Package ‘bigalgebra’

April 8, 2022

Type Package

Title 'BLAS' and 'LAPACK' Routines for Native R Matrices and 'big.matrix' Objects

Version 1.1.0

Date 2022-04-07

Depends bigmemory (>= 4.0.0)

Imports methods

LinkingTo bigmemory, BH, Rcpp

Author Frederic Bertrand [cre, ctb] (<https://orcid.org/0000-0002-0837-8281>),
Michael J. Kane [aut],
Bryan Lewis [aut],
John W. Emerson [aut]

Maintainer Frederic Bertrand <frederic.bertrand@utt.fr>

Description Provides arithmetic functions for R matrix and 'big.matrix' objects as well as functions for QR factorization, Cholesky factorization, General eigenvalue, and Singular value decomposition (SVD). A method matrix multiplication and an arithmetic method -for matrix addition, matrix difference- allows for mixed type operation -a matrix class object and a big.matrix class object- and pure type operation for two big.matrix class objects.

License LGPL-3 | Apache License 2.0

Encoding UTF-8

LazyLoad yes

NeedsCompilation yes

RoxygenNote 7.1.1

URL https://fbertran.github.io/bigalgebra/,
https://github.com/fbertran/bigalgebra/

BugReports https://github.com/fbertran/bigalgebra/issues/

Repository CRAN

Date/Publication 2022-04-08 06:42:37 UTC

R topics documented:

`bigalgebra-package` *Arithmetic routines for native R matrices and big.matrix objects.*

Description

This package provides arithmetic functions for native R matrices and `big.matrix` objects.

Details

This package provides arithmetic functions for native R matrices and `big.matrix` objects.

The package defines a number of global options that begin with `bigalgebra`. They include:

Option	Default value
`bigalgebra.temp_pattern`	"matrix_"
`bigalgebra.tempdir`	`tempdir`
`bigalgebra.mixed_arithmetic_returns_R_matrix`	TRUE
`bigalgebra.DEBUG`	FALSE

The `bigalgebra.tempdir` option must be a function that returns a temporary directory path used to big matrix results of BLAS and LAPACK operations. The deault value is simply the default R `tempdir` function.

The `bigalgebra.temp_pattern` is a name prefix for file names of generated big matrix objects output as a result of BLAS and LAPACK operations.

The `bigalgebra.mixed_arithmetic_returns_R_matrix` option determines whether arithmetic operations involving an R matrix or vector and a big.matrix matrix or vector return a big matrix (when the option is `FALSE`), or return a normal R matrix (`TRUE`).

The package is built, by default, with R's native BLAS libraries, which use 32-bit signed integer indexing. The default build is limited to vectors of at most 2**31 - 1 entries and matrices with at most 2**31 - 1 rows and 2**31 - 1 columns (note that standard R matrices are limtied to 2**31 - 1 total entries).

The package includes a reference BLAS implementation that supports 64-bit integer indexing, relaxing the limitation on vector lengths and matrix row and column limits. Installation of this package with the 64-bit reference BLAS implementation may be performed from the command-line install:

`REFBLAS=1 R CMD INSTALL bigalgebra`

where "`bigalgebra`" is the source package (for example, `bigalgebra_0.8.4.tar.gz`).

The package may also be build with user-supplied external BLAS and LAPACK libraries, in either 32- or 64-bit varieties. This is an advanced topic that requires additional Makevars modification, and may include adjustment of the low-level calling syntax depending on the library used.

Feel free to contact us for help installing and running the package.

Author(s)

Frédéric Bertrand, Michael J. Kane, Bryan Lewis, John W. Emerson

Maintainer: Frédéric Bertrand <frederic.bertrand@utt.fr>

References

https://www.netlib.org/blas/

https://www.netlib.org/lapack/

See Also

bigmemory, big.matrix

Examples

```
# Testing the development of the user-friendly operators:
# if you have any problems, please email us!  - Jay & Mike 4/29/2010

library("bigmemory")
A <- big.matrix(5,4, type="double", init=0,
                dimnames=list(NULL, c("alpha", "beta")))
B <- big.matrix(4,4, type="double", init=0,
                dimnames=list(NULL, c("alpha", "beta")))

C <- A
D <- A[]

print(C - D)       # Compare the results (subtraction of an R matrix from a
                   # big.matrix)

# The next example illustrates mixing R and big.matrix objects. It returns by
# default (see # options("bigalgebra.mixed_arithmetic_returns_R_matrix")
D <- matrix(rnorm(16),4)
E <- A
```

balgebra-methods *Class* "big.matrix" *arithmetic methods*

Description

Arithmetic operations for big.matrices

Methods

%*% `signature{x="big.matrix",y="big.matrix"}`: ...

