Package ‘expm’

March 21, 2019

Type Package

Title Matrix Exponential, Log, 'etc'

Version 0.999-4

Date 2019-03-20

Author Vincent Goulet, Christophe Dutang, Martin Maechler,
David Firth, Marina Shapira, Michael Stadelmann

Contact expm-developers@lists.R-forge.R-project.org

Maintainer Martin Maechler <maechler@stat.math.ethz.ch>

Description Computation of the matrix exponential, logarithm, sqrt,
and related quantities.

Depends Matrix

Imports methods

Suggests RColorBrewer, sfsmisc, Rmpfr

BuildResaveData no

License GPL (>= 2)

URL http://R-Forge.R-project.org/projects/expm/

Encoding UTF-8

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-03-21 10:10:02 UTC

R topics documented:

balance 2
expAtv 3
expm 5
expm.Higham08 9
expmCond 12
expmFrechet 14
logm 15

matpow 17
matStig 18
sqrtm 19

Index **20**

balance *Balance a Square Matrix via LAPACK's DGEBAL*

Description

Balance a square matrix via LAPACK's `DGEBAL`. This is an R interface, mainly used for experimentation.

This LAPACK routine is used internally for Eigenvalue decompositions, but also, in Ward(1977)'s algorithm for the matrix exponential.

The name `balance()` is preferred nowadays, where "dgebal" will probably become deprecated.

Usage

```
balance(A, job = c("B", "N", "P", "S"))
 dgebal(A, job = c("B", "N", "P", "S"))
```

Arguments

`A` a square ($n \times n$) numeric matrix.

`job` a one-letter string specifying the 'job' for DGEBAL.

P **P**ermutation

S **S**caling

B **B**oth permutation and scaling

N **N**one

Details

An excerpt of the LAPACK documentation about DGEBAL(), describing the result

i1 ("ILO") (output) integer

i2 ("IHI") (output) integer
`i1` and `i2` are set to integers such that on exit `z[i,j] = 0` if $i > j$ and $j = 1, ..., i1 - 1$ or $i = i2 + 1, ..., n$.
If `job = 'N'` or `'S'`, `i1 = 1` and `i2 = n`.

scale (output) numeric vector of length `n`. Details of the permutations and scaling factors applied to `A`. If `P[j]` is the index of the row and column interchanged with row and column `j` and `D[j]` is the scaling factor applied to row and column j, then `scale[j] = P[j]` for $j = 1, ..., i1 - 1$
`= D[j]` for $j = i1, ..., i2$,
`= P[j]` for $j = i2 + 1, ..., n$.
The order in which the interchanges are made is `n` to `i2+1`, then `1` to `i1-1`.

Look at the LAPACK documentation for more details.

Value

A list with components

`z`	the transformation of matrix `A`, after permutation and or scaling.
`scale`	numeric vector of length n, containing the permutation and/or scale vectors applied.
`i1,i2`	integers (length 1) in $\{1, 2, \ldots, n\}$, denoted by `ILO` and `IHI` respectively in the LAPACK documentation. Only relevant for `"P"` or `"B"`, they describe where permutations and where scaling took place; see the Details section.

Author(s)

Martin Maechler

References

LAPACK Reference Manual

See Also

`eigen`, `expm`.

Examples

```
m4 <- rbind(c(-1,-1, 0, 0),
            c( 0, 0,10,10),
            c( 0, 0,10, 0),
            c( 0,10, 0, 0))
(b4 <- balance(m4))

## --- for testing and didactical reasons : ----

demo(balanceTst) # also defines the balanceTst() function
                 # which in its tests ``defines'' what
                 # the return value means, notably (i1,i2,scale)
```

`expAtv` *Compute Matrix Exponential exp(A t) * v directly*

Description

Compute $\exp(At) * v$ directly, without evaluating $\exp(A)$.

