To appear in ACM - Transactions on Mathematical Software (1998)

EXPOKIT: Software Package for Computing Matrix Exponentials

Roger B. SIDJE
UNIVERSITY OF QUEENSLAND

EXPOKIT provides a set of routines aimed at computing matrix exponentials. More precisely, it computes either a small matrix exponential in full, the action of a large sparse matrix exponential on an operand vector, or the solution of a system of linear ODEs with constant inhomogeneity. The backbone of the sparse routines consists of Krylov subspace projection methods (Arnoldi and Lanczos processes) and that is why the toolkit is capable of coping with sparse matrices of large dimension. The software handles real and complex matrices and provides specific routines for symmetric and Hermitian matrices. The computation of matrix exponentials is a numerical issue of critical importance in the area of Markov chains and furthermore, the computed solution is subject to probabilistic constraints. In addition to addressing general matrix exponentials, a distinct attention is assigned to the computation of transient states of Markov chains.

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra; G.1.7 [**Numerical Analysis**]: Ordinary Differential Equations—***initial value problems***; G.4 [**Mathematics of Computing**]: Mathematical Software

General Terms: Algorithms

Additional Key Words and Phrases: Matrix exponential, Krylov methods, Markov chains

1. INTRODUCTION

Evaluating the action of the matrix exponential operator on an arbitrary vector is an important problem that arises in mathematics, physics and engineering. For example, in control theory, a central problem consists in solving the system of ODEs

$$\begin{cases} \dfrac{dw(t)}{dt} &= Aw(t) + Bu(t), \;\; t \geq 0 \\ w(0) &= v, \qquad\qquad\qquad \text{initial condition} \end{cases}$$

Address: Department of Mathematics, University of Queensland, Brisbane QLD 4072, Australia, rbs@maths.uq.edu.au, http://www.maths.uq.edu.au/expokit.

where $w(t)$ is the *state vector*, $A \in \mathrm{IR}^{n\times n}$ is the *state companion matrix*, $u(t) \in \mathrm{IR}^m$ is the *control* and $B \in \mathrm{IR}^{n\times m}$. The state vector $w(t)$ is given explicitly by

$$w(t) = e^{tA}v + \int_0^t e^{(t-s)A}Bu(s)ds.$$

As another example, a successful and widely used way of modeling the behavior of many physical systems consists in enumerating the (mutually exclusive) states in which the system may be at a given time and then, describing the interaction between these states. The analysis of performance of these systems requires quantifying several attributes, including reliability, availability, and performability. Frequently, finite state Markov chains constitute powerful mathematical tools that are used to achieve these purposes [Berman and Plemmons 1979; Ciardo et al. 1990; Neuts 1981; Seneta 1981; Stewart 1994]. Under suitable hypotheses, the evolution of several physical systems may be governed by the Chapman-Kolmogorov system of differential equations:

$$\begin{cases} \dfrac{dw(t)}{dt} &= Aw(t),\ t \in [0,\ T] \\ w(0) &= v, \qquad \text{initial probability distribution} \end{cases}.$$

Its analytic solution is $w(t) = e^{tA}v$ and represents the *transient probability distribution* of the Markov chain. The coefficient matrix A is an *infinitesimal generator* of order n, where n is the number of states in the Markov chain. Thus $A \in \mathrm{IR}^{n\times n}$, with elements $a_{ij} \geq 0$ when $i \neq j$, and $a_{jj} = -\sum_{\substack{i=1\\ i\neq j}}^{n} a_{ij}$. Useful performance parameters can be derived after computing transient solutions. In particular, the ith component of $w(t)$ is the probability that the physical system will be in the state numbered i at time t. An in-depth analysis of the behavior of a given system is usually done by investigating the influence of model parameters (e.g., the initial condition v) on the solution vector $w(t)$. As the number of trials grows, the amount of computation explodes. Moreover, in real-time simulations, calculations must be done within a very short delay and, as the size of the problem grows, only efficient computers, running efficient software, are able to perform at the desired speed.

The computation of the matrix exponential is often not an end in itself. It can also be used in a preconditioner-like manner to find the rightmost eigenvalues, e.g., Saad [1992b, p.277], Meerbergen and Sadkane [1996], and it can be the elemental building-block in some solution techniques of ODEs and DAEs, e.g., Lawson [1967], Hochbruck and Lubich [1996], Hochbruck et al. [1996].

A direct way to define the matrix exponential $\exp(A)$ is undoubtedly through the exponential power series expansion $\exp(A) = \sum_{k=0}^{\infty} A^k/k!$ whose convergence is guaranteed for any square matrix A. Other definitions exist; each alternative being of theoretical and/or practical interest depending on the desired end and also on the class of matrices under consideration. Although papers and textbooks dealing with matrix exponentials are abundant in the literature, Moler and Van Loan [1978] cautioned that practical implementations are 'dubious' in the sense that a sole implementation might not be entirely reliable for all classes of problems. The explicit computation of the matrix exponential function is difficult when the argument matrix is of large norm or of wide departure from normality. Besides, the difficulty grows worse when the order of the matrix is large.

The case where the matrix is of moderate dimension has benefited from extensive studies such as Ward [1977], Parlett and Ng [1985], Fassino [1993], just to name a few, which resulted into implementations that appear satisfactory in several uncontrived practical instances. Traditional approaches are either based on Taylor series, rational Padé approximations or the Schur decomposition – the later being used in conjunction with a recurrence relationship amongst the elements of the exponential of the triangular factor [Parlett 1976; Parlett and Ng 1985]. Apart from the Taylor series approach, these techniques require the full matrix and thus they are applicable only when the order of the matrix does not exceed a few hundreds.

EXPOKIT deals with small matrices as well as large sparse matrices. The software package handles real and complex matrices and supplies specific routines for symmetric and Hermitian matrices. A distinct attention is given to matrices arising from Markov chains. The backbone of the sparse routines consists of Krylov subspace projection techniques (the well-known Arnoldi and Lanczos processes) and that is why the toolkit is capable of coping with sparse matrices of large dimension. It seems these techniques were long established amongst Chemical Physicists without much justification other than their satisfactory behavior. But since the theoretical characterization of Gallopoulous and Saad [1992] and Saad [1992a], they are gaining wide acceptance and their use here and elsewhere shows remarkable success, see e.g., Gallopoulous and Saad [1992], Saad [1992a], Leyk and Roberts [1995], Sidje and Stewart [1996], and related references. Recently, Hochbruck and Lubich [1996] made a major contribution by improving the *a priori* error bounds considerably, thus yielding a better understanding as to why the techniques are working so well. Within the field of Markov chains, relevant studies appear in Sidje [1994] and Philippe and Sidje [1995].

The knowledge gathered in recent research warrants now the production of intelligible software that will assist scientists in tackling some of the significantly large problems arising in modeling and engineering. An attempt in this direction briefly appears in Saad [1994] but EXPOKIT is the first comprehensive package aiming specifically at this purpose right from the outset. A number of scientists have already shown interest in using the software and their positive feedback somewhat reflects the appropriateness and usefulness of the approach. Investigations towards parallel algorithms that will enable for addressing much larger problems have already been initiated in Sidje [1994, 1996] but we shall concentrate here on the serial framework. We start by presenting the foundation of the algorithms (Sections 2 through 5). We subsequently illustrate their utilization (Section 6) and indicate where the portable routines implementing them can be retrieved (Section 7).

2. THE FULL CASE

EXPOKIT provides routines for computing small matrix exponentials. They are based on rational Chebyshev or Padé approximations. The rational Chebyshev approximation comes from extending the minimax Chebyshev theory to rational fractions. Cody, Meinardus and Varga [1969] have stated and solved the problem: find $C_{pp}(x) \equiv N_{pp}(x)/D_{pp}(x)$ such that

$$||C_{pp}(x) - e^{-x}||_{L^\infty[0,+\infty)} = \min_{r_{pp}\in\mathcal{R}_{pp}} \max_{x\in[0,+\infty)} |r_{pp}(x) - e^{-x}| \qquad (1)$$

where $\mathcal{R}_{pq}$ denotes the class of (p,q)-degree rational functions. For $p = 1, 2, ..., 14$, the coefficients of the best approximants $N_{pp}(x)$ and $D_{pp}(x)$ were computed and listed. Subsequently, Carpenter, Ruttan and Varga [1984] extended the list up to $p = 30$. The positive point here is that it is possible to compute (and keep) the poles $\{\theta_i\}$ and to consider the partial fraction expansion $C_{pp}(x) = \alpha_0 + \sum_{i=1}^{p} \alpha_i/(x - \theta_i)$ to obtain directly

$$\exp(\tau H)y \approx C_{pp}(-\tau H)y = \alpha_0 y - \sum_{i=1}^{p} (\tau H + \theta_i I)^{-1} \alpha_i y.$$