%*% `signature{x="matrix",y="big.matrix"}`: ...

%*% `signature{x="big.matrix",y="matrix"}`: ...

Arith `signature{x="big.matrix",y="big.matrix"}`: ...

Arith `signature{x="big.matrix",y="matrix"}`: ...

Arith `signature{x="matrix",y="big.matrix"}`: ...

Arith `signature{x="big.matrix",y="numeric"}`: ...

Arith `signature{x="numeric",y="big.matrix"}`: ...

Notes

Miscellaneous arithmetic methods for matrices and big.matrices. See also `options("bigalgebra.mixed_arithmetic_retu`

Author(s)

B. W. Lewis <blewis@illposed.net>

daxpy *BLAS daxpy functionality*

Description

This function implements the function Y := A * X + Y where X and Y may be either native double-precision valued R matrices or numeric vectors, or double-precision valued `big.matrix` objects, and A is a scalar.

Usage

```
daxpy(A = 1, X, Y)
```

Arguments

`A`	Optional numeric scalar value to scale the matrix `X` by, with a default value of 1.
`X`	Requried to be either a native R `matrix` or numeric vector, or a `big.matrix` object
`Y`	Optional native R `matrix` or numeric vector, or a `big.matrix` object

Details

At least one of either X or Y must be a big.matrix. All values must be of type double (the only type presently supported by the bigalgebra package).

This function is rarely necessary to use directly since the bigalgebra package defines standard arithmetic operations and scalar multiplication. It is more efficient to use daxpy directly when both scaling and matrix addition are required, in which case both operations are performed in one step.

Value

The output value depends on the classes of input values X and Y and on the value of the global option bigalgebra.mixed_arithmetic_returns_R_matrix.

If X and Y are both big matrices, or Y is missing, options("bigalgebra.mixed_arithmetic_returns_R_matrix") is FALSE, then a big.matrix is returned. The returned big.matrix is backed by a temporary file mapping that will be deleted when the returned result is garbage collected by R (see the examples).

Otherwise, a standard R matrix is returned. The dimensional shape of the output is taken from X. If input X is dimensionless (that is, lacks a dimension attribute), then the output is a column vector.

Author(s)

Michael J. Kane

References

https://www.netlib.org/blas/daxpy.f

See Also

bigmemory

Examples

```
require(bigmemory)
A = matrix(1, nrow=3, ncol=2)
B <- big.matrix(3, 2, type="double", init=0,
               dimnames=list(NULL, c("alpha", "beta")), shared=FALSE)
C = B + B   # C is a new big matrix
D = A + B   # D defaults to a regular R matrix, to change this, set the option:
# options(bigalgebra.mixed_arithmetic_returns_R_matrix=FALSE)
E = daxpy(A=1.0, X=B, Y=B)  # Same kind of result as C
print(C[])
print(D)
print(E[])

# The C and E big.matrix file backings will be deleted when garbage collected:
# (We enable debugging to see this explicitly)
options(bigalgebra.DEBUG=TRUE)
rm(C,E)
gc()
```

dcopy *Copy a vector.*

Description

Copy double precision DX to double precision DY. For I = 0 to N-1, copy DX(LX+I*INCX) to DY(LY+I*INCY), where LX = 1 if INCX .GE. 0, else LX = 1+(1-N)*INCX, and LY is defined in a similar way using INCY.

Usage

```
dcopy(N = NULL, X, INCX = 1, Y, INCY = 1)
```

Arguments

N	number of elements in input vector(s)
X	double precision vector with N elements
INCX	storage spacing between elements of DX
Y	double precision vector with N elements
INCY	storage spacing between elements of DY

Value

DY copy of vector DX (unchanged if N .LE. 0)

References

C. L. Lawson, R. J. Hanson, D. R. Kincaid and F. T. Krogh, Basic linear algebra subprograms for Fortran usage, Algorithm No. 539, Transactions on Mathematical Software 5, 3 (September 1979), pp. 308-323.

