Package: nnls (via r-universe)

September 5, 2024

Type Package

Title The Lawson-Hanson Algorithm for Non-Negative Least Squares (NNLS)

Version 1.5

Author Katharine M. Mullen and Ivo H. M. van Stokkum

Maintainer Katharine Mullen <mullenkate@gmail.com>

Suggests bvls, quadprog

Description An R interface to the Lawson-Hanson implementation of an algorithm for non-negative least squares (NNLS). Also allows the combination of non-negative and non-positive constraints.

License GPL (>= 2)

Date/Publication 2023-09-11 07:20:07 UTC

NeedsCompilation yes

Repository https://k-m-m.r-universe.dev

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

RemoteRef HEAD

RemoteSha 75aac85d5e03f8373d164860facc097d92c0b46f

Contents

`nnls-package` *The Lawson-Hanson NNLS implementation of non-negative least squares*

Description

An R interface to the Lawson-Hanson NNLS implementation of an algorithm for non-negative linear least squares that solves the least squares problem $\min \parallel Ax = b \parallel_2$ with the constraint $x \geq 0$ where $x \in R^n, b \in R^m$ and A is an $m \times n$ matrix. Also allows the combination of non-negative and non-positive constraints on x.

References

Lawson CL, Hanson RJ (1974). Solving Least Squares Problems. Prentice Hall, Englewood Cliffs, NJ.

Lawson CL, Hanson RJ (1995). Solving Least Squares Problems. Classics in Applied Mathematics. SIAM, Philadelphia.

See Also

nnls, nnnpls, the method `"L-BFGS-B"` for optim, solve.QP, bvls

`nnls` *The Lawson-Hanson NNLS implemention of non-negative least squares*

Description

An R interface to the Lawson-Hanson NNLS implementation of an algorithm for non-negative linear least squares that solves $\min \parallel Ax - b \parallel_2$ with the constraint $x \geq 0$, where $x \in R^n, b \in R^m$ and A is an $m \times n$ matrix.

Usage

```
nnls(A, b)
```

Arguments

`A`	numeric matrix with `m` rows and `n` columns
`b`	numeric vector of length `m`

Value

nnls returns an object of class "nnls".

The generic accessor functions coefficients, fitted.values, deviance and residuals extract various useful features of the value returned by nnls.

An object of class "nnls" is a list containing the following components:

x	the parameter estimates.
deviance	the residual sum-of-squares.
residuals	the residuals, that is response minus fitted values.
fitted	the fitted values.
mode	a character vector containing a message regarding why termination occured.
passive	vector of the indices of x that are not bound at zero.
bound	vector of the indices of x that are bound at zero.
nsetp	the number of elements of x that are not bound at zero.

Source

This is an R interface to the Fortran77 code distributed with the book referenced below by Lawson CL, Hanson RJ (1995), obtained from Netlib (file 'lawson-hanson/all'), with a trivial modification to return the variable NSETP.

References

Lawson CL, Hanson RJ (1974). Solving Least Squares Problems. Prentice Hall, Englewood Cliffs, NJ.

Lawson CL, Hanson RJ (1995). Solving Least Squares Problems. Classics in Applied Mathematics. SIAM, Philadelphia.

