

# Package 'nlsic'

May 16, 2025

**Title** Non Linear Least Squares with Inequality Constraints

**Version** 1.1.0

**Maintainer** Serguei Sokol <sokol@insa-toulouse.fr>

**Description** We solve non linear least squares problems with optional equality and/or inequality constraints. Non linear iterations are globalized with back-tracking method. Linear problems are solved by dense QR decomposition from 'LAPACK' which can limit the size of treated problems. On the other side, we avoid condition number degradation which happens in classical quadratic programming approach. Inequality constraints treatment on each non linear iteration is based on 'NNLS' method (by Lawson and Hanson). We provide an original function 'lsi\_In' for solving linear least squares problem with inequality constraints in least norm sens. Thus if Jacobian of the problem is rank deficient a solution still can be provided. However, truncation errors are probable in this case. Equality constraints are treated by using a basis of Null-space. User defined function calculating residuals must return a list having residual vector (not their squared sum) and Jacobian. If Jacobian is not in the returned list, package 'numDeriv' is used to calculate finite difference version of Jacobian. The 'NLSIC' method was first published in Sokol et al. (2012) <doi:10.1093/bioinformatics/btr716>.

**License** GPL-2

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**Depends** nnls, dotty

**Suggests** numDeriv, RUnit, limSolve

**URL** <https://github.com/MathsCell/nlsic>

**BugReports** <https://github.com/MathsCell/nlsic/issues>

**NeedsCompilation** no

**Author** Serguei Sokol [aut, cre] (ORCID:  
<<https://orcid.org/0000-0002-5674-3327>>)

**Repository** CRAN

**Date/Publication** 2025-05-16 15:20:02 UTC

## Contents

equa2vecmat . . . . .	2
join . . . . .	3
ldp . . . . .	3
lsi . . . . .	4
lsie_ln . . . . .	5
lsi_ln . . . . .	6
lsi_reg . . . . .	6
ls_0 . . . . .	7
ls_ln . . . . .	8
ls_ln_svd . . . . .	8
nlsic . . . . .	9
Nulla . . . . .	12
pnull . . . . .	13
tls . . . . .	14
uplo2uco . . . . .	14
<b>Index</b>	<b>16</b>

---

equa2vecmat	<i>Parse linear equations/inequalities</i>
-------------	--

---

### Description

parse a text vector of linear equations and produce a corresponding matrix and right hand side vector

### Usage

```
equa2vecmat(nm_par, linear, sep = "=")
```

### Arguments

nm_par	a string vector of variable names. It will be used in the symbolic derivation.
linear	string vector of linear equations like "a+2*c+3*b = 0"
sep	separator of two parts of equations. Use for example ">=" for linear inequalities

### Value

an augmented matrix. Its first column is the rhs vector. Other columns are named by nm\_par. If the vector linear is NULL or its content is empty a NULL is returned

### Examples

```
equa2vecmat(c("a", "b", "c"), "a+2*c+3*b = 0", "=")
```

---

join *Join elements into a string*

---

**Description**

convert elements of vector  $v$  (and all following arguments) in strings and join them using  $sep$  as separator.

**Usage**

```
join(sep, v, ...)
```

**Arguments**

$sep$	A string used as a separator
$v$	A string vector to be joined
$\dots$	other variables to be converted to strings and joined

**Value**

A joined string

**Examples**

```
join(" ", c("Hello", "World"))
```

---

ldp *Least Distance Problem*

---

**Description**

Solve least distance programming: find  $x$  satisfying  $u \leq x \leq co$  and s.t.  $\min(\|x\|)$  by passing to nnls (non negative least square) problem.

**Usage**

```
ldp(u, co, rcond = 1e+10)
```

**Arguments**

$u$	a dense matrix of inequality constraints
$co$	a right hand side vector of inequality constraints
$rcond$	maximal condition number for determining rank deficient matrix

**Value**

solution vector or NULL if constraints are unfeasible

lsl

*Linear Least Squares with Inequality constraints (LSI)***Description**

solve linear least square problem ( $\min \|A\%x-b\|$ ) with *inequality constraints*  $u\%x \geq co$

