

# Package ‘gslnls’

May 1, 2024

**Type** Package

**Title** GSL Multi-Start Nonlinear Least-Squares Fitting

**Version** 1.3.2

**Date** 2024-04-30

**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 dog-leg 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). 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 (>= 2.2)

**Biarch** true

**RoxygenNote** 7.2.3

**NeedsCompilation** yes

**Author** Joris Chau [aut, cre]

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

**Repository** CRAN

**Date/Publication** 2024-05-01 14:02:06 UTC

## R topics documented:

anova.gsl_nls	2
coef.gsl_nls	3
confint.gsl_nls	4
confintd	5
confintd.gsl_nls	6
deviance.gsl_nls	7
df.residual.gsl_nls	8
fitted.gsl_nls	9
formula.gsl_nls	10
gsl_nls	11
gsl_nls_control	18
gsl_nls_large	22
hatvalues.gsl_nls	26
logLik.gsl_nls	27
nls_test_list	28
nls_test_problem	29
nobs.gsl_nls	30
predict.gsl_nls	31
residuals.gsl_nls	33
sigma.gsl_nls	34
summary.gsl_nls	35
vcov.gsl_nls	36

<b>Index</b>	<b>37</b>
--------------	-----------

---

anova.gsl_nls	<i>Anova tables</i>
---------------	---------------------

---

### 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	<i>Confidence interval for model parameters</i>
-----------------	---

---

**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**

<code>object</code>	A fitted model object.
<code>expr</code>	An expression or character vector that can be transformed to an <a href="#">expression</a> giving the function(s) of the parameters to be evaluated. Each expression should evaluate to a numeric scalar.
<code>level</code>	A numeric scalar between 0 and 1 giving the level of the derived parameter confidence intervals.
<code>...</code>	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	<i>Confidence intervals for derived parameters</i>
------------------	--

---

**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 <a href="#">expression</a> 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 <i>symbolic</i> differentiation of expr with <a href="#">deriv</a> , or "numeric" for <i>numeric</i> differentiation of expr with <a href="#">numericDeriv</a> 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"))
```

---

deviance.gsl_nls	<i>Model deviance</i>
------------------	-----------------------

---

**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	<i>Extract model fitted values</i>
----------------	------------------------------------

---

## 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

<code>object</code>	An object inheriting from class "gsl_nls".
<code>...</code>	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	<i>Extract model formula</i>
-----------------	------------------------------

---

### 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"),  
  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"),  
  control = gsl_nls_control(),  
  lower,  
  upper,  
  jac = NULL,  
  fvv = NULL,  
  trace = FALSE,  
  weights,  
  ...  
)
```

**Arguments**

fn	a nonlinear model defined either as a two-sided <a href="#">formula</a> including variables and parameters, or as a <a href="#">function</a> 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 <a href="#">formula</a> . Can also be a list or an environment, but not a matrix.
y	numeric response vector if fn is defined as a <a href="#">function</a> , 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 <a href="#">selfStart</a> model. The following choices are supported: <ul style="list-style-type: none"> <li>• 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 <i>only</i> 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.</li> <li>• 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. <math>c(0, \text{Inf})</math> or <math>c(\text{NA}, \text{NA})</math>), the multi-start algorithm considers dynamically changing starting ranges for the parameters with infinite and/or missing ranges.</li> </ul>
algorithm	character string specifying the algorithm to use. The following choices are supported: <ul style="list-style-type: none"> <li>• "lm" Levenberg-Marquardt algorithm (default).</li> <li>• "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.</li> <li>• "dogleg" Powell's dogleg algorithm.</li> <li>• "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.</li> <li>• "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.</li> </ul>
control	an optional list of control parameters to tune the least squares iterations and multistart algorithm. See <a href="#">gsl_nls_control</a> for the available control parameters and their default values.

lower	a named list or named numeric vector of parameter lower bounds. If missing (default), the parameters are unconstrained from below.
upper	a named list or named numeric vector of parameter upper bounds. If missing (default), the parameters are unconstrained from above.
jac	either NULL (default) or a <a href="#">function</a> 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 <a href="#">deriv</a> , this only works if fn is defined as a (non-selfstarting) formula. If fn is a <a href="#">selfStart</a> model, the Jacobian specified in the "gradient" attribute of the self-start model is used instead.
fvv	either NULL (default) or a <a href="#">function</a> 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 <a href="#">deriv</a> , this only works if fn is defined as a (non-selfstarting) formula. If the model <a href="#">function</a> in fn also returns a "hessian" attribute (similar to the "gradient" attribute in a <a href="#">selfStart</a> 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 <a href="#">formula</a> .
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 <a href="#">options</a> , and is <a href="#">na.fail</a> if that is unset. The 'factory-fresh' default is <a href="#">na.omit</a> . Value <a href="#">na.exclude</a> can be useful. This argument is only used if fn is defined as a <a href="#">formula</a> .
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 <a href="#">formula</a> .
...	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` 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` 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  $p < 41$  and 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 algorithm 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 or equal to `mstart_r` times `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 attained 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>