The poles $\{\theta_i\}$ and the coefficients $\{\alpha_i\}$ were computed and listed in Gallopoulos and Saad [1992] for $p = 10$ and $p = 14$. Expokit only implements the case $p = 14$. When τH is real, an LU decomposition is saved for complex conjugate pair of poles. Thus the whole cost can be reduced by half. When $\|\tau H\|_2$ becomes large, we have found this approach less stable than the diagonal Padé approximations which are theoretically A-acceptable, see e.g., Iserles and Nørsett [1991]. Given the domain of applicability specified in (1), the approximations above are more suited for symmetric negative definite matrices, in which case

$$\| \exp(\tau H) - C_{pp}(-\tau H) \|_2 \leq \lambda_{pp}$$

where λ_{pp} is an explicitly known constant referred to as the *uniform rational Chebyshev constant* [Varga 1990]. It is now known that $\lambda_{pp} \approx 10^{-p}$ which means that a type (p, p)-approximation yields p-digit accuracy. However, this bound does not hold for a general matrix H having a complex spectrum and/or a poorly conditioned system of eigenvectors. As shown in Fig. 1, if employed where not intended, these Chebyshev approximants become invalid and large errors occur. Arguably, one can always apply a shift to the exponential as $e^z = e^s e^{(z-s)} \approx e^s C_{pp}(-(z-s))$, but e^s quickly becomes large and magnifies the error.

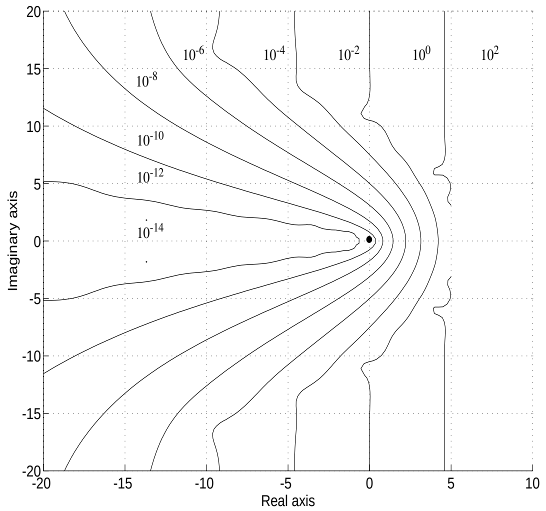

Fig. 1. Contour plot of $|e^z - C_{14,14}(-z)|$ for z in the left-plane.

The type (p,q) Padé approximation for e^x is the (p,q)-degree rational function $P_{pq}(x) \equiv N_{pq}(x)/D_{pq}(x)$ obtained by solving the algebraic equation $\sum_{k=0}^{\infty} x^k/k! - N_{pq}(x)/D_{pq}(x) = \mathcal{O}(x^{p+q+1})$, i.e., $P_{pq}(x)$ must match the Taylor series expansion up to order $p+q$. For stability and economy of computation, it is advantageous to set $p=q$. Indeed, *for the exponential function*, we fortuitously have

$$P_{pp}(x) = \frac{N_{pp}(x)}{N_{pp}(-x)} \tag{2}$$

where $N_{pp}(x) = \sum_{k=0}^{p} c_k x^k$ with $c_0 = 1, c_k = c_{k-1}\frac{p+1-k}{(2p+1-k)k}$. As noted in Sidje [1994], it is more economical to use the following irreducible form rather than (2):

$$P_{pp}(x) = \begin{cases} 1 + 2\dfrac{x\sum_{k=0}^{p/2-1} c_{2k+1}x^{2k}}{\sum_{k=0}^{p/2} c_{2k}x^{2k} - x\sum_{k=0}^{p/2-1} c_{2k+1}x^{2k}} & \text{if } p \text{ is even} \\ -1 - 2\dfrac{\sum_{k=0}^{(p-1)/2} c_{2k}x^{2k}}{x\sum_{k=0}^{(p-1)/2} c_{2k+1}x^{2k} - \sum_{k=0}^{(p-1)/2} c_{2k}x^{2k}} & \text{if } p \text{ is odd} \end{cases} . \tag{3}$$

The Horner evaluations of the numerator and the denominator in (3) need half the operations of (2) and a careful implementation uses only four extra matrices with no shuffling of data (see the routine **_PADM** in this toolkit). However beyond these considerations, there is a major drawback of the Padé approximations: they are only accurate near the origin so that the approximation of $\exp(\tau H)$ is not valid when $\|\tau H\|_2$ is too large. Moreover when τH has widely spread eigenvalues, the computation of $P_{pp}(\tau H)$ involves an ill-conditioned linear system. Fortunately these problems disappear and an acceptable accuracy can be obtained even for a small degree p if we make use of the exponential property $\exp(\tau H) = (\exp(2^{-s}\tau H))^{2^s}$ which is referred to as 'scaling and squaring'. We then use the approximation $\exp(\tau H) \approx (P_{pp}(2^{-s}\tau H))^{2^s}$ evaluated by repeated squaring. The principal drawback of the resulting algorithm may come from the fact that if $s \gg 1$ then, the computed squares can be contaminated by rounding errors and the cost becomes large. An inverse error analysis made in Moler and Van Loan [1978] has shown that if $\|2^{-s}\tau H\|_\infty \le 1/2$ then

$$\left(P_{pp}\left(2^{-s}\tau H\right)\right)^{2^s} = \exp(\tau H + E) \tag{4}$$

where

$$\frac{\|E\|_\infty}{\|\tau H\|_\infty} \le \frac{(p!)^2}{(2p)!(2p+1)!}\left(\frac{1}{2}\right)^{2p-3} \approx \begin{cases} 0.34 \times 10^{-15} & (\mathrm{p} = 6) \\ 0.11 \times 10^{-18} & (\mathrm{p} = 7) \\ 0.27 \times 10^{-22} & (\mathrm{p} = 8) \end{cases} .$$

Thus a value of $p = 6$ is generally satisfactory. Other roundoff error considerations are studied in detail in Ward [1977].

As we shall see below, large sparse techniques rely upon small dense methods. Of the two small dense methods just described, EXPOKIT uses by default the Padé method at the core of its large sparse techniques – users can decide otherwise. The preference went to the Padé method because it resolves the definite and indefinite cases equally while the Chebyshev method is apt to compute the direct action of the matrix exponential, $\exp(\tau H)y$, when it is known in advance that H is symmetric negative definite.

3. THE SPARSE CASE

ALGORITHM 3.1. *Compute* $w(t) = \exp(tA)v$

$w := v$; $t_k := 0$;
$\overline{H}_{m+2} := \mathrm{zeros}[m+2, m+2]$;
while $t_k < t$ **do**
  $v := w$; $\beta := \|v\|_2$;
  $v_1 := v/\beta$;
  for $j := 1 : m$ **do** {*Arnoldi process*}
    $p := Av_j$;
    for $i := 1 : j$ **do**
      $h_{ij} := v_i^* p$;
      $p := p - h_{ij} v_i$;
    end
    $h_{j+1,j} := \|p\|_2$;
    if $h_{j+1,j} \leq tol\|A\|$ **happy-breakdown**
    $v_{j+1} := p/h_{j+1,j}$;
  end
  $\overline{H}(m+2, m+1) := 1$;
  repeat
    $\tau :=$ *step-size;*
    $F_{m+2} := \exp(\tau \overline{H}_{m+2})$;
    $w := \beta V_{m+1} F(1{:}m+1, 1)$;
    err_loc := local error estimate;
  until *err_loc* $\leq \delta\, tol$;
  $t_k := t_k + \tau$;
end

Consider a large sparse n-by-n matrix A (real or complex), a n-vector v and a scalar $t \in \mathbb{R}$ (presumably the 'time'). For ease of presentation, consider $t > 0$ since $\exp(-tA) = \exp(t(-A))$. Alg. 3.1 computes an approximation of $w(t) = \exp(tA)v$. The algorithm purposely sets out to compute the matrix exponential times a vector rather than the matrix exponential in isolation. The underlying principle is to approximate

$$w(t) = e^{tA}v = v + \frac{(tA)}{1!}v + \frac{(tA)^2}{2!}v + \cdots \tag{5}$$

by an element of the Krylov subspace $\mathcal{K}_m(tA, v) = \mathrm{Span}\{v, (tA)v, \ldots, (tA)^{m-1}v\}$, where m, the dimension of the Krylov subspace, is small compared to n, the order of the principal matrix (usually $m \leq 50$ while n can exceed many thousands). The approximation being used is

$$\tilde{w}(t) = \beta V_{m+1} \exp(t\overline{H}_{m+1}) e_1 \tag{6}$$

where e_1 is the first unit basis vector, $\beta = \|v\|_2$; $V_{m+1} = [v_1, \ldots, v_{m+1}]$ and $\overline{H}_{m+1} = [h_{ij}]$ are, respectively, the orthonormal basis and the upper Hessenberg matrix resulting from the well-known Arnoldi process, see e.g., Saad [1992b], Golub and Van Loan [1996]. When the matrix is symmetric or Hermitian, the Arnoldi process is replaced by the Lanczos process and computational savings occur. If the minimal degree of v is some integer ν ($\nu \leq m \leq n$) then, an invariant subspace is found and the approximation $\beta V_\nu \exp(tH_\nu)e_1$ is *exact.* This situation is usually referred to as a 'happy breakdown' and, in exact arithmetic, happens at least when $m = n$.