Usage

```
expAtv(A, v, t = 1,
       method = "Sidje98",
       rescaleBelow = 1e-6,
       tol = 1e-07, btol = 1e-07, m.max = 30, mxrej = 10,
       verbose = getOption("verbose"))
```

Arguments

`A`	n x n matrix
`v`	n - vector
`t`	number (scalar);
`method`	a string indicating the method to be used; there's only one currently; we would like to add newer ones.
`rescaleBelow`	if `norm(A,"I")` is smaller than `rescaleBelow`, rescale `A` to norm 1 and `t` such that At remains unchanged. This step is in addition to Sidje's original algorithm and easily seen to be necessary even in simple cases (e.g., $n = 3$).
`tol, btol`	tolerances; these are tuning constants of the "Sidje1998" method which the user should typically *not* change.
`m.max, mxrej`	integer constants you should only change if you know what you're doing
`verbose`	flag indicating if the algorithm should be verbose..

Value

a list with components

`eAtv`	.....fixme...

Author(s)

Ravi Varadhan, Johns Hopkins University; Martin Maechler (cosmetic, generalization to sparse matrices; rescaling (see `rescaleBelow`).

References

Roger B. Sidje (1998) EXPOKIT: Software Package for Computing Matrix Exponentials. *ACM - Transactions On Mathematical Software* **24**(1), 130–156.

((NOT yet available!))
Al-Mohy, A. and Higham, N. (2011). Computing the Action of the Matrix Exponential, with an Application to Exponential Integrators. *SIAM Journal on Scientific Computing*, **33**(2), 488–511.

See Also

`expm`

Examples

```
source(system.file("demo", "exact-fn.R", package = "expm"))
##-> rnilMat() ; xct10
set.seed(1)
(s5 <- Matrix(m5 <- rnilMat(5))); v <- c(1,6:9)
(em5 <- expm(m5))
r5  <- expAtv(m5, v)
r5. <- expAtv(s5, v)
stopifnot(all.equal(r5, r5., tolerance = 1e-14),
```

```
          all.equal(c(em5 %*% v), r5$eAtv))

v <- 10:1
with(xct10, all.equal(expm(m), expm))
all.equal(c(xct10$expm %*% v),
          expAtv(xct10$m, v)$eAtv)
```

---

expm *Matrix Exponential*

Description

This function computes the exponential of a square matrix A, defined as the sum from $r = 0$ to infinity of $A^r/r!$. Several methods are provided. The Taylor series and Padé approximation are very importantly combined with scaling and squaring.

Usage

```
expm(x, method = c("Higham08.b", "Higham08",
                   "AlMohy-Hi09",
                   "Ward77", "PadeRBS", "Pade", "Taylor", "PadeO", "TaylorO",
                   "R_Eigen", "R_Pade", "R_Ward77", "hybrid_Eigen_Ward"),
     order = 8, trySym = TRUE, tol = .Machine$double.eps, do.sparseMsg = TRUE,
     preconditioning = c("2bal", "1bal", "buggy"))
```

Arguments

`x`	a square matrix.
`method`	`"Higham08.b"`, `"Ward77"`, `"Pade"` or `"Taylor"`, etc; The default is now `"Higham08.b"` which uses Higham's 2008 algorithm with *additional* balancing preconditioning, see `expm.Higham08`. The versions with "*O" call the original Fortran code, whereas the first ones call the BLAS-using and partly simplified newer code. `"R_Pade"` uses an R-code version of `"Pade"` for didactical reasons, and `"R_Ward77"` uses an R version of `"Ward77"`, still based on LAPACK's `dgebal`, see R interface `dgebal`. This has enabled us to diagnose and fix the bug in the original octave implementation of `"Ward77"`. `"R_Eigen"` tries to diagonalise the matrix `x`, if not possible, `"R_Eigen"` raises an error. `"hybrid_Eigen_Ward"` method also tries to diagonalise the matrix `x`, if not possible, it uses `"Ward77"` algorithm.
`order`	an integer, the order of approximation to be used, for the "Pade" and "Taylor" methods. The best value for this depends on machine precision (and slightly on `x`) but for the current double precision arithmetic, one recommendation (and the Matlab implementations) uses `order = 6` unconditionally; our default, `8`, is from Ward(1977, p.606)'s recommendation, but also used for `"AlMohy-Hi09"` where a high order `order=12` may be more appropriate (and slightly more expensive).

trySym — logical indicating if `method = "R_Eigen"` should use `isSymmetric(x)` and take advantage for (almost) symmetric matrices.

tol — a given tolerance used to check if `x` is computationally singular when `method = "hybrid_Eigen_Ward"`.

do.sparseMsg — logical allowing a message about sparse to dense coercion; setting it `FALSE` suppresses that message.

preconditioning — a string specifying which implementation of Ward(1977) should be used when `method = "Ward77"`.

