A Krylov subspace algorithm for evaluating the φ-functions appearing in exponential integrators

JITSE NIESEN
University of Leeds
and
WILL M. WRIGHT
La Trobe University

We develop an algorithm for computing the solution of a large system of linear ordinary differential equations (ODEs) with polynomial inhomogeneity. This is equivalent to computing the action of a certain matrix function on the vector representing the initial condition. The matrix function is a linear combination of the matrix exponential and other functions related to the exponential (the so-called φ-functions). Such computations are the major computational burden in the implementation of exponential integrators, which can solve general ODEs. Our approach is to compute the action of the matrix function by constructing a Krylov subspace using Arnoldi or Lanczos iteration and projecting the function on this subspace. This is combined with time-stepping to prevent the Krylov subspace from growing too large. The algorithm is fully adaptive: it varies both the size of the time steps and the dimension of the Krylov subspace to reach the required accuracy. We implement this algorithm in the MATLAB function `phipm` and we give instructions on how to obtain and use this function. Various numerical experiments show that the `phipm` function is often significantly more efficient than the state-of-the-art.

Categories and Subject Descriptors: G.1.3 [**Numerical linear algebra**]: ; G.4 [**Mathematical Software**]: Algorithm design and analysis; Documentation; G.1.7 [**Ordinary Differential Equations**]: Initial value problems; G.1.8 [**Partial Differential Equations**]: Finite difference methods; Method of lines; Spectral methods

General Terms: Theory, Algorithms
Additional Key Words and Phrases: Krylov subspace methods, exponential integrators, matrix exponential

1. INTRODUCTION

In recent years there has been a resurgence of interest in a class of numerical methods for the solution of ordinary differential equations (ODEs) known as exponential integrators. These are intended to be used on ODEs which can be split into a stiff linear part and a non-stiff nonlinear part. This splitting can be done once or sev-

Author's addresses: Jitse Niesen, School of Mathematics, University of Leeds, Leeds, LS2 9JT, United Kingdom. Email: jitse@maths.leeds.ac.uk, Will M. Wright, Department of Mathematics and Statistics, La Trobe University, 3086 Victoria, Australia. Email: w.wright@latrobe.edu.au.
JN was supported by Australian Research Council grant DP0559083.

eral times, as need be. As their name suggests, exponential integrators use the matrix exponential and various related matrix functions, generally referred to as φ-functions, within the numerical integrator. The computational cost of exponential integrators is dominated by the need to evaluate these φ-functions, and this task is the subject of this paper. For a recent review of exponential integrators, we refer the reader to Minchev and Wright [2005].

ODEs of the form exploited by exponential integrators often arise when semi-discretizing a partial differential equation. Typically, the matrix appearing in the linear part is large and sparse. For such matrices, Krylov subspace methods provide an extremely efficient means of evaluating the action of an arbitrary matrix function on a vector, without the need to evaluate the matrix function itself. The use of Krylov subspace approximations for the action of matrix exponential was pioneered by several authors in the late eighties and early nineties, notably Friesner et al. [1989] and Gallopoulos and Saad [1992]. Hochbruck and Lubich [1997] show that for particular classes of matrices, the convergence of the action of the matrix exponential on a vector is faster than that for the solution of the corresponding linear system. This suggests that exponential integrators can be faster than implicit methods, because computing the action of the matrix exponential and solving linear systems are the two fundamental operations for exponential integrators and implicit methods, respectively. This analysis breathed life back in the class of exponential integrators invented in the early sixties, which had been abandoned in the eighties due to their excessive computational expense.

In their well-known paper on computing the matrix exponential, Moler and Van Loan [2003] write: "The most extensive software for computing the matrix exponential that we are aware of is EXPOKIT." This refers to the work of Sidje [1998], who wrote software for the computation of the matrix exponential of both small dense and large sparse matrices. The software uses a Krylov subspace approach in the large sparse setting. Computing the action of the matrix exponential is equivalent to solving a linear ODE, and Sidje uses this equivalence to apply time-stepping ideas from numerical ODE methods in EXPOKIT. The time-steps are chosen with the help of an error estimate due to Saad [1992]. Sidje [1998] extends this approach to the computation of the first φ-function, which appears in the solutions of linear ODEs with constant inhomogeneity. This was further generalized by Sofroniou and Spaletta [2007] to polynomial inhomogeneities (or, from another point of view, to general φ-functions); their work is included in the latest version of MATHEMATICA.

In all the approaches described above the size of the Krylov subspace is fixed, generally to thirty. Hochbruck et al. [1998] explain in their landmark paper one approach to adapt the size of the Krylov subspace. In this paper, we develop a solver which combines the time-stepping ideas of Sidje [1998] and Sofroniou and Spaletta [2007] with the adaptivity of the dimension of the Krylov subspace as described by Hochbruck et al. [1998]. We do not examine the case of small dense matrices; this has been studied by Koikari [2007] and Skaflestad and Wright [2009]. This algorithm described in this paper can be used as a kernel for the efficient implementation of certain classes of exponential integrators. We also recommend the recent book by Higham [2008], which discusses many of the issues related to matrix functions and their computation.

The approach followed in this paper, reducing large matrices to smaller ones by projecting them on Krylov subspace, is not the only game in town. Other possibilities are restricted-denominator rational Krylov methods [Moret 2007], the real Leja point method [Caliari and Ostermann 2009], quadrature formulas based on numerical inversion of sectorial Laplace transforms [López-Fernández 2010], and contour integration [Schmelzer and Trefethen 2007]. These methods are outside the scope of this paper, but we intend to study and compare them in future work.

The outline of this paper follows. In Section 2 we present several useful results regarding the φ-functions. The algorithm we have developed is explained in Section 3, where we present the Krylov subspace method, show how error estimation, time-stepping and adaptivity are handled in our algorithm, and finally give some instructions on how to use our implementation. Several numerical experiments are given in Section 4 followed by some concluding remarks and pointers towards future work in Section 5.