Examples

```
## Not run:
set.seed(4669)
A = big.matrix(3, 2, type="double", init=1, dimnames=list(NULL,
c("alpha", "beta")), shared=FALSE)
B = big.matrix(3, 2, type="double", init=0, dimnames=list(NULL,
c("alpha", "beta")), shared=FALSE)

dcopy(X=A,Y=B)
A[,]-B[,]

# The big.matrix file backings will be deleted when garbage collected.
rm(A,B)
gc()

## End(Not run)
```

dgeev *DGEEV computes eigenvalues and eigenvectors.*

Description

DGEEV computes the eigenvalues and, optionally, the left and/or right eigenvectors for GE matrices.

DGEEV computes for an N-by-N real nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors. The right eigenvector v(j) of A satisfies A * v(j) = lambda(j) * v(j) where lambda(j) is its eigenvalue. The left eigenvector u(j) of A satisfies u(j)**H * A = lambda(j) * u(j)**H where u(j)**H denotes the conjugate-transpose of u(j).

The computed eigenvectors are normalized to have Euclidean norm equal to 1 and largest component real.

Usage

```
dgeev(
  JOBVL = "V",
  JOBVR = "V",
  N = NULL,
  A,
  LDA = NULL,
  WR,
  WI,
  VL,
  LDVL = NULL,
  VR = NULL,
  LDVR = NULL,
  WORK = NULL,
  LWORK = NULL
)
```

Arguments

JOBVL	a character. **= 'N':** left eigenvectors of A are not computed; **= 'V':** left eigenvectors of A are computed.
JOBVR	a character. **= 'N':** right eigenvectors of A are not computed; **= 'V':** right eigenvectors of A are computed.
N	an integer. The order of the matrix A. N >= 0.
A	a matrix of dimension (LDA,N), the N-by-N matrix A.
LDA	an integer. The leading dimension of the matrix A. LDA >= max(1,N).

WR
: a vector of dimension (N). WR contain the real part of the computed eigenvalues. Complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part first.

WI
: a vector of dimension (N). WI contain the imaginary part of the computed eigenvalues. Complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part first.

VL
: a matrx of dimension (LDVL,N)

  If JOBVL = 'V', the left eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues.

  If JOBVL = 'N', VL is not referenced.

  If the j-th eigenvalue is real, then u(j) = VL(:,j), the j-th column of VL.

  If the j-th and (j+1)-st eigenvalues form a complex conjugate pair, then u(j) = VL(:,j) + i*VL(:,j+1) and u(j+1) = VL(:,j) - i*VL(:,j+1).

LDVL
: an integer. The leading dimension of the array VL. LDVL >= 1; if JOBVL = 'V', LDVL >= N.

VR
: a matrix of dimension (LDVR,N).

  If JOBVR = 'V', the right eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues.

  If JOBVR = 'N', VR is not referenced.

  If the j-th eigenvalue is real, then v(j) = VR(:,j), the j-th column of VR.

  If the j-th and (j+1)-st eigenvalues form a complex conjugate pair, then v(j) = VR(:,j) + i*VR(:,j+1) and v(j+1) = VR(:,j) - i*VR(:,j+1).

LDVR
: an integer. The leading dimension of the array VR. LDVR >= 1; if JOBVR = 'V', LDVR >= N.

WORK
: a matrix of dimension (MAX(1,LWORK))

LWORK
: an integer. The dimension of the array WORK.LWORK >= max(1,3*N), and if JOBVL = 'V' or JOBVR = 'V', LWORK >= 4*N. For good performance, LWORK must generally be larger. If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

Value

WR, WI, VR, VL and Work. On exit, A has been overwritten.