Details

The exponential of a matrix is defined as the infinite Taylor series

$$e^M = \sum_{k=1}^{\infty} \frac{M^k}{k!}.$$

For the "Pade" and "Taylor" methods, there is an `"accuracy"` attribute of the result. It is an upper bound for the L2 norm of the Cauchy error `expm(x, *, order + 10) - expm(x, *, order)`.

Currently, only algorithms which are *"R-code only"* accept *sparse* matrices (see the `sparseMatrix` class in package **Matrix**), i.e., currently only `"R_Eigen"` and `"Higham08"`.

Value

The matrix exponential of `x`.

Note

For a good general discussion of the matrix exponential problem, see Moler and van Loan (2003).

Author(s)

The `"Ward77"` method:
Vincent Goulet `<vincent.goulet@act.ulaval.ca>`, and Christophe Dutang, based on code translated by Doug Bates and Martin Maechler from the implementation of the corresponding Octave function contributed by A. Scottedward Hodel `<A.S.Hodel@eng.auburn.edu>`.

The `"PadeRBS"` method:
Roger B. Sidje, see the EXPOKIT reference.

The `"PadeO"` and `"TaylorO"` methods:
Marina Shapira (U Oxford, UK) and David Firth (U Warwick, UK);

The `"Pade"` and `"Taylor"` methods are slight modifications of the "*O" ([O]riginal versions) methods, by Martin Maechler, using BLAS and LINPACK where possible.

The `"hybrid_Eigen_Ward"` method by Christophe Dutang is a C translation of `"R_Eigen"` method by Martin Maechler.

The `"Higham08"` and `"Higham08.b"` (current default) were written by Michael Stadelmann, see `expm.Higham08`.

The `"AlMohy-Hi09"` implementation (R code interfacing to stand-alone C) was provided and donated by Drew Schmidt, U. Tennesse.

References

Ward, R. C. (1977). Numerical computation of the matrix exponential with accuracy estimate. *SIAM J. Num. Anal.* **14**, 600–610.

Roger B. Sidje (1998). EXPOKIT: Software package for computing matrix exponentials. ACM - Transactions on Mathematical Software **24**(1), 130–156.

Moler, C and van Loan, C (2003). Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review* **45**, 3–49. At http://epubs.siam.org/doi/pdf/10.1137/S00361445024180

Awad H. Al-Mohy and Nicholas J. Higham (2009) A New Scaling and Squaring Algorithm for the Matrix Exponential. *SIAM. J. Matrix Anal. & Appl.*, **31**(3), 970–989.

See Also

The package vignette for details on the algorithms and calling the function from external packages.

`expm.Higham08` for `"Higham08"`.

`expAtv(A,v,t)` computes $e^{At}v$ (for scalar t and n-vector v) *directly* and more efficiently than computing e^{At}.

