A Catalogue of Software for Matrix Functions.  
Version 2.0

Higham, Nicholas J. and Deadman, Edvin

2016

MIMS EPrint: 2016.3

Manchester Institute for Mathematical Sciences
School of Mathematics
The University of Manchester

Reports available from: http://eprints.maths.manchester.ac.uk/
And by contacting: The MIMS Secretary
School of Mathematics
The University of Manchester
Manchester, M13 9PL, UK

ISSN 1749-9097
A Catalogue of Software for Matrix Functions.
Version 2.0

Nicholas J. Higham*  Edvin Deadman†


Abstract
A catalogue of software for computing matrix functions and their Fréchet derivatives is presented. For a wide variety of languages and for software ranging from commercial products to open source packages we describe what matrix function codes are available and which algorithms they implement.

Contents
1 Introduction 2
2 Applications of Matrix Functions 2
3 Matrix Function Algorithms 3
4 MATLAB Built-in Functions 5
5 Symbolic Math Toolbox 5
6 The Advanpix Multiprecision Computing Toolbox 6
7 The Matrix Function Toolbox 6
8 Other MATLAB Functions 7
8.1 $f(A)$ . . . . . . . . . . . . . . . 8
8.2 $f(A)b$ . . . . . . . . . . . . . . . 8
9 Expokit 9
10 EXPINT 9
11 GNU Octave 9
12 Scilab 10
13 Maple 10
14 Mathematica 10
15 R 11
15.1 Expm . . . . . . . . . . . . . . 11
15.2 pbdDMAT . . . . . . . . . . . 11
15.3 Matrix . . . . . . . . . . . . . . 11
16 $\varphi$ Functions in Fortran 95 11
17 NAG Library 12
18 SLICOT 12
19 Python 15
19.1 SciPy . . . . . . . . . . . . . . 15
19.2 Python: SymPy . . . . . . . . 15
19.3 Other Python Functions . . . 16
20 Java 16
21 Julia 16
22 C++: Eigen 16
23 The GNU Scientific Library 17

*School of Mathematics, The University of Manchester, Manchester, M13 9PL, UK (nick.higham@manchester.ac.uk, http://www.maths.manchester.ac.uk/~higham).
†Numerical Algorithms Group Ltd., Peter House, Oxford Street, Manchester, M1 5AN (edvin.deadman@nag.co.uk, http://www.maths.manchester.ac.uk/~edeadman).
1 Introduction

The earliest widely available software for computing functions of matrices is probably the function named \texttt{fun} in the original 1984 Fortran version of MATLAB:

\begin{verbatim}
< MATLAB >
Version of 01/10/84
<>
help fun
\end{verbatim}

\texttt{FUN} For matrix arguments \(X\), the functions \texttt{SIN}, \texttt{COS}, \texttt{ATAN}, \texttt{SQRT}, \texttt{LOG}, \texttt{EXP} and \(X^{**}p\) are computed using eigenvalues \(D\) and eigenvectors \(V\). If \(\langle V,D\rangle = \text{EIG}(X)\) then \(f(X) = V*f(D)/V\). This method may give inaccurate results if \(V\) is badly conditioned. Some idea of the accuracy can be obtained by comparing \(X^{**}1\) with \(X\).

For vector arguments, the function is applied to each component.

Since then, and especially in the last decade or so, the quantity of software for matrix functions has grown tremendously—to such an extent that it is hard to keep track of what is available. This document is an attempt to produce a catalogue of software for matrix functions available in different languages and packages.

The document lists what is available with a brief description of and reference to the algorithms that are used (where known). We make no attempt to judge the quality of the software. We also do not specify version numbers; for software still under development we are referring to the version current at the time of writing. This document is not intended to be exhaustive. For example, if a code or package has been superseded or is a translation of an existing code to another language we will usually omit it.

This catalogue is a revised and updated version of [51]. We welcome notification of errors and omissions, which will be incorporated into future versions.

For background on functions of matrices see [47], [49], or [55].

2 Applications of Matrix Functions

Matrix functions have applications in a diverse and growing range of areas of science, engineering, and the social sciences. We list a selection of areas in which we are aware of the use of matrix function software.

- Multizone models of pollutant transport in buildings take the form of linear systems of ordinary differential equations, which can be effectively solved using the matrix exponential [72].

- In Markov models in finance, statistics and social science [47, Sec. 2.3] transition probability matrices are related to the transition intensity matrix via the matrix exponential. Transition matrices for shorter time scales can be generated by taking matrix roots, but there are open questions about the existence and uniqueness of stochastic roots [52], [59].
• NMR spectroscopy involves evaluating the exponential of a symmetric diagonally dominant relaxation matrix [43], [68]. The package SIMPSON (http://nmr.au.dk/software/simpson) for numerical simulations of NMR experiments includes several methods for evaluating the matrix exponential.

• In control theory, linear dynamical systems can be expressed as continuous-time systems or as discrete-time state-space systems. The matrix exponential and logarithm can be used to convert between the two forms [47, Sec. 2.4]. In the Control System Toolbox for MATLAB, functions `c2d` and `d2c` carry out these conversions.

• In nuclear engineering the burnup equations are a first-order system of linear ordinary differential equations that are usually solved by time-stepping with the matrix exponential [74]. The Python Nuclear Engineering Toolkit (http://pynesim.org) uses the SciPy function `linalg.expm` (see Section 19.1).