## 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)) ## 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)      ## unknown start
)

## 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 ~ SSasympt(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,
    ...
)

```

## Arguments

maxiter	positive integer, termination occurs when the number of iterations reaches <code>maxiter</code> .
scale	character, scaling method or damping strategy determining the diagonal scaling matrix $D$ . The following options are supported: <ul style="list-style-type: none"> <li>• "more" Moré rescaling (default). This method makes the problem scale-invariant and has been proven effective on a large class of problems.</li> <li>• "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 <i>parameter evaporation</i> (parameters going to infinity).</li> <li>• "marquardt" Marquardt rescaling. This method is scale-invariant, but it is generally considered inferior to both the Levenberg and Moré strategies.</li> </ul>
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 <code>gsl_nls_large</code> , the Cholesky solver ("cholesky") is always selected and this parameter is not used. For least squares problems fitted with <code>gsl_nls</code> the following choices are supported: <ul style="list-style-type: none"> <li>• "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.</li> <li>• "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.</li> <li>• "svd" SVD decomposition of the Jacobian. This method will produce the most reliable solutions for ill-conditioned Jacobians but is also the slowest.</li> </ul>
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 <code>gsl_nls_large</code> , 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 $\sqrt{\text{.Machine\$double.eps}}$ .
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 <code>gsl_nls</code> or <code>gsl_nls_large</code> .
xtol	numeric value, termination occurs when the relative change in parameters between iterations is $\leq$ xtol. A general guideline for selecting the step tolerance is to choose $\text{xtol} = 10^{-d}$ where d is the number of accurate decimal digits desired in the parameters, defaults to $\sqrt{\text{.Machine\$double.eps}}$ .
ftol	numeric value, termination occurs when the relative change in sum of squared residuals between iterations is $\leq$ ftol, defaults to $\sqrt{\text{.Machine\$double.eps}}$ .
gtol	numeric value, termination occurs when the relative size of the gradient of the sum of squared residuals is $\leq$ gtol, indicating a local minimum, defaults to $\text{.Machine\$double.eps}^{(1/3)}$
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 $\text{mstart\_n} \%/\% 10$ .
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 $(1 + \text{mstart\_tol})$ 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_minisp	positive integer, minimum number of detected stationary points before the multi-start algorithm terminates, defaults to 1.
...	any additional arguments (currently not used).

**Value**

A list with exactly twenty-one 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

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 <code>fn</code> if defined as a <a href="#">formula</a> . Can also be a list or an environment, but not a matrix.
y	numeric response vector if <code>fn</code> is defined as a <a href="#">function</a> , equal in length to the vector returned by evaluation of the function <code>fn</code> .
start	a named list or named numeric vector of starting estimates. <code>start</code> is only allowed to be missing if <code>fn</code> is a <a href="#">selfStart</a> model. If <code>fn</code> is a formula, a naive guess for <code>start</code> is tried, but this should not be relied on.
algorithm	character string specifying the algorithm to use. The following choices are supported: <ul style="list-style-type: none"> <li>• "lm" Levenberg-Marquardt algorithm (default).</li> <li>• "lmaccel" Levenberg-Marquardt algorithm with geodesic acceleration. Can be faster than "lm" but less stable. Stability is controlled by the <code>avmax</code> parameter in <code>control</code>, setting <code>avmax</code> to zero is analogous to not using geodesic acceleration.</li> <li>• "dogleg" Powell's dogleg algorithm.</li> <li>• "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.</li> <li>• "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.</li> <li>• "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.</li> </ul>
control	an optional list of control parameters to tune the least squares iterations and multistart algorithm. See <a href="#">gsl_nls_control</a> for the available control parameters and their default values.
jac	a <a href="#">function</a> returning the $n$ by $p$ dimensional Jacobian matrix of the nonlinear model <code>fn</code> , 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 <code>jac</code> is derived symbolically with <a href="#">deriv</a> , this only works if <code>fn</code> is defined as a (non-selfstarting) formula. If <code>fn</code> is a <a href="#">selfStart</a> model, the Jacobian specified in the "gradient" attribute of the self-start model is used instead.
fvv	a <a href="#">function</a> returning an $n$ dimensional vector containing the second directional derivatives of the nonlinear model <code>fn</code> , with $n$ the number of observations. This argument is only used if geodesic acceleration is enabled ( <code>algorithm = "lmaccel"</code> ). 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 <code>fvv</code> is derived symbolically with <a href="#">deriv</a> , this only works if <code>fn</code>

	is defined as a (non-selfstarting) formula. If the model <code>function</code> in <code>fn</code> also returns a "hessian" attribute (similar to the "gradient" attribute in a <code>selfStart</code> model), this Hessian matrix is used to evaluate the second directional derivatives instead.
<code>trace</code>	logical value indicating if a trace of the iteration progress should be printed. Default is <code>FALSE</code> . If <code>TRUE</code> , 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.
<code>subset</code>	an optional vector specifying a subset of observations to be used in the fitting process. This argument is only used if <code>fn</code> is defined as a <code>formula</code> .
<code>weights</code>	an optional numeric vector of (fixed) weights. When present, the objective function is weighted least squares.
<code>na.action</code>	a function which indicates what should happen when the data contain NAs. The default is set by the <code>na.action</code> setting of <code>options</code> , and is <code>na.fail</code> if that is unset. The 'factory-fresh' default is <code>na.omit</code> . Value <code>na.exclude</code> can be useful. This argument is only used if <code>fn</code> is defined as a <code>formula</code> .
<code>model</code>	a logical value. If <code>TRUE</code> , the model frame is returned as part of the object. Defaults to <code>FALSE</code> . This argument is only used if <code>fn</code> is defined as a <code>formula</code> .
<code>...</code>	additional arguments passed to the calls of <code>fn</code> , <code>jac</code> and <code>fvv</code> 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` 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` 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
)