The mathematical basis of the selected approximation has been documented [Gallopoulos and Saad 1992; Saad 1992a; Sidje 1994; Hochbruck and Lubich 1996]. In particular, it has been established that this approximation is better than the m-fold Taylor expansion – which highlights that for the same amount of matrix-vector products, the Krylov approximation is better than the Taylor approximation (even though the Krylov approach involves more local calculation, notably the Gram-Schmidt sweeps). In fact, Hochbruck and Lubich [1996] have shown that the error in the Krylov method behaves like $\mathcal{O}(e^{m-t\|A\|_2}(t\|A\|_2/m)^m)$ when $m \geq 2t\|A\|_2$. They gave sharper bounds depending on the class of the matrix and/or the location and shape of its spectrum which illustrated that the technique can work quite well even for moderate m. Using Chebyshev series expansion, Druskin and Knizhnerman [1995] and Stewart and Leyk [1996] have also provided other relevant insights on the error bound for the symmetric case.

The distinctive feature to underscore in the Krylov approximation is that the original large problem (5) has been converted to the small problem (6) which is more desirable. The explicit computation of $\exp(t\overline{H}_{m+1})$ is performed using known dense algorithms. In the present toolkit, this nucleus is handled either with the irreducible rational Padé method combined with scaling-and-squaring, or the uniform rational Chebyshev approximation as shown earlier.

The description of the algorithm would be incomplete if we omit to mention that, in reality, due to stability and accuracy requirements, $w(t)$ is not computed in one stretch. On the contrary, a time-stepping strategy along with error estimations is embedded within the process. Typically, the algorithm evolves with the integration scheme

$$\begin{cases} w(0) & = v \\ w(t_{k+1}) & = e^{(t_k+\tau_k)A}v \;=\; e^{\tau_k A}w(t_k), \quad k = 0, 1, ..., s \end{cases} \tag{7}$$

where

$$\tau_k = t_{k+1} - t_k, \qquad 0 = t_0 < t_1 < \cdots < t_s < t_{s+1} = t.$$

Consequently, in the course of the integration, one can output discrete observations (if they are needed) at no extra cost. Nevertheless, it is clear from (7) that the crux of the problem remains an operation of the form $e^{\tau A}v$, with different v's. The selection of a specific step-size τ is made so that $e^{\tau A}v$ is now effectively approximated by $\beta V_{m+1}\exp(\tau\overline{H}_{m+1})e_1$. Following the procedures of ODEs solvers, an *a posteriori* error control is carried out to ensure that the intermediate approximation is acceptable with respect to expectations on the global error. The starting point of all these critical issues is the following expansion series established in Saad [1992a]:

$$\exp(\tau A)v \;=\; \beta V_m \exp(\tau H_m)e_1 + \beta\tau h_{m+1,m}\sum_{j=1}^{\infty} e_m^* \varphi_j(\tau H_m)e_1(\tau A)^{j-1}v_{m+1} \tag{8}$$

$$= \beta V_{m+1}\exp(\tau\overline{H}_{m+1})e_1 + \beta\sum_{j=2}^{\infty} h_{m+1,m}\tau^j e_m^* \varphi_j(\tau H_m)e_1 A^{j-1}v_{m+1} \tag{9}$$

where $\varphi_0(z) \equiv e^z, \varphi_j(z) \equiv (\varphi_{j-1}(z) - \varphi_{j-1}(0))/z = \sum_{i=0}^{\infty} z^i/(i+j)!, j \geq 1$. The functions φ_j are positive, increasing, and $\varphi_{j+1} \leq \varphi_j/j$ in $[0, +\infty)$. Moreover they become smoother as j increases. A way to control the error and the step-size has been described in Sidje [1994] as we shall now summarize. For a small step-size the next term in (8) is an appropriate estimate of the local truncation error. But this estimate is insufficiently accurate for a large step-size. This is due to the fact that in that case, in magnitude, the terms of the expansion series grow before they decay.

ALGORITHM 3.2. *Local truncation error estimation*

$err1 := \beta |h_{m+1,m} \tau e_m^* \varphi_1(\tau H_m) e_1|$;
$err2 := \beta |h_{m+1,m} \tau^2 e_m^* \varphi_2(\tau H_m) e_1| \, \|A v_{m+1}\|_2$;
if $err1 \gg err2$ **then** $\{$*small step-size: quick convergence*$\}$
  $err := err2$;
else if $err1 > err2$ **then** $\{$*slow convergence*$\}$
  $err := err2 * \frac{1}{1-\frac{err2}{err1}}$;
else $\{$*err1* $<$ *err2: asymptotic convergence*$\}$
  $err := err1$;
endif
$err\_loc := \max(err, roundoff)$;

A study of the asymptotic behavior of the error term in (8) suggested using the estimator above. It is an emulation of the problem of approximating an infinite series by its pth partial sum:

(1) if the terms of the series decrease rapidly (in magnitude) then the next $(p+1)$-st term is an appropriate estimate of the error

(2) if the terms decrease slowly then, the geometric limit is considered

(3) if the terms start decreasing at order P where $P \gg p$, then the $(p+1)$-st term has no meaning. We take the pth term as an estimate of the error. This choice is enlightened in Hochbruck et al. [1996, §6.3].

Now an effective way to cheaply compute the coefficients $h_{m+1,m} \tau^j e_m^* \varphi_j(\tau H_m) e_1$ is shown in the following assertions homologous to Sidje [1994, p.99].

THEOREM 1. *Let* $c \in \mathbb{C}^m$ *and*

$$\widetilde{H}_{m+p} = \begin{pmatrix} H_m & c & 0 & \cdots & 0 \\ & 0 & 1 & \ddots & \vdots \\ & & 0 & \ddots & 0 \\ & & & \ddots & 1 \\ 0 & & & & 0 \end{pmatrix} \in \mathbb{C}^{(m+p)\times(m+p)}$$

then

$$\exp(\tau\widetilde{H}_{m+p}) = \begin{pmatrix} \exp(\tau H_m) & \tau\varphi_1(\tau H_m)c & \tau^2\varphi_2(\tau H_m)c & \cdots & \tau^p\varphi_p(\tau H_m)c \\ & 1 & \frac{\tau}{1!} & \cdots & \frac{\tau^{p-1}}{(p-1)!} \\ & & 1 & \ddots & \vdots \\ & & & \ddots & \frac{\tau}{1!} \\ 0 & & & & 1 \end{pmatrix}.$$

PROOF. Consider the block upper triangular decomposition

$$\widetilde{H}_{m+p} = \begin{pmatrix} H_m & ce_1^* \\ 0 & J \end{pmatrix}, \quad J = \begin{pmatrix} 0 & 1 & & 0 \\ & 0 & \ddots & \\ & & \ddots & 1 \\ 0 & & & 0 \end{pmatrix} = \begin{pmatrix} e_2^* \\ \vdots \\ e_p^* \\ 0 \end{pmatrix}.$$

Then

$$e^{\tau\widetilde{H}_{m+p}} = \begin{pmatrix} e^{\tau H_m} & F \\ 0 & e^{\tau J} \end{pmatrix}, \quad e^{\tau J} = \begin{pmatrix} 1 & \frac{\tau}{1!} & \cdots & \frac{\tau^{p-1}}{(p-1)!} \\ & 1 & \ddots & \vdots \\ & & \ddots & \frac{\tau}{1!} \\ 0 & & & 1 \end{pmatrix}, \quad F = [f_1, ..., f_p].$$

