown point of 0.5 of the S-estimator).

$$\rho(x, k) = \int_0^x \text{sign}(u)\left(-\frac{\phi'(|u|) + k}{\phi(|u|)}\right)^+ du$$

with ϕ the standard normal density.

`hampel` Hampel loss function with a single scaling constant k, setting $a = 1.5k$, $b = 3.5k$ and $r = 8k$. $k = 0.902$ by default, resulting in 95% efficiency of the regression estimator, ($k = 0.212$ gives a breakdown point of 0.5 of the S-estimator).

$$\rho(x, a, b, r) = \begin{cases} \frac{1}{2}x^2/C & \text{if } |x| \leq a \\ \left(\frac{1}{2}a^2 + a(|x| - a)\right)/C & \text{if } a < |x| \leq b \\ \frac{a}{2}\left(2b - a + (|x| - b)\left(1 + \frac{r-|x|}{r-b}\right)\right)/C & \text{if } b < |x| \leq r \\ 1 & \text{if } r < |x| \end{cases}$$

with $C = \rho(\infty) = \frac{a}{2}(b - a + r)$.

`ggw` Generalized Gauss-Weight loss function, a generalization of the Welsh/Leclerc loss with tuning constants a, b, c, defaulting to $a = 1.387$, $b = 1.5$, $c = 1.063$ for 95% efficiency of the regression estimator, ($a = 0.204, b = 1.5, c = 0.296$ gives a breakdown point of 0.5 of the S-estimator).

$$\rho(x, a, b, c) = \int_0^x \psi(u, a, b, c) du$$

with,

$$\psi(x, a, b, c) = \begin{cases} x & \text{if } |x| \leq c \\ \exp\left(-\frac{1}{2}\frac{(|x|-c)^b}{a}\right) x & \text{if } |x| > c \end{cases}$$

`lqq` Linear Quadratic Quadratic loss function with tuning constants b, c, s, defaulting to $b = 1.473$, $c = 0.982$, $s = 1.5$ for 95% efficiency of the regression estimator, ($b = 0.402, c = 0.268, s = 1.5$ gives a breakdown point of 0.5 of the S-estimator).

$$\rho(x, b, c, s) = \int_0^x \psi(u, b, c, s) du$$

with,

$$\psi(x, b, c, s) = \begin{cases} x & \text{if } |x| \leq c \\ \text{sign}(x)\left(|x| - \frac{s}{2b}(|x| - c)^2\right) & \text{if } c < |x| \leq b + c \\ \text{sign}(x)\left(c + b - \frac{bs}{2} + \frac{s-1}{a}\left(\frac{1}{2}\tilde{x}^2 - a\tilde{x}\right)\right) & \text{if } b + c < |x| \leq a + b + c \\ 0 & \text{otherwise} \end{cases}$$

where $\tilde{x} = |x| - b - c$ and $a = (2c + 2b - bs)/(s - 1)$.

References

J.T. Barron (2019). *A general and adaptive robust loss function*. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 4331-4339).

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

R.A. Maronna et al., *Robust Statistics: Theory and Methods*, ISBN 0470010924.

See Also

https://CRAN.R-project.org/package=robustbase

Examples

```
## Huber loss with default scale k
gsl_nls_loss(rho = "huber")

## L1-L2 loss (alpha = 1)
gsl_nls_loss(rho = "barron", cc = c(1, 1.345))

## Cauchy loss (alpha = 0)
gsl_nls_loss(rho = "barron", cc = c(k = 1.345, alpha = 0))
```

`hatvalues.gsl_nls` *Calculate leverage values*

Description

Returns leverage values (hat values) from a fitted `"gsl_nls"` object based on the estimated variance-covariance matrix of the model parameters.

Usage

```
## S3 method for class 'gsl_nls'
hatvalues(model, ...)
```

Arguments

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

Value

Numeric vector of leverage values similar to `hatvalues`.

See Also

`hatvalues`

Examples

```
## data
set.seed(1)
n <- 25
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))

hatvalues(obj)
```

`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 <- 25
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)
```

`nls_test_list` *Available NLS test problems*

Description

Returns an overview of 59 NLS test problems originating primarily from the NIST Statistical Reference Datasets (StRD) archive; Bates and Watts (1988); and More, Garbow and Hillstrom (1981).

Usage

```
nls_test_list(fields = c("name", "class", "p", "n", "check"))
```

Arguments

`fields`	optional character vector to return a subset of columns in the data.frame.

Value

a data.frame with high-level information about the available test problems. The following columns are returned by default:

- `"name"` Name of the test problem for use in `nls_test_problem`.
- `"class"` Either `"formula"` if the model is defined as a formula or `"function"` if defined as a function.
- `"p"` Default number of parameters in the test problem.
- `"n"` Default number of residuals in the test problem.
- `"check"` One of the following three options: (1) `"p, n fixed"` if the listed values for `p` and `n` are the only ones possible; (2) `"p <= n free"` if the values for `p` and `n` are not fixed, but `p` must be smaller or equal to `n`; (3) `"p == n free"` if the values for `p` and `n` are not fixed, but `p` must be equal to `n`.

References

D.M. Bates and Watts, D.G. (1988). *Nonlinear Regression Analysis and Its Applications*, Wiley, ISBN: 0471816434.

J.J. Moré, Garbow, B.S. and Hillstrom, K.E. (1981). *Testing unconstrained optimization software*, ACM Transactions on Mathematical Software, 7(1), 17-41.

See Also

`nls_test_problem`

https://www.itl.nist.gov/div898/strd/nls/nls_main.shtml

https://people.math.sc.edu/Burkardt/f_src/test_nls/test_nls.html

Examples

