Package ‘lsei’

October 7, 2015

Title  Solving Least Squares Problems under Equality/Inequality
       Constraints

Version  1.1-1

Date  2015-10-04

Description  It contains functions that solve least squares linear
              regression problems under linear equality/inequality
              constraints. It is developed based on the 'Fortran' program of
              Lawson and Hanson (1974, 1995), which is public domain and

License  GPL (>= 2)

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

NeedsCompilation  yes

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

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

Repository  CRAN

Date/Publication  2015-10-07 09:13:29

R topics documented:

  hfti ................................................................. 2
  lsei ............................................................... 3
  nnls ............................................................... 5

Index  8
**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](http://www.netlib.org/lawson-hanson).

**Usage**

`hfti(a, b, tol=1e-7)`

**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:

\[
\text{minimize } \|ax - b\|.
\]

**Value**

- `b`: first `k` elements contains the solution
- `k`: pseudorank
- `rnorm`: Euclidean norm of the residual vector.

**Author(s)**

Yong Wang <yongwang@auckland.ac.nz>

**References**


**See Also**

`lsei`, `nnls`.
Examples

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

---

### lsei

**Least Squares Solution under Equality and Inequality Constraints**

#### Description

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

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

The `ldp` function solves a least distance programming problem under inequality constraints and implements the LDP algorithm 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 Fortran implementation used internally is downloaded from [http://www.netlib.org/lawson-hanson](http://www.netlib.org/lawson-hanson).

#### Usage

```r
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, tol=1e-15)
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.
- **q**: Matrix of numeric coefficients for the quadratic term of a quadratic programming problem.
- **p**: Vector of numeric values for the linear term of a quadratic programming problem.
lower, upper  Bounds on the solutions, as a way to specify such simple inequality constraints.
tol  Tolerance, for checking compatibility of inequalities in lsi() and for calculating pseudo-rank in qp().

Details

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:

\[
\text{minimize } \frac{1}{2} x^T qx + p^T x,
\]
\[
\text{subject to } cx = d, ex \geq f.
\]

Value

A vector of the solution values

Author(s)

Yong Wang <yongwang@auckland.ac.nz>

References


See Also

nnls, hfti.
nnls

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.170
a = cbind(c(.25,.5,.5,.8),rep(1,4))
b = c(.5,.5,.7,1.2)
e = cbind(c(1,0,-1),c(0,1,-1))
f = c(0,0,-1)
lsei(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

nnls

Least Squares and Quadratic Programming under Nonnegativity Constraints

Description

The nnls function 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.

The pnnls function solves the least squares problem when coefficients are partly subject to nonnegativity constraints. It also allows the NN-restricted coefficients to be further restricted to have a fixed positive sum.

The pnnqp function solves the quadratic programming problem when solution values are partly subject to nonnegativity constraints. It also allows the NN-restricted coefficients to be further restricted to have a fixed positive sum.

These functions are particularly useful for finding zeros exactly.
Usage

nnls(a, b)

pnnls(a, b, k=0, sum=NULL)

pnnqp(q, p, k=0, sum=NULL, tol=1e-15)

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 for calculating pseudo-rank of q.

Details

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

\[
\text{minimize } ||ax - b||,
\]

subject to \( x \geq 0 \).

Function *pnnls* can also solve 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 different from the input, 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

\[
\text{minimize } \frac{1}{2} x^T q x + p^T x,
\]

when only some or all coefficients are restricted by nonnegativity. The output value of k can be different from the input, if q 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. 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*.

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

- **x**  
  Solution

- **rnorm**  
  Euclidean norm of the residual vector.

- **index**  
  Indices of the columns; those in the positive set are first given, and then those in the zero set.

- **mode**  
  = 1, successful computation;  
  = 2, bad dimensions of the problem;  
  = 3, iteration count exceeded (more than 3 times the number of variables iterations).

- **r**  
  The upper-triangular matrix $Q^a$, pivoted by variables in the order of `index`.

- **b**  
  The vector $Q^b$, pivoted by variables in the order of `index`.

Author(s)

Yong Wang <yongwang@auckland.ac.nz>

References


See Also

- `lsei, hfti`.

Examples

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

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

*Topic algebra
  hfti, 2
  lsei, 3
  nnls, 5
*Topic array
  hfti, 2
  lsei, 3
  nnls, 5

hfti, 2, 4, 7
ldp (lsei), 3
lsei, 2, 3, 7
lsi (lsei), 3
nnls, 2, 4, 5
pnnls (nnls), 5
pnnqp (nnls), 5
qp (lsei), 3