2. THE φ-FUNCTIONS

Central to the implementation of exponential integrators is the efficient and accurate evaluation of the matrix exponential and other φ-functions. These φ-functions are defined for scalar arguments by the integral representation

$$\varphi_0(z) = \mathrm{e}^z, \quad \varphi_\ell(z) = \frac{1}{(\ell-1)!}\int_0^1 \mathrm{e}^{(1-\theta)z}\theta^{\ell-1}\,\mathrm{d}\theta, \qquad \ell = 1, 2, \ldots,\ z \in \mathbf{C}. \tag{1}$$

For small values of ℓ, these functions are

$$\varphi_1(z) = \frac{\mathrm{e}^z - 1}{z}, \quad \varphi_2(z) = \frac{\mathrm{e}^z - 1 - z}{z^2}, \quad \varphi_3(z) = \frac{\mathrm{e}^z - 1 - z - \frac{1}{2}z^2}{z^3}.$$

The φ-functions satisfy the recurrence relation

$$\varphi_\ell(z) = z\varphi_{\ell+1}(z) + \frac{1}{\ell!}, \qquad \ell = 1, 2, \ldots. \tag{2}$$

The definition can then be extended to matrices instead of scalars using any of the available definitions of matrix functions, such as that based on the Jordan canonical form [Horn and Johnson 1991; Higham 2008].

Every stage in an exponential integrator can be expressed as a linear combination of φ-functions acting on certain vectors:

$$\varphi_0(A)b_0 + \varphi_1(A)b_1 + \varphi_2(A)b_2 + \cdots + \varphi_p(A)b_p. \tag{3}$$

Here p is related to the order of the exponential integrator, typically taking values less than five. A is a matrix, often the Jacobian for exponential Rosenbrock-type methods or an approximation to it for methods based on the classical linear/nonlinear splitting; usually, A is large and sparse.

We need to compute expressions of the form (3) several times in each step that the integrator takes, so there is a need to evaluate these expressions efficiently and accurately. This is the problem taken up in this paper. We would like to stress that any procedure for evaluating (3), such as the one described here, is independent of the specific exponential integrator used and can thus be re-used in different integrators. The exponential integrators differ in the vectors $b_0, \ldots, b_p$ appearing in (3).

The following lemma gives a formula for the exact solution of linear differential equations with polynomial inhomogeneity. This result partly explains the important role that φ-functions play in exponential integrators (see Minchev and Wright [2005] for more details). The lemma also provides the background for the time-stepping procedure for the evaluation of (3) which we develop in §3.3.

LEMMA 2.1 (SKAFLESTAD AND WRIGHT [2009]). *The solution of the non-autonomous linear initial value problem*

$$u'(t) = Au(t) + \sum_{j=0}^{p-1} \frac{t^j}{j!} b_{j+1}, \qquad u(t_k) = u_k, \tag{4}$$

is given by

$$u(t_k + \tau_k) = \varphi_0(\tau_k A)u_k + \sum_{j=0}^{p-1} \sum_{\ell=0}^{j} \frac{t_k^{j-\ell}}{(j-\ell)!} \tau_k^{\ell+1} \varphi_{\ell+1}(\tau_k A) b_{j+1},$$

where the functions φ_ℓ are defined in (1).

PROOF. Recall that φ_0 denotes the matrix exponential. Using $\varphi_0((t_k - t)A)$ as an integrating factor for (4) we arrive at

$$\begin{aligned} u(t_k + \tau_k) &= \varphi_0(\tau_k A)u_k + \varphi_0(\tau_k A) \int_0^{\tau_k} \varphi_0(-sA) \sum_{j=0}^{p-1} \frac{(t_k + \tau_k)^j}{j!} b_{j+1} \, \mathrm{d}s \\ &= \varphi_0(\tau_k A)u_k + \varphi_0(\tau_k A) \int_0^{\tau_k} \varphi_0(-sA) \sum_{j=0}^{p-1} \sum_{\ell=0}^{j} \frac{t_k^{j-\ell} s^\ell}{\ell!(j-\ell)!} b_{j+1} \, \mathrm{d}s. \end{aligned}$$

Now change the integration variable, $s = \theta\tau_k$, and apply the definition (1) of the φ-functions.

$$\begin{aligned} &= \varphi_0(\tau_k A)u_k + \sum_{j=0}^{d-1} \sum_{\ell=0}^{j} \frac{t_k^{j-\ell}}{(j-\ell)!} \tau_k^{\ell+1} \left(\frac{1}{\ell!} \int_0^1 \varphi_0((1-\theta)\tau_k A)\theta^\ell \, \mathrm{d}\theta \right) b_{j+1} \\ &= \varphi_0(\tau_k A)u_k + \sum_{j=0}^{p-1} \sum_{\ell=0}^{j} \frac{t_k^{j-\ell}}{(j-\ell)!} \tau_k^{\ell+1} \varphi_{\ell+1}(\tau_k A) b_{j+1}. \end{aligned}$$

□

3. THE ALGORITHM

This section describes the details of our algorithm for evaluating expressions of the form (3) as implemented in the MATLAB function `phipm`. In the first part of this section we explain how Krylov subspace techniques can be used to reduce large matrices to small ones when evaluating matrix functions. An estimate of the error committed in the Krylov subspace approximation is essential for an adaptive solver; this is dealt with in the second part. Then we discuss how to split up the computation of the φ-functions into several steps. Part four concerns the possibility of adapting the Krylov subspace dimension and the size of the steps using the error

estimate from the second part, and the interaction between both forms of adaptivity. Finally, we explain how to use the implementation provided in the `phipm` function.

3.1 The basic method

We start by considering how to compute $\varphi_p(A)v$, where A is an $n \times n$ matrix (with n large) and $v \in \mathbf{R}^n$. This is one of the terms in (3). We will use a Krylov subspace approach for this task.

The idea behind this Krylov subspace approach is quite simple. The vector $\varphi_p(A)v$ lives in $\mathbf{R}^n$, which is a big space. We approximate it in a smaller space of dimension m. This smaller space is the Krylov subspace, which is given by

$$K_m = \operatorname{span}\{v, Av, A^2v, \ldots, A^{m-1}v\}.$$

However, the vectors $A^j v$ form a bad basis for the Krylov subspace because they point in almost the same direction as the dominant eigenvector of A; thus, the basis vectors are almost linearly dependent. In fact, computing these successive products is the power iteration method for evaluating the dominant eigenvector. Therefore, we apply the (stabilized) Gram–Schmidt procedure to get an orthonormal basis of the Krylov subspace:

$$K_m = \operatorname{span}\{v_1, v_2, \ldots, v_m\}.$$

Let V_m denote the n-by-m matrix whose columns are $v_1, \ldots, v_m$. Then the m-by-m matrix $H_m = V_m^T A V_m$ is the projection of the action of A to the Krylov subspace, expressed in the basis $\{v_1, \ldots, v_m\}$. The *Arnoldi iteration* (Algorithm 1) computes the matrices H_m and V_m, see Saad [1992].