• In social and information networks the elements of either the exponential or the resolvent of the adjacency matrix of the network can be used to quantify the importance of nodes within the network [30]. Recent research and software development has focused on computing these elements, including in cases with special structure; see [15] and the references therein. In time-varying networks the matrix logarithm is required [39].

• A number of problems in imaging make use of the matrix logarithm, including image registration [12], patch modeling-based skin detection [57], and in-betweening in computer animations [75].

• In optics, the Mueller matrix \( M \) is a real \( 4 \times 4 \) matrix associated with an element that alters the polarization of light. One method for determining the diattenuation, retardance, and depolarization properties of \( M \) involves computing a \( p \)th root with \( p \approx 10^5 \) [22], [70]. The logarithm of \( M \) also provides understanding of the underlying medium that \( M \) describes [71]. A related Jones matrix can be represented in terms of the matrix exponential [13].

3 Matrix Function Algorithms

There is now a large literature on matrix function algorithms, of which a survey as of 2010 is given in [50]. It may not be clear to users from different fields which algorithms represent the current state-of-the-art. We list the algorithms that we consider to be preferred for a few common matrix functions, for the case where a factorization of \( A \) can be explicitly computed and full precision is required.

• Exponential: scaling and squaring algorithm (Al-Mohy and Higham, 2009) [4].

• Logarithm: inverse scaling and squaring algorithm (Al-Mohy, Higham, and Relton, 2012, 2013) [7], [8].

• Square root: Schur algorithm (Bjöck and Hammarling, 1983) [19], or real version for real matrices (Higham, 1987) [45]. An algorithm with blocking provides performance improvements (Deadman, Higham, and Ralha, 2013) [26].

• Real matrix power \( A^t \) with \( t \in \mathbb{R} \): Schur–Padé algorithm (Higham and Lin, 2013) [54].

• Cosine and sine (Al-Mohy, Higham, and Relton, 2015) [9].

Table 1: Availability of recommended algorithms.

<table>
<thead>
<tr>
<th>Function</th>
<th>MATLAB built-in</th>
<th>MATLAB Third party</th>
<th>NAG Library</th>
<th>SciPy Sec. 17</th>
<th>Julia Sec. 19.1</th>
<th>R Sec. 21</th>
<th>Sec. 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e^A$ [4]</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>$\log A$ [7], [8]</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>$\cos A, \sin A$ [9]</td>
<td>×</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>$\arccos(h)(A)$, $\arcsin(h)(A)$ [11]</td>
<td>×</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>$A^{1/2}$ [19], [26], [45]</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>$A^t$ [54]</td>
<td>×</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>$f(A)$ [23]</td>
<td>√</td>
<td>–</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Estimation of $\text{cond}(f, A)$</td>
<td>×</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>$e^{Ah}$ [6]</td>
<td>×</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>$L_{\exp}$ [3]</td>
<td>×</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>$L_{\log}$ [8]</td>
<td>×</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td>√</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>$L_{xt}$ [54]</td>
<td>×</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td>√</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

$^{a}$Krylov methods are also available; see the following sections.

- General matrix function with derivatives of the underlying scalar function available: Schur–Parlett algorithm (Davies and Higham, 2003) [23]. This uses the recurrence of Parlett (1976) [73].

- Function of a symmetric or Hermitian matrix: diagonalization (spectral decomposition).

In many applications of matrix functions the matrix is not known exactly, due to data errors or errors in previous computations. Even with exact data the computation of a matrix function is subject to rounding errors. It is therefore important to understand the sensitivity of the matrix function to perturbations in the data, which is determined by the Fréchet derivative, denoted by $L_f$. The recommended algorithms for computing the Fréchet derivative are as follows.

- Exponential: scaling and squaring algorithm (Al-Mohy and Higham, 2009) [3].
- Logarithm: inverse scaling and squaring algorithm (Al-Mohy, Higham, and Relton, 2013) [8].
- Real matrix power $A^t$ with $t \in \mathbb{R}$: Schur–Padé algorithm (Higham and Lin, 2013) [54].
- General matrix function: complex step algorithm (Al-Mohy and Higham, 2010) [5] or use of a block $2 \times 2$ matrix formula [50, Sec. 7.3].

Table 1 summarizes the availability of the above algorithms in several key sources of software. Details are provided in the following sections.

The worst-case sensitivity of a matrix function over all perturbations is measured by the condition number, $\text{cond}(f, A)$ [47, Chap. 3]. The recommended way to estimate the condition number is by using one of the above algorithms for the Fréchet derivative in conjunction with [47, Alg. 3.22] and the block matrix 1-norm estimator of [56]; an implementation is the function $\text{funm_condest1}$ in the Matrix Function Toolbox (see Section 7). We encourage users to compute a condition number estimate whenever possible.

A rather different problem is to compute $f(A)b$, where $b$ is a vector—the action of $f(A)$ on a vector—without explicitly forming $f(A)$. Such problems can involve very large, sparse
matrices, in which case matrix factorization may not be possible and methods that require only matrix–vector products with $A$ are needed. The sensitivity of this problem is investigated by Deadman [24].

Codes for the $f(A)b$ problem are described in some of the following sections.