Examples

```
## Not run:
set.seed(4669)
A = matrix(rnorm(16),4)
WR= matrix(0,nrow=4,ncol=1)
WI= matrix(0,nrow=4,ncol=1)
VL = matrix(0,ncol=4,nrow=4)
eigen(A)
dgeev(A=A,WR=WR,WI=WI,VL=VL)
```

```
VL
WR
WI

rm(A,WR,WI,VL)

A = as.big.matrix(matrix(rnorm(16),4))
WR= matrix(0,nrow=4,ncol=1)
WI= matrix(0,nrow=4,ncol=1)
VL = as.big.matrix(matrix(0,ncol=4,nrow=4))
eigen(A[,])
dgeev(A=A,WR=WR,WI=WI,VL=VL)
VL[,]
WR[,]
WI[,]

rm(A,WR,WI,VL)
gc()

## End(Not run)
```

---

dgemm *Matrix Multiply*

Description

This is function provides dgemm functionality, which DGEMM performs one of the matrix-matrix operations. C := ALPHA * op(A) * op(B) + BETA * C.

Usage

```
dgemm(
  TRANSA = "N",
  TRANSB = "N",
  M = NULL,
  N = NULL,
  K = NULL,
  ALPHA = 1,
  A,
  LDA = NULL,
  B,
  LDB = NULL,
  BETA = 0,
  C,
  LDC = NULL,
  COFF = 0
)
```

Arguments

TRANSA	a character. TRANSA specifies the form of op(A) to be used in the matrix multiplication as follows: **TRANSA =** 'N' or 'n', op(A) = A. **TRANSA =** 'T' or 't', op(A) = A**T. **TRANSA =** 'C' or 'c', op(A) = A**T.
TRANSB	a character. TRANSB specifies the form of op(B) to be used in the matrix multiplication as follows: #' **TRANSA =** 'N' or 'n', op(B) = B. **TRANSA =** 'T' or 't', op(B) = B**T. **TRANSA =** 'C' or 'c', op(B) = B**T.
M	an integer. M specifies the number of rows of the matrix op(A) and of the matrix C. M must be at least zero.
N	an integer. N specifies the number of columns of the matrix op(B) and of the matrix C. N must be at least zero.
K	an integer. K specifies the number of columns of the matrix op(A) and the number of rows of the matrix op(B). K must be at least zero.
ALPHA	a real number. Specifies the scalar alpha.
A	a matrix of dimension (LDA, ka), where ka is k when TRANSA = 'N' or 'n', and is m otherwise. Before entry with TRANSA = 'N' or 'n', the leading m by k part of the array A must contain the matrix A, otherwise the leading k by m part of the array A must contain the matrix A.
LDA	an integer.
B	a matrix of dimension (LDB, kb), where kb is n when TRANSB = 'N' or 'n', and is k otherwise. Before entry with TRANSB = 'N' or 'n', the leading k by n part of the array B must contain the matrix B, otherwise the leading n by k part of the array B must contain the matrix B.
LDB	an integer.
BETA	a real number. Specifies the scalar beta
C	a matrix of dimension (LDC, N). Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the array C is overwritten by the m by n matrix (alpha*op(A)*op(B) + beta*C).
LDC	an integer.
COFF	offset for C.

Value

Update C with the result.

Examples

```
require(bigmemory)
A = as.big.matrix(matrix(1, nrow=3, ncol=2))
B <- big.matrix(2, 3, type="double", init=-1,
                dimnames=list(NULL, c("alpha", "beta")), shared=FALSE)
C = big.matrix(3, 3, type="double", init=1,
               dimnames=list(NULL, c("alpha", "beta", "gamma")), shared=FALSE)
2*A[,]%*%B[,]+0.5*C[,]
E = dgemm(ALPHA=2.0, A=A, B=B, BETA=0.5, C=C)
E[,] # Same result

# The big.matrix file backings will be deleted when garbage collected.
rm(A,B,C,E)
gc()
```

`dgeqrf` *QR factorization*

Description

DGEQRF computes a QR factorization of a real M-by-N matrix A: A = Q * R.

Usage

```
dgeqrf(
  M = NULL,
  N = NULL,
  A,
  LDA = NULL,
  TAU = NULL,
  WORK = NULL,
  LWORK = NULL
)
```

Arguments

`M`	an integer. The number of rows of the matrix A. M >= 0.
`N`	an integer. The number of columns of the matrix A. N >= 0.
`A`	the M-by-N big matrix A.
`LDA`	an integer. The leading dimension of the array A. LDA >= max(1,M).
`TAU`	a min(M,N) matrix. The scalar factors of the elementary reflectors.
`WORK`	a (MAX(1,LWORK)) matrix. On exit, if INFO = 0, WORK(1) returns the optimal LWORK.
`LWORK`	an integer. The dimension of th array WORK.