From $\widetilde{H}_{m+p}e^{\tau\widetilde{H}_{m+p}} = e^{\tau\widetilde{H}_{m+p}}\widetilde{H}_{m+p}$ we have $H_mF - FJ = e^{\tau H_m}ce_1^* - ce_1^*e^{\tau J}$. Multiplying on the right by e_j to extract the j-th column, we obtain the recurrence relations

$$\begin{cases} f_1 &= \tau\varphi_1(\tau H_m)c, \\ H_m f_j &= f_{j-1} - \frac{\tau^{j-1}}{(j-1)!}c, \quad 1 < j \le p \end{cases}$$

and by induction $f_j = \tau^{j-1}\Big(\varphi_{j-1}(\tau H_m) - \varphi_{j-1}(0)I\Big)H_m^{-1}c = \tau^j\varphi_j(\tau H_m)c$. $\square$

COROLLARY 1. *Let* $c \in \mathbb{C}^m$ *and*

$$\overline{H}_{m+p} = \begin{pmatrix} H_m & & & & 0 \\ c^* & 0 & & & \\ 0 & 1 & 0 & & \\ \vdots & \ddots & \ddots & \ddots & \\ 0 & \cdots & 0 & 1 & 0 \end{pmatrix} \in \mathbb{C}^{(m+p)\times(m+p)}$$

then

$$\exp(\tau\overline{H}_{m+p}) = \begin{pmatrix} \exp(\tau H_m) & & & & 0 \\ \tau c^*\varphi_1(\tau H_m) & 1 & & & \\ \tau^2 c^*\varphi_2(\tau H_m) & \frac{\tau}{1!} & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ \tau^p c^*\varphi_p(\tau H_m) & \frac{\tau^{p-1}}{(p-1)!} & \cdots & \frac{\tau}{1!} & 1 \end{pmatrix}.$$

PROOF. Applying Theorem 1 with H_m replaced by H_m^* and taking the conjugate transpose of the result yields the assertion. □

Notice that these results remain valid without assumptions on H_m (it needs not be Hessenberg and/or invertible). When setting $c^* = h_{m+1,m}e_m^*$ in particular, the desired coefficients of the expansion series (8) and (9) can be recovered in the first column just below the m-th row of $\exp(\tau\overline{H}_{m+p})$. The monitoring of the step-size τ and the size of the Krylov basis m then follows heuristics of ODE solvers, see e.g., Gustafsson [1991]. We base the step-size selection on the measure

$$\epsilon_k = \|\tilde{\varepsilon}_k\|/\tau_{k-1}, \quad \text{error per unit step (EPUS)}$$

where $\|\tilde{\varepsilon}_k\|$ is the local truncation error approximated by *err*. Unless otherwise specified, $\|\cdot\|$ denotes the euclidian norm. If *tol* denotes the prescribed tolerance to be achieved, the first step-size is chosen to satisfy a known theoretical bound and within the integration process, the step-size is selected by means of the formula

$$\tau_k = \gamma\ (tol/\epsilon_k)^{1/r}\ \tau_{k-1} \tag{10}$$

rounded to 2 significant digits to prevent numerical noise. The value of r is $m-1$ or m depending on whether the error estimate *err* comes from *err1* or *err2* respectively (see Alg. 3.2). A step is rejected if $\epsilon_{k+1} > \delta\, tol$. (The scalars γ and δ are *safety factors* intended to reduce the risk of rejection of the step. They have been assigned the classical values 0.9 and 1.2 respectively – they can be modified by the user.) With this step-size mechanism, the 'accumulated global error' is upper bounded for a fixed integration domain regardless of the number of steps used:

$$\textstyle\sum_k \|\tilde{\varepsilon}_k\| \le \delta \cdot t \cdot tol. \tag{11}$$

Notice that since $\|\tilde{\varepsilon}_k\|$ is estimated by *err* which itself is subject to (4), the error measurement should be understood in a weighted sense. Furthermore, since for $j = 1, ..., k$, $\tilde{\varepsilon}_j$ are local truncation error vectors, i.e., $\tilde{\varepsilon}_j = e^{\tau_{j-1}A}\tilde{w}(t_{j-1}) - \tilde{w}(t_j)$, we obtain $w(t_k) - \tilde{w}(t_k) = \sum_{j=1}^{k} e^{(t_k - t_j)A}\tilde{\varepsilon}_j$ and therefore the error actually satisfies (letting $t \equiv t_k$)

$$\|w(t) - \tilde{w}(t)\| \le \max_j \|e^{(t-t_j)A}\| \cdot \sum_{j=1}^{k} \|\tilde{\varepsilon}_j\| \le \max_{\tau\in[0,t]} \|e^{\tau A}\| \cdot \sum_{j=1}^{k} \|\tilde{\varepsilon}_j\|. \tag{12}$$

It is well-known that $\lim_{\tau\to+\infty} e^{\tau A} = 0$ iff $\alpha(A) < 0$, where $\alpha(A) = \max\{Re(\lambda), \lambda \in \lambda(A)\}$. An upper bound on (12) can be obtained by using $\max_{\tau\in[0,t]} \|e^{\tau A}\| \le \max_{\tau\in[0,t]} e^{\|\tau A\|} \le e^{\|tA\|}$ but this straight bound is very pessimistic and besides, it does not decay with $\alpha(A)$. A review of various bounds for $\|e^{\tau A}\|$ is provided in Van Loan [1977] where it is shown that an effective bound is $e^{\alpha(A)\tau} \le \|e^{\tau A}\| \le e^{\alpha(A)\tau}\sum_{k=0}^{n-1} \|\tau N\|^k/k!$ with N being the strict upper part of the triangular factor of the Schur decomposition of A. The lower bound $\|e^{\tau A}\| = e^{\alpha(A)\tau}$ is attained if A is normal. In general, even if $\alpha(A) < 0$, $\|e^{\tau A}\|$ can still grow before it decays (the so-called 'hump' effect) and whether this situation occurs or not depends on the sign of $\mu(A)$, i.e, $\sup_{\tau\ge 0} \|e^{\tau A}\| = 1$ iff $\mu(A) \le 0$, where $\mu(A) = \lambda_{\max}((A + A^*)/2)$ is the logarithmic norm of A and we recall that $\alpha(A) \le \mu(A)$. Fig. 2 is a detailed characterization of the behavior of $\|e^{\tau A}\|$.

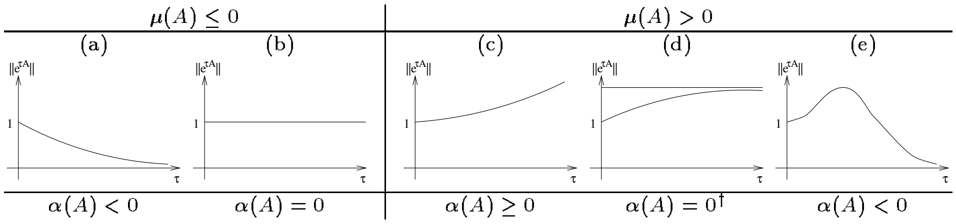

Fig. 2. Behavior of $||e^{\tau A}||$ in relation to $\mu(A)$ and $\alpha(A)$. The dagger (†) means eigenvalues have nonpositive real part and those which are purely imaginary or zero are nondefective.

The routines in EXPOKIT attempt to achieve (11) and thus the user may bear in mind the aforementioned issues, especially the potential discrepancies that may result from (4) or (12). These issues are related to the conditioning of the matrix exponential in itself [Van Loan 1977]. Our control (11) is conservative and together with Alg. 3.2, they prove nicely effective if the problem is not exaggeratedly ill-conditioned. It is also meaningful to observe that the relative error satisfies (recall that $\beta = ||v||$ and $v_1 = v/\beta$)

$$\frac{||w(t) - \tilde{w}(t)||}{||w(t)||} \leq \frac{\max_{\tau \in [0,t]} ||e^{\tau A}||}{||w(t)||} \sum_{j=1}^{k} ||\tilde{\varepsilon}_j|| = \frac{1}{\beta} \frac{||e^{\hat{t}A}\hat{v}||}{||e^{tA}v_1||} \sum_{j=1}^{k} ||\tilde{\varepsilon}_j|| \qquad (13)$$

for some $\hat{t} \in [0, t]$ and $\hat{v}$ such that $||\hat{v}|| = 1$. Therefore if $||e^{\hat{t}A}\hat{v}||/||e^{tA}v_1|| \approx 1$, we can relate (11) to a bound on the relative error. Otherwise if $||e^{\hat{t}A}\hat{v}|| \gg 1$ or $||e^{tA}v_1|| \ll 1$, the upper bound in (13) becomes too large and has little practical value. Upon exit $||e^{tA}v_1||$ is readily approximated by $||\tilde{w}(t)||/\beta$ and as a by-product of the computations, EXPOKIT also returns the following approximation:

$$hump \equiv ||e^{\hat{t}A}\hat{v}|| = \max_{\tau \in [0,t]} ||e^{\tau A}|| = \max_{\tau \in [0,t],\ x \neq 0} \frac{||e^{\tau A}x||}{||x||} \approx \max_{t_j \in [0,t]} \frac{||\tilde{w}(t_j)||}{\beta}.$$

When a step-size is rejected it is reduced by using formula (10) once more but there is also the opportunity (not implemented in this release) of adjusting (increase or decrease) the dimension of the Krylov subspace by the way of the relation

$$m_k = m_{k-1} + \left\lceil \frac{\log(tol/\epsilon_k)}{\log \tau_{k-1}} \right\rceil.$$

As an epilogue to the description of the algorithm, it is worth reminding that the primary matrix interacts only through matrix-vector products. Hence the method is a matrix-free method and conveys all the associated advantages.

