Package ‘kexpmv’

January 11, 2018

Type Package

Title Matrix Exponential using Krylov Subspace Routines

Version 0.0.3

Author Meabh G. McCurdy `<mmccurdy01@qub.ac.uk>`

Maintainer Meabh G. McCurdy `<mmccurdy01@qub.ac.uk>`

Depends methods, SparseM, R (>= 3.0.2)

LinkingTo Rcpp (>= 0.11.0)

Copyright See kexpmv/inst/notes/LAPACK_LICENSE.txt for src/lapack.f.

Description Implements functions from 'EXPOKIT'
(<https://www.maths.uq.edu.au/expokit/>) to calculate
matrix exponentials, Sidje RB, (1998) <doi:10.1145/285861.285868>.
Includes functions for small dense matrices along with functions
for large sparse matrices. The functions for large sparse matrices
implement Krylov subspace methods which help minimise the
computational complexity for matrix exponentials. 'Kexpmv' can be
utilised to calculate both the matrix exponential in isolation
along with the product of the matrix exponential and a vector.

License GPL (>= 2)

LazyData yes

ByteCompile true

NeedsCompilation yes

Repository CRAN

Collate 'kexpmv.R'

RoxygenNote 6.0.1

Date/Publication 2018-01-11 14:19:23 UTC

R topics documented:

`kexpmv-package` *Matrix exponentiation using Krylov subspace routines*

Description

Matrix Exponential using Krylov subspace routines

Details

This package utilises some of the matrix exponential routines included in EXPOKIT (`http://www.maths.uq.edu.au/expokit/`), which is software designed to calculate matrix exponentials for both small dense and large sparse matrices. The use of Krylov subspace methods implemented within these routines should result in more efficient computations.

The EXPOKIT software was developed by Roger B. Sidje. Nicholas J. Matzke went on to adapt this software for the R package Rexpokit, which enables users to make use of EXPOKIT, a FORTRAN library, within an R environment. Kexpmv uses the foundations of Rexpokit with the aim of making computing matrix exponentials more efficient, especially for large sparse matrices which can often be computationally complex.

Permission to distribute the EXPOKIT source under GPL was obtained from Roger B. Sidje.

EXPOKIT includes functions for exponentiating both small dense matrices and large sparse matrices. Fast and efficient matrix exponentiation is needed for different application areas, i.e. Markov models. This package allows the user to calculate both the matrix exponential in isolation as well as the matrix exponential with the product of a vector, which is essential for multi-state Markov models. Both `expokit_dgexpv` and `expokit_dmexpv` functions can compute both of these calculations with the use of the `vector` arguement.

When the matrix has large dimensions and is sparse in nature, this means the matrix has a high volume of elements equal to zero. Similiar to EXPOKIT this package transforms the matrix into Compressed Row Storage (CRS) format before the matrix exponentiation is performed. See functions `mat2crs` and `crs2mat` for more details.

Acknowledgements/sources

1. Nicholas Matzke `<nickmatzke.ncse@gmail.com>` helped greatly with the initial setup of the package. See his `Rexpokit` for another R implementation of EXPOKIT routines.

2. EXPOKIT, original FORTRAN package, by Roger B. Sidje `<rbs@maths.uq.edu.au>`, Department of Mathematics, University of Queensland, Brisbane, QLD-4072, Australia, (c) 1996-2013 All Rights Reserved. Sidje has given permission to include EXPOKIT code in this R package under the usual GPL license for CRAN R packages. For the full EXPOKIT copyright and license, see `expokit_copyright.txt` under `inst/notes`.

3. The development of this package was helped by advice and discussions with my PhD supervisors Adele Marshall and Karen Cairns.

Examples

```
library(kexpmv)

# Construct a square (nxn) matrix
mat=matrix(c(-0.071207, 0.065573, 0.005634, 0, -0.041206, 0.041206, 0, 0, 0),
nrow=3, byrow=TRUE)

# Define value for t
t = 15

# Exponentiate this matrix using DGPADM for t = 15
OutputMat = expokit_dgpadm( mat = mat, t = t, transpose_needed = TRUE)
print(OutputMat)

# Construct column (nx1) vector
v = matrix(0,3,1)
v[1] = 1

# Exponentiate the matrix using DMEXPV for t = 15 with the product of vector v.
OutputMat = expokit_dmexpv( mat = mat, t = t, vector = v, transpose_needed = TRUE,
transform_to_crs = TRUE)

# Print corresponding (nx1) results vector
print(OutputMat$output_probs)

# Print message to determine whether the mxstep value needs increased. If NULL
# then mxstep value is valid.
print(OutputMat$message)

# Exponentiate the matrix using DGEXPV for t = 15.
OutputMat = expokit_dgexpv( mat = mat, t = t, vector = NULL, transpose_needed = TRUE,
transform_to_crs = TRUE)
print(OutputMat$output_mat)
print(OutputMat$message)

# Functions DMEXPV and DGEXPV are very similiar, with the only difference being DMEXPV
# carries out an additional check for Markov chains.
```