Examples

```
x <- matrix(c(-49, -64, 24, 31), 2, 2)
expm(x)
expm(x, method = "AlMohy-Hi09")
## ----------------------------
## Test case 1 from Ward (1977)
## ----------------------------
test1 <- t(matrix(c(
    4, 2, 0,
    1, 4, 1,
    1, 1, 4), 3, 3))
expm(test1, method="Pade")
## Results on Power Mac G3 under Mac OS 10.2.8
##                    [,1]               [,2]               [,3]
## [1,] 147.86662244637000 183.76513864636857  71.79703239999643
## [2,] 127.78108552318250 183.76513864636877  91.88256932318409
## [3,] 127.78108552318204 163.67960172318047 111.96810624637124
## -- these agree with ward (1977, p608)

## Compare with the naive "R_Eigen" method:
try(
expm(test1, method="R_Eigen")
) ## platform depently, sometimes gives an error from solve
## or is accurate or one older result was
##                   [,1]                [,2]               [,3]
##[1,] 147.86662244637003  88.500223574029647 103.39983337000028
##[2,] 127.78108552318220 117.345806155250600  90.70416537273444
##[3,] 127.78108552318226  90.384173332156763 117.66579819582827
## -- hopelessly inaccurate in all but the first column.
##
```

```
## ----------------------------
## Test case 2 from Ward (1977)
## ----------------------------
test2 <- t(matrix(c(
    29.87942128909879, .7815750847907159, -2.289519314033932,
    .7815750847907159, 25.72656945571064,  8.680737820540137,
   -2.289519314033932, 8.680737820540137,  34.39400925519054),
           3, 3))
expm(test2, method="Pade")
##                   [,1]                [,2]                [,3]
##[1,]   5496313853692357 -18231880972009844 -30475770808580828
##[2,] -18231880972009852  60605228702227024 101291842930256144
##[3,] -30475770808580840 101291842930256144 169294411240859072
## -- which agrees with Ward (1977) to 13 significant figures
expm(test2, method="R_Eigen")
##                   [,1]                [,2]                [,3]
##[1,]   5496313853692405 -18231880972009100 -30475770808580196
##[2,] -18231880972009160  60605228702221760 101291842930249376
##[3,] -30475770808580244 101291842930249200 169294411240850880
## -- in this case a very similar degree of accuracy.
##
## ----------------------------
## Test case 3 from Ward (1977)
## ----------------------------
test3 <- t(matrix(c(
    -131, 19, 18,
    -390, 56, 54,
    -387, 57, 52), 3, 3))
expm(test3, method="Pade")
##                    [,1]                [,2]                [,3]
##[1,] -1.5096441587713636 0.36787943910439874 0.13533528117301735
##[2,] -5.6325707997970271 1.47151775847745725 0.40600584351567010
##[3,] -4.9349383260294299 1.10363831731417195 0.54134112675653534
## -- agrees to 10dp with Ward (1977), p608.
expm(test3, method="R_Eigen")
##                   [,1]               [,2]                [,3]
##[1,] -1.509644158796182 0.3678794391103086 0.13533528117547022
##[2,] -5.632570799902948 1.4715177585023838 0.40600584352641989
##[3,] -4.934938326098410 1.1036383173309319 0.54134112676302582
## -- in this case, a similar level of agreement with Ward (1977).
##
## ----------------------------
## Test case 4 from Ward (1977)
## ----------------------------
test4 <-
    structure(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-10,
                1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
```

```
                0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 1, 0),
    .Dim = c(10, 10))
attributes(expm(test4, method="Pade"))
max(abs(expm(test4, method="Pade") - expm(test4, method="R_Eigen")))
##[1] 8.746826694186494e-08
## -- here mexp2 is accurate only to 7 d.p., whereas mexp
##    is correct to at least 14 d.p.
##
## Note that these results are achieved with the default
## settings order=8, method="Pade" -- accuracy could
## presumably be improved still further by some tuning
## of these settings.

##
## example of computationally singular matrix
##
m <- matrix(c(0,1,0,0), 2,2)
try(
expm(m, m="R_Eigen")
)
## error since m is computationally singular
expm(m, m="hybrid")
## hybrid use the Ward77 method
```

---

expm.Higham08 *Matrix Exponential [Higham 2008]*

Description

Calculation of matrix exponential e^A with the 'Scaling & Squaring' method with balancing.

Implementation of Higham's Algorithm from his book (see references), Chapter 10, Algorithm 10.20.

The balancing option is an extra from Michael Stadelmann's Masters thesis.

Usage

```
expm.Higham08(A, balancing = TRUE)
```

Arguments

A	square matrix, may be a "sparseMatrix", currently only if balancing is false.
balancing	logical indicating if balancing should happen (before and after scaling and squaring).