4 MATLAB Built-in Functions

MATLAB has a number of built-in commands for evaluating functions of matrices.

- **expm**: matrix exponential by scaling and squaring algorithm (Al-Mohy and Higham, 2009) [4]. MATLAB R2006a–R2015a used the earlier algorithm of (Higham, 2005, 2009) [46], [48], which could suffer from overscaling [65].
  
  Also included for pedagogical and historical interest are three older algorithms.
  
  - **expmdemo1**: matrix exponential by an older scaling and squaring algorithm [36, Alg. 9.3.1]. This is an M-file implementation of the algorithm that was used by **expm** in MATLAB 7 (R14SP3) and earlier versions.
  
  - **expmdemo2**: matrix exponential by Taylor series.
  
  - **expmdemo3**: matrix exponential by eigenvalue decomposition.


- **mpower**, `^`: arbitrary matrix power via eigendecomposition. Not that this approach can be numerically unstable for noninteger powers of highly nonnormal matrices.

- **sqrtm**: matrix square root by Schur method with recursive blocking (Björck and Hammarling, 1983) [19], (Deadman, Higham, and Ralha, 2013) [26] and with condition number estimate.

- **funm**: Schur–Parlett algorithm for general functions (Davies and Higham, 2003) [23]. It has built-in support for the matrix cosine, sine, hyperbolic cosine, and hyperbolic sine.

- **polyvalm**: evaluate polynomial with matrix argument.

5 Symbolic Math Toolbox

The Symbolic Math Toolbox [64] is a MATLAB toolbox that carries out computations with symbolic variables and also provides variable precision arithmetic. The toolbox overloads the following functions for both symbolic and variable precision matrix arguments. For variable precision arguments the function **digits** can be used to specify the number of digits of precision required.

- **expm**: matrix exponential.

- **logm**: matrix logarithm.

- **sqrtm**: matrix square root.

- **mpower**, `^`: arbitrary matrix power via eigendecomposition.
• **funm**: general matrix function.

The Symbolic Math Toolbox also contains the MuPAD computer algebra system, which provides some additional matrix function capabilities for matrices with numeric (not symbolic) entries.

• **numeric::expMatrix**: computes the matrix exponential or the action of the matrix exponential on another matrix or vector. The numerical precision used can be specified by the environment variable **DIGITS**. The exponential is evaluated using a choice of diagonalization, interpolation, a Taylor series (apparently without scaling and squaring), or (for \( e^{A}b \) only) a Krylov subspace method.

• **numeric::fMatrix**: for a diagonalizable matrix, computes an arbitrary function of the matrix via a diagonalization.

### 6 The Advanpix Multiprecision Computing Toolbox

The Advanpix Multiprecision Toolbox [1] is an extension to MATLAB for computing with arbitrary precision. The toolbox provides arbitrary precision analogues to the built-in MATLAB matrix functions as well as some trigonometric matrix functions. It is specifically optimized for quadruple precision.

The following matrix function routines are available in the toolbox: **funm**, **expm**, **sqrtm**, **logm**, **sinm**, **cosm**, **sinhm**, and **coshm**. The first four have the same calling sequences as their MATLAB counterparts.

### 7 The Matrix Function Toolbox

The Matrix Function Toolbox (Higham, 2008) [44] contains MATLAB implementations of many of the algorithms described in the book *Functions of Matrices: Theory and Computation* [47], including

- trigonometric matrix functions,
- condition number evaluation and estimation,
- Fréchet derivative evaluation,
- polar decomposition,
- iterative methods for computing matrix roots,
- \( f(A)b \) via Arnoldi method.