4. THE MARKOVIAN CASE

The Markovian context brings other probabilistic considerations. The approximation of $w(t) = \exp(tA)v$ is subject to the constraint that the resulting vector must be a probability vector, thus with components in the range $[0, 1]$ and with sum equal to 1. Since the analytic solution of the Chapman-Kolmogorov system of differential equations is $w(t)$, its computation can be addressed totally in the perspective of ODEs. But general-purpose ODEs solvers do not bring any guarantee either.

Practice shows that they can even produce negative components – which have no meaning in terms of probabilities.

A number of results of interest were provided in Sidje [1994] and Philippe and Sidje [1995] by exploiting the fact that the matrix A is an infinitesimal generator of a Markov chain and satisfies some inherent properties. With the Krylov approach, the sum condition is fulfilled, i.e., $\mathbb{1}^T\tilde{w}(t_k) = 1$ where $\mathbb{1} = (1, ..., 1)^T$. If during the integration process (7) some components of an intermediate approximation happen to be negative, the components under concern are less than the error tolerance (in magnitude), otherwise this intermediate solution would be rejected by the error control. Therefore a handy way to overcome any difficulty is to set the negative components to zero and perform a normalization afterwards. Another way to proceed is to reduce the step-size. It is mathematically guaranteed that the Krylov approximation is a probability vector for small enough step-sizes. Markov chains conform to Fig. 2-(d) and it is shown that the global error in the approximation grows at most linearly, i.e., (12) becomes $\|w(t_k) - \tilde{w}(t_k)\|_1 \leq \sum_{j=1}^{k} \|\tilde{\varepsilon}_j\|_1$. Additionally, it is possible to detect and cope with excessive roundoff errors as follows. Letting $\bar{w}(t_k)$ be the computed value of $\tilde{w}(t_k)$ and using the fact that $\mathbb{1}^T\tilde{w}(t_k) = 1$ and the Hölder inequality $\|\mathbb{1}\|_1\|\tilde{w}(t_k) - \bar{w}(t_k)\|_\infty \geq |\mathbb{1}^T(\tilde{w}(t_k) - \bar{w}(t_k))|$, we have

$$\|\tilde{w}(t_k) - \bar{w}(t_k)\|_\infty \geq \frac{|1 - \mathbb{1}^T\bar{w}(t_k)|}{n} \equiv \textit{roundoff}. \tag{14}$$

Hence if the quantity *roundoff* is far from the machine precision then, the computed approximation $\bar{w}(t_k)$ is too contaminated and the process should be stopped. This is a sufficient condition to indicate a high level of roundoff errors. The indicator is deficient when the vector $\tilde{w}(t_k) - \bar{w}(t_k)$ is orthogonal to $\mathbb{1}$. This may happen but it is expected to be rare. Another interesting feature which is also worth mentioning is the capacity to detect the *stationary probability* distribution $w_\infty = \lim_{t\to\infty} e^{tA}v$ (i.e., the ith component of w_∞ is the probability that the Markov chain will be in the state numbered i at statistical equilibrium). The detection of the steady-state is made possible by the 'happy breakdown'.

The ensuing customization of the generic algorithm incorporating all these aspects proves to be a fairly reliable and versatile algorithm for the computation of transient states of Markov chains. Extensive experiments conducted in Sidje and Stewart [1996] have shown the benefit of the method.

5. THE NONHOMOGENEOUS CASE

Quite often the linear system of ODEs from which the matrix exponential arises is nonhomogeneous and has a constant forcing term:

$$\begin{cases} \dfrac{dw(t)}{dt} &= Aw(t) + u \\ w(0) &= v, \qquad \text{initial condition.} \end{cases} \tag{15}$$

If the matrix A is nonsingular the solution can be written as $w(t) = e^{tA}(v + A^{-1}u) - A^{-1}u$ and so with one extra linear system solve, it can be recovered as described earlier. However, a much cheaper and unrestricted approach which does not depend on the invertibility of A is also possible through techniques similar to that of the homogeneous case. The explicit solution of (15) is

$$w(t) = e^{tA}v + \int_0^t e^{(t-s)A}u ds = e^{tA}v + t\varphi(tA)u \tag{16}$$

where $\varphi(z) \equiv \varphi_1(z) = (e^z - 1)/z = \sum_{i=0}^{\infty} z^i/(i+1)!$. Therefore an integration scheme can be obtained as

$$\begin{cases} w(0) & = v \\ w(t_{k+1}) & = w(t_k + \tau_k) \;=\; \tau_k \varphi(\tau_k A)(Aw(t_k) + u) + w(t_k) \end{cases} . \tag{17}$$

Indeed (16) yields $w(t+\tau) = e^{\tau A}w(t) + \int_0^\tau e^{(\tau-s)A}u ds = e^{\tau A}w(t) + \tau\varphi(\tau A)u = (\tau A\varphi(\tau A) + I)w(t) + \tau\varphi(\tau A)u = \tau\varphi(\tau A)(Aw(t) + u) + w(t)$. Hence the crux of the integration scheme (17) is now an operation of the form $\varphi(\tau A)v$ which can be approximated in a way similar to $\exp(\tau A)v$. In a more precise sense we have the following generalizations.

THEOREM 2. *For every $p \geq 0$,*

$$\begin{aligned} \tau^p\varphi_p(\tau A)v &= \tau^p\beta V_m\varphi_p(\tau H_m)e_1 + \beta\sum_{j=p+1}^{\infty} h_{m+1,m}\tau^j e_m^*\varphi_j(\tau H_m)e_1 A^{j-p-1}v_{m+1} \\ &= \tau^p\beta V_{m+1}\varphi_p(\tau\overline{H}_{m+1})e_1 + \beta\sum_{j=p+2}^{\infty} h_{m+1,m}\tau^j e_m^*\varphi_j(\tau H_m)e_1 A^{j-p-1}v_{m+1}. \end{aligned}$$

PROOF. For $p = 0$, we recover directly (8) and (9). Then from (8) we can write

$$(\tau A\varphi_1(\tau A)+I)v = \beta V_m(\tau H_m\varphi_1(\tau H_m)+I)e_1+\beta\sum_{j=1}^{\infty} h_{m+1,m}\tau^j e_m^*\varphi_j(\tau H_m)e_1 A^{j-1}v_{m+1}.$$

Therefore using the fundamental relation $V_m H_m = AV_m - h_{m+1,m}v_{m+1}e_m^*$, we have

$$A\Big(\tau\varphi_1(\tau A)v - \tau\beta V_m\varphi_1(\tau H_m)e_1 - \beta\sum_{j=2}^{\infty} h_{m+1,m}\tau^j e_m^*\varphi_j(\tau H_m)e_1 A^{j-2}v_{m+1}\Big) = 0$$

and since this holds for any v, we necessarily have

$$\begin{aligned} \tau\varphi_1(\tau A)v &= \tau\beta V_m\varphi_1(\tau H_m)e_1 + \beta\sum_{j=2}^{\infty} h_{m+1,m}\tau^j e_m^*\varphi_j(\tau H_m)e_1 A^{j-2}v_{m+1} && (18) \\ &= \tau\beta V_{m+1}\varphi_1(\tau\overline{H}_{m+1})e_1 + \beta\sum_{j=3}^{\infty} h_{m+1,m}\tau^j e_m^*\varphi_j(\tau H_m)e_1 A^{j-2}v_{m+1} && (19) \end{aligned}$$

with the 'corrected' equality (19) following from the fact that

$$\varphi_1(\tau\overline{H}_{m+1}) = \begin{pmatrix} \varphi_1(\tau H_m) & 0 \\ h_{m+1,m}\tau e_m^*\varphi_2(\tau H_m) & 1 \end{pmatrix}$$

which can be proved thanks to the relation $\varphi_1(\tau\overline{H}_{m+1})\overline{H}_{m+1} = \overline{H}_{m+1}\varphi_1(\tau\overline{H}_{m+1})$. The proof for higher p proceeds by induction in a similar way. □