Details

The algorithm comprises the following steps

1. 0.Balancing
2. 1.Scaling
3. 2.Padé-Approximation
4. 3.Squaring
5. 4.Reverse Balancing

Value

a matrix of the same dimension as `A`, the matrix exponential of `A`.

Author(s)

Michael Stadelmann (final polish by Martin Maechler).

References

Higham, N.~J. (2008). *Functions of Matrices: Theory and Computation*; Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Michael Stadelmann (2009). *Matrixfunktionen; Analyse und Implementierung*. [in German] Master's thesis and Research Report 2009-12, SAM, ETH Zurich; http://www.sam.math.ethz.ch/reports/2009, or ftp://ftp.sam.math.ethz.ch/pub/sam-reports/reports/reports2009/2009-12.pdf.

See Also

For now, the other algorithms `expm`. **This will change there will be one function with optional arguments to chose the method !**.

`expmCond`, to compute the exponential-*condition* number.

Examples

```
## The *same* examples as in ../expm.Rd  {FIXME} --
x <- matrix(c(-49, -64, 24, 31), 2, 2)
expm.Higham08(x)

## ----------------------------
## Test case 1 from Ward (1977)
## ----------------------------
test1 <- t(matrix(c(
    4, 2, 0,
    1, 4, 1,
    1, 1, 4), 3, 3))
expm.Higham08(test1)
##                   [,1]               [,2]               [,3]
## [1,] 147.86662244637000 183.76513864636857  71.79703239999643
## [2,] 127.78108552318250 183.76513864636877  91.88256932318409
```

```
## [3,] 127.78108552318204 163.67960172318047 111.96810624637124
## -- these agree with ward (1977, p608)


## ----------------------------
## Test case 2 from Ward (1977)
## ----------------------------
test2 <- t(matrix(c(
    29.87942128909879, .7815750847907159, -2.289519314033932,
    .7815750847907159, 25.72656945571064,  8.680737820540137,
   -2.289519314033932, 8.680737820540137,  34.39400925519054),
           3, 3))
expm.Higham08(test2)
expm.Higham08(test2, balancing = FALSE)
##                  [,1]               [,2]               [,3]
##[1,]   5496313853692405 -18231880972009100 -30475770808580196
##[2,] -18231880972009160  60605228702221760 101291842930249376
##[3,] -30475770808580244 101291842930249200 169294411240850880
## -- in this case a very similar degree of accuracy.

## ----------------------------
## Test case 3 from Ward (1977)
## ----------------------------
test3 <- t(matrix(c(
    -131, 19, 18,
    -390, 56, 54,
    -387, 57, 52), 3, 3))
expm.Higham08(test3)
expm.Higham08(test3, balancing = FALSE)
##                    [,1]                [,2]                [,3]
##[1,] -1.5096441587713636 0.36787943910439874 0.13533528117301735
##[2,] -5.6325707997970271 1.47151775847745725 0.40600584351567010
##[3,] -4.9349383260294299 1.10363831731417195 0.54134112675653534
## -- agrees to 10dp with Ward (1977), p608. ??? (FIXME)

## ----------------------------
## Test case 4 from Ward (1977)
## ----------------------------
test4 <-
    structure(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-10,
                1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 1, 0),
    .Dim = c(10, 10))

E4 <- expm.Higham08(test4)
Matrix(zapsmall(E4))
```

```
S4 <- as(test4, "sparseMatrix") # some R based expm() methods work for sparse:
ES4 <- expm.Higham08(S4, bal=FALSE)
stopifnot(all.equal(E4, unname(as.matrix(ES4))))
## NOTE: Need much larger sparse matrices for sparse arith to be faster!

##
## example of computationally singular matrix
##
m <- matrix(c(0,1,0,0), 2,2)
eS <- expm.Higham08(m)  # "works"  (hmm ...)
```

`expmCond` *Exponential Condition Number of a Matrix*

Description

Compute the exponential condition number of a matrix, either with approximation methods, or exactly and very slowly.