The toolbox is documented in [47, App. D] and its contents are summarized in Table 2.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arnoldi</td>
<td>Arnoldi iteration</td>
</tr>
<tr>
<td>ascent_seq</td>
<td>Ascent sequence for square (singular) matrix.</td>
</tr>
<tr>
<td>cosm</td>
<td>Matrix cosine by double angle algorithm.</td>
</tr>
<tr>
<td>cosm_pade</td>
<td>Evaluate Padé approximation to the matrix cosine.</td>
</tr>
<tr>
<td>cosmsinhm</td>
<td>Matrix cosine and sine by double angle algorithm.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>cosmsinm_pade</td>
<td>Evaluate Padé approximations to matrix cosine and sine.</td>
</tr>
<tr>
<td>expm_cond</td>
<td>Relative condition number of matrix exponential.</td>
</tr>
<tr>
<td>expm_frechet_pade</td>
<td>Fréchet derivative of matrix exponential via Padé approximation.</td>
</tr>
<tr>
<td>expm_frechet_quad</td>
<td>Fréchet derivative of matrix exponential via quadrature.</td>
</tr>
<tr>
<td>fab_arnoldi</td>
<td>$f(A)b$ approximated by Arnoldi method.</td>
</tr>
<tr>
<td>funm_condest1</td>
<td>Estimate of 1-norm condition number of matrix function.</td>
</tr>
<tr>
<td>funm_condest_fro</td>
<td>Estimate of Frobenius norm condition number of matrix function.</td>
</tr>
<tr>
<td>funm_ev</td>
<td>Evaluate general matrix function via eigensystem.</td>
</tr>
<tr>
<td>funm_simple</td>
<td>Simplified Schur–Parlett method for function of a matrix.</td>
</tr>
<tr>
<td>logm_cond</td>
<td>Relative condition number of matrix logarithm.</td>
</tr>
<tr>
<td>logm_frechet_pade</td>
<td>Fréchet derivative of matrix logarithm via Padé approximation.</td>
</tr>
<tr>
<td>logm_iss</td>
<td>Matrix logarithm by inverse scaling and squaring method.</td>
</tr>
<tr>
<td>logm_pade_pf</td>
<td>Evaluate Padé approximant to matrix logarithm by partial fraction form.</td>
</tr>
<tr>
<td>mft_test</td>
<td>Test the Matrix Function Toolbox.</td>
</tr>
<tr>
<td>mft_tolerance</td>
<td>Convergence tolerance for matrix iterations.</td>
</tr>
<tr>
<td>polar_newton</td>
<td>Polar decomposition by scaled Newton iteration.</td>
</tr>
<tr>
<td>polar_svd</td>
<td>Canonical polar decomposition via singular value decomposition.</td>
</tr>
<tr>
<td>polyvalm_ps</td>
<td>Evaluate polynomial at matrix argument by Paterson–Stockmeyer algorithm.</td>
</tr>
<tr>
<td>power_binary</td>
<td>Power of matrix by binary powering (repeated squaring).</td>
</tr>
<tr>
<td>quasitriang_struct</td>
<td>Block structure of upper quasitriangular matrix.</td>
</tr>
<tr>
<td>riccati_xaxb</td>
<td>Solve Riccati equation $XAX = B$ in positive definite matrices.</td>
</tr>
<tr>
<td>rootpm_newton</td>
<td>Coupled Newton iteration for matrix $p$th root.</td>
</tr>
<tr>
<td>rootpm_real</td>
<td>$p$th root of real matrix via real Schur form.</td>
</tr>
<tr>
<td>rootpm_schur_newton</td>
<td>Matrix $p$th root by Schur–Newton method.</td>
</tr>
<tr>
<td>rootpm_sign</td>
<td>Matrix $p$th root via matrix sign function.</td>
</tr>
<tr>
<td>signm</td>
<td>Matrix sign decomposition.</td>
</tr>
<tr>
<td>signm_newton</td>
<td>Matrix sign function by Newton iteration.</td>
</tr>
<tr>
<td>sqrtm_db</td>
<td>Matrix square root by Denman–Beavers iteration.</td>
</tr>
<tr>
<td>sqrtm_dbp</td>
<td>Matrix square root by product form of Denman–Beavers iteration.</td>
</tr>
<tr>
<td>sqrtm_newton</td>
<td>Matrix square root by Newton iteration (unstable).</td>
</tr>
<tr>
<td>sqrtm_newton_full</td>
<td>Matrix square root by full Newton method.</td>
</tr>
<tr>
<td>sqrtm_pulay</td>
<td>Matrix square root by Pulay iteration.</td>
</tr>
<tr>
<td>sqrtm_real</td>
<td>Square root of real matrix by real Schur method.</td>
</tr>
<tr>
<td>sqrtm_triangular_min_norm</td>
<td>Estimated minimum norm square root of triangular matrix.</td>
</tr>
<tr>
<td>sylvsol</td>
<td>Solve Sylvester equation.</td>
</tr>
</tbody>
</table>

### 8 Other MATLAB Functions

Various other MATLAB functions implementing algorithms developed in research papers are freely available online. We organize them according to the problem that they treat: $f(A)$ or the action of $f(A)$ on a vector, $f(A)b$. 
8.1 \( f(A) \)


8.2 \( f(A)b \)


• Güttel and Knizhnerman (2011) [40]: a black-box rational Arnoldi method for computing $f(A)b$, where $f$ is a Markov matrix function. Available from http://guettel.com/markovfunmv. See also [41].

• Hale, Higham and Trefethen (2008) [42]: algorithms for evaluating $f(A)$ and $f(A)b$ by contour integration. MATLAB code is embedded in the paper.


9 Expokit

Expokit (Sidje, 1998) [79], [80] is a package of Fortran and MATLAB codes to compute $e^{A}$ (using scaling and squaring) and $e^{A}b$ (using Krylov subspace methods). An R interface to Expokit is available at http://cran.r-project.org/web/packages/expokit.

10 EXPINT

EXPINT (Berland, Skaflestad and Wright, 2007) [16], [17] is a MATLAB package providing exponential integrators for ordinary differential equations. A large range of Runge–Kutta, multistep, and general linear integrators is available. The functions $\varphi_k(z) = \sum_{j=0}^{\infty} z^j/(j + k)!$ underlying the methods are evaluated at matrix arguments using Padé approximants with a scaling and squaring scheme.

11 GNU Octave

GNU Octave [34] is an open source problem-solving environment (PSE)\footnote{A PSE provides a programming language, an interactive command window with the display of graphics, and the ability to export graphics and more generally publish documents to HTML, PDF, \TeX, and so on.} with a high-level programming language similar to (and mostly compatible with) MATLAB. It contains several matrix function routines.

• $\text{expm}$: matrix exponential by Ward’s version of the scaling and squaring algorithm (1977) [83].

• $\text{logm}$: matrix logarithm by an inverse scaling and squaring algorithm (Higham, 2008) [47].