Value

M-by-N big matrix A. The elements on and above the diagonal of the array contain the min(M,N)-by-N upper trapezoidal matrix R (R is upper triangular if m >= n); the elements below the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of min(m,n) elementary reflectors.

Examples

```
## Not run:
#' hilbert <- function(n) { i <- 1:n; 1 / outer(i - 1, i, "+") }
h9 <- hilbert(9); h9
qr(h9)$rank           #--> only 7
qrh9 <- qr(h9, tol = 1e-10)
qrh9$rank
C <- as.big.matrix(h9)
dgeqrf(A=C)

# The big.matrix file backings will be deleted when garbage collected.
rm(C)
gc()

## End(Not run)
```

---

dgesdd — *DGESDD computes the singular value decomposition (SVD) of a real matrix.*

Description

DGESDD computes the singular value decomposition (SVD) of a real M-by-N matrix A, optionally computing the left and right singular vectors. If singular vectors are desired, it uses a divide-and-conquer algorithm.

The SVD is written

A = U * SIGMA * transpose(V)

where SIGMA is an M-by-N matrix which is zero except for its min(m,n) diagonal elements, U is an M-by-M orthogonal matrix, and V is an N-by-N orthogonal matrix. The diagonal elements of SIGMA are the singular values of A; they are real and non-negative, and are returned in descending order. The first min(m,n) columns of U and V are the left and right singular vectors of A.

Note that the routine returns VT = V**T, not V.

Usage

```
dgesdd(
  JOBZ = "A",
  M = NULL,
  N = NULL,
  A,
```

```
    LDA = NULL,
    S,
    U,
    LDU = NULL,
    VT,
    LDVT = NULL,
    WORK = NULL,
    LWORK = NULL
)
```

Arguments

`JOBZ` a character. Specifies options for computing all or part of the matrix U:

- **= 'A':** all M columns of U and all N rows of V**T are returned in the arrays U and VT;
- **= 'S':** the first min(M,N) columns of U and the first min(M,N) rows of V**T are returned in the arrays U and VT;
- **= 'O':** If M >= N, the first N columns of U are overwritten on the array A and all rows of V**T are returned in the array VT; otherwise, all columns of U are returned in the array U and the first M rows of V**T are overwritten in the array A;
- **= 'N':** no columns of U or rows of V**T are computed.

`M` an integer. The number of rows of the input matrix A. M >= 0.

`N` an integer. The number of columns of the input matrix A. N >= 0.

`A` the M-by-N matrix A.

`LDA` an integer. The leading dimension of the matrix A. LDA >= max(1,M).

`S` a matrix of dimension (min(M,N)). The singular values of A, sorted so that S(i) >= S(i+1).

`U` U is a matrx of dimension (LDU,UCOL)

- **UCOL = M if** JOBZ = 'A' or JOBZ = 'O' and M < N; UCOL = min(M,N) if JOBZ = 'S'.
- **If** JOBZ = 'A' or JOBZ = 'O' and M < N, U contains the M-by-M orthogonal matrix U;
- **if** JOBZ = 'S', U contains the first min(M,N) columns of U (the left singular vectors, stored columnwise);
- **if** JOBZ = 'O' and M >= N, or JOBZ = 'N', U is not referenced.

`LDU` an integer. The leading dimension of the matrix U. LDU >= 1; if JOBZ = 'S' or 'A' or JOBZ = 'O' and M < N, LDU >= M.

`VT` VT is matrix of dimension (LDVT,N)

- **If** JOBZ = 'A' or JOBZ = 'O' and M >= N, VT contains the N-by-N orthogonal matrix V**T;
- **if** JOBZ = 'S', VT contains the first min(M,N) rows of V**T (the right singular vectors, stored rowwise);
- **if** JOBZ = 'O' and M < N, or JOBZ = 'N', VT is not referenced.

`LDVT` an integer. The leading dimension of the matrix VT. LDVT >= 1; if JOBZ = 'A' or JOBZ = 'O' and M >= N, LDVT >= N; if JOBZ = 'S', LDVT >= min(M,N).

