Package: lsei (via r-universe)

September 8, 2024

Title Solving Least Squares or Quadratic Programming Problems under Equality/Inequality Constraints

Version 1.3-0

Date 2020-09-07

Description It contains functions that solve least squares linear regression problems under linear equality/inequality constraints. Functions for solving quadratic programming problems are also available, which transform such problems into least squares ones first. It is developed based on the 'Fortran' program of Lawson and Hanson (1974, 1995), which is public domain and available at <http://www.netlib.org/lawson-hanson/>.

Encoding UTF-8

License GPL (>= 2)

URL https://www.stat.auckland.ac.nz/~yongwang/

RoxygenNote 7.1.1

NeedsCompilation yes

Author Yong Wang [aut, cre], Charles L. Lawson [aut], Richard J. Hanson [aut]

Maintainer Yong Wang <yongwang@auckland.ac.nz>

Date/Publication 2020-09-17 08:20:03 UTC

Repository https://yong3738.r-universe.dev

RemoteUrl https://github.com/cran/lsei

RemoteRef HEAD

RemoteSha 9d6fd620cdf85cb18b83682c202817ec7ba82c37

Contents

`hfti` *Least Squares Solution using Householder Transformation*

Description

Solves the least squares problem using Householder forward triangulation with column interchanges. It is an R interface to the HFTI function that is described in Lawson and Hanson (1974, 1995). Its Fortran implementation is public domain and is available at `http://www.netlib.org/lawson-hanson/`.

Usage

```
hfti(a, b, tol = 1e-07)
```

Arguments

`a`	Design matrix.
`b`	Response vector or matrix.
`tol`	Tolerance for determining the pseudorank.

Details

Given matrix `a` and vector `b`, `hfti` solves the least squares problem:

$$\text{minimize} \quad ||ax - b||.$$

Value

`b`	first `krank` elements contains the solution
`krank`	psuedo-rank
`rnorm`	Euclidean norm of the residual vector.

Author(s)

Yong Wang <yongwang@auckland.ac.nz>

References

Lawson and Hanson (1974, 1995). Solving least squares problems. Englewood Cliffs, N.J., Prentice-Hall.

See Also

lsei, nnls.

Examples

```
a = matrix(rnorm(10*4), nrow=10)
b = a %*% c(0,1,-1,1) + rnorm(10)
hfti(a, b)
```

---

indx *Index-finding in a Sorted Vector*

Description

For each of given values, `indx` finds the index of the value in a vector sorted in ascending order that the given value is barely greater than or equal to.

Usage

```
indx(x, v)
```

Arguments

x	vector of numeric values, the indices of which are to be found.
v	vector of numeric values sorted in ascending order.

Details

For each x[i], the function returns integer j such that

$$v_j \leq x_i < v_{j+1}$$

where $v_0 = -\infty \mathrm{and} v_{n+1} = \infty$.

Value

Returns a vector of integers, that are indices of x-values in vector v.

Author(s)

Yong Wang <yongwang@auckland.ac.nz>

Examples

```
indx(0:6,c(1:5,5))
indx(sort(rnorm(5)),-2:2)
```

`lsei` *Least Squares and Quadratic Programming under Equality and Inequality Constraints*

Description

These functions can be used for solving least squares or quadratic programming problems under general equality and/or inequality constraints.

Usage

```
lsei(a, b, c=NULL, d=NULL, e=NULL, f=NULL, lower=-Inf, upper=Inf)
lsi(a, b, e=NULL, f=NULL, lower=-Inf, upper=Inf)
ldp(e, f)
qp(q, p, c=NULL, d=NULL, e=NULL, f=NULL, lower=-Inf, upper=Inf, tol=1e-15)
```

Arguments

`a`	Design matrix.
`b`	Response vector.
`c`	Matrix of numeric coefficients on the left-hand sides of equality constraints. If it is NULL, `c` and `d` are ignored.
`d`	Vector of numeric values on the right-hand sides of equality constraints.
`e`	Matrix of numeric coefficients on the left-hand sides of inequality constraints. If it is NULL, `e` and `f` are ignored.
`f`	Vector of numeric values on the right-hand sides of inequality constraints.
`lower`, `upper`	Bounds on the solutions, as a way to specify such simple inequality constraints.
`q`	Matrix of numeric values for the quadratic term of a quadratic programming problem.
`p`	Vector of numeric values for the linear term of a quadratic programming problem.
`tol`	Tolerance, for calculating pseudo-rank in `qp`.