• $\text{sqrtm}$: matrix square root by the Schur method (Björck and Hammarling, 1983) [19].
An extra package `linear-algebra` is available (http://octave.sourceforge.net/linear-algebra), which contains some additional matrix function routines.

- **thfm**: trigonometric and hyperbolic functions and their inverses. It implements textbook definitions, in terms of `expm`, `logm`, and `sqrtm`.
- **funm**: general matrix function via diagonalization.

### 12 Scilab

Scilab [78] is another open source PSE. Scilab syntax is similar to that of MATLAB and a code translator is available to convert code from MATLAB to Scilab. Scilab contains several matrix function routines.

- **expm**: matrix exponential using block diagonalization with a Padé approximant applied to each block.
- **logm**: matrix logarithm via diagonalization.
- **sqrtm**: matrix square root via diagonalization.
- Trigonometric and hyperbolic functions, which are implemented via their definitions in terms of the matrix exponential.
- **power**: matrix power using diagonalization for non-integer powers.

### 13 Maple

Maple contains some matrix function routines in its `LinearAlgebra` package. The matrix functions are computed symbolically using polynomial interpolation at the matrix eigenvalues.

- **MatrixExponential**: exponential of a matrix.
- **MatrixPower**: general (non-integer) power of a matrix.
- **MatrixFunction**: general function of a matrix. The function is supplied in the form of an analytic expression by the user.

### 14 Mathematica

Mathematica evaluates the functions listed below for numeric or symbolic matrices.

- **MatrixFunction**: evaluates a general matrix function. The Schur–Parlett algorithm (Davies and Higham, 2003) [23] is used for numeric matrices (derivatives are computed symbolically), and the Jordan form is used for symbolic matrices.
- **MatrixExp**: the matrix exponential is computed by scaling and squaring (Higham, 2005) [46], [48]. This function can also compute the action of the matrix exponential on a vector, using Krylov methods.
- **MatrixLog**: the matrix logarithm is evaluated via a Schur decomposition for numeric matrices or the Jordan form for symbolic matrices.
15 R

Three packages from CRAN (the Comprehensive R Archive Network) include codes for matrix functions.

15.1 Expm

Goulet, Dutang, Maechler, Firth, Shapira, and Stadelmann have written the R package expm [37]. It contains codes not only for the matrix exponential but also for the logarithm and square root.

- **expm**: the default method corresponds to option `Higham08.b`, which uses the algorithm of (Higham, 2005) [46], [48] with balancing. The option `AlMohy-Hi09` implements the algorithm of Al-Mohy and Higham [4]. Also available are options for using an eigendecomposition and other Padé and Taylor-based methods.

- **expAtv**: uses a Krylov method of Sidje (1998) [80] to compute the action of the matrix exponential on a vector.

- **expmCond**: computes or approximates the 1-norm or Frobenius norm condition number of the matrix exponential using the algorithm of Al-Mohy and Higham (2009) [3] or methods from [47, Sec. 3.4].

- **expmFrechet**: Fréchet derivative of matrix exponential (Al-Mohy and Higham, 2009) [3].

- **logm**: matrix logarithm by inverse scaling and squaring (Higham, 2008) [47, Alg. 11.9].

- **sqrtm**: matrix square root by the Schur method (Björck and Hammarling, 1983) [19].

- **matpow**: positive integer powers via binary powering.

15.2 pbdDMAT


15.3 Matrix

The Matrix R package [14] contains a code `expm` that is “a translation of the implementation of the corresponding Octave function contributed to the Octave project by A. Scottedward Hodel”. The `expm` code in the Expm package (see Section 15.1) is to be preferred.

16 ϕ Functions in Fortran 95

Koikari (2009) [63] has written Fortran 95 software for computing the functions \( \varphi_\ell(z) = \sum_{k=0}^{\infty} \frac{z^k}{(k + \ell)!} \) by scaling and squaring and by a block Schur–Parlett algorithm. The code is available as a supplement to the paper on the ACM website.
17 NAG Library

The NAG Library [67] has a large set of matrix function routines in its Chapter F01, covering computation of matrix functions and their Fréchet derivatives and estimation of the condition numbers of matrix functions.

- NAG Fortran Library Mark 23 and NAG Toolbox for MATLAB Mark 23 (released 2011):
  - Matrix exponential using scaling and squaring algorithm (Higham, 2005) [46], [48].
  - Function of real symmetric or Hermitian matrix via eigendecomposition.

- NAG C Library Mark 23, released 2012, also contains:
  - Schur–Parlett algorithm for general functions and for cos, sin, cosh, sinh, exp (Davies and Higham, 2003) [23].
  - Matrix logarithm by Schur–Parlett algorithm with inverse scaling and squaring algorithm (Higham, 2008) [47].

- NAG Fortran Library Mark 24 and the NAG Toolbox for MATLAB Mark 24 (released 2013) also contain:
  - Action of the matrix exponential by scaled Taylor series algorithm (Al-Mohy and Higham, 2011) [6].
  - Condition number estimation in the 1-norm for general matrix functions and for cos, sin, cosh, sinh, exp.