**Usage**

```
lsl(a, b, u = NULL, co = NULL, rcond = 1e+10, mnorm = NULL, x0 = NULL)
```

**Arguments**

a	dense matrix A or its QR decomposition
b	right hand side vector. Rows containing NA are dropped.
u	dense matrix of inequality constraints
co	right hand side vector of inequality constraints
rcond	maximal condition number for determining rank deficient matrix
mnorm	dummy parameter
x0	dummy parameter

**Details**

Method:

1. reduce the problem to ldp ( $\min(xat*xa) \Rightarrow$  least distance programming)
2. solve ldp
3. change back to x  
If b is all NA, then a vector of NA is returned.

mnorm, and x0 are dummy parameters which are here to make lsl() compatible with lsl\_ln() argument list

**Value**

solution vector whose attribute 'mes' may contain a message about possible numerical problems

**See Also**

[lsl\\_ln](#), [ldp](#), [base::qr](#)

---

lsie_ln	<i>Linear Least Squares problem with inequality and equality constraints, least norm solution</i>
---------	---

---

**Description**

solve linear least square problem ( $\min \|A\%x-b\|$ ) with *inequality constraints*  $u\%x \geq co$  and equality constraints  $e\%x = ce$  Method: reduce the pb to lsi\_ln on the null-space of e

**Usage**

```
lsie_ln(
  a,
  b,
  u = NULL,
  co = NULL,
  e = NULL,
  ce = NULL,
  rcond = 1e+10,
  mnorm = NULL,
  x0 = NULL
)
```

**Arguments**

a	dense matrix A or its QR decomposition
b	right hand side vector
u	dense matrix of inequality constraints
co	right hand side vector of inequality constraints
e	dense matrix of equality constraints
ce	right hand side vector of equality constraints
rcond	maximal condition number for determining rank deficient matrix
mnorm	matrix to be used to minimize the norm $\ mnorm\%x\ $ in case of under-determined system. If mnorm is NULL (default), it is
x0	vector from which minimal norm should be searched when required

**Value**

solution vector whose attribute 'mes' may contain a message about possible numerical problems

**See Also**

[lsi\\_ln](#)

---

lsi\_ln *Linear Least Squares with Inequality constraints, least norm solution*

---

### Description

solve linear least square problem  $\min_x \|A*x-b\|$  with inequality constraints  $u*x \geq co$  If A is rank deficient, least norm solution  $\|mnorm*(x-x0)\|$  is used. If the parameter mnorm is NULL, it is treated as an identity matrix. If the vector x0 is NULL, it is treated as 0 vector.

### Usage

```
lsi_ln(a, b, u = NULL, co = NULL, rcond = 1e+10, mnorm = NULL, x0 = NULL)
```

### Arguments

a	dense matrix A or its QR decomposition
b	right hand side vector
u	dense matrix of inequality constraints
co	right hand side vector of inequality constraints
rcond	maximal condition number for determining rank deficient matrix
mnorm	norm matrix (can be dense or sparse) for which <code>%%</code> operation with a dense vector is defined
x0	optional vector from which a least norm distance is searched for

### Value

solution vector whose attribute 'mes' may contain a message about possible numerical problems

### See Also

[lsi](#), [ldp](#), [base::qr](#)

---

lsi\_reg *Regularized Linear Least Squares*

---

### Description

solve linear least square problem  $(\min_x \|a*x-b\|)$  with inequality constraints  $ux \geq co$  If a is rank deficient, regularization term  $\lambda^2 * \|mnorm*(x-x0)\|^2$  is added to  $\|a*x-b\|^2$ .

### Usage

```
lsi_reg(a, b, u = NULL, co = NULL, rcond = 1e+10, mnorm = NULL, x0 = NULL)
```

**Arguments**

a	dense matrix A or its QR decomposition
b	right hand side vector
u	dense matrix of inequality constraints
co	right hand side vector of inequality constraints
rcond	used for calculating $\lambda = d[1] / \sqrt{\text{rcond}}$ where $d[1]$ is maximal singular value of a
mnorm	norm matrix (can be dense or sparse) for which <code>%*%</code> operation with a dense vector is defined
x0	optional vector from which a least norm distance is searched for

**Details**

The rank of a is estimated as number of singular values above  $d[1] * 1.e-10$  where  $d[1]$  is the highest singular value. The scalar lambda is an positive number and is calculated as  $d[1] / \sqrt{\text{rcond}}$  ('rcond' parameter is preserved for compatibility with others `lsi_...()` functions). At return, lambda can be found in attributes of the returned vector x. NB. lambda is set to NA

- if  $\text{rank}(a) == 0$  or a is of full rank
- or if there is no inequality. If the matrix mnorm is NULL, it is supposed to be an identity matrix. If the vector x0 is NULL, it is treated as 0 vector.