`WORK` a matrix of dimension (MAX(1,LWORK))

`LWORK` an integer. The dimension of the array WORK. LWORK >= 1. If LWORK = -1, a workspace query is assumed. The optimal size for the WORK array is calculated and stored in WORK(1), and no other work except argument checking is performed.

Let mx = max(M,N) and mn = min(M,N).

If JOBZ = 'N', LWORK >= 3*mn + max(mx, 7*mn).

If JOBZ = 'O', LWORK >= 3*mn + max(mx, 5*mn*mn + 4*mn).

If JOBZ = 'S', LWORK >= 4*mn*mn + 7*mn.

If JOBZ = 'A', LWORK >= 4*mn*mn + 6*mn + mx.

These are not tight minimums in all cases; see comments inside code. For good performance, LWORK should generally be larger; a query is recommended.

Value

IWORK an integer matrix dimension of (8*min(M,N)) A is updated.

if JOBZ = 'O', A is overwritten with the first N columns of U (the left singular vectors, stored columnwise) if M >= N; A is overwritten with the first M rows of V**T (the right singular vectors, stored rowwise) otherwise.

if JOBZ .ne. 'O', the contents of A are destroyed.

INFO an integer

= 0: successful exit.

< 0: if INFO = -i, the i-th argument had an illegal value.

> 0: DBDSDC did not converge, updating process failed.

Examples

```
## Not run:
set.seed(4669)
A = matrix(rnorm(12),4,3)
S = matrix(0,nrow=3,ncol=1)
U = matrix(0,nrow=4,ncol=4)
VT = matrix(0,ncol=3,nrow=3)
dgesdd(A=A,S=S,U=U,VT=VT)
S
U
VT

rm(A,S,U,VT)

A = as.big.matrix(matrix(rnorm(12),4,3))
S = as.big.matrix(matrix(0,nrow=3,ncol=1))
U = as.big.matrix(matrix(0,nrow=4,ncol=4))
VT = as.big.matrix(matrix(0,ncol=3,nrow=3))
```

```
dgesdd(A=A,S=S,U=U,VT=VT)
S[,]
U[,]
VT[,]

rm(A,S,U,VT)
gc()

## End(Not run)
```

---

dpotrf *Cholesky factorization*

Description

DPOTRF computes the Cholesky factorization of a real symmetric positive definite matrix A.

The factorization has the form

A = U**T * U, if UPLO = 'U', or

A = L * L**T, if UPLO = 'L',

where U is an upper triangular matrix and L is lower triangular.

This is the block version of the algorithm, calling Level 3 BLAS.

Usage

```
dpotrf(UPLO = "U", N = NULL, A, LDA = NULL)
```

Arguments

UPLO	a character. **'U':** Upper triangle of A is stored; **'L':** Lower triangle of A is stored.
N	an integer. The order of the matrix A. N >= 0.
A	a big.matrix, dimension (LDA,N).
LDA	an integer. Dimension of the array A. LDA >= max(1,N).

Value

updates the big matrix A with the result, INFO is an integer

= 0: successful exit

< 0: if INFO = -i, the i-th argument had an illegal value

> 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

Terms laying out of the computed triangle should be discarded.

Examples

```
set.seed(4669)
A = matrix(rnorm(16),4)
B = as.big.matrix(A %*% t(A))
C = A %*% t(A)
chol(C)
dpotrf(UPLO='U', N=4, A=B, LDA=4)
D <- B[,]
D[lower.tri(D)]<-0
D
D-chol(C)
t(D)%*%D-C

#' # The big.matrix file backings will be deleted when garbage collected.
rm(A,B,C,D)
gc()
```

`dscal` *Scales a vector by a constant.*

Description

Scales a vector by a constant.

Usage

```
dscal(N = NULL, ALPHA, Y, INCY = 1)
```

Arguments

N	an integer. Number of elements in input vector(s)
ALPHA	a real number. The scalar alpha
Y	a big matrix to scale by ALPHA
INCY	an integer. Storage spacing between elements of Y.

Value

Update Y.

Examples

```
set.seed(4669)
A = big.matrix(3, 2, type="double", init=1, dimnames=list(NULL,
c("alpha", "beta")), shared=FALSE)
dscal(ALPHA=2,Y=A)
A[,]

# The big.matrix file backings will be deleted when garbage collected.
rm(A)
gc()
```

Index