- The NAG C Library Mark 24 (2014), NAG Fortran Library Mark 25 (2015), and NAG Toolbox for MATLAB Mark 25 (2015) also contain:
  - Improved scaling and squaring algorithms for matrix exponential and logarithm (Al-Mohy and Higham, 2009, 2012) [4], [7], with computation of the logarithm of a real matrix in real arithmetic (Al-Mohy, Higham, and Relton, 2013) [8].
  - Matrix square root using Schur method with blocking (including real arithmetic algorithm of Higham [45]) [19], [26].
  - Matrix power $A^p, p \in \mathbb{R}$, via Schur–Padé algorithm (Higham and Lin, 2011, 2013) [53], [54].
  - Latest Fréchet derivative and condition number algorithms for the matrix exponential, logarithm, and real powers [3], [8], [54].

Documentation can be found at: http://www.nag.co.uk/support_documentation.asp. A complete list of all of the NAG matrix function routines, together with their Mark of introduction and the algorithms used, is given in Table 3.

18 SLICOT

SLICOT (Subroutine Library in Systems and Control Theory) [81] includes Fortran 77 codes for three matrix functions.

- MB05MD computes the exponential of a real matrix by diagonalization, optionally with balancing.
Table 3: Matrix Function Routines in the NAG Library.

<table>
<thead>
<tr>
<th>Short Name</th>
<th>Long Name</th>
<th>Purpose</th>
<th>Arithmetic</th>
<th>Algorithms Used</th>
<th>Marks of Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>F01EC</td>
<td>nag_real_gen_matrix_exp</td>
<td>Exponential</td>
<td>Real</td>
<td>Scaling &amp; squaring [4]</td>
<td>FL22 &amp; CL9</td>
</tr>
<tr>
<td>F01ED</td>
<td>nag_real_symm_matrix_exp</td>
<td>Exponential, symmetric matrix</td>
<td>Real</td>
<td>Diagonalization</td>
<td>FL23 &amp; CL9</td>
</tr>
<tr>
<td>F01EF</td>
<td>nag_matop_real_symm_matrix_fun</td>
<td>Symmetric matrix function</td>
<td>Real</td>
<td>Diagonalization</td>
<td>FL23 &amp; CL23</td>
</tr>
<tr>
<td>F01EK</td>
<td>nag_matop_real_gen_matrix_fun_std</td>
<td>sin, cos, sinh, cosh or exp</td>
<td>Real</td>
<td>Schur–Parlett [23]</td>
<td>FL24 &amp; CL23</td>
</tr>
<tr>
<td>F01EL</td>
<td>nag_matop_real_gen_matrix_fun_num</td>
<td>General user-provided function</td>
<td>Real</td>
<td>Schur–Parlett [23]</td>
<td>FL24 &amp; CL24</td>
</tr>
<tr>
<td>F01EM</td>
<td>nag_matop_real_gen_matrix_fun_usd</td>
<td>General user-provided function</td>
<td>Real</td>
<td>Schur–Parlett [23]</td>
<td>FL24 &amp; CL23</td>
</tr>
<tr>
<td>F01EN</td>
<td>nag_matop_real_gen_matrix_sqrt</td>
<td>Square root</td>
<td>Real</td>
<td>[19], [45], [26]</td>
<td>FL25 &amp; CL24</td>
</tr>
<tr>
<td>F01EP</td>
<td>nag_matop_real_tri_matrix_sqrt</td>
<td>Upper triangular square root</td>
<td>Real</td>
<td>[19], [45], [26]</td>
<td>FL25 &amp; CL24</td>
</tr>
<tr>
<td>F01EQ</td>
<td>nag_matop_real_gen_matrix_pow</td>
<td>General real power</td>
<td>Real</td>
<td>[53], [54]</td>
<td>FL25 &amp; CL24</td>
</tr>
<tr>
<td>F01FD</td>
<td>nag_complex_gen_matrix_exp</td>
<td>Exponential, Hermitian matrix</td>
<td>Complex</td>
<td>Diagonalization</td>
<td>FL23 &amp; CL23</td>
</tr>
<tr>
<td>F01FF</td>
<td>nag_matop_complex_symm_matrix_fun</td>
<td>Hermitian matrix function</td>
<td>Complex</td>
<td>Diagonalization</td>
<td>FL23 &amp; CL23</td>
</tr>
<tr>
<td>F01FK</td>
<td>nag_matop_real_gen_matrix_fun_std</td>
<td>sin, cos, sinh, cosh or exp</td>
<td>Complex</td>
<td>Schur–Parlett [23]</td>
<td>FL24 &amp; CL23</td>
</tr>
<tr>
<td>F01FL</td>
<td>nag_matop_real_gen_matrix_fun_num</td>
<td>General user-provided function</td>
<td>Complex</td>
<td>Schur–Parlett [23]</td>
<td>FL24 &amp; CL24</td>
</tr>
<tr>
<td>F01FM</td>
<td>nag_matop_real_gen_matrix_fun_usd</td>
<td>General user-provided function</td>
<td>Complex</td>
<td>Schur–Parlett [23]</td>
<td>FL24 &amp; CL23</td>
</tr>
<tr>
<td>F01FN</td>
<td>nag_matop_real_gen_matrix_sqrt</td>
<td>Square root</td>
<td>Complex</td>
<td>[19], [26]</td>
<td>FL25 &amp; CL24</td>
</tr>
<tr>
<td>F01FP</td>
<td>nag_matop_real_tri_matrix_sqrt</td>
<td>Upper triangular square root</td>
<td>Complex</td>
<td>[19], [26]</td>
<td>FL25 &amp; CL24</td>
</tr>
<tr>
<td>F01FQ</td>
<td>nag_matop_real_gen_matrix_pow</td>
<td>General real power</td>
<td>Complex</td>
<td>[53], [54]</td>
<td>FL25 &amp; CL24</td>
</tr>
<tr>
<td>F01GB</td>
<td>nag_matop_real_gen_matrix_actexp_rcomm</td>
<td>Action of matrix exponential</td>
<td>Real</td>
<td>rev comm¹</td>
<td>FL24 &amp; CL24</td>
</tr>
<tr>
<td>F01HB</td>
<td>nag_matop_complex_gen_matrix_actexp_rcomm</td>
<td>Action of matrix exponential</td>
<td>Complex</td>
<td>rev comm¹</td>
<td>FL24 &amp; CL24</td>
</tr>
<tr>
<td>F01JA</td>
<td>nag_matop_real_gen_matrix_cond_std</td>
<td>sin, cos, sinh, cosh, exp cond</td>
<td>Real</td>
<td>Schur–Parlett [23]</td>
<td>FL24 &amp; CL24</td>
</tr>
<tr>
<td>F01JB</td>
<td>nag_matop_real_gen_matrix_cond_num</td>
<td>User function, condition</td>
<td>Real</td>
<td>Schur–Parlett [23]</td>
<td>FL24 &amp; CL24</td>
</tr>
<tr>
<td>F01JC</td>
<td>nag_matop_real_gen_matrix_cond_usd</td>
<td>User function, condition</td>
<td>Real</td>
<td>Schur–Parlett [23]</td>
<td>FL24 &amp; CL24</td>
</tr>
<tr>
<td>F01JD</td>
<td>nag_matop_real_gen_matrix_cond_sqrt</td>
<td>Square root, condition</td>
<td>Real</td>
<td>[19], [45], [26]</td>
<td>FL25 &amp; CL24</td>
</tr>
</tbody>
</table>