Usage

```
expmCond(A, method = c("1.est", "F.est", "exact"),
         expm = TRUE, abstol = 0.1, reltol = 1e-6,
         give.exact = c("both", "1.norm", "F.norm"))
```

Arguments

`A`	a square matrix
`method`	a string; either compute 1-norm or F-norm *approximations*, or compte these *exactly*.
`expm`	logical indicating if the matrix exponential itself, which is computed anyway, should be returned as well.
`abstol, reltol`	for `method = "F.est"`, numerical ≥ 0, as *absolute* and *relative* error tolerance.
`give.exact`	for `method = "exact"`, specify if only the 1-norm, the Frobenius norm, or both are to be computed.

Details

`method = "exact"`, aka Kronecker-Sylvester algorithm, computes a Kronecker matrix of dimension $n^2 \times n^2$ and hence, with $O(n^5)$ complexity, is prohibitely slow for non-small n. It computes the *exact* exponential-condition numbers for both the Frobenius and/or the 1-norm, depending on `give.exact`.

The two other methods compute approximations, to these norms, i.e., **est**imate them, using algorithms from Higham, chapt.~3.4, both with complexity $O(n^3)$.

Value

when expm = TRUE, for method = "exact", a list with components

expm containing the matrix exponential, expm.Higham08(A).

expmCond(F|1) numeric scalar, (an approximation to) the (matrix exponential) condition number, for either the 1-norm (expmCond1) or the Frobenius-norm (expmCondF).

When expm is false and method one of the approximations ("*.est"), the condition number is returned directly (i.e., numeric of length one).

Author(s)

Michael Stadelmann (final polish by Martin Maechler).

References

Awad H. Al-Mohy and Nicholas J. Higham (2009). *Computing Fréchet Derivative of the Matrix Exponential, with an application to Condition Number Estimation*; MIMS EPrint 2008.26; Manchester Institute for Mathematical Sciences, U. Manchester, UK. http://eprints.ma.man.ac.uk/1218/01/covered/MIMS_ep2008_26.pdf

Higham, N.~J. (2008). *Functions of Matrices: Theory and Computation*; Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Michael Stadelmann (2009) *Matrixfunktionen* ... Master's thesis; see reference in expm.Higham08.

See Also

expm.Higham08 for the matrix exponential.

Examples

```
set.seed(101)
(A <- matrix(round(rnorm(3^2),1), 3,3))

eA <- expm.Higham08(A)
stopifnot(all.equal(eA, expm::expm(A), tolerance= 1e-15))

C1 <- expmCond(A, "exact")
C2 <- expmCond(A, "1.est")
C3 <- expmCond(A, "F.est")
all.equal(C1$expmCond1, C2$expmCond, tolerance= 1e-15)# TRUE
all.equal(C1$expmCondF, C3$expmCond)# relative difference of 0.001...
```

`expmFrechet` *Frechet Derivative of the Matrix Exponential*

Description

Compute the Frechet (actually 'Fréchet') derivative of the matrix exponential operator.

Usage

```
expmFrechet(A, E, method = c("SPS", "blockEnlarge"), expm = TRUE)
```

Arguments

`A`	square matrix ($n \times n$).
`E`	the "small Error" matrix, used in $L(A, E) = f(A + E, A)$
`method`	string specifying the method / algorithm; the default `"SPS"` is "Scaling + Pade + Squaring" as in the algorithm 6.4 below; otherwise see the 'Details' section.
`expm`	logical indicating if the matrix exponential itself, which is computed anyway, should be returned as well.

Details

Calculation of e^A and the Exponential Frechet-Derivative $L(A, E)$.

When `method = "SPS"` (by default), the with the Scaling - Padé - Squaring Method is used, in an R-Implementation of Al-Mohy and Higham (2009)'s Algorithm 6.4.

Step 1: Scaling (of A and E)

Step 2: Padé-Approximation of e^A and $L(A, E)$

Step 3: Squaring (reversing step 1)

`method = "blockEnlarge"` uses the matrix identity of

$$f([AE; 0A]) = [f(A)Df(A); 0f(A)]$$

for the $2n \times 2n$ block matrices where $f(A) := expm(A)$ and $Df(A) := L(A, E)$. Note that `"blockEnlarge"` is much simpler to implement but slower (CPU time is doubled for $n = 100$).