`crs2mat` *Convert a CRS-formatted matrix to standard square format*

Description

The EXPOKIT's DMEPXV and DGEXPV functions both deal with sparse matrices. These matrices have a lot of zeros, and can therefore be stored more efficiently by converting the matrix into CRS (Compressed Row Storage) format.

Usage

```
crs2mat(mat, n)
```

Arguments

mat	a 3 column list including the vectors; ia, ja and a.
n	the dimension of the square (nxn) matrix.

Details

In EXPOKIT and its wrapper functions, a CRS-formatted matrix is input as 3 vectors:

ia = row pointer. This vector stores the location in the 'a' vector that is the first non-zero element in a row.
ja = column indices for non-zero elements.
a = non-zero elements of the matrix.

This function takes a 3-column list CRS matrix and converts it back to standard square format.

Author(s)

Meabh G. McCurdy <mmccurdy01@qub.ac.uk>

Examples

```
# Create a CRS format matrix
ia = c(1, 2, 4, 6)
ja = c(1, 1, 2, 1, 2)
a  = c(-0.071207,  0.065573, -0.041206,  0.005634,  0.041206)
crsmat=list(ia, ja, a)

# Convert CRS format matrix to square format
n = 3
mat = crs2mat(crsmat, n)
print(mat)
```

expokit_dgexpv *EXPOKIT DGEXPV matrix exponentiation on a square matrix*

Description

This function converts a matrix to CRS format and exponentiates it via the EXPOKIT dgexpv function with the wrapper functions `wrapalldgexpv_` and `dgexpv_` around DGEXPV. This function can be used to calculate both the matrix exponential in isolation or the product of the matrix exponential with a vector. This can be achieved by modifying the `vector` variable as shown below.

Usage

```
expokit_dgexpv(mat = NULL, t = 15, vector = NULL,
  transpose_needed = TRUE, transform_to_crs = TRUE, crs_n = NULL,
  anorm = NULL, mxstep = 10000, tol = as.numeric(1e-10))
```

Arguments

`mat`	an input square matrix.
`t`	a time value to exponentiate by.
`vector`	If NULL (default), the full matrix exponential is returned. However, in order to fully exploit the efficient speed of EXPOKIT on sparse matrices, this vector argument should be equal to a vector, v. This vector is an n dimensional vector, which in the Markovian case, can represent the starting probabilities.
`transpose_needed`	If TRUE (default), matrix will be transposed before the exponential operator is performed.
`transform_to_crs`	If the input matrix is in square format then the matrix will need transformed into CRS format. This is required for EXPOKIT's sparse-matrix functions DMEXPV and DGEXPV. Default TRUE; if FALSE, then the `mat` argument must be a CRS-formatted matrix.
`crs_n`	If a CRS matrix is input, `crs_n` specifies the order (# of rows or # of columns) of the matrix. Default is NULL.
`anorm`	The `expokit_dgexpv` routine requires an initial guess at the norm of the matrix. Using the R function `norm` might get slow with large matrices. Default is NULL.
`mxstep`	The EXPOKIT code performs integration steps and `mxstep` is the maximum number of integration steps performed. Default is 10000 steps; May need to increase this value if function outputs a warning message.
`tol`	the tolerance defined for the calculations.

Details

NOTE: When looking at DGEXPV vs. DMEXPV. According to the EXPOKIT documentation, the DGEXPV routine should be faster than DMEXPV. This is a result of the additional check the DMEXPV routine runs to check whether the output values satisfy the conditions needed for a Markovian model, which is not done in DGEXPV.

From EXPOKIT:

```
*     The method used is based on Krylov subspace projection
*     techniques and the matrix under consideration interacts only
*     via the external routine 'matvec' performing the matrix-vector
*     product (matrix-free method).
```

It should be noted that both the DMEXPV and DGEXPV functions within EXPOKIT require the matrix-vector product y = A*x = Q'*x i.e, where A is the transpose of Q. Failure to remember this leads to wrong results.