Algorithm 1 The Arnoldi iteration.

```
v_1 = v/‖v‖
for j = 1, ..., m do
   w = A v_j
   for i = 1, ..., j do
      h_{i,j} = v_i^T w; w = w − h_{i,j} v_i
   end for
   h_{j+1,j} = ‖w‖; v_{j+1} = w/h_{j+1,j}
end for
```

It costs $\frac{3}{2}(m^2 - m + 1)n$ floating point operations (flops) and m products of A with a vector to compute the matrices H_m and V_m. The cost of one of these matrix-vector products depends on the sparsity of A; the straightforward approach uses $2N_A$ flops where N_A is the number of nonzero entries in the matrix A.

The projection of the action of A on the Krylov subspace K_m in the standard basis of $\mathbf{R}^n$ is $V_m H_m V_m^T$. We now approximate $\varphi_p(A)v$ by $\varphi_p(V_m H_m V_m^T)v$. Since $V_m^T V_m = I_m$ and $V_m V_m^T v = v$ we have $\varphi_p(V_m H_m V_m^T)v = V_m \varphi_p(H_m) V_m^T v$. Finally, $V_m^T v = \|v\| e_1$, where e_1 is the first vector in the standard basis. Taking everything together, we arrive at the approximation

$$\varphi_p(A)v \approx \beta V_m \varphi_p(H_m) e_1, \quad \beta = \|v\|. \tag{5}$$

The advantage of this formulation is that the matrix H_m has size m-by-m and thus it is much cheaper to evaluate $\varphi_p(H_m)$ than $\varphi_p(A)$.

The matrix H_m is Hessenberg, meaning that the (i, j) entry vanishes whenever $i > j + 1$. It is related to the matrix A by the relation $H_m = V_m^T A V_m$. If A is symmetric, then H_m is both symmetric and Hessenberg, which means that it is tridiagonal. In that case, we denote the matrix by T_m, and we only need traverse the i-loop in the Arnoldi iteration twice: once for $i = j - 1$ and once for $i = j$. The resulting algorithm is known as the *Lanczos iteration* (Algorithm 2); see Trefethen and Bau [1997, Alg 36.1]. It takes $3(2m - 1)n$ flops and m products of A with a vector to compute T_m and V_m when A is symmetric.

Algorithm 2 The Lanczos iteration.

$t_{1,0} = 0$; $v_0 = 0$; $v_1 = v/\|v\|$
for $j = 1, \dots, m$ **do**
  $w = Av_j$; $t_{j,j} = v_j^T w$
  $w = w - t_{j,j-1} v_{j-1} - t_{j,j} v_j$
  $t_{j,j-1} = \|w\|$; $t_{j-1,j} = \|w\|$
  $v_{j+1} = w/t_{j,j-1}$
end for

The Krylov subspace algorithm reduces the problem of computing $\varphi_p(A)v$ where A is a big n-by-n matrix to that of computing $\varphi_p(H_m)e_1$ where H_m is a smaller m-by-m matrix. Skaflestad and Wright [2009] describe a modified scaling-and-squaring method for the computation of $\varphi_p(H_m)$. However, this method has the disadvantage that one generally also has to compute $\varphi_0(H_m), \dots, \varphi_{p-1}(H_m)$. It is usually cheaper to compute the matrix exponential $\exp(\hat{H}_m)$ of a slightly larger matrix $\hat{H}_m$, following an idea of Saad [1992, Prop. 2.1], generalized by Sidje [1998, Thm. 1] to $p > 1$. Indeed, if we define the augmented matrix $\hat{H}_m$ by

$$\hat{H}_m = \begin{bmatrix} H_m & e_1 & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{bmatrix} \begin{matrix} m \text{ rows} \\ p-1 \text{ rows} \\ 1 \text{ row} \end{matrix} \tag{6}$$

then the top m entries of the last column of $\exp(\hat{H}_m)$ yield the vector $\varphi_p(H_m)e_1$. Finally, we compute the matrix exponential $\exp(\hat{H}_m)$ using the degree-13 diagonal Padé approximant combined with scaling and squaring as advocated by Higham [2005]; this is the method implemented in the function `expm` in MATLAB Version 7.2 (R2006a) and later. In contrast, EXPOKIT uses the degree-14 uniform rational Chebyshev approximant for symmetric negative-definite matrices and the degree-6 diagonal Padé approximant for general matrices, combined with scaling and squaring. We choose not to deal with the negative-definite matrices separately, but consider this as a possible extension to consider at a later date. Koikari [2009] recently developed a new variant of the Schur–Parlett algorithm using three-by-three blocking.

The computation of $\exp(\hat{H}_m)$ using Higham's method requires one matrix division (costing $\frac{8}{3}(m+p)^3$ flops) and $6 + \lceil \log_2(\|\hat{H}_m\|_1/5.37) \rceil_+$ matrix multiplications (costing $2(m+p)^3$ flops each), where $\lceil x \rceil_+$ denotes the smallest nonnegative integer

larger than x. Thus, the total cost of computing the matrix exponential of $\hat{H}_m$ is $M(\hat{H}_m)\,(m+p)^3$ where

$$M(A) = \frac{44}{3} + 2\left\lceil \log_2 \frac{\|A\|_1}{5.37} \right\rceil_+ . \tag{7}$$

Only the last column of the matrix exponential $\exp(\hat{H}_m)$ is needed; it is natural to ask if this can be exploited. Alternatively one could ask whether the computation of $\exp(\hat{H}_m)$ using the scaling-and-squaring algorithm can be modified to take advantage of the fact that $\hat{H}_m$ is Hessenberg (Higham [2008, Prob. 13.6] suggests this as a research problem). The algorithm suggested by Higham [2005, Alg. 2.3] computes the matrix exponential to machine precision (Higham considers IEEE single, double and quadruple precision). The choice of the degree of the Padé approximation and the number of scaling steps is intimately connected with this choice of precision. However, we generally do not require this accuracy. It would be interesting to see if significant computational savings can be gained in computing the matrix exponential by developing an algorithm with several other choices of precision.