Value

a list with components

`expm`	if `expm` is true, the matrix exponential ($n \times n$ matrix).
`Lexpm`	the Exponential-Frechet-Derivative $L(A, E)$, a matrix of the same dimension.

Author(s)

Michael Stadelmann (final polish by Martin Maechler).

References

see expmCond.

See Also

expm.Higham08 for the matrix exponential. expmCond for exponential condition number computations which are based on `expmFrechet`.

Examples

```
(A <- cbind(1, 2:3, 5:8, c(9,1,5,3)))
E <- matrix(1e-3, 4,4)
(L.AE <- expmFrechet(A, E))
all.equal(L.AE, expmFrechet(A, E, "block"), tolerance = 1e-14) ## TRUE
```

---

logm — *Matrix Logarithm*

Description

This function computes the (principal) matrix logarithm of a square matrix. A logarithm of a matrix A is L such that $A = e^L$ (meaning `A == expm(L)`), see the documentation for the matrix exponential, expm, which can be defined as

$$e^L := \sum_{r=0}^{\infty} L^r/r!.$$

Usage

```
logm(x, method = c("Higham08", "Eigen"),

     tol = .Machine$double.eps)
```

Arguments

`x`	a square matrix.
`method`	a string specifying the algorithmic method to be used. The default uses the algorithm by Higham(2008). The simple `"Eigen"` method tries to diagonalise the matrix `x`; if that is not possible, it raises an error.
`tol`	a given tolerance used to check if `x` is computationally singular when `method = "Eigen"`.

Details

The exponential of a matrix is defined as the infinite Taylor series

$$e^M = \sum_{k=1}^{\infty} \frac{M^k}{k!}.$$

The matrix logarithm of A is a matrix M such that $exp(M) = A$. Note that there typically are an infinite number number of such matrices, and we compute the *prinicipal* matrix logarithm, see the references.

Method "Higham08" works via "*inverse* scaling and squaring", and from the Schur decomposition, applying a matrix square root computation. It is somewhat slow but also works for non-diagonalizable matrices.

Value

A matrix 'as x' with the matrix logarithm of x, i.e., `all.equal( expm(logm(x)), x, tol)` is typically true for quite small tolerance `tol`.

Author(s)

Method "Higham08" was implemented by Michael Stadelmann as part of his master thesis in mathematics, at ETH Zurich; the "Eigen" method by Christophe Dutang.

References

Higham, N.~J. (2008). *Functions of Matrices: Theory and Computation*; Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

The Matrix Logarithm is very nicely defined by Wikipedia, http://en.wikipedia.org/wiki/Matrix_logarithm.

See Also

expm

Examples

```
m <- diag(2)
logm(m)
expm(logm(m))

## Here, logm() is barely defined, and Higham08 has needed an amendment
## in order for not to loop forever:
D0 <- diag(x=c(1, 0.))
(L. <- logm(D0))
stopifnot( all.equal(D0, expm(L.)) )

## A matrix for which clearly no logm(.) exists:
(m <- cbind(1:2, 1))
(l.m <- try(logm(m))) ## all NA or even error {on Solaris}
```

```
## The "Eigen" method ``works'' but wrongly :
expm(logm(m, "Eigen"))
```

`matpow` *Matrix Power*

Description

Compute the k-th power of a matrix. Whereas `x^k` computes *element wise* powers, `x %^% k` corresponds to $k-1$ matrix multiplications, `x %*% x %*% ... %*% x`.

Usage

```
x %^% k
```

Arguments

`x`	a square `matrix`.
`k`	an integer, $k \geq 0$.

Details

Argument k is coerced to integer using `as.integer`.

The algorithm uses $O(log_2(k))$ matrix multiplications.

Value

A matrix of the same dimension as `x`.

Note

If you think you need `x^k` for $k < 0$, then consider instead `solve(x %^% (-k))`.

Author(s)

Based on an R-help posting of Vicente Canto Casasola, and Vincent Goulet's C implementation in **actuar**.

See Also

`%*%` for matrix multiplication.