Details

The `lsei` function solves a least squares problem under both equality and inequality constraints. It is an implementation of the LSEI algorithm described in Lawson and Hanson (1974, 1995).

The `lsi` function solves a least squares problem under inequality constraints. It is an implementation of the LSI algorithm described in Lawson and Hanson (1974, 1995).

The `ldp` function solves a least distance programming problem under inequality constraints. It is an R wrapper of the LDP function which is in Fortran, as described in Lawson and Hanson (1974, 1995).

The qp function solves a quadratic programming problem, by transforming the problem into a least squares one under the same equality and inequality constraints, which is then solved by function `lsei`.

The NNLS and LDP Fortran implementations used internally is downloaded from `http://www.netlib.org/lawson-hanson/`.

Given matrices `a`, `c` and `e`, and vectors `b`, `d` and `f`, function `lsei` solves the least squares problem under both equality and inequality constraints:

$$\text{minimize } \; ||ax - b||,$$
$$\text{subject to } \; cx = d, ex \geq f.$$

Function `lsi` solves the least squares problem under inequality constraints:

$$\text{minimize } \; ||ax - b||,$$
$$\text{subject to } \; ex \geq f.$$

Function `ldp` solves the least distance programming problem under inequality constraints:

$$\text{minimize } \; ||x||,$$
$$\text{subject to } \; ex \geq f.$$

Function qp solves the quadratic programming problem:

$$\text{minimize } \; \frac{1}{2} x^T q x + p^T x,$$
$$\text{subject to } \; cx = d, ex \geq f.$$

Value

A vector of the solution values

Author(s)

Yong Wang <yongwang@auckland.ac.nz>

References

Lawson and Hanson (1974, 1995). Solving least squares problems. Englewood Cliffs, N.J., Prentice-Hall.

See Also

`nnls`,`hfti`.

Examples

```
beta = c(rnorm(2), 1)
beta[beta<0] = 0
beta = beta / sum(beta)
a = matrix(rnorm(18), ncol=3)
b = a %*% beta + rnorm(3,sd=.1)
c = t(rep(1, 3))
d = 1
e = diag(1,3)
f = rep(0,3)
lsei(a, b)                        # under no constraint
lsei(a, b, c, d)                  # under eq. constraints
lsei(a, b, e=e, f=f)              # under ineq. constraints
lsei(a, b, c, d, e, f)            # under eq. and ineq. constraints
lsei(a, b, rep(1,3), 1, lower=0)  # same solution
q = crossprod(a)
p = -drop(crossprod(b, a))
qp(q, p, rep(1,3), 1, lower=0)    # same solution

## Example from Lawson and Hanson (1974), p.140
a = cbind(c(.4302,.6246), c(.3516,.3384))
b = c(.6593, .9666)
c = c(.4087, .1593)
d = .1376
lsei(a, b, c, d)   # Solution: -1.177499  3.884770

## Example from Lawson and Hanson (1974), p.170
a = cbind(c(.25,.5,.5,.8),rep(1,4))
b = c(.5,.6,.7,1.2)
e = cbind(c(1,0,-1),c(0,1,-1))
f = c(0,0,-1)
lsi(a, b, e, f)      # Solution: 0.6213152 0.3786848

## Example from Lawson and Hanson (1974), p.171:
e = cbind(c(-.207,-.392,.599), c(2.558, -1.351, -1.206))
f = c(-1.3,-.084,.384)
ldp(e, f)            # Solution: 0.1268538 -0.2554018
```

---

matMaxs *Row or Column Maximum Values of a Matrix*

Description

Finds either row or column maximum values of a matrix.

Usage

```
matMaxs(x, dim = 1)
```

Arguments

`x`	numeric matrix.
`dim`	`=1`, for row maximum values; `=2`, for column maximum values.

Details

Matrix `x` may contain `Inf` or `-Inf`, but not `NA` or `NaN`.

Value

Returns a numeric vector with row or column maximum values.

The function is very much the same as using `apply(x, 1, max)` or `apply(x, 2, max)`, but faster.

Author(s)

Yong Wang <yongwang@auckland.ac.nz>

Examples

```
x = cbind(c(1:4,Inf), 5:1)
matMaxs(x)
matMaxs(x, 2)
```

---

nnls — *Least Squares and Quadratic Programming under Nonnegativity Constraints*

Description

These functions are particularly useful for solving least squares or quadratic programming problems when some or all of the solution values are subject to nonnegativity constraint. One may further restrict the NN-restricted coefficients to have a fixed positive sum.