CRS (Compressed Row Storage) format is a compressed format that is required for EXPOKIT's sparse-matrix functions such as DGEXPV and DMEXPV. However this format is not necessary in EXPOKIT's padm-related functions.

This function is recommended for large sparse matrices, however the infinity norm of the matrix proves to be crucial when selecting the most efficient routine. If the infinity norm of the large sparse matrix is approximately >100 may be of benefit to use the `expokit_dgpadm` function for faster computations.

Author(s)

Meabh G. McCurdy <mmccurdy01@qub.ac.uk>

Examples

```
# Make a square (n x n) matrix A
# Use expokit_dgexpv to calculate both exp(At) and exp(At)v, where t is a
# time value and v is an n dimensional column vector.
mat=matrix(c(-0.071207, 0.065573, 0.005634, 0, -0.041206, 0.041206, 0, 0, 0),
nrow=3, byrow=TRUE)

# Set the time variable t
 t=15

# Exponentiate with EXPOKIT's dgexpv to obtain the full matrix exponential
OutputMat = expokit_dgexpv(mat=mat, t=t, transpose_needed=TRUE, vector=NULL)

print(OutputMat$output_mat)
print(OutputMat$message)

# Can also calculate the matrix exponential with the product of a vector.
# Create the n dimensional vector
v = matrix(0,3,1)
v[1] = 1

# Exponentiate with EXPOKIT's dgexpv
OutputMat = expokit_dgexpv(mat=mat, t=t, transpose_needed=TRUE, vector=v)

print(OutputMat$output_probs)
print(OutputMat$message)

# If message is 'NULL' then no error has occured and the number of
# mxsteps defined in the function is acceptable.
```

---

expokit_dgpadm *EXPOKIT dgpadm matrix exponentiation on a square matrix*

Description

This function exponentiates a matrix via the EXPOKIT padm function.

Usage

```
expokit_dgpadm(mat = NULL, t = 15, transpose_needed = TRUE)
```

Arguments

mat	an input square matrix.
t	time value to exponentiate by.
transpose_needed	If TRUE (default), matrix will be transposed.

Details

From EXPOKIT:

```
*     Computes exp(t*H), the matrix exponential of a general matrix in
*     full, using the irreducible rational Pade approximation to the
*     exponential function exp(x) = r(x) = (+/-)( I + 2*(q(x)/p(x)) ),
*     combined with scaling-and-squaring.
```

This function is recommended when dealing with small dense matrices. However it can also be used for large sparse matrices when the infinity norm is approximately >100.

Author(s)

Meabh G. McCurdy <mmccurdy01@qub.ac.uk>
Nicholas J. Matzke <nickmatzke.ncse@gmail.com>

Examples

```
# Define input matrix to be exponentiated
mat=matrix(c(-0.071207, 0.065573, 0.005634, 0, -0.041206, 0.041206, 0, 0, 0),
nrow=3, byrow=TRUE)

# Define value of t
t=15

# Exponentiate with EXPOKIT's dgpadm
Pmat = expokit_dgpadm(mat=mat, t=t, transpose_needed=TRUE)
print(Pmat)
```

`expokit_dmexpv` *EXPOKIT dmexpv matrix exponentiation on a square matrix*

Description

This function converts a matrix to CRS format and exponentiates it via the EXPOKIT dmexpv function with the wrapper functions `wrapalldmexpv_` and `dmexpv_` around dmexpv. This function can be used to calculate both the matrix exponential in isolation or the product of the matrix exponential with a vector. This can be achieved by modifying the `vector` variable as shown below.

Usage

```
expokit_dmexpv(mat = NULL, t = 15, vector = NULL,
  transpose_needed = TRUE, transform_to_crs = TRUE, crs_n = NULL,
  anorm = NULL, mxstep = 10000, tol = as.numeric(1e-10))
```

Arguments

`mat`	an input square matrix.
`t`	a time value to exponentiate by.
`vector`	If NULL (default), the full matrix exponential is returned. However, in order to fully exploit the efficient speed of EXPOKIT on sparse matrices, this vector argument should be equal to a vector, v. This vector is an n dimensional vector, which in the Markovian case, can represent the starting probabilities.
`transpose_needed`	If TRUE (default), matrix will be transposed before the exponential operator is performed.
`transform_to_crs`	If the input matrix is in square format then the matrix will need transformed into CRS format. This is required for EXPOKIT's sparse-matrix functions DMEXPV and DGEXPV. Default TRUE; if FALSE, then the `mat` argument must be a CRS-formatted matrix.