¹“Rev comm” denotes a reverse communication interface.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Type</th>
<th>Context</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>F01JE</td>
<td>General real power, condition</td>
<td>Real</td>
<td>[54]</td>
<td>FL25 &amp; CL24</td>
</tr>
<tr>
<td>F01JF</td>
<td>Real power, Fréchet derivative</td>
<td>Real</td>
<td>[54]</td>
<td>FL25 &amp; CL24</td>
</tr>
<tr>
<td>F01JG</td>
<td>Exponential, condition number</td>
<td>Real</td>
<td>[3]</td>
<td>FL25 &amp; CL24</td>
</tr>
<tr>
<td>F01JJ</td>
<td>Logarithm, condition number</td>
<td>Real</td>
<td>[7], [8]</td>
<td>FL25 &amp; CL24</td>
</tr>
<tr>
<td>F01JK</td>
<td>Logarithm, Fréchet derivative</td>
<td>Real</td>
<td>[7], [8]</td>
<td>FL25 &amp; CL24</td>
</tr>
<tr>
<td>F01KA</td>
<td>sin, cos, sinh, cosh, exp cond</td>
<td>Complex</td>
<td>Schur–Parlett [23]</td>
<td>FL24 &amp; CL24</td>
</tr>
<tr>
<td>F01KB</td>
<td>User function, condition</td>
<td>Complex</td>
<td>Schur–Parlett [23]</td>
<td>FL24 &amp; CL24</td>
</tr>
<tr>
<td>F01KC</td>
<td>User function, condition</td>
<td>Complex</td>
<td>Schur–Parlett [23]</td>
<td>FL24 &amp; CL24</td>
</tr>
<tr>
<td>F01KD</td>
<td>Square root, condition</td>
<td>Complex</td>
<td>[19], [26]</td>
<td>FL25 &amp; CL24</td>
</tr>
<tr>
<td>F01KE</td>
<td>General real power, condition</td>
<td>Complex</td>
<td>[54]</td>
<td>FL25 &amp; CL24</td>
</tr>
<tr>
<td>F01KF</td>
<td>Real power, Fréchet derivative</td>
<td>Complex</td>
<td>[54]</td>
<td>FL25 &amp; CL24</td>
</tr>
<tr>
<td>F01KG</td>
<td>Exponential, condition number</td>
<td>Complex</td>
<td>[3]</td>
<td>FL25 &amp; CL24</td>
</tr>
<tr>
<td>F01KH</td>
<td>Exponential, Fréchet derivative</td>
<td>Complex</td>
<td>[3]</td>
<td>FL25 &amp; CL24</td>
</tr>
<tr>
<td>F01KJ</td>
<td>Logarithm, condition number</td>
<td>Complex</td>
<td>[7], [8]</td>
<td>FL25 &amp; CL24</td>
</tr>
<tr>
<td>F01KK</td>
<td>Logarithm, Fréchet derivative</td>
<td>Complex</td>
<td>[7], [8]</td>
<td>FL25 &amp; CL24</td>
</tr>
</tbody>
</table>
• **MB05ND** computes the matrix exponential and an integral involving the matrix exponential for a real matrix

• **MB05OD** computes the exponential of a real matrix by Ward’s version of the scaling and squaring algorithm (1977) [83] and returns an accuracy estimate.