See Also

nnnpls, the method "L-BFGS-B" for optim, solve.QP, bvls

Examples

```
## simulate a matrix A
## with 3 columns, each containing an exponential decay
t <- seq(0, 2, by = .04)
k <- c(.5, .6, 1)
A <- matrix(nrow = 51, ncol = 3)
Acolfunc <- function(k, t) exp(-k*t)
for(i in 1:3) A[,i] <- Acolfunc(k[i],t)

## simulate a matrix X
## with 3 columns, each containing a Gaussian shape
## the Gaussian shapes are non-negative
X <- matrix(nrow = 51, ncol = 3)
wavenum <- seq(18000,28000, by=200)
```

```
location <- c(25000, 22000, 20000)
delta <- c(3000,3000,3000)
Xcolfunc <- function(wavenum, location, delta)
  exp( - log(2) * (2 * (wavenum - location)/delta)^2)
for(i in 1:3) X[,i] <- Xcolfunc(wavenum, location[i], delta[i])

## set seed for reproducibility
set.seed(3300)

## simulated data is the product of A and X with some
## spherical Gaussian noise added
matdat <- A %*% t(X) + .005 * rnorm(nrow(A) * nrow(X))

## estimate the rows of X using NNLS criteria
nnls_sol <- function(matdat, A) {
  X <- matrix(0, nrow = 51, ncol = 3)
  for(i in 1:ncol(matdat))
     X[i,] <- coef(nnls(A,matdat[,i]))
  X
}
X_nnls <- nnls_sol(matdat,A)

matplot(X_nnls,type="b",pch=20)
abline(0,0, col=grey(.6))

## Not run:
## can solve the same problem with L-BFGS-B algorithm
## but need starting values for x
bfgs_sol <- function(matdat, A) {
  startval <- rep(0, ncol(A))
  fn1 <- function(par1, b, A) sum( ( b - A %*% par1)^2)
  X <- matrix(0, nrow = 51, ncol = 3)
  for(i in 1:ncol(matdat))
    X[i,] <-  optim(startval, fn = fn1, b=matdat[,i], A=A,
                   lower = rep(0,3), method="L-BFGS-B")$par
    X
}
X_bfgs <- bfgs_sol(matdat,A)

## the RMS deviation under NNLS is less than under L-BFGS-B
sqrt(sum((X - X_nnls)^2)) < sqrt(sum((X - X_bfgs)^2))

## and L-BFGS-B is much slower
system.time(nnls_sol(matdat,A))
system.time(bfgs_sol(matdat,A))

## can also solve the same problem by reformulating it as a
## quadratic program (this requires the quadprog package; if you
## have quadprog installed, uncomment lines below starting with
## only 1 "#" )

# library(quadprog)
```

```
# quadprog_sol <- function(matdat, A) {
#  X <- matrix(0, nrow = 51, ncol = 3)
#  bvec <- rep(0, ncol(A))
#  Dmat <- crossprod(A,A)
#  Amat <- diag(ncol(A))
#  for(i in 1:ncol(matdat)) {
#    dvec <- crossprod(A,matdat[,i])
#    X[i,] <- solve.QP(dvec = dvec, bvec = bvec, Dmat=Dmat,
#                      Amat=Amat)$solution
#  }
#  X
# }
# X_quadprog <- quadprog_sol(matdat,A)

## the RMS deviation under NNLS is about the same as under quadprog
# sqrt(sum((X - X_nnls)^2))
# sqrt(sum((X - X_quadprog)^2))

## and quadprog requires about the same amount of time
# system.time(nnls_sol(matdat,A))
# system.time(quadprog_sol(matdat,A))


## End(Not run)
```

---

nnnpls *An implementation of least squares with non-negative and non-positive constraints*

Description

An implementation of an algorithm for linear least squares problems with non-negative and non-positive constraints based on the Lawson-Hanson NNLS algorithm. Solves $\min \parallel Ax - b \parallel_2$ with the constraint $x_i \geq 0$ if $con_i \geq 0$ and $x_i \leq 0$ otherwise, where $x, con \in R^n, b \in R^m$, and A is an $m \times n$ matrix.

Usage

```
nnnpls(A, b, con)
```

Arguments

A	numeric matrix with `m` rows and `n` columns
b	numeric vector of length `m`
con	numeric vector of length `m` where element `i` is negative if and only if element `i` of the solution vector `x` should be constrained to non-positive, as opposed to non-negative, values.

Value

nnnpls returns an object of class "nnnpls".

The generic accessor functions coefficients, fitted.values, deviance and residuals extract various useful features of the value returned by nnnpls.

An object of class "nnnpls" is a list containing the following components:

x	the parameter estimates.
deviance	the residual sum-of-squares.
residuals	the residuals, that is response minus fitted values.
fitted	the fitted values.
mode	a character vector containing a message regarding why termination occured.
passive	vector of the indices of x that are not bound at zero.
bound	vector of the indices of x that are bound at zero.
nsetp	the number of elements of x that are not bound at zero.

Source

This is an R interface to Fortran77 code distributed with the book referenced below by Lawson CL, Hanson RJ (1995), obtained from Netlib (file 'lawson-hanson/all'), with some trivial modifications to allow for the combination of constraints to non-negative and non-positive values, and to return the variable NSETP.