```
## available test problems
nls_test_list()
```

`nls_test_problem` *Retrieve an NLS test problem*

Description

Fetches the model definition and model data required to solve a single NLS test problem with `gsl_nls` (or `nls` if the model is defined as a formula). Use `nls_test_list` to list the names of the available NLS test problems.

Usage

```
nls_test_problem(name, p = NA, n = NA)
```

Arguments

`name`	Name of the NLS test problem, as returned in the `"name"` column of `nls_test_list`.
`p`	The number of parameters in the NLS test problem constrained by the `check` condition returned by `nls_test_list`. If `NA` (default), the default number of parameters as listed by `nls_test_list` is used.
`n`	The number of residuals in the NLS test problem constrained by the `check` condition returned by `nls_test_list`. If `NA` (default), the default number of residuals as listed by `nls_test_list` is used.

Value

If the model is defined as a formula, a list of class `"nls_test_formula"` with elements:

- `data` a data.frame with `n` rows containing the data (predictor and response values) used in the regression problem.
- `fn` a formula defining the test problem model.
- `start` a named vector of length `p` with suggested starting values for the parameters.
- `target` a named vector of length `p` with the certified target values for the parameters corresponding to the *best-available* solutions.

If the model is defined as a function, a list of class `"nls_test_function"` with elements:

- `fn` a function defining the test problem model. `fn` takes a vector of parameters of length `p` as its first argument and returns a numeric vector of length `n`. `fn`
- `y` a numeric vector of length `n` containing the response values.
- `start` a numeric named vector of length `p` with suggested starting values for the parameters.
- `jac` a function defining the analytic Jacobian matrix of the model `fn`. `jac` takes a vector of parameters of length `p` as its first argument and returns an `n` by `p` dimensional matrix.
- `target` a numeric named vector of length `p` with the certified target values for the parameters, or a vector of `NA`'s if no target solution is available.

Note

For several problems the optimal least-squares objective of the target solution can be obtained at multiple different parameter locations.

References

D.M. Bates and Watts, D.G. (1988). *Nonlinear Regression Analysis and Its Applications*, Wiley, ISBN: 0471816434.

J.J. Moré, Garbow, B.S. and Hillstrom, K.E. (1981). *Testing unconstrained optimization software*, ACM Transactions on Mathematical Software, 7(1), 17-41.

See Also

nls_test_list

https://www.itl.nist.gov/div898/strd/nls/nls_main.shtml

https://people.math.sc.edu/Burkardt/f_src/test_nls/test_nls.html

Examples

```
## example regression problem
ratkowsky2 <- nls_test_problem(name = "Ratkowsky2")
with(ratkowsky2,
     gsl_nls(
       fn = fn,
       data = data,
       start = start
     )
)

## example optimization problem
rosenbrock <- nls_test_problem(name = "Rosenbrock")
with(rosenbrock,
     gsl_nls(
       fn = fn,
       y = y,
       start = start,
       jac = jac
     )
)
```

---

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 <- 25
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 <- 25
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 <- 25
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)
```

`sigma.gsl_nls` *Residual standard deviation*

Description

Returns the estimated (unweighted) residual standard deviation of a fitted "gsl_nls" object.

Usage

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

Arguments

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

Value

Numeric residual standard deviation value similar to `sigma`

See Also

`sigma`

Examples

```
## data
set.seed(1)
n <- 25
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))

sigma(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.gsl_nls" similar to summary.nls

See Also

summary.nls

Examples

```
## data
set.seed(1)
n <- 25
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 <- 25
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

anova, *3*
anova.gsl_nls, 2

class, *5*
coef, *3*
coef.gsl_nls, 3, *41*
confint, *5–7*
confint.gsl_nls, 4
confint.nls, *4, 5*
confintd, 5
confintd.gsl_nls, 6
cooks.distance, *7*
cooks.distance.gsl_nls, 7

data.frame, *33*
deriv, *6*, *14*, *25*
deviance, *8*
deviance.gsl_nls, 8
df.residual, *9*
df.residual.gsl_nls, 9

expression, *5, 6*

fitted, *10*
fitted.gsl_nls, 10
formula, *11*, *13*, *15*, *24–26*, *33, 34*
formula.gsl_nls, 11
function, *11*, *13, 14*, *24–26*, *33, 34*

gsl_nls, 12, *20, 21*, *27, 28*, *34*
gsl_nls_control, *14*, 20, *25*
gsl_nls_large, *20, 21*, 24
gsl_nls_loss, *14*, 28

hatvalues, *31*
hatvalues.gsl_nls, 31

list, *34*
logLik, *32*
logLik.gsl_nls, 32

na.exclude, *15*, *26*
na.fail, *15*, *26*
na.omit, *15*, *26*
nls, *16*, *34*
nls.control, *23*
nls_test_list, 33, *34, 35*
nls_test_problem, *33*, 34
nobs, *36*
nobs.gsl_nls, 35
numericDeriv, *6*

options, *15*, *26*

predict.gsl_nls, 36
predict.nls, *37*

residuals, *38*
residuals.gsl_nls, 38

selfStart, *13, 14*, *25*
sigma, *39*
sigma.gsl_nls, 39
summary.gsl_nls, 40
summary.nls, *40*
symnum, *40*

vcov, *41*
vcov.gsl_nls, 41