It is then justified from (19) that one can consider the approximation

$$\tau\varphi_1(\tau A)v \approx \tau\beta V_{m+1}\varphi_1(\tau\overline{H}_{m+1})e_1 \tag{20}$$

so that in essence, the original large problem (16) reduces to the computation of the small-sized problem $\varphi_1(\tau\overline{H}_{m+1})e_1$ and techniques discussed in Section 2 can be applied. For instance, $\varphi_1(z)$ can be recovered from the Chebyshev partial fraction expansion of e^z as $\varphi_1(z) = (e^z - 1)/z \approx (C_{pp}(-z) - 1)/z = \sum_{i=1}^{p}(\alpha_i/\theta_i)/(z + \theta_i)$ which allows for obtaining $\varphi_1(\tau\overline{H}_{m+1})e_1$ directly. However, the Chebyshev approximations are not always valid and so it is desirable to use the robust alternative proposed in Theorem 1 which provides the information needed for the approximation as well as the extra coefficients of the expansion series useful for the estimation of the error. Interestingly also, no special coding is needed. The computation of all these is done at once with a direct call to one of the small matrix exponential routine within the package. Finally, if for $j = 1, ..., k$, we consider local truncation error vectors $\tilde{\varepsilon}_j = \tau_{j-1}\varphi(\tau_{j-1}A)(A\tilde{w}(t_{j-1}) + u) + \tilde{w}(t_{j-1}) - \tilde{w}(t_j)$ then, we can infer $w(t_k) - \tilde{w}(t_k) = \sum_{j=1}^{k} e^{(t_k - t_j)A}\tilde{\varepsilon}_j$ and a similar analysis to that done earlier holds.

6. EXAMPLES

This section shows experiments with some routines of EXPOKIT. All the examples are executed on a standard SUN4 workstation. The size of the Krylov basis is $m = 30$ everywhere. Drivers reproducing each example are available in the package: *sample_m.f*, *sample_z.f*, *sample_g.f*, *sample_b.f*, *sample_p.f* for the first through to the fifth example respectively. Another driver (not shown here), *sample_d.f*, is available and it depicts the utilization of routines targeted to small dense problems.

6.1 A binary Markov example

A Markov system is being modeled in this first example. The system consists of a collection of components that are binary, i.e., they have only two states, 'good' and 'bad'. If c is the number of components in the system, the number of states in which the system can be is $n = 2^c$. Rules for constructing automatically the infinitesimal generator are given in Clarotti [1984]. The matrix chosen for the example is of order $n = 1,024$ with $nz = 11,264$ (i.e., the number of components is $c = 10$). We set $t = 10$, $v = (1, 0, ..., 0)^T$ and $tol = 10^{-10}$.

Output with DMEXPV
$w(1{:}5) =$
0.13051504168006
1.2181403890139D-02
1.3517629316864D-02
1.2616454029073D-03
1.7712755656580D-02
CPU Time = 81 sec

Output with DGEXPV
$w(1{:}5) =$
0.13051504168006
1.2181403890139D-02
1.3517629316864D-02
1.2616454029073D-03
1.7712755656580D-02
CPU Time = 76 sec

This example aims at illustrating that DGEXPV (the version for General matrices) is often slightly faster than DMEXPV (the version for Markov matrices). There is an overhead when enforcing the probability constraint in DMEXPV. Both versions are provided in EXPOKIT and so in the end, the choice of which one to use is left to the satisfaction of the Markovian analyst – depending on whether the impending priority is guaranteed reliability or speed at some risk.

6.2 A Hermitian example

Output with ZGEXPV	Output with ZHEXPV
$w(1{:}5)$ =	$w(1{:}5)$ =
(305250.75378623, -113.89587171937)	(305250.75378623, -113.89587171924)
(6.7491920417580D-08, 6.8657867370494D-08)	(6.7491920417570D-08, 6.8657867370493D-08)
(-3.0966858825505, 4.6959293376943)	(-3.0966858825505, 4.6959293376943)
(-1.6776892228097D-06, -7.8892940444656D-06)	(-1.6776892228097D-06, -7.8892940444658D-06)
(9.1675656352188D-06, -2.6860802938635D-06)	(9.1675656352187D-06, -2.6860802938635D-06)
CPU Time = 49 sec	CPU Time = 13 sec

The matrix of this example comes from a symmetric pattern of order $n = 5,300$ of the Harwell-Boeing collection. We take the pattern BCSPWR10 and fill-in a Hermitian matrix. Both real and imaginary parts are filled using uniform random numbers within the range $[-5, +5]$. The Coordinates (COO) storage is used to hold the matrix and this yields $nz = 21,842$ as the effective number of non-zero elements (the conjugate transpose is included explicitly). We set $t = 1$, $v = (1, 0, ..., 0, 1)^T$ and $tol = 10^{-5}$. The example illustrates the gain of the routine tailored for Hermitian matrices over the general routine.

6.3 A nonsymmetric example

DGEXPV with COO	DGEXPV with CCS	DGEXPV with CRS
$w(1{:}5)$ =	$w(1{:}5)$ =	$w(1{:}5)$ =
6464.4009480840	6464.4009480840	6464.4009480840
4828.8899856184	4828.8899856184	4828.8899856184
5450.5845968253	5450.5845968253	5450.5845968253
5592.7251537899	5592.7251537899	5592.7251537899
4575.6820139573	4575.6820139573	4575.6820139573
CPU Time = 17 sec	CPU Time = 14 sec	CPU Time = 15 sec

The matrix of this example is the unsymmetric matrix ORANI678 of the Harwell-Boeing collection [Duff et al. 1989]. The order is $n = 2,529$ and the number of non-zero elements is $nz = 90,158$. This example illustrates the impact of the sparse data storage. DGEXPV is executed with $t = 10$, $v = (1, ..., 1)^T$, $tol = 0$, using three different data structures (see [Barret et al. 1994; Saad 1994]): Coordinates (COO), Compressed Row Storage (CRS), Compressed Column Storage (CCS). By inputing tol as zero, the code automatically set tol to the square-root of the machine epsilon which is about $1.5\,10^{-8}$ actually. It appears that the CCS format fits better the sparsity pattern of ORANI678. It is often hard to predict which structure fits better an arbitrary sparse matrix. Our observation on various computer architectures suggests that the CRS format is usually a compromise with unknown erratic sparsity patterns.

6.4 A forward-backward example

Forward with DSEXPV	Backward with DSEXPV
$w^+(1{:}5)$ =	$w^-(1{:}5)$ =
3456.5698306801	1.0000000000001
7.3427169843682	1.0000000000003
4094.7323184931	1.0000000000003
1275.0417533589	1.0000000000003
2939.0163458165	1.0000000000003
CPU Time = 11 sec	CPU Time = 10 sec

The matrix of this example is the symmetric matrix GR3030 of the Harwell-Boeing collection. The order is $n = 900$ and the number of non-zero elements is $nz = 4,322$. We set $t = 1$, $v = (1, ..., 1)^T$ and $tol = 10^{-10}$. This example shows the computation of $e^{-tA}e^{tA}v$. Two runs are performed in tandem. The first run computes $w^+ = e^{tA}v$ (forward) and the output of this computation is passed as the input operand of the next run which computes $w^- = e^{-tA}w^+$ (backward). In exact arithmetic, the final result should be v. The output obtained is very close. We see also that the runs took comparable time.

6.5 A nonhomogeneous example

DGPHIV with $u = 0, v = \mathbb{1}$	DGPHIV with $u = \mathbb{1}, v = 0$	DGPHIV with $u = \mathbb{1}, v = \mathbb{1}$
$w(1:5) =$	$w(1:5) =$	$w(1:5) =$
6464.4009480834	4962.2511930989	11426.652141182
4828.8899856180	3658.2951908942	8487.1851765123
5450.5845968249	4152.9628069217	9603.5474037466
5592.7251537895	4279.0497069393	9871.7748607288
4575.6820139570	3442.7780495196	8018.4600634766
CPU Time = 13 sec	CPU Time = 13 sec	CPU Time = 14 sec

This is an illustration of an integration with respect to the phi function, i.e., it computes $w = \exp(tA)v + t\varphi(tA)u$, where $\varphi(z) = (\exp(z) - 1)/z$. The matrix of this example is the unsymmetric matrix ORANI678 (CCS format) that was used in §6.3. We also set $t = 10$, $tol = 0$ and thus the square-root of the machine unit is the accuracy tolerance that will be effectively used. We have set the input parameters so as to make a certain number of comparisons. By setting $u = 0$ in the first run, the expected answer is $w^{[1]} = \exp(tA)v$. This is confirmed by a look at §6.3. With the second run, the answer should be $w^{[2]} = t\varphi(tA)u$. To check this we compute the difference $\Delta = \|(u + Aw^{[2]}) - w^{[1]}\|/\|w^{[1]}\| \approx 10^{-14}$ which correlates quite well with the tolerance used. The third run is a general run where neither u nor v are null. We observe that the runs took comparable times. It has been our observation that computing $\exp(tA)v$ via the PHI routine may sometimes be faster than via the EXP routine. This may be related to the fact that the right-hand side function in (17), i.e., the phi function, is smoother than the right-hand side function in (7), i.e., the exponential function – thus allowing for bigger step-sizes. However, they produce slightly different numerical results and the EXP routine should be preferred.