19 **Python**

19.1 **SciPy**

SciPy [60] is a Python package for scientific computing. It has a number of matrix function codes.

• **sparse.linalg.expm**: matrix exponential by scaling and squaring algorithm (Al-Mohy and Higham, 2009) [4].

• **linalg.expm3, linalg.expm2, linalg.expm**: matrix exponential using Taylor series, eigenvalue decomposition, and scaling and squaring (Higham, 2005) [46], respectively.

• **linalg.logm**: matrix logarithm via inverse scaling and squaring algorithm (Al-Mohy and Higham, 2012) [7].

• **linalg.sinm, linalg.cosm, linalg.tanm, linalg.sinhm, linalg.coshm, linalg.tanhm**: implemented in terms of **linalg.expm**.

• **linalg.fsum, linalg.fmul, linalg.fadd, linalg.fatanh, linalg.fasinh, linalg.facosh, linalg.fatanh**: implemented in terms of **linalg.expm**.

• **linalg.funm**: unblocked Schur–Parlett algorithm [36, Alg. 9.1.1], [47, Alg. 4.1.3].

• **linalg.fractional_matrix_power**: arbitrary real power of matrix by Schur–Padé algorithm (Higham and Lin, 2011, 2013) [53], [54].

• **linalg.signm**: matrix sign function via Newton iteration [47, Sec. 5.3].

• **linalg.expm_frechet**: Fréchet derivative of matrix exponential by scaling and squaring algorithm (Al-Mohy and Higham, 2009) [3].

• **linalg.expm_cond**: Frobenius norm condition number of matrix exponential.

• **linalg.sqrtm**: matrix square root by blocked version of Schur method of Björck and Hammarling (1983) [19], [26].

• **linalg.expm_multiply**: action of matrix exponential on a vector or matrix via scaled Taylor series algorithm (Al-Mohy and Higham, 2011) [6].

• **linalg.polar**: polar decomposition via the singular value decomposition [47, Chap. 8].

19.2 **Python: SymPy**

SymPy [82] is a Python library for symbolic mathematics. It contains some numerical matrix function capabilities in its **mpmath** module and variable precision arithmetic is supported. The precision is set using either **mp.prec** (to set the binary precision, measured in bits) or **mp.dps** (to set the decimal precision).

• **mpmath.expm**: matrix exponential by scaling and squaring with Taylor series or Padé approximant.
• `mpmath.logm`: matrix logarithm evaluated via inverse scaling and squaring and a Taylor series.

• `mpmath.sinm, mpmath.cosm`: matrix sine and cosine implemented via the matrix exponential.

• `mpmath.sqrtm`: matrix square root evaluated using the Denman–Beavers (1976) iteration [27].

• `mpmath.powm`: $A^p$ for $p \in \mathbb{C}$, implemented as $e^{p \log A}$.

Note that both SymPy and SciPy are available in Sage [76], an open source Python-based mathematical software package.

19.3 Other Python Functions

Python implementations of the algorithms in Deadman (2015) [24], for computing the condition number of $f(A)b$, are available at https://github.com/edvindeadman/fAbcond.

20 Java


21 Julia

Julia [18], [61] is an open-source, high-level, dynamic programming language designed specifically for high-performance numerical and scientific computing. Its extensive mathematical function library is largely written in Julia itself, but also includes calls to other libraries such as LAPACK and OpenBLAS. Four matrix function routines are available in the Julia Standard Library:

• `expm`: matrix exponential using the scaling and squaring algorithm of Higham (2005) [46], [48].

• `logm`: matrix logarithm by inverse scaling and squaring algorithm of Al-Mohy and Higham (2012) [7].

• `sqrtm`: matrix square root using the Schur method of Björck and Hammarling (1983) [19].

22 C++: Eigen

Niesen has written a matrix functions module for the Eigen C++ template library for linear algebra [69].

• `MatrixBase::exp()`: matrix exponential by scaling and squaring algorithm (Higham, 2005) [46], [48].

• `MatrixBase::sin(), MatrixBase::sinh(), MatrixBase::cos(), MatrixBase::cosh()` and `MatrixBase::matrixFunction()` are all based on the Schur–Parlett algorithm (Davies and Higham, 2003) [23].
• **MatrixBase::log()**: matrix logarithm by the Schur–Parlett algorithm with inverse scaling and squaring [47, Alg. 11.11].

• **MatrixBase::pow()**: real matrix powers $A^t (t \in \mathbb{R})$ using the Schur–Padé algorithm (Higham and Lin, 2011) [53].

• **MatrixBase::sqrt()**: matrix square root by Schur method (Björck and Hammarling, 1983) [19] and the real Schur method (Higham, 1987) [45].

## 23 The GNU Scientific Library

The GNU Scientific Library (GSL) is an open-source numerical library written in C (although wrappers exist for many other programming languages) [35]. GSL includes an undocumented function `gsl_linalg_exponential_ss` to compute the matrix exponential. This routine uses scaling and squaring and a truncated Taylor series. The scaling and truncation parameters are chosen as in Moler and Van Loan (2003) [66, Method 3].

### Acknowledgements

We thank Awad Al-Mohy, Michael Croucher, Massimiliano Fasi, Stefan Güttel, Sven Hammarling, Peter Kandolf, Lijing Lin, and Samuel Relton for comments and suggestions.

### References