3.2 Error estimation

Saad [1992, Thm. 5.1] derives a formula for the error in the Krylov subspace approximation (5) to $\varphi_p(A)v$ in the case $p = 0$. This result was generalized by Sidje [1998, Thm. 2] to $p > 0$. Their result states that

$$\varphi_p(A)v - \beta V_m \varphi_p(H_m) e_1 = \beta \sum_{j=p+1}^{\infty} h_{m+1,m} e_m^T \varphi_j(H_m) e_1 A^{j-p-1} v_{m+1}. \tag{8}$$

The first term in the series on the right-hand side does not involve any multiplications with the matrix A and the Arnoldi iteration already computes vector v_{m+1}, so this term can be computed without too much effort. We use this term as an error estimate for the approximation (5):

$$\varepsilon = \|\beta h_{m+1,m} e_m^T \varphi_{p+1}(H_m) e_1 v_{m+1}\| = \beta |h_{m+1,m}|\, [\varphi_{p+1}(H_m)]_{m,1}. \tag{9}$$

Further justification for this error estimate is given by Hochbruck et al. [1998].

We also use this error estimate as a corrector: instead of (5), we use

$$\varphi_p(A)v \approx \beta V_m \varphi_p(H_m) e_1 + \beta h_{m+1,m} e_m^T \varphi_{p+1}(H_m) e_1 v_{m+1}. \tag{10}$$

If $\varphi_{p+1}(H_m)e_1$ is computed using the augmented matrix (6), then $\varphi_p(H_m)e_1$ also appears in the result, so we only need to exponentiate a matrix of size $m+p+1$. The approximation (10) is more accurate, but ε is no longer a real error estimate. This is acceptable because, as explained below, it is not used as an error estimate but only for the purpose of adaptivity.

Sidje [1998] proposes a more accurate error estimate which also uses the second term of the series in (8). However, the computation of this requires a matrix-vector product and the additional accuracy is in our experience limited. We thus do not use Sidje's error estimate.

3.3 Time-stepping

Saad [1992, Thm. 4.7] proves that the error of the approximation (5) satisfies the bound

$$\|\text{Error}\| \leq \frac{2\beta(\rho(A))^m}{m!}(1 + o(1)) \tag{11}$$

for sufficiently large m, where $\rho(A)$ denotes the spectral radius of A. This bound deteriorates as $\rho(A)$ increases, showing that the dimension m of the Krylov subspace has to be large if $\rho(A)$ is large. Sidje [1998] proposes an alternative method for computing $\varphi_1(A)v$ when $\rho(A)$ is large, based on time-stepping. This approach is generalized by Sofroniou and Spaletta [2007] for general φ-functions.

The main idea behind this time-stepping procedure is that $\varphi_p(A)v$ solves a non-autonomous linear ODE. More generally, Lemma 2.1 states that the function

$$u(t) = \varphi_0(tA)b_0 + t\varphi_1(tA)b_1 + t^2\varphi_2(tA)b_2 + \cdots + t^p\varphi_p(tA)b_p, \tag{12}$$

is the solution of the differential equation

$$u'(t) = Au(t) + b_1 + tb_2 + \cdots + \frac{t^{p-1}}{(p-1)!}b_p, \qquad u(0) = b_0. \tag{13}$$

We now use a time-stepping method to calculate $u(t_{\text{end}})$ for some $t_{\text{end}} \in \mathbf{R}$. If we want to compute an expression of the form (3), we set $t_{\text{end}} = 1$. Split the time interval $[0, t_{\text{end}}]$ by introducing a grid $0 = t_0 < t_1 < \ldots < t_n = t_{\text{end}}$. To advance the solution, say from t_k to t_{k+1}, we need to solve the differential equation (13) with the value of $u(t_k)$ as initial condition. The relation between $u(t_k)$ and $u(t_{k+1})$ is given in Lemma 2.1. Rearranging this expression gives

$$u(t_{k+1}) = \varphi_0(\tau_k A)u(t_k) + \sum_{i=1}^{p} \tau_k^i \varphi_i(\tau_k A) \sum_{j=0}^{p-i} \frac{t_k^j}{j!} b_{i+j}, \tag{14}$$

where $\tau_k = t_{k+1} - t_k$. However, we do not need to evaluate all the φ-functions. The recurrence relation $\varphi_q(A) = \varphi_{q+1}(A)A + \frac{1}{q!}I$ implies that

$$\varphi_q(A) = \varphi_p(A)A^{p-q} + \sum_{j=0}^{p-q-1} \frac{1}{(q+j)!}A^j, \qquad q = 0, 1, \ldots, p-1.$$

Substituting this in (14) yields

$$u(t_{k+1}) = \tau_k^p \varphi_p(\tau_k A)w_p + \sum_{j=0}^{p-1} \frac{\tau_k^j}{j!} w_j, \tag{15}$$

where the vectors w_j are given by

$$w_j = A^j u(t_k) + \sum_{i=1}^{j} A^{j-i} \sum_{\ell=0}^{j-i} \frac{t_k^\ell}{\ell!} b_{i+\ell}, \qquad j = 0, 1, \ldots, p.$$

This is the time-stepping method for computing (3).

The computational cost of this method is as follows. At every step, we need to compute the vectors w_j for $j = 0, \ldots, p$, the action of $\varphi_p(\tau_k A)$ on a vector,

$p+1$ scalar multiplications of a vector of length n and p vector additions. The vectors w_j satisfy the recurrence relation

$$w_0 = u(t_k) \quad \text{and} \quad w_j = Aw_{j-1} + \sum_{\ell=0}^{p-j} \frac{t_k^\ell}{\ell!} b_{j+\ell}, \qquad j = 1, \ldots, p, \tag{16}$$

and hence their computation requires p multiplications of A with a vector, p scalar multiplications, and p vector additions.

One reason for developing this time-stepping method is to reduce the dimension of the Krylov subspace. If the spectrum of A is very large, multiple time-steps may be required. We intend, in the future, to compare this approach with the approach described by Skaflestad and Wright [2009] which evaluates the matrices $\varphi_0(H_m), \ldots, \varphi_{p+1}(H_m)$ directly. The latter method may have computational advantages, particularly if $b_0, \ldots, b_p$ are equal or zero. For values of p that we are interested in, that is less than five, we have not noticed any loss of accuracy using this approach. We intend to look into this issue more thoroughly in future investigations, especially in light of the paper [Al-Mohy and Higham 2010], which appeared during the review process, which noticed that for large values of p accuracy can be lost.