`crs_n`	If a CRS matrix is input, `crs_n` specifies the order (# of rows or # of columns) of the matrix. Default is NULL.
`anorm`	The `expokit_dmexpv` requires an initial guess at the norm of the matrix. Using the R function `norm` might get slow with large matrices. Default is NULL.
`mxstep`	The EXPOKIT code performs integration steps and `mxstep` is the maximum number of integration steps performed. Default is 10000 steps; May need to increase this value if function outputs a warning message.
`tol`	the tolerance defined for the calculations.

Details

From EXPOKIT:

```
*     The method used is based on Krylov subspace projection
*     techniques and the matrix under consideration interacts only
*     via the external routine 'matvec' performing the matrix-vector
*     product (matrix-free method).
*
*     This is a customised version for Markov Chains. This means that a
*     check is done within this code to ensure that the resulting vector
*     w is a probability vector, i.e., w must have all its components
*     in [0,1], with sum equal to 1. This check is done at some expense
*     and the user may try DGEXPV which is cheaper since it ignores
*     probability constraints.
```

This check assumes that the transition matrix Q, satisfies Qe = 0 where e is a column vector of 1's. If this condition does not hold then use the DGEPXV function instead. It should be noted that both the DMEXPV and DGEXPV functions within EXPOKIT require the matrix-vector product y = A*x = Q'*x i.e, where A is the transpose of Q. Failure to remember this leads to wrong results.

CRS (Compressed Row Storage) format is a compressed format that is required for EXPOKIT's sparse-matrix functions such as DGEXPV and DMEXPV. However this format is not necessary in EXPOKIT's padm-related functions.

This function is recommended for large sparse matrices, however the infinity norm of the matrix proves to be crucial when selecting the most efficient routine. If the infinity norm of the large sparse matrix is approximately >100 may be of benefit to use the `expokit_dgpadm` function for faster computations.

Author(s)

Meabh G. McCurdy <`mmccurdy01@qub.ac.uk`>

Examples

```
# Make a square matrix A
# Use expokit_dmexpv to calculate both exp(At) and exp(At)v, where t is a
# time value and v is an n dimensional column vector.
mat=matrix(c(-0.071207, 0.065573, 0.005634, 0, -0.041206, 0.041206, 0, 0, 0),
nrow=3, byrow=TRUE)

# Set the time variable t
t=15

# Exponentiate with EXPOKIT's dmexpv to obtain the full matrix exponential
OutputMat = expokit_dmexpv(mat=mat, t=t, transpose_needed=TRUE, vector=NULL)

print(OutputMat$output_mat)
print(OutputMat$message)

# Can also calculate the matrix exponential with the product of a vector.
# Create the n dimensional vector
v = matrix(0,3,1)
v[1] = 1

# Exponentiate with EXPOKIT's dmexpv
OutputMat = expokit_dmexpv(mat=mat, t=t, transpose_needed=TRUE, vector=v)

print(OutputMat$output_probs)
print(OutputMat$message)

# If message is 'NULL' then no error has occured and the number of
# mxsteps defined in the function is acceptable.
```

mat2crs — *Convert matrix to CRS format using SparseM function*

Description

The EXPOKIT's expokit_dmexpv and expokit_dgexpv functions both deal with sparse matrices. These matrices have a lot of zeros, and can therefore be stored more efficiently by converting the matrix into CRS (Compressed Row Storage) format.

Usage

```
mat2crs(mat)
```

Arguments

mat	A square matrix.

Details

In EXPOKIT and its wrapper functions, a CRS-formatted matrix is input as 3 vectors:

ia = row pointer. This vector stores the location in the 'a' vector that is the first non-zero element in a row.
ja = column indices for non-zero elements.
a = non-zero elements of the matrix.

Author(s)

Meabh G. McCurdy <mmccurdy01@qub.ac.uk>

Examples

```
# Make a square matrix
mat=matrix(c(-0.071207, 0, 0, 0.065573, -0.041206, 0, 0.005634, 0.014206, 0), nrow=3, byrow=TRUE)

# Covert to CRS format
mat2crs(mat)
print(mat)
```

Index