

# Package ‘nlWaldTest’

May 9, 2026

**Version** 1.1.3

**Date** 2016-03-22

**Title** Wald Test of Nonlinear Restrictions and Nonlinear CI

**Description** Wald Test for nonlinear restrictions on model parameters and confidence intervals for nonlinear functions of parameters using delta-method. Applicable after ANY model, provided parameters estimates and their covariance matrix are available.

**Author** Oleh Komashko

**Maintainer** Oleh Komashko <oleg\_komashko@ukr.net>

**Depends** R (>= 3.0.2)

**License** GPL (>= 2)

**LazyData** yes

**Repository** CRAN

**NeedsCompilation** no

**Date/Publication** 2016-03-25 00:12:23

## Contents

CESdata . . . . .	1
nlConfint . . . . .	2
nlWaldtest . . . . .	4

<b>Index</b>	<b>7</b>
--------------	----------

---

CESdata	<i>Data for testing CES production function</i>
---------	---

---

## Description

Data for estimation and testing CES production function: q-output, l-labor, k-capital

**Usage**

```
CESdata
```

**Format**

A data frame with 25 observations on the following 3 variables.

k capital

l labor

q output

**Source**

EViews, coef\_test.wf1

**Examples**

```
attach(CESdata)
```

---

nlConfint

*Confidence intervals for nonlinear functions of parameters*


---

**Description**

Computes confidence intervals for nonlinear functions of a model parameters. Delta method is used to compute standard errors. Applicable after any model provided estimates of parameters and their covariance matrix are available.

**Usage**

```
nlConfint(obj = NULL, texts, level = 0.95, coeff = NULL,
          Vcov = NULL, df2 = NULL, x = NULL)
# Standard:
# nlConfint(obj, texts) # based on z-statistics
# nlWaldtest(obj, texts, df2 = T) # based on z-statistics

# If coef(obj) and vcov(obj) are not available
# nlWaldtest(texts = funciones, coeff = vector, Vcov = matrix)
```

**Arguments**

obj	model object of any class, for which <code>vcov.class(obj)</code> and <code>coef.class(obj)</code> methods are defined. Otherwise, both <code>coeff</code> and <code>Vcov</code> should be inputted directly.
texts	function(s) of parameters, $b[i]$ , as string or vector of strings. Several functions can be inputted as a string, separated by semicolon, or as a character vector, e.g. <code>texts = "b[1]^b[2]-1; b[3]"</code> , or <code>texts = c("b[1]^b[2]-1", "b[3]")</code> ; b's should be numbered as in <code>coeff</code> vector.

level	confidence level, a number in (0, 1). Default is 0.95.
coeff	vector of parameter estimates. If missing, it is set for <code>coef(obj)</code> when available. It allows, for example, to compute CI for functions of marginal effects and elasticities provided their covariance matrix is inputted.
Vcov	covariance matrix of parameters. If missing, it is set to <code>coef(obj)</code> when available. If <code>coeff</code> and/or <code>Vcov</code> are inputted, theirs counterparts from <code>obj</code> are superseded.
df2	defines whether CI will be computed based on z (the default method) or t statistics. To compute t-based intervals, one can use <code>df2 = T</code> , provided a method for <code>df.residual</code> is available. Otherwise, one could input <code>df2 = n</code> , where n is a natural number. <code>df2</code> is the df in the t statistics. If <code>df2 = T</code> but <code>df.residuals(obj)</code> doesn't exist, z-based intervals are forced, followed by a message.
x	number, or numeric vector. Provides a way to supply cumbersome coefficients into functions, e.g. <code>texts = "b[1]^x[1] + x[2]"</code> , <code>x = c(0.1234, 5.6789)</code> to compute CI for <code>b[1]^0.1234 + 5.6789</code> .

### Details

The function should be applicable after (almost) any regression-type model, estimated using cross-section, time series, or panel data. If there are no methods for `coef(obj)` and/or `vcov(obj)`, `coeff` and `Vcov` arguments should be inputted directly. To realize the delta-method, the function first tries to compute analytical derivatives using `deriv`. If failed, it computes numerical derivatives, calling `numericDeriv`.

### Value

an `r` by 3 matrix, where `r` is the number of functions in `texts` argument. The first column is formed of values of the functions computed at parameters estimates. The two last columns are confidence bounds.

### Author(s)

Oleh Komashko

### References

Greene, W.H. (2011). *Econometric Analysis*, 7th edition. Upper Saddle River, NJ: Prentice Hall

### See Also

[nlWaldtest](#)

### Examples

```
set.seed(13)
x1<-rnorm(30);x2<-rnorm(30);x3<-rnorm(30);y<-rnorm(30)
set.seed(NULL)
lm1a<-lm(y~x1+x2+x3)
nlConfint(lm1a, c("b[2]^3+b[3]*b[1]", "b[2]"))
```

---

nlWaldtest                      *Nonlinear restriction(s) Wald test*


---

### Description

Tests restriction(s) on model parameters of the form  $R(b)=q$ , where  $R$  is vector or scalar valued (non)linear function of  $b$ , the vector of model parameters, and  $q$  is numeric vector or scalar. Delta method is used for covariance matrix. Applicable after any model provided parameters estimates and their covariance matrix are available.