We choose an initial step size similar to the one suggested in EXPOKIT, except we increase the rather conservative estimate by an order of magnitude, to give

$$\tau_0 = \frac{10}{\|A\|_\infty} \left(\frac{\mathrm{Tol}\left((m_{\mathrm{ave}}+1)/\mathrm{e}\right)^{m_{\mathrm{ave}}+1} \sqrt{2\pi(m_{\mathrm{ave}}+1)}}{4\|A\|_\infty \|b_0\|_\infty} \right)^{1/m_{\mathrm{ave}}}, \tag{17}$$

where Tol is the user defined tolerance and m_{ave} is the average of the input and maximum allowed size of the Krylov subspace.

3.4 Adaptivity

The procedure described above has two key parameters, the dimension m of the Krylov subspace and the time-step τ. These need to be chosen appropriately. As we cannot expect the user to make this choice, and the optimal values may change, the algorithm needs to determine m and τ adaptively.

We are using a time-stepping method, so adapting the step size τ is similar to adaptivity in ODE solvers. This has been studied extensively. It is described by Butcher [2008, §39] and Hairer et al. [1993, §II.4], among others. The basic idea is as follows. We assume that the time-stepping method has order q, so that the error is approximately $C\tau^{q+1}$ for some constant C. We somehow compute an error estimate ε and choose a tolerance Tol that the algorithm should satisfy. Then the optimal choice for the new step size is

$$\tau_{\mathrm{new}} = \tau_k \left(\frac{1}{\omega} \right)^{1/(q+1)} \quad \text{where} \quad \omega = \frac{t_{\mathrm{end}} \|\varepsilon\|}{\tau_k \cdot \mathrm{Tol}}. \tag{18}$$

The factor t_{end} is included so that the method is invariant under time scalings. Usually, a safety factor γ is added to ensure that the error will probably satisfy the error tolerance, changing the formula to $\tau_{\mathrm{new}} = \tau_k(\gamma/\omega)^{1/(q+1)}$. Common choices are $\gamma = 0.25$ and $\gamma = 0.38$. However, in our case the consequence of rejecting a

step is that we computed the matrix exponential in vain, while in ODE solvers the whole computation has to be repeated when a step is rejected. We may thus be more adventurous and therefore we take $\gamma = 0.8$.

In our scheme, the error estimate ε is given by (9). However, what is the order q for our scheme? The *a priori* estimate (11) suggests that the order equals the dimension m of the Krylov subspace. Experiments confirm that the error is indeed proportional to τ^{m+1} in the limit $\tau \to 0$. However, for finite step size the error is better described by τ^q with a smaller exponent q. Let us call the exponent q which provides the best fit around a given value step size τ the "heuristic order", for lack of a better term.

We can estimate the heuristic order if we have attempted two step sizes during the same step, which happens if we have just rejected a step and reduced the step size. The estimate for the heuristic order is then

$$\hat{q} = \frac{\log(\tau/\tau_{\text{old}})}{\log(\|\varepsilon\|/\|\varepsilon_{\text{old}}\|)} - 1, \tag{19}$$

where ε and ε_{old} denote the error estimates produced when attempting step size τ and τ_{old}, respectively. In all other cases, we use $\hat{q} = \frac{1}{4}m$; there is no rigorous argument behind the choice of $\frac{1}{4}$ but it seems to yield good performance in practice.

With this estimate for the heuristic order, we compute the suggested new step size as

$$\tau_{\text{new}} = \tau_k \left(\frac{\gamma}{\omega}\right)^{1/(\hat{q}+1)}. \tag{20}$$

The other parameter that we want to adapt is the Krylov subspace dimension m. The error bound (11) suggests that, at least for modest changes of m, the error is approximately equal to $C\kappa^{-m}$ for some values of C and k. Again, we can estimate κ if we have error estimates corresponding to two different values of m:

$$\hat{\kappa} = \left(\frac{\|\varepsilon\|}{\|\varepsilon_{\text{old}}\|}\right)^{1/(m_{\text{old}}-m)}. \tag{21}$$

If this formula cannot be used, then we take $\hat{\kappa} = 2$. Given this estimate, the minimal m which satisfies the required tolerance is given by

$$m_{\text{new}} = m + \frac{\log(\omega/\gamma)}{\log \hat{\kappa}}. \tag{22}$$

We now have to choose between two possibilities: either we keep m constant and change τ to τ_{new}, or we keep τ constant and change m to m_{new}. We will pick the cheapest option. To advance from t_k to t_{k+1}, we need to evaluate (15). Computation of the vectors w_j requires $2(p-1)(N_A+n)$ flops. Then, we need to do m steps of the Arnoldi algorithm, for a cost of $\frac{3}{2}(m^2-m+1)n + 2mN_A$ flops. If A is symmetric, we will use the Lanczos algorithm and the costs drops to $3(2m-1)n+2mN_A$ flops. To compute $\varphi_p(\tau_k A)w_p$ in (15) using (10), we need to exponentiate a matrix of size $m+p+1$, costing $M(\hat{H}_m)\,(m+p+1)^3$ flops with $M(\hat{H}_m)$ given by (7). Finally, the scalar multiplications and vector additions in (15) requires a further

$(2p+1)n$ flops. All together, we find that the cost of a single step is

$$C_1(m) = \begin{cases} (m+p)N_A + 3(m+p)n + M(H_m)\,(m+p+1)^3, & \text{for Lanczos;} \\ (m+p)N_A + (m^2+3p+2)n + M(H_m)\,(m+p+1)^3, & \text{for Arnoldi.} \end{cases} \tag{23}$$

This needs to be multiplied with the number of steps required to go from the current time t_k to the end point $t = t_{\text{end}}$. So the total cost is

$$C(\tau, m) = \left\lceil \frac{t_{\text{end}} - t_k}{\tau} \right\rceil C_1(m). \tag{24}$$

We compute $C(\tau_{\text{new}}, m)$ and $C(\tau, m_{\text{new}})$ according to this formula. If $C(\tau_{\text{new}}, m)$ is smaller, then we change the time-step to τ_{new} and leave m unchanged. However, to prevent too large changes in τ we restrict it to change by no more than a factor 5. Similarly, if $C(\tau_{\text{new}}, m)$ is smaller, then τ remains as it is and we change m to m_{new}, except that we restrict it to change by no more as a factor $\frac{4}{3}$ (this factor is chosen following Hochbruck et al. [1998]).