```

---

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

[https://people.math.sc.edu/Burkardt/f\\_src/test\\_nls/test\\_nls.html](https://people.math.sc.edu/Burkardt/f_src/test_nls/test_nls.html)

**Examples**

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

---

nls_test_problem	<i>Retrieve an NLS test problem</i>
------------------	-------------------------------------

---

**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 <a href="#">nls_test_list</a> .
p	The number of parameters in the NLS test problem constrained by the check condition returned by <a href="#">nls_test_list</a> . If NA (default), the default number of parameters as listed by <a href="#">nls_test_list</a> is used.
n	The number of residuals in the NLS test problem constrained by the check condition returned by <a href="#">nls_test_list</a> . If NA (default), the default number of residuals as listed by <a href="#">nls_test_list</a> 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 Hillstom, 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://www.itl.nist.gov/div898/strd/nls/nls_main.shtml)

[https://people.math.sc.edu/Burkardt/f\\_src/test\\_nls/test\\_nls.html](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	<i>Calculate model predicted values</i>
-----------------	---

---

**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	<i>Residual standard deviation</i>
---------------	------------------------------------

---

### 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	<i>Model summary</i>
-----------------	----------------------

---

## 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 <a href="#">symnum</a> ) 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 <- 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, [2](#), [3](#)  
anova.gsl\_nls, [2](#)

class, [5](#)  
coef, [3](#)  
coef.gsl\_nls, [3](#), [36](#)  
confint, [5–7](#)  
confint.gsl\_nls, [4](#)  
confint.nls, [4](#), [5](#)  
confintd, [5](#)  
confintd.gsl\_nls, [6](#)

data.frame, [28](#)  
deriv, [6](#), [13](#), [23](#)  
deviance, [7](#)  
deviance.gsl\_nls, [7](#)  
df.residual, [8](#)  
df.residual.gsl\_nls, [8](#)

expression, [5](#), [6](#)

fitted, [9](#)  
fitted.gsl\_nls, [9](#)  
formula, [10](#), [12](#), [13](#), [22–24](#), [28](#), [29](#)  
formula.gsl\_nls, [10](#)  
function, [10](#), [12](#), [13](#), [22–24](#), [28](#), [29](#)

gsl\_nls, [11](#), [18–20](#), [25](#), [29](#)  
gsl\_nls\_control, [12](#), [18](#), [23](#)  
gsl\_nls\_large, [18–20](#), [22](#)

hatvalues, [26](#)  
hatvalues.gsl\_nls, [26](#)

list, [29](#)  
logLik, [27](#)  
logLik.gsl\_nls, [27](#)

na.exclude, [13](#), [24](#)  
na.fail, [13](#), [24](#)  
na.omit, [13](#), [24](#)

nls, [15](#), [29](#)  
nls.control, [21](#)  
nls\_test\_list, [28](#), [29](#), [30](#)  
nls\_test\_problem, [28](#), [29](#)  
nobs, [31](#)  
nobs.gsl\_nls, [30](#)  
numericDeriv, [6](#)

options, [13](#), [24](#)

predict.gsl\_nls, [31](#)  
predict.nls, [32](#)

residuals, [33](#)  
residuals.gsl\_nls, [33](#)

selfStart, [12](#), [13](#), [23](#)  
sigma, [34](#)  
sigma.gsl\_nls, [34](#)  
summary.gsl\_nls, [35](#)  
summary.nls, [35](#)  
symnum, [35](#)

vcov, [36](#)  
vcov.gsl\_nls, [36](#)