### Usage

```
nlWaldtest(obj = NULL, texts, rhss = NULL, coeff = NULL,
           Vcov = NULL, df2 = NULL, x = NULL)
# Standard:
# nlWaldtest(obj, texts) # Chi square test
# nlWaldtest(obj, texts, df2 = T) # F test

# Force different covariance matrix:
# nlWaldtest(obj, texts, Vcov = vcovHC(obj))

# If coef(obj) and vcov(obj) are not available
# nlWaldtest(texts = restrictions, coeff = vector, Vcov = matrix)

# Backward compatibility:
# nlWaldtest(obj, texts, rhss)
```

### Arguments

obj	model object of any class, for which <code>vcov.class(obj)</code> and <code>coef.class(obj)</code> methods are defined. If missing, both <code>coeff</code> and <code>Vcov</code> should be inputted.
texts	left-side(s) of normalized restriction(s), $R(b)$ , as string or vector of strings. Multiple restrictions can be inputted as a character vector or as a character, separated by semicolon. Right-hand sides can be included either separated by "=", or subtracted, e.g. <code>texts = "b[1]^b[2] = 1; b[3] = 2"</code> , or, the same, <code>texts = c("a[1]^a[2] - 1", "a[3] = 2")</code> ; $b$ 's should be numbered as in <code>coeff</code> vector.
rhss	right-side(s) of normalized restriction(s) as number or vector. Retained mostly for backward compatibility. Set to zero(s), if missing.
coeff	vector of parameter estimates. If missing, it is set to <code>coef(obj)</code> when available. It allows, for example, to test hypotheses in terms of marginal effects and elasticities provided their covariance matrix is inputted.
Vcov	covariance matrix of parameters. If missing, it is set to <code>vcov(obj)</code> when available. If <code>coeff</code> and/or <code>Vcov</code> are inputted, theirs counterparts from <code>obj</code> are superseded.

df2	defines the type of the test. By default, Chi square test is performed. To perform F test one can use <code>df2 = T</code> , if a method for <code>df.residual</code> is available. Otherwise, one could input <code>df2 = n</code> , where <code>n</code> is a natural number. <code>df2</code> is the denominator <code>df</code> in the F statistics. If <code>df2 = T</code> but <code>df.residuals(obj)</code> doesn't exist, Chi square test is forced, followed by a message.
x	number, or numeric vector. Provides a way to supply cumbersome coefficients into restrictions, e.g. <code>texts = "b[1]^x[1] = x[2]"</code> , <code>x = c(0.1234, 5.6789)</code> to test $b[1]^{0.1234} = 5.6789$ . Instead of "b", one can use any valid variable name excluding "x". The "cumbersome" coefficients must be named only as <code>x[i]</code> .

### Details

The test should be applicable after (almost) any regression-type model, estimated using cross-section, time series, or panel data. If there are no methods for `coef(obj)` and/or `vcov(obj)`, `coef` and `Vcov` arguments should be inputted directly. To realize the delta-method, the function first tries to compute analytical derivatives using `deriv`. If failed, it computes numerical derivatives, calling `numericDeriv`.

### Value

an object of "htest" class.

### Author(s)

Oleh Komashko

### References

Greene, W.H. (2011). *Econometric Analysis*, 7th edition. Upper Saddle River, NJ: Prentice Hall

### See Also

[nlConfint](#)

### Examples

```
set.seed(13)
x1<-rnorm(30);x2<-rnorm(30);x3<-rnorm(30);y<-rnorm(30)
set.seed(NULL)
lm1<-lm(y~x1+x2+x3)
nlConfint(lm1, "b[2]^3+b[3]*b[1];b[2]")
nlWaldtest(lm1,"a[2]^3+a[3]*a[1] = x[1]; a[2]", x = -0.07)
nlWaldtest(lm1,c("b[2]^3+b[3]*b[1]+0.07", "b[2]"))

# Reproduce example in EVIEWS 8 Users Guide II, pp. 149-151.

## Not run:
require(nlme)
nl1<-nlsl(log(q)~c1+c2*log(c3*(k^c4)+(1-c3)*(1^c4)),
```

```
data=CESdata,start=list(c1=-2.6,c2=1.8,c3=0.0001,c4=-6),
nls.control(maxiter = 100, tol = 1e-05,minFactor = 1/2^15)
nIWaldtest(nl1,"b[2]-1/b[4]",0)
nIWaldtest(nl1,"b[2]*b[4]",1)

## End(Not run)
```

# Index

CESdata, 1

deriv, 3, 5

df.residual, 3, 5

nlConfint, 2, 5

nlWaldtest, 3, 4

numericDeriv, 3, 5