**Value**

solution vector whose attribute 'mes' may contain a message about possible numerical problems and 'lambda' is regularization parameter used in solution.

**See Also**

[lsi\\_ln](#)

---

ls\_0

*Particular solution of rank-deficient least squares*

---

**Description**

Particular solution of rank-deficient least squares

**Usage**

`ls_0(aq, b)`

**Arguments**

aq	QR decomposition of a matrix
b	right-hand-side of the LS system

**Value**

solution vector where free variables are set to 0.

---

ls_ln	<i>Linear Least Squares, least norm solution</i>
-------	--

---

**Description**

Linear Least Squares, least norm solution

**Usage**

```
ls_ln(a, b, rcond = 1e+10, mnorm = NULL, x0 = NULL)
```

**Arguments**

a	matrix or its QR decomposition
b	vector of right hand side
rcond	maximal condition number for rank definition
mnorm	norm matrix (can be dense or sparse) for which %% operation with a dense vector is defined
x0	optional vector from which a least norm distance is searched for

**Value**

solution vector

---

ls_ln_svd	<i>Linear Least Squares, least norm solution (by svd)</i>
-----------	---

---

**Description**

Least squares  $a\%* \%x \approx b$  of least norm  $\|x\|$  by using `svd(a)`

**Usage**

```
ls_ln_svd(a, b, rcond = 1e+10)
```

**Arguments**

a	dense matrix
b	right hand side vector
rcond	maximal condition number for determining rank deficient matrix

**Value**

solution vector



**Description**

Solve non linear least squares problem  $\min_{\text{par}} ||r(\text{par}, \dots)||$  with optional inequality constraints  $u \cdot \text{par} \geq \text{co}$  and optional equality constraints  $e \cdot \text{par} = \text{eco}$

**Usage**

```
nlsic(
  par,
  r,
  u = NULL,
  co = NULL,
  control = list(),
  e = NULL,
  eco = NULL,
  flsi = lsi,
  ...
)
```

**Arguments**

par	initial values for parameter vector. It can be in non feasible domain, i.e. for which $u \cdot \text{par} \geq \text{co}$ is not always respected.
r	a function calculating residual vector by a call to $r(\text{par}, \text{cjac}, \dots)$ where par is a current parameter vector, and cjac is set to TRUE if Jacobian is required or FALSE if not. A call to $r()$ must return a list having a field "res" containing the residual vector and optionally a field "jacobian" when cjac is set to TRUE.
u	constraint matrix in $u \cdot \text{par} \geq \text{co}$
co	constraint right hand side vector
control	a list of control parameters ('=' indicates default values): <ul style="list-style-type: none"> <li>• <math>\text{errx}=1.e-7</math> error on l2 norm of the iteration step <math>\sqrt{\text{pt} \cdot \text{p}}</math>.</li> <li>• <math>\text{maxit}=100</math> maximum of newton iterations</li> <li>• <math>\text{btstart}=1</math> starting value for backtracking coefficient</li> <li>• <math>\text{bfrac}=0.5</math> (0;1) by this value we diminish the step till tight up to the quadratic model of norm reduction in backtrack (bt) iterations</li> <li>• <math>\text{btdesc}=0.1</math> (0;1) how good we have to tight up to the quadratic model 0 - we are very relaxe, 1 - we are very tight (may need many bt iterations)</li> <li>• <math>\text{btmaxit}=15</math> maximum of backtrack iterations</li> <li>• <math>\text{btkmin}=1.e-7</math> a floor value for backtracking fractioning</li> <li>• <math>\text{trace}=0</math> no information is printed during iterations (1 - print is active)</li> </ul>