7. THE SOFTWARE

A World Wide Web site, *http://www.maths.uq.edu.au/expokit*, has been set up and it allows for accessing the package (or parts of it). A mailing list has also been set up, *expokit@maths.uq.edu.au*, and it provides a medium through which users of the package are kept informed of upgrades in the software. Users can subscribe/unsubscribe in this open forum, exchange ideas and contribute in the extension of the package.

Coding was done in Fortran 77 while using building blocks from the two well-known scientific libraries BLAS and LAPACK. As a precaution, necessary steps have been taken to ensure an unconstrained portability of the software. We include an essential minimal substitute to BLAS and LAPACK. In this way the software is self-contained and can still operate even if these libraries are not yet installed in the environment of the user. The naming scheme used in LAPACK has inspired

Table 1. Nomenclature.

T	indicates the data type
D	double precision
Z	complex*16

Z	indicates the task scope
M	Matrix output
V	Vector output

X	indicates the matrix
G	General
H	Hermitian
S	Symmetric
M	Markov chain matrix
N	Hessenberg

YYY	indicates the task
PAD	Padé
CHB	Chebyshev
EXP	Exponential evaluation
PHI	φ evaluation (nonhomogeneous poblem)

the naming convention used for the routines in the package. However, because the chart of matrices involved in EXPOKIT is not as exhaustive as that of LAPACK (e.g., all sparse routines in EXPOKIT are matrix-free), we introduced slight differences. Unless otherwise stated, the names of the Fortran 77 routines in EXPOKIT match the pattern **TXYYYZ** in accordance with the description on Table 1.

Not all combinations are possible. The choice of the pattern above complies with the 6-length Fortran 77 recommendation and achieves the dual purpose of avoiding conflicts with LAPACK's names (a cross-check has been done to ascertain this) and facilitating/harmonizing forthcoming extensions of the package. The header in Table 2 below is prototype to the routines in the package. Whenever possible readability and user-friendliness are in the first place – provided efficiency is not impeded (e.g., the ordinary user is relieved from handling bulky parameters while the advanced user can fetch them if needed).

To sustain flexibility, the MATLAB counterparts of the algorithms have been coded and are also included in the distribution. Thus research codes written in MATLAB can perform better by taking advantage of the new techniques. More to the point, we supply MATLAB scripts allowing experimental matrices composed within the user-friendly environment of MATLAB to be stored into files suitably formatted for loading directly into the Fortran versions. It is not our intention to elaborate on operating directives. A README file is included in the distribution for this purpose. We shall instead outline some top-level routines of relevance that will give a sense of concreteness to the reader.

__EXPV computes $w = \exp(tA)v$, t can be positive or negative. The action of the matrix exponential operator on the operand vector is evaluated directly, i.e., $\exp(tA)$ is not computed in isolation before being applied to v. The variants dealing with symmetric/Hermitian matrices use the cheaper Lanczos process instead of the Arnoldi process. Of particular interest is **DMEXPV**, the customised version for Markov Chains. This means that a check is done within this program to ensure that the resulting vector w is a probability vector.
IMPORTANT: The well-known transition rate matrix Q of a Markov chain satisfies $Q\mathbb{1} = 0$ where $\mathbb{1} = (1, ..., 1)^T$; **DMEXPV** uses the matrix-vector product $y = Ax = Q^T x$, i.e., the *transpose* of Q times a vector. Failure to remember this leads to wrong results.

__PHIV computes $w = \exp(tA)v + t\varphi(tA)u$ which is the solution of the nonhomogeneous linear ODE problem $w' = Aw + u$, $w_0 = v$. The input parameter t can be positive or negative. If $u = 0$ this procedure is mathematically equivalent to **__EXPV** and if $v = 0$ it computes $t\varphi(tA)u$. The variants dealing with symmetric/Hermitian matrices use the cheaper Lanczos process instead of the Arnoldi process.

__PADM computes the matrix exponential $\exp(tH)$ in full where H is a relatively small matrix. The underlying method is the irreducible rational Padé approximation to the exponential function combined with scaling-and-squaring. This procedure can be used in its own right to compute the exponential of a small matrix in full. Computational savings are made in the symmetric/Hermitian variants.

__CHBV computes $\exp(tH)y$ where H is a relatively small matrix using the partial fraction expansion of the uniform rational Chebyshev approximation of type $(14, 14)$ to the exponential function over the negative real axis. The calculation of each fraction is done using Gaussian elimination with pivoting. The Chebyshev method is intended to compute the direct action of the matrix exponential on a vector when H is negative definite.

blas.f/lapack.f minimal substitute to BLAS and LAPACK. In this way EXPOKIT can be used even if these library packages are not yet installed in the environment of the user. Guidelines for their use (or non-use) are provided in the Makefile.

expv.m/mexpv.m MATLAB counterparts of **__EXPV** and **DMEXPV**.

mat2coo.m utility program aimed at storing a MATLAB matrix into a text file that can be exploited directly. The matrix is stored under the Coordinates storage format (COO).

mat2crs.m/mat2ccs.m utility programs aimed at storing a MATLAB matrix into a text file under the Compressed Row Storage (CRS) format and the Compressed Column Storage (CCS) format.

NOTE: Fortran subroutines for matrix-vector multiplication when the matrix is stored using either of the above formats are included in this distribution. They are referred to as **__coov**, **__crsv**, and **__ccsv**. For optimum performances, however, if users have an a priori knowledge of the structure of their matrices, it is recommended to store them using the most advantageous format and to devise the most advantageous matrix-vector multiplication routine. A Fortran converter subroutine, **__cnvr**, that transforms any of the above formats to another one is supplied in this distribution.

loadcoo.m/loadcrs.m/loadccs.m utility Matlab scripts for loading matrices stored using either **mat2coo.m**, **mat2crs.m**, or **mat2ccs.m.**

8. CONCLUSION

This paper has detailed the main current ingredients of the EXPOKIT package. The work has benefited from latest results in Krylov approximation methods to the matrix exponential and new results and extensions were established that enabled effective local and global error estimation, step-size control, efficient implementation and tailorization (e.g., Markov chains). The software is self-contained and accepts real and complex data. It deals with small matrices as well as large sparse matrices and it makes computational advantage of symmetric/Hermitian matrices. Since the computation of the matrix exponential is difficult, the paper has also outlined the issues that can contribute in the understanding of the limitations of the methods if they fail to produce accurate enough results.

It is hoped that numerical methods that need the small matrix exponential in full or that make use of the action of the large sparse matrix exponential and/or the phi function in their own right or as a building-block will find in the software robust and ready-to-use routines that will be helpful. Examples of such methods include Lawson [1967], Leyk and Roberts [1995], Hochbruck et al. [1996], Meerbegen and Sadkane [1996]. Feedback from current users of the package certainly shows that the package is quite propitious. In addition to the inevitable conventional maintenance, efforts will be undertaken to include refinements and extensions that will keep the software up-to-date with new discoveries. In particular, studies in Sidje [1994, 1996] have already illustrated how an efficient parallelization can be achieved.

REFERENCES

BARRET, R., BERRY, M., CHAN, T. F., DEMMEL, J., DONATO, J., DONGARRA, J., EIJKHOUT, V., POZO, R., ROMINE, C., AND VAN DER VORST, H. 1994. *Templates for the solutions of linear systems: building blocks for iterative methods.* SIAM.

BERMAN, A. AND PLEMMONS, R. J. 1979. *Nonnegative Matrices in the Mathematical Sciences.* Academic Press, New York.

CARPENTER, A. J., RUTTAN, A., AND VARGA, R. S. 1984. Extended numerical computations on the 1/9 conjecture in rational approximation theory. In *Lecture Notes in Mathematics 1105* (Berlin, 1984), pp. 383–411. Springer-Verlag.

CIARDO, G., MARIE, R. A., SERICOLA, B., AND TRIVEDI, K. S. 1990. Performability analysis using semi-Markov reward processes. *IEEE Trans. Comput. 39*, 10, 1251–1264.