Examples

```
A <- cbind(1, 2 * diag(3)[,-1])
A
A %^% 2
stopifnot(identical(A, A %^% 1),
          A %^% 2 == A %*% A)
```

matStig *Stig's "infamous" Example Matrix*

Description

Stig Mortensen wrote on Oct 22, 2007 to the authors of the **Matrix** package with subject "Strange result from expm". There, he presented the following 8×8 matrix for which the Matrix `expm()` gave a "strange" result. As we later researched, the result indeed was wrong: the correct entries were wrongly permuted. The reason has been in the underlying source code in Octave from which it had been ported to **Matrix**.

Usage

```
data(matStig)
```

Author(s)

Martin Maechler

Examples

```
data(matStig)

as(matStig, "sparseMatrix") # since that prints more nicely.

## For more compact printing:
op <- options(digits = 4)

E1 <- expm(matStig, "Ward77", preconditioning="buggy") # the wrong result
as(E1, "sparseMatrix")
str(E2 <- expm(matStig, "Pade"))# the correct one (has "accuracy" attribute)
as(E2, "sparseMatrix")
attr(E2,"accuracy") <- NULL   # don't want it below
E3 <- expm(matStig, "R_Eigen")  # even that is fine here
all.equal(E1,E2) # not at all equal (rel.difference >~= 1.)
stopifnot(all.equal(E3,E2)) # ==

##________ The "proof" that "Ward77" is wrong _________
M <- matStig
Et1 <- expm(t(M), "Ward77", precond= "buggy")
Et2 <- expm(t(M), "Pade"); attr(Et2,"accuracy") <- NULL
all.equal(Et1, t(E1)) # completely different (rel.diff ~ 1.7 (platform dep.))
stopifnot(all.equal(Et2, t(E2))) # the same (up to tolerance)

options(op)
```

`sqrtm` *Matrix Square Root*

Description

This function computes the matrix square root of a square matrix. The sqrt of a matrix A is S such that $A = SS$.

Usage

```
sqrtm(x)
```

Arguments

`x`	a square matrix.

Details

The matrix square root S of M, $S = sqrtm(M)$ is defined as one (the “principal”) S such that $SS = S^2 = M$, (in R, `all.equal( S %*% S , M )`).

The method works from the Schur decomposition.

Value

A matrix ‘as `x`’ with the matrix sqrt of `x`.

Author(s)

Michael Stadelmann wrote the first version.

References

Higham, N.~J. (2008). *Functions of Matrices: Theory and Computation*; Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

See Also

`expm`, `logm`

Examples

```
m <- diag(2)
sqrtm(m) == m # TRUE

(m <- rbind(cbind(1, diag(1:3)),2))
sm <- sqrtm(m)
sm
zapsmall(sm %*% sm) # Zap entries ~= 2e-16
stopifnot(all.equal(m, sm %*% sm))
```

Index

∗Topic **algebra**
- expAtv, 3
- expm, 5
- expm.Higham08, 9
- expmCond, 12
- expmFrechet, 14
- logm, 15
- sqrtm, 19

∗Topic **arith**
- balance, 2
- matpow, 17

∗Topic **array**
- balance, 2
- matpow, 17
- matStig, 18

∗Topic **datasets**
- matStig, 18

∗Topic **math**
- expAtv, 3
- expm, 5
- expm.Higham08, 9
- expmCond, 12
- expmFrechet, 14
- logm, 15
- sqrtm, 19

%^% (matpow), 17
%*%, *17*

as.integer, *17*

balance, 2

dgebal, *5*
dgebal (balance), 2

eigen, *3*
expAtv, 3, *7*
expm, *3*, *4*, 5, *10*, *15*, *16*, *19*
expm.Higham08, *5–7*, 9, *13*, *15*
expmCond, *10*, 12, *15*
expmFrechet, 14
expmv (expAtv), 3

isSymmetric, *6*

list, *13*
logm, 15, *19*

matpow, 17
matrix, *17*
matStig, 18
mexp (expm), 5

norm, *4*
numeric, *13*

sparseMatrix, *6*, *9*
sqrtm, 19