- `rcond=1.e10` maximal condition number in QR decomposition starting from which a matrix is considered as numerically rank deficient. The inverse of this number is also used as a measure of very small number.
  - `ci` = list of options relative to confidence interval estimation
    - `p=0.95` p-value of confidence interval. If you need a plain sd value, set p-value to 0.6826895
    - `report=T` report (or not and hence calculate or not) confidence interval information (in the field `hci`, as 'half confidence interval' width)
  - `history=TRUE` report or not the convergence history
  - `adaptbt=FALSE` use or not adaptive backtracking
  - `mnorm=NULL` a norm matrix for a case `sln==TRUE` (cf next parameter)
  - `sln=FALSE` use or not (default) least norm of the solution (instead of least norm of the increment)
  - `maxstep=NULL` a maximal norm for increment step (if not NULL), must be positive
  - `monotone=FALSE` should or not the cost decrease be monotone. If TRUE, then at first non decrease of the cost, the iterations are stopped with a warning message.
- `e` linear equality constraint matrix in `e**%par = eco` (dense)
- `eco` right hand side vector in equality constraints
- `flsi` function solving linear least squares problem. For a custom function, see interfaces in `lsi` or `lsi_ln` help pages.
- `...` parameters passed through to `r()`

### Details

Solving method consist in sequential LSI problems globalized by backtracking technique. If `e`, `eco` are not NULL, reduce jacobian to basis of `e`'s kernel before `lsi()` call.

NB. If the function `r()` returns a list having a field "jacobian" it is supposed to be equal to the jacobian `dr/dpar`. If not, numerical derivation `numDeriv::jacobian()` is automatically used for its calculation.

NB2. `nlsic()` does not call `stop()` on possible errors. Instead, 'error' field is set to 1 in the returned result. This is done to allow a user to examine the current state of data and possibly take another path then to merely stop the program. So, a user must allways check this field at return from `nlsic()`.

NB3. User should test that field 'mes' is not NULL even when error is 0. It may contain a warning message.

### Value

a list with following components (some components can be absent depending on 'control' parameter)

- 'par' estimated values of `par` as vector
- 'paro' the same but in original structure (i.e. matrix if `par` is a matrix)
- 'lastp' the last LSI solution during non linear iterations
- 'hci' vector of half-width confidence intervals for `par`

- 'ci\_p' p-value for which CI was calculated
- 'ci\_fdeg' freedom degree used for CI calculation
- 'sd\_res' standard deviation of residuals
- 'covpar' covariance matrix for par
- 'laststep' the last increment after possible back-tracking iterations
- 'normp' the norm of lastp
- 'res' the last residual vector
- 'prevres' residual vector from previous non linear iteration
- 'jacobian' the last used jacobian
- 'retres' last returned result of r() call
- 'it' non linear iteration number where solution was obtained
- 'btit' back-tracking iteration number done during the last non linear iteration
- 'history' list with convergence history information
- 'error' error code: 0 - normal end, 1 - some error occurred, see message in 'mes'
- 'mes' textual message explaining what problem was in case of error

### See Also

[lsi](#), [lsi\\_ln](#), [uplo2uco](#)

### Examples

```
# solve min_{a,b} ||exp(a*x+b)-meas||, a,b>=1
a_true=1; b_true=2; x=0:5
# simulation function
sim=function(par, x) exp(par[["a"]]*x+par[["b"]])
# residual function to be passed to nlsic()
resid=function(par, cjac, ...) {
  dots=list(...)
  s=sim(par, dots$x)
  result=list(res=s-dots$meas)
  if (cjac) {
    result$jacobian=cbind(a=s*dots$x, b=s)
  }
  result
}
# simulated noised measurements for true parameters
set.seed(7) # for reproducible results
meas=sim(c(a=a_true, b=b_true), x)+rnorm(x)
# starting values for par
par=c(a=0, b=0)
# prepare constraints
uco=uplo2uco(par, lower=c(a=1, b=1))
# main call: solve the problem
fit=nlsic(par, resid, uco$u, uco$c, control=list(trace=1), x=x, meas=meas)
if (fit$error == 1) {
  stop(fit$mes)
}
```

```

} else {
  print(fit$par) # a=1.001590, b=1.991194
  if (! is.null(fit$mes)) {
    warning(fit$mes)
  }
}

```

---

Nulla

*Null-space basis*


---

### Description

use Lapack for null space basis (derived from MASS::Null)

### Usage

```
Nulla(M, rcond = 1e+10)
```

### Arguments

M	matrix such that $t(M) \%*\% B = 0$ where B is a basis of $t(M)$ 's kernel (aka Null-space)
rcond	maximal condition number for rank definition

### Value

numeric matrix whose columns are basis vectors. Its attribute 'qr' contains QR decomposition of M.