CLAROTTI, C. A. 1984. The Markov approach to calculating system reliability: Computational problems. In A. SERRA AND R. E. BARLOW Eds., *International School of Physics "Enrico Fermi", Varenna, Italy* (Amsterdam, 1984), pp. 54–66. North-Holland Physics Publishing.

CODY, W. J., MEINARDUS, G., AND VARGA, R. S. 1969. Chebyshev rational approximation to $\exp(-x)$ in $[0,+\infty)$ and applications to heat conduction problems. *J. Approx. Theory 2*, 50–65.

DRUSKIN AND KNIZHNERMAN. 1995. Krylov subspace approximations of eigenpairs and matrix functions in exact and computer arithmetic. *Num. Lin. Alg. Appl. 2*, 205–217.

DUFF, I. S., GRIMES, R. G., AND LEWIS, J. G. 1989. Sparse matrix test problems. *ACM Trans. Math. Softw. 15*, 1–14.

FASSINO, C. 1993. *Computation of Matrix Functions.* Ph. D. thesis, Università di Pisa, Genova - Udine, Italia.

GALLOPOULOS, E. AND SAAD, Y. 1992. Efficient solution of parabolic equations by Krylov approximation methods. *SIAM J. Sci. Stat. Comput. 13*, 5, 1236–1264.

GOLUB, G. H. AND VAN LOAN, C. F. 1996. *Matrix Computations* (third ed.). The Johns Hopkins University Press, Baltimore and London.

GUSTAFSSON, K. 1991. Control theoretic techniques for stepsize selection in explicit Runge-Kutta methods. *ACM Trans. Math. Softw. 17*, 4, 533–554.

HOCHBRUCK, M. AND LUBICH, C. 1996. On Krylov subspace approximations to the matrix exponential operator. *SIAM J. Numer. Anal. (to appear)*.

HOCHBRUCK, M., LUBICH, C., AND SELHOFER, H. 1996. Exponential integrators for large systems of differential equations. *SIAM J. Sci. Comput. (to appear)*.

ISERLES, A. AND NØRSETT, S. P. 1991. *Order Stars.* Chapman & Hall.

LAWSON, J. D. 1967. Generalized Runge-Kutta processes for stable systems with large Lipschitz constants. *SIAM J. Numer. Anal. 4*, 372–380.

LEYK, T. AND ROBERTS, S. G. 1995. Numerical solving of a free boundary phase field model using Krylov subspace method. In W. SCIENTIFIC Ed., *Computational Techniques and Applications: CTAC95* (Melbourne, Australia, 1995).

MEERBERGEN, K. AND SADKANE, M. 1996. Using Krylov approximations to the matrix exponential operator in Davidson's method. Technical Report TW 238, Katholieke Universiteit Leuven. Department of Computer Science.

MOLER, C. B. AND VAN LOAN, C. F. 1978. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Review. 20*, 4, 801–836.

Table 2. Header of DGEXPV.

subroutine	DGEXPV(n, m, t, v, w, tol, anorm, wsp,lwsp, iwsp,liwsp, matvec, itrace,iflag) implicit none integer n, m, lwsp, liwsp, itrace, iflag, iwsp(liwsp) double precision t, tol, anorm, v(n), w(n), wsp(lwsp) external matvec
Purpose	DGEXPV computes w = exp(t·A)·v - for a General matrix A
Arguments	
n	(input) order of the principal matrix A.
m	(input) maximum size for the Krylov basis.
t	(input) time at which the solution is needed (can be < 0).
v(n)	(input) given operand vector.
w(n)	(output) computed approximation of exp(t·A)·v.
tol	(input/output) the requested accuracy tolerance on w. If on input tol = 0.0d0 or tol is too small (tol $\leq$ eps) the internal value sqrt(eps) is used, and tol is set to sqrt(eps) on output ('eps' denotes the machine epsilon). ('Happy breakdown' is assumed if $h_{j+1,j} \leq$ anorm·tol)
anorm	(input) an approximation of some norm of A.
wsp(lwsp)	(workspace) lwsp $\geq$ n·(m+1)+n+(m+2)^2+4·(m+2)^2+ideg+1 (actually, ideg = 6)
iwsp(liwsp)	(workspace) liwsp $\geq$ m+2
matvec	external subroutine for matrix-vector multiplication. synopsis: matvec(x, y) double precision x(*), y(*) computes: y(1:n) $\leftarrow$ A·x(1:n) where A is the principal matrix.
itrace	(input) running mode. 0=silent, 1=print step-by-step info
iflag	(output) exit flag. < 0 - bad input arguments $= 0$ - no problem $= 1$ - maximum number of steps reached without convergence $= 2$ - requested tolerance was too high
Accounts	Upon exit, an interested user may retrieve accounts on the computations. They are located in wsp and iwsp as indicated below:

location	mnemonic	description
iwsp(1)	nmult	number of matrix-vector multiplications used
iwsp(2)	nexph	number of Hessenberg matrix exponential evaluated
iwsp(3)	nscale	number of repeated squaring involved in Pade
iwsp(4)	nstep	number of integration steps used up to completion
iwsp(5)	nreject	number of rejected step-sizes
iwsp(6)	ibrkflag	set to 1 if 'happy breakdown' and 0 otherwise
iwsp(7)	mbrkdwn	if 'happy breakdown', basis-size when it occurred
wsp(1)	step_min	minimum step-size used during integration
wsp(2)	step_max	maximum step-size used during integration
wsp(3)	dummy	
wsp(4)	dummy	
wsp(5)	x_error	maximum among all local truncation errors
wsp(6)	s_error	global sum of local truncation errors
wsp(7)	tbrkdwn	if 'happy breakdown', time when it occurred
wsp(8)	t_now	integration domain successfully covered
wsp(9)	hump	approximation of the 'hump', i.e., $\max_{\tau\in[0,t]} \|\|e^{\tau A}\|\|$
wsp(10)	$\|\|w\|\|/\|\|v\|\|$	scaled euclidian norm of the solution w.

NEUTS, M. F. 1981. *Matrix Geometric Solutions in Stochastic Models: An Algorithmic Approach.* The Johns Hopkins University Press, Baltimore.

PARLETT, B. N. 1976. A recurrence among the elements of functions of triangular matrices. *Linear Algebra Appl. 14*, 117–121.

PARLETT, B. N. AND NG, K. C. 1985. Development of an accurate algorithm for $\exp(B\tau)$. Technical Report PAM-294, Center for Pure and Applied Mathematics, University of California, Berkeley.

PHILIPPE, B. AND SIDJE, R. B. 1995. Transient solutions of Markov processes by Krylov subspaces. In W. J. STEWART Ed., *2nd International Workshop on the Numerical Solution of Markov Chains* (Raleigh, NC, USA, 1995).

SAAD, Y. 1992a. Analysis of some Krylov subspace approximations to the matrix exponential operator. *SIAM J. Numer. Anal. 29*, 1, 208–227.

SAAD, Y. 1992b. *Numerical Methods for Large Eigenvalue Problems.* John Wiley & Sons, Manchester University Press.

SAAD, Y. 1994. SPARSKIT: a basic tool kit for sparse matrix computation, version 2. Technical report, Computer Science Department, University of Minnesota, Minneapolis MN55455.

SENETA, E. 1981. *Non-negative Matrices and Markov Chains* (2nd ed.). Springer-Verlag, New York.

SIDJE, R. B. 1994. *Parallel Algorithms for Large Sparse Matrix Exponentials: application to numerical transient analysis of Markov processes.* Ph. D. thesis, Univ. of Rennes 1.

SIDJE, R. B. 1996. Alternatives for parallel Krylov basis computation. *Num. Lin. Alg. Appl. (to appear).*

SIDJE, R. B. AND STEWART, W. J. 1996. A survey of methods for computing large sparse matrix exponentials arising in Markov chains. Technical Report TR-96-06, Computer Science Department, North Carolina State University, Raleigh NC 27695-8206.

STEWART, D. E. AND LEYK, T. S. 1996. Error estimates for Krylov subspace approximations of matrix exponentials. *J. Comput. Appl. Math. (to appear).*

STEWART, W. J. 1994. *Introduction to the Numerical Solution of Markov Processes.* Princeton University Press.

VAN LOAN, C. F. 1977. The sensitivity of the matrix exponential. *SIAM J. Numer. Anal. 14*, 6, 971–981.

VARGA, R. S. 1990. *Scientific Computation on Mathematical Problems and Conjectures.* No. 60 in CBMS-NSF regional conference series in applied mathematics. SIAM.

WARD, R. C. 1977. Numerical computation of the matrix exponential with accuracy estimate. *SIAM J. Numer. Anal. 14*, 4, 600–610.