References

Lawson CL, Hanson RJ (1974). Solving Least Squares Problems. Prentice Hall, Englewood Cliffs, NJ.

Lawson CL, Hanson RJ (1995). Solving Least Squares Problems. Classics in Applied Mathematics. SIAM, Philadelphia.

See Also

nnls, the method "L-BFGS-B" for optim, solve.QP, bvls

Examples

```
## simulate a matrix A
## with 3 columns, each containing an exponential decay
t <- seq(0, 2, by = .04)
k <- c(.5, .6, 1)
A <- matrix(nrow = 51, ncol = 3)
Acolfunc <- function(k, t) exp(-k*t)
for(i in 1:3) A[,i] <- Acolfunc(k[i],t)

## simulate a matrix X
## with 3 columns, each containing a Gaussian shape
## 2 of the Gaussian shapes are non-negative and 1 is non-positive
X <- matrix(nrow = 51, ncol = 3)
```

```
wavenum <- seq(18000,28000, by=200)
location <- c(25000, 22000, 20000)
delta <- c(3000,3000,3000)
Xcolfunc <- function(wavenum, location, delta)
  exp( - log(2) * (2 * (wavenum - location)/delta)^2)
for(i in 1:3) X[,i] <- Xcolfunc(wavenum, location[i], delta[i])
X[,2] <- -X[,2]

## set seed for reproducibility
set.seed(3300)

## simulated data is the product of A and X with some
## spherical Gaussian noise added
matdat <- A %*% t(X) + .005 * rnorm(nrow(A) * nrow(X))

## estimate the rows of X using NNNPLS criteria
nnnpls_sol <- function(matdat, A) {
  X <- matrix(0, nrow = 51, ncol = 3)
  for(i in 1:ncol(matdat))
     X[i,] <- coef(nnnpls(A,matdat[,i],con=c(1,-1,1)))
  X
}
X_nnnpls <- nnnpls_sol(matdat,A)

## Not run:
## can solve the same problem with L-BFGS-B algorithm
## but need starting values for x and
## impose a very low/high bound where none is desired
bfgs_sol <- function(matdat, A) {
  startval <- rep(0, ncol(A))
  fn1 <- function(par1, b, A) sum( ( b - A %*% par1)^2)
  X <- matrix(0, nrow = 51, ncol = 3)
  for(i in 1:ncol(matdat))
    X[i,] <-  optim(startval, fn = fn1, b=matdat[,i], A=A,
              lower=rep(0, -1000, 0), upper=c(1000,0,1000),
              method="L-BFGS-B")$par
    X
}
X_bfgs <- bfgs_sol(matdat,A)

## the RMS deviation under NNNPLS is less than under L-BFGS-B
sqrt(sum((X - X_nnnpls)^2)) < sqrt(sum((X - X_bfgs)^2))

## and L-BFGS-B is much slower
system.time(nnnpls_sol(matdat,A))
system.time(bfgs_sol(matdat,A))

## can also solve the same problem by reformulating it as a
## quadratic program (this requires the quadprog package; if you
## have quadprog installed, uncomment lines below starting with
## only 1 "#" )

# library(quadprog)
```

```
# quadprog_sol <- function(matdat, A) {
#  X <- matrix(0, nrow = 51, ncol = 3)
#  bvec <- rep(0, ncol(A))
#  Dmat <- crossprod(A,A)
#  Amat <- diag(c(1,-1,1))
#  for(i in 1:ncol(matdat)) {
#    dvec <- crossprod(A,matdat[,i])
#    X[i,] <- solve.QP(dvec = dvec, bvec = bvec, Dmat=Dmat,
#                      Amat=Amat)$solution
#  }
#  X
# }
# X_quadprog <- quadprog_sol(matdat,A)

## the RMS deviation under NNNPLS is about the same as under quadprog
# sqrt(sum((X - X_nnnpls)^2))
# sqrt(sum((X - X_quadprog)^2))

## and quadprog requires about the same amount of time
# system.time(nnnpls_sol(matdat,A))
# system.time(quadprog_sol(matdat,A))

## End(Not run)
```

Index