### See Also

[MASS::Null](#)

### Examples

```
Nulla(1:3)
```

---

pnull

*Particular least-squares solution and Null-space basis*


---

### Description

use Lapack to find a particular solution  $x_p$  of under-determined least squares system  $Ax=b$  and build null space basis  $B$  of  $A$  (derived from `MASS::Null`). In such a way, a general solution is given by  $x=x_p+Bz$  where  $z$  is an arbitrary vector of size  $n_{col}(A)-rank(A)$ .

### Usage

```
pnull(A, b = NULL, qrat = NULL, rcond = 1e+10, keepqr = FALSE)
```

### Arguments

<code>A</code>	matrix (or its QR decomposition) such that $A\%*\%B=0$ where $B$ is a basis of $\text{Kern}(A)$ (aka Null-space).
<code>b</code>	is the right hand side of the least squares system $Ax=b$ .
<code>qrat</code>	is QR decomposition of $t(A)$ .
<code>rcond</code>	maximal condition number for rank definition
<code>keepqr</code>	if TRUE store <code>qr</code> and <code>qrat</code> as attribute of <code>B</code>

### Value

alist with `xp` and `B`, particular solution and numeric matrix whose columns are basis vectors. If requested, attributes `'qr'` and `'qrat'` of `B` contain QR decomposition of  $A$  and  $t(A)$  respectively.

### See Also

[MASS::Null Nulla](#)

### Examples

```
A=diag(nrow=3L)[1:2,,drop=FALSE]
b=A%*(1:3)
res=pnull(A, b)
stopifnot(all.equal(res$xp, c(1:2,0)))
stopifnot(all.equal(c(res$B), c(0,0,1)))
```

---

tls	<i>Total Least Squares <math>a\%*x \sim b</math></i>
-----	--

---

**Description**

Total Least Squares  $a\%*x \sim b$

**Usage**

```
tls(a, b)
```

**Arguments**

a	matrix
b	right hand side vector

**Value**

solution vector

---

uplo2uco	<i>Transform box-type inequalities into matrix and vector form</i>
----------	--

---

**Description**

Transform a set of inequalities  $\text{param}["\text{name}"] \geq \text{lower}["\text{name}"]$  and  $\text{param}["\text{name}"] \leq \text{upper}["\text{name}"]$  into a list with matrix  $u$  and vector  $co$  such that  $u\%*\%param \geq co$ . In addition to box inequalities above, user can provide linear inequalities in a form like  $"a+2*c+3*b \geq 0"$  where 'a', 'b' and 'c' must be names of param components. Numeric and symbolic coefficients and right hand sides are allowed in these expressions. However, symbols must be defined at the moment of calling `uplo2uco()` so that expressions containing such symbols could be `eval()`-ed to numerical values. All inequalities must be written with '>=' sign (not with '<=', '>', ...).

**Usage**

```
uplo2uco(param, upper = NULL, lower = NULL, linear = NULL)
```

**Arguments**

param	a named vector whose names will be used for parsing inequalities
upper	a named numeric vector of upper limits
lower	a named numeric vector of lower limits
linear	a string vector of linear inequalities

**Value**

a list with numeric matrix 'u' and vector 'co' such that  $u \cdot \text{param} - \text{co} \geq 0$

**See Also**

[equa2vecmat](#) for parsing linear expressions

# Index

base::qr, [4](#), [6](#)

equa2vecmat, [2](#), [15](#)

join, [3](#)

ldp, [3](#), [4](#), [6](#)

ls\_0, [7](#)

ls\_ln, [8](#)

ls\_ln\_svd, [8](#)

lsi, [4](#), [6](#), [11](#)

lsi\_ln, [4](#), [5](#), [6](#), [7](#), [11](#)

lsi\_reg, [6](#)

lsie\_ln, [5](#)

MASS::Null, [12](#), [13](#)

nlsic, [9](#)

Nulla, [12](#), [13](#)

pnull, [13](#)

tls, [14](#)

uplo2uco, [11](#), [14](#)