Usage

```
nnls(a, b)
pnnls(a, b, k=0, sum=NULL)
pnnqp(q, p, k=0, sum=NULL, tol=1e-20)
```

Arguments

`a`	Design matrix.
`b`	Response vector.
`k`	Integer, meaning that the first `k` coefficients are not NN-restricted.

sum	= NULL, if NN-restricted coefficients are not further restricted to have a fixed sum; = a positive value, if NN-restricted coefficients are further restricted to have a fixed positive sum.
q	Positive semidefinite matrix of numeric values for the quadratic term of a quadratic programming problem.
p	Vector of numeric values for the linear term of a quadratic programming problem.
tol	Tolerance used for calculating pseudo-rank of q.

Details

Function nnls solves the least squares problem under nonnegativity (NN) constraints. It is an R interface to the NNLS function that is described in Lawson and Hanson (1974, 1995). Its Fortran implementation is public domain and available at http://www.netlib.org/lawson-hanson/ (with slight modifications by Yong Wang for compatibility with the lastest Fortran compiler.)

Given matrix a and vector b, nnls solves the nonnegativity least squares problem:

$$\mathrm{minimize} \ \ ||ax - b||,$$

$$\mathrm{subject\ to} \ \ x \geq 0.$$

Function pnnls also solves the above nonnegativity least squares problem when k=0, but it may also leave the first k coefficients unrestricted. The output value of k can be smaller than the input one, if a has linearly dependent columns. If sum is a positive value, pnnls solves the problem by further restricting that the NN-restricted coefficients must sum to the given value.

Function pnnqp solves the quadratic programming problem

$$\mathrm{minimize} \ \ \frac{1}{2} x^T q x + p^T x,$$

when only some or all coefficients are restricted by nonnegativity. The quadratic programming problem is solved by transforming the problem into a least squares one under the same constraints, which is then solved by function pnnls. Arguments k and sum have the same meanings as for pnnls.

Functions nnls, pnnls and pnnqp are able to return any zero-valued solution as 0 exactly. This differs from functions lsei and qp, which may produce very small values for exactly 0s, thanks to numerical errors.

Value

x	Solution
r	The upper-triangular matrix Q*a, pivoted by variables in the order of index, when sum=NULL. If sum > 0, r is for the transformed a.
b	The vector Q*b, pivoted by variables in the order of index, when sum=NULL. If sum > 0, b is for the transformed b.

index	Indices of the columns of r; those unrestricted and in the positive set are first given, and then those in the zero set.
rnorm	Euclidean norm of the residual vector.
mode	= 1, successful computation; = 2, bad dimensions of the problem; = 3, iteration count exceeded (more than 3 times the number of variables iterations).
k	Number of the first few coefficients that are truly not NN-restricted.

Author(s)

Yong Wang <yongwang@auckland.ac.nz>

References

Lawson and Hanson (1974, 1995). Solving Least Squares Problems. Englewood Cliffs, N.J., Prentice-Hall.

Dax (1990). The smallest point of a polytope. Journal of Optimization Theory and Applications, 64, pp. 429-432.

Wang (2010). Fisher scoring: An interpolation family and its Monte Carlo implementations. Computational Statistics and Data Analysis, 54, pp. 1744-1755.

See Also

lsei, hfti.

Examples

```
a = matrix(rnorm(40), nrow=10)
b = drop(a %*% c(0,1,-1,1)) + rnorm(10)
nnls(a, b)$x                     # constraint x >= 0
pnnls(a, b, k=0)$x               # same as nnls(a, b)
pnnls(a, b, k=2)$x               # first two coeffs are not NN-constrained
pnnls(a, b, k=2, sum=1)$x        # NN-constrained coeffs must sum to 1
pnnls(a, b, k=2, sum=2)$x        # NN-constrained coeffs must sum to 2
q = crossprod(a)
p = -drop(crossprod(b, a))
pnnqp(q, p, k=2, sum=2)$x        # same solution

pnnls(a, b, sum=1)$x             # zeros found exactly
pnnqp(q, p, sum=1)$x             # zeros found exactly
lsei(a, b, rep(1,4), 1, lower=0) # zeros not so exact
```

Index