Package 'matpow'

May 8, 2026

Type Package

Title Matrix Powers

Version 0.1.2

Description A general framework for computing powers of matrices. A key feature is the capability for users to write callback functions, called after each iteration, thus enabling customization for specific applications. Diverse types of matrix classes/matrix multiplication are accommodated. If the multiplication type computes in parallel, then the package computation is also parallel.

License GPL (>= 2)

Suggests bigmemory

NeedsCompilation no

ByteCompile yes

Author Norm Matloff, Jack Norman

Maintainer Norm Matloff <normmatloff@gmail.com>

Repository CRAN

Date/Publication 2022-03-12 10:40:05 UTC

Contents

cgraph *Callback Examples*

Description

Callback examples for matpow.

Usage

```
cgraph(ev,cbinit=FALSE,mindist=FALSE)
eig(ev,cbinit=FALSE,x=NULL,eps=1e-08)
mc(ev,cbinit=FALSE,eps=1e-08)
mexp(ev,cbinit=FALSE,eps=1e-08)
```

Arguments

ev	R environment as in the return value of matpow.
cbinit	matpow will first call the callback with cbinit set to TRUE before iterations begin, then to FALSE during iterations.
mindist	if TRUE, the matrix of minimum intervertex distances will be calculated.
x	initial guess for the principal eigenvector.
eps	convergence criterion.

Details

Note that these functions are not called directly. The user specifies the callback function (whether one of the examples here or one written by the user) in his/her call to matpow, which calls the callback after each iteration.

- cgraph: Determines the connectivity of a graph, and optionally the minimum intervertex distance matrix. The matrix m in the call to matpow should be an adjacency matrix, 1s and 0s.
- eig: Calculates the principal eigenvector of the input matrix.
- mc: Calculates the long-run distribution vector for an aperiodic, discrete-time Markov chain; the input matrix is the transition matrix for the chain.
- mexp: Calculates the exponential of the input matrix, as in e.g. expm of the **Matrix** package.

In cgraph, it is recommended that squaring be set to TRUE in calling matpow, though this cannot be done if the mindist option is used. Use of squaring is unconditionally recommended for eig and mc. Do not use squaring with mexp.

Restrictions: These functions are currently set up only for ordinary R matrix multiplication or use with gputools.

Value

Callback functions don't normally return values, but they usually do maintain data in the R environment ev that is eventually returned by matpow, including the following components as well as the application-independent ones:

- cgraph: Graph connectedness is returned in a boolean component connected. If the mindist option had been chosen, the dists component will show the minimum intervertex distances.
- eig: The x component will be the principal eigenvector.
- mc: The pivec component will be the long-run distribution vector.
- mexp: The esum component will be the matrix exponential.

Examples

```
## Not run:
m <- rbind(c(1,0,0,1),c(1,0,1,1),c(0,1,0,0),c(0,0,1,1))
ev <- matpow(m,callback=cgraph,mindist=T)
ev$connected  # prints TRUE
ev$dists  # prints, e.g. that min dist from 1 to 2 is 3
m <- rbind(1:2,3:4)
# allow for 1000 iterations max
ev <- matpow(m,1000,callback=eig,squaring=TRUE)
# how many iterations did we actually need?
ev$i  # only 8
ev$x  # prints eigenvec; check by calling R's eigen()

## End(Not run)
```

---

dup *Deep-Copy*

Description

Functions to perform deep copies of matrices.

Usage

```
dup.vanilla(mat)
dup.bigmemory(mat)
```

Arguments

mat	matrix to be copied.

Details

One of the arugments to matpow is dup, a function to do deep copying of the type of matrix being used. The user may supply a custom one, or use either dup.vanilla or dup.bigmemory.

Value

The matrix copy.

`genmulcmd` *Generate Multiplication Command*

Description

Functions to form quoted multiplication commands.

Usage

```
genmulcmd.vanilla(a,b,c)
genmulcmd.bigmemory(a,b,c)
```

Arguments

a	a quoted string.
b	a quoted string.
c	a quoted string.

Details

One of the arugments to `matpow` is `genmulcmd`, a function to generate a string containing the command the multiply matrices. The string is fed into `parse` and `eval` for execution. The user may supply a custom function, or use either `genmulcmd.vanilla` or `genmulcmd.bigmemory`.

Value

A quoted string for c = a * b for the given type of matrix/multiplication.

`matpow` *Matrix Powers*

Description

Computes matrix powers, with optional application-specific callbacks. Accommodates (external) parallel multiplication mechanisms.

Usage

```
matpow(m,k=NULL,squaring=FALSE,genmulcmd=NULL,dup=NULL,callback=NULL,...)
```

Arguments

m	input matrix.
k	desired power. If NULL, it is expected that the initialization portion of the user's callback function will set k.
squaring	if TRUE, saves time by first squaring m, then squaring the result and so on, until a power is reached of k or more.
genmulcmd	function to generate multiplication commands, in quoted string form. For the ordinary R "matrix" class this is function(a,b,c) paste(c," <- ",a," %*% ",b), supplied as genmulcmd.vanilla with the package.
dup	function to make a deep copy of a matrix.
callback	application-specific callback function.
...	application-specific arguments

Details

Multiplication is iterated until the desired power k is reached, with these exceptions: (a) If squaring is TRUE, k may be exceeded. (b) The callback function can set stop in ev, halting iterations; this is useful, for instance, if some convergence criterion has been reached.

One key advantage of using matpow rather than direct iteration is that parallel computation can be accommodated, by specifying genmulcmd. (The word "accommodated" here means the user must have available a mechanism for parallel computation; matpow itself contains no parallel code.)

For instance, if one is using GPU with gputools, one sets genmulcmd to genmulcmd.gputools, which calls gpuMatMult() instead of the usual %*%. So, one can switch from serial to parallel by merely changing this one argument. If genmulcmd is not specified, the code attempts to sense the proper function by inspecting class(m), in the cases of classes "matrix" and "big.matrix".

Of course, if the user's R is configured to use a parallel BLAS, such as OpenBLAS, this is automatically handled via the ordinary R "matrix" class.

Another important advantage of matpow is the ability to write a callback function, which enables much flexibility. The callback, if present, is called by matpow after each iteration, allowing application-specific operations to be applied. For instance, cgraph determines the connectivity of a graph, by checking whether the current power has all of its entries nonzero.

The call form is callbackname(ev,init,...) where ev is the R environment described above, and init must be set to TRUE on the first call, and FALSE afterward.

Since some types of matrix multiplication do not allow the product to be in the same physical location as either multiplicand, a "red and black" approach is taken to the iteration process: Storage space for powers in ev alternatives between prod1 and prod2, for odd-numbered and even-numbered iterations, respectively.

Value

An R environment ev, including the following components:

prod1	matrix, the final power.
stop	boolean value, indicating whether the iterations were stopped before the final power was to be computed.

`i` number of the last iteration performed.

Application-specific data, maintained by the callback function, can be stored here as well.

Examples

```
## Not run:
m <- rbind(1:2,3:4)
ev <- matpow(m,16)
ev$prod1
# prints
#              [,1]        [,2]
# [1,] 115007491351 1.67615e+11
# [2,] 251422553235 3.66430e+11

ev$i  # prints 15
matpow(m,16,squaring=TRUE)$i  # prints 4, same prod1

## End(Not run)

# see further examples in the callbacks
```

`normvec` *Vector Norm*

Description

Ordinary L2 vector norm.

Usage

```
normvec(x)
```

Arguments

`x` R vector.

Value

Vector norm.

Index