Finally, the step is accepted if $\omega > \delta$ where $\delta = 1.2$. Thus, we allow that the tolerance is slightly exceeded. The idea is that our adaptivity procedure aims to get ω down to $\gamma = 0.8$, so that usually we stay well below the tolerance and hence we may permit ourselves to exceed it occasionally. The resulting algorithm is summarized in Algorithms 3 and 4.

Algorithm 3 Computing the linear combination (3).

```
t = 0; k = 0; u_k = b_0
Evaluate initial τ using (17);
Initial guess m = 10
repeat
  Compute w_0, ..., w_p according to (16)
  repeat
    Compute H_m and B_m using Algorithm 1 or 2
    F = approximation to φ_p(τA)w_p given by (10)
    ε = error estimate given by (9)
    Compute ω according to (18)
    Compute τ_new and m_new using Algorithm 4
    Compute C(τ_new, m) and C(τ, m_new) according to (7), (23) and (24)
    if C(τ_new, m) < C(τ, m_new) then
      τ = min{max{τ_new, (1/5)τ}, 2τ, 1 − t}
    else
      m = min{max{m_new, ⌊(3/4)m⌋, 1}, ⌈(4/3)m⌉}
    end if
  until ω ≤ δ
  Compute u_{k+1} according to (15)
  t = t + τ; k = k + 1
until t = 1
return u_k
```

Algorithm 4 Computing τ_{new} and m_{new}.

if previous step was rejected and τ was reduced **then**
  Compute $\hat{q}$ according to (19)
else if previous step was rejected and $\hat{q}$ was computed in previous step **then**
  Keep old value of $\hat{q}$
else
  $\hat{q} = \frac{1}{4}m$
end if
Compute τ_{new} according to (20)
if previous step was rejected and m was reduced **then**
  Compute $\hat{\kappa}$ according to (21)
else if previous step was rejected and $\hat{\kappa}$ was computed in previous step **then**
  Keep old value of $\hat{\kappa}$
else
  $\hat{\kappa} = 2$
end if
Compute m_{new} according to (22)

3.5 The MATLAB function `phipm`

The algorithm described above is implemented in a MATLAB function called `phipm`. In terms of computing power, the system requirements are modest. Any computer capable of running a moderately up-to-date version of MATLAB is sufficient.

A call of the `phipm` function has the form

```
[u, stats] = phipm(t, A, b, tol, symm, m)
```

There are three mandatory input arguments and one mandatory output argument; the other arguments are optional. The first input argument is `t`, the final time, t_{end}, for the differential equation (13). This is generally chosen to be $t = 1$ because the solution (12) of (13) at $t = 1$ equals the linear combination (3). The second argument is the n-by-n matrix argument of the φ-functions. The `phipm` function can also be used without forming the matrix A explicitly, by setting the argument `A` to a function which, given a vector b, computes Ab. Finally, `b` is an n-by-$(p + 1)$ matrix with columns representing the vectors $b_0, b_1, \ldots, b_p$ to be multiplied by the corresponding φ-functions. There is one mandatory output argument: `u`, the numerical approximation to the solution of (13) at the final time t_{end}.

There are three optional input arguments. The first one is `tol`, the tolerance Tol in (19). The default tolerance is 10^{-7}. Then comes `symm`, a boolean indicating whether A is symmetric (`symm=1`) or not (`symm=0`). If not supplied, the code determines itself whether A is symmetric if the matrix is passed explicitly, and assumes that A is not symmetric if the matrix is given implicitly. The final input argument is `m`, the initial choice for the dimension of the Krylov subspace. The initial choice is $m = 1$ by default. There is also one optional output argument: `stats`, for providing the user with various statistics of the computation. It is a vector with four entries: `stats(1)` is the number of steps needed to complete the integration, `stats(2)` is the number of rejected steps, `stats(3)` is the number of matrix-vector products, and `stats(4)` is the number of matrix exponentials computed.

4. NUMERICAL EXPERIMENTS

In this section we perform several numerical experiments, which illustrate the advantages of the approach that has been outlined in the previous sections. We compare the function `phipm` with various state-of-the-art numerical algorithms. The first experiment compares several numerical ODE solvers on a large system of linear ODEs, resulting from the finite-difference discretization of the Heston PDE, a common example from the mathematical finance literature. The second experiment compares our function `phipm` and the `expv` and `phiv` functions from EXPOKIT on various large sparse matrices. This repeats the experiment reported by Sidje [1998].

All experiments use a MacBookPro with 2.66 GHz Intel Core 2 Duo processor and 4GB 1067 MHz DDR3 memory. We use MATLAB Version 7.11 (R2010b) for all experiments, which computes the matrix exponential as described in Higham [2005]. We have noticed that the results described in this section depend on the specifications of the computer but the overall nature of the numerical experiments remains the same.

4.1 The Heston equation in financial mathematics

A European call option gives its owner the right (but not obligation) to buy a certain asset for a certain price (called the strike price) at a certain time (the expiration date). European call options with stochastic volatility, modeled by a stochastic mean-reverting differential equation, have been successfully priced by Heston [1993]. The Heston pricing formulae are a natural extension of the celebrated Black–Scholes–Merton pricing formulae. Despite the existence of a semi-closed form solution, which requires the numerical computation of an indefinite integral, the so-called Heston PDE is often used as a test example to compare various numerical integrators. This example follows closely In 't Hout [2007] and In 't Hout and Foulon [2010].

Let $U(s, v, t)$ denote the European call option price at time $T - t$, where s is the price of the underlying asset and v the variance in the asset price at that time. Here, T is the expiration date of the option. Heston's stochastic volatility model ensures that the price of a European call option satisfies the time-dependent convection-diffusion-reaction equation

$$U_t = \frac{1}{2} v s^2 U_{ss} + \rho \lambda v s U_{vs} + \frac{1}{2} \lambda^2 v U_{vv} + (r_d - r_f) s U_s + \kappa(\eta - v) U_v - r_d U.$$

This equation is posed on the unbounded spatial domain $s > 0$ and $v \geq 0$, while t ranges from 0 to T. Here, $\rho \in [-1, 1]$ represents the correlation between the Wiener processes modeling the asset price and its variance, λ is a positive scaling constant, r_d and r_f are constants representing the risk-neutral domestic and foreign interest rates respectively, η represents the mean level of v and κ the rate at which v reverts to η. The payoff of the European call option provides the initial condition

$$U(s, v, 0) = \max(s - K, 0),$$

where $K \geq 0$ denotes the strike price.

The unbounded spatial domain must be restricted in size in computations and we choose the sufficiently large rectangle $[0, S] \times [0, V]$. Usually S and V are chosen much larger than the values of s and v of practical interest. This is a commonly

used approach in financial modeling so that if the boundary conditions are imperfect their effect is minimized, see Tavella and Randall [2000, p. 121]. Suitable boundary conditions for $0 < t \leq T$ are

$$\begin{aligned}
&U(0, v, t) = 0,\\
&U(s, V, t) = s,\\
&U_s(S, v, t) = 1,\\
&U_t(s, 0, t) - (r_d - r_f)sU_s(s, 0, t) - \kappa\eta U_v(s, 0, t) + rU(s, 0, t) = 0.
\end{aligned}$$

Note that the boundary and initial conditions are inconsistent or non-matching, that is the boundary and initial conditions at $t = 0$ do not agree.

We discretize the spatial domain using a uniform rectangular mesh with mesh lengths Δs and Δv and use standard second-order finite differences to approximate the derivatives as follows:

$$\begin{aligned}
(U_s)_{i,j} &\approx \frac{U_{i+1,j} - U_{i-1,j}}{2\Delta s},\\
(U_{ss})_{i,j} &\approx \frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{\Delta s^2},\\
(U_v)_{i,j} &\approx \frac{U_{i,j+1} - U_{i,j-1}}{2\Delta v},\\
(U_v)_{i,0} &\approx \frac{-3U_{i,0} + 4U_{i,1} - U_{i,2}}{2\Delta v},\\
(U_{vv})_{i,j} &\approx \frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{\Delta v^2},\\
(U_{sv})_{i,j} &\approx \frac{U_{i+1,j+1} + U_{i-1,j-1} - U_{i-1,j+1} - U_{i+1,j-1}}{4\Delta v \Delta s}.
\end{aligned}$$

The boundary associated to $v = 0$ is included in the mesh, but the other three boundaries are not. Combining the finite-difference discretization and the boundary conditions leads to a large system of ODEs of the form

$$u'(t) = Au(t) + b_1, \qquad u(0) = b_0.$$

The exact solution for this system is given in Lemma 2.1. This is a natural problem for the `phipm` solver.

We compare five methods: the scheme of Crank and Nicolson [1947], two Alternating Direction Implicit (ADI) schemes, `ode15s` from MATLAB and the `phipm` method described in this paper. The two ADI schemes are the method due to Douglas and Rachford [1956] with $\theta = \frac{1}{2}$, and the method of Hundsdorfer and Verwer [2003] with $\theta = \frac{3}{10}$ and $\mu = \frac{1}{2}$. The first ADI method is of order one and the second method is of order two. In 't Hout [2007] lists each of the ADI methods described above and explains necessary implementation details; the stability of these methods is discussed in In 't Hout and Welfert [2009]. To compare `phipm` with an adaptive solver we choose the in-built MATLAB solver `ode15s`, which is a variable step size, variable order implementation of the backward differentiation formulae (BDF). The Jacobian of the right-hand side (that is, the matrix A) is passed to `ode15s`; this makes `ode15s` considerably faster.

We choose the same problem parameters as in the paper by [In 't Hout 2007]: namely $\kappa = 2$, $\nu = 0.2$, $\lambda = 0.3$, $\rho = 0.8$, $r_d = 0.03$, $r_f = 0.0$, the option

maturity $T = 1$ and the strike price $K = 100$. The spatial domain is truncated to $[0, 8K] \times [0, 5]$. We use a grid with 100 points in the s-direction and 51 points in the v-direction (recall that $v = 0$ is included in the grid and the other borders are not). This results in a system with 5100 degrees of freedom, with $nnz = 44,800$ non-zero elements. Figure 1 shows the error of the solvers against the CPU time.

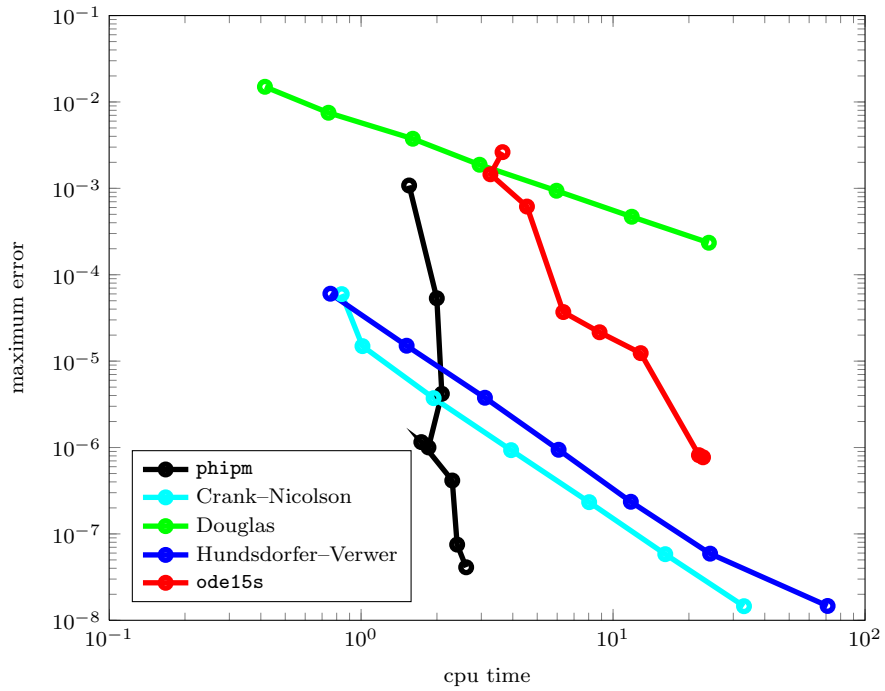

Fig. 1. Plots of maximum error (on the domain $[0, 2K] \times [0, 1]$) against CPU time for the system of ODEs from the discretized Heston PDE. The two ADI schemes are the represented by the: green line (Douglas); blue line (Hundsdorfer and Verwer), the cyan line is Crank–Nicolson, the red line is `ode15s` and the black line is `phipm`.

The Crank–Nicolson and ADI schemes are run with the step size decreasing in powers of two from 2^{-8} to 2^{-14}, while `phipm` and `ode15s` are run with the tolerance decreasing geometrically from 10^{-1} to 10^{-6}. The error is computed by comparing the numerical solution against the "exact" solution, as computed using two different methods with very small stepsizes so the the solution was accurate to within 10^{-10}. We measure the maximum error at time $t = T$ of the numerical solution satisfying $[0, 2K] \times [0, 1]$, a smaller domain than the computational domain.

The first surprising result is that the Crank–Nicolson method outperforms the ADI methods. This requires the use of column reordering and row scaling in the LU decomposition as implemented in MATLAB. A call to this function takes the form `[L,U,P,Q,R] = lu(X)`. Practitioners are interested in this problem for accuracy levels of around 10^{-4} or one basis point. The `phipm` method is the most efficient for an accuracy of around 10^{-6}. Similar results hold for the four parameter sets

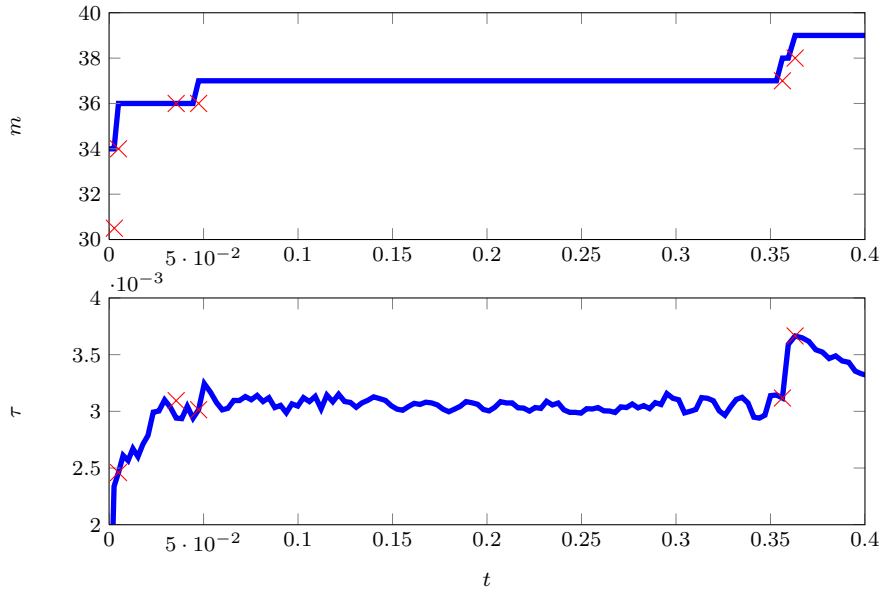

Fig. 2. Plots showing how the dimension m of the Krylov subspace and the step size τ change during the integration of the system of ODEs from the discretized Heston PDE, as solved by `phipm` with a tolerance of 10^{-4}. The red crosses represent rejected steps.

listed in In 't Hout and Foulon [2010], with the cross-over point close to 10^{-6}. Note that for these four parameter sets the U_v term was discretized using upwinding when $v > 1$. Recently, we have applied Krylov subspace methods to a variety of option pricing problems. We find that Krylov subspace methods significantly outperform ADI methods for dimension higher than two; we refer the interested reader to the forthcoming paper [Niesen and Wright 2011].

Figure 2 shows how the Krylov subspace size (top graph) and the step size (bottom graph) change during the integration interval. They both vary during the integration, which shows that the adaptivity presented in Section 3.4 is effective. Figure 3 plots the error estimate and the actual error during the integration interval. The error estimate is always larger than the actual error in this experiment.

One worrying aspect in these figures is that the step size sequence in Figure 2 zig-zags and that the estimated error in Figure 3 varies alot from step to step. This might indicate stability problems, perhaps caused by augmenting the matrix in (6) or by the matrix-vector multiplications in (16). The MATLAB solver `ode15s` also has a rather strange stepsize pattern for this problem, where stepsizes are constant or rapidly increasing. The error behaviour of Krylov methods is a difficult problem and we intend to investigate this further in future work; perhaps the use of control theory techniques can be useful in smoothing the error estimate and choice of Krylov subspace size. Also, given that this equation has a semi-closed form solution, the application to more exotic options, such as barrier, American and Asian options, is of more practical significance and in our minds for future work.

Finally, we draw attention to the fact that the linear algebra in `ode15s` the

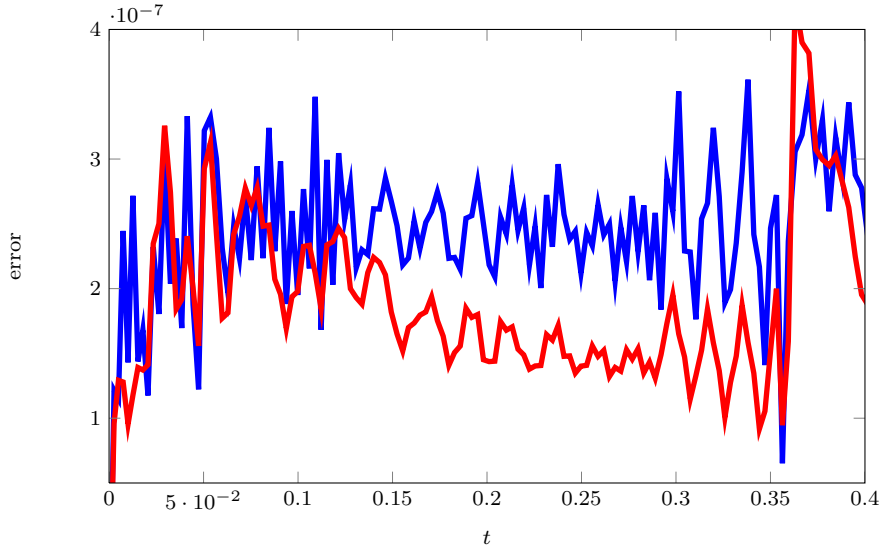

Fig. 3. The error estimate (blue) and the actual error (red) computed during the integration of the system of ODEs from the discretized Heston PDE. Only accepted steps are shown. Again, the tolerance is 10^{-4}.

Crank–Nicolson and all the ADI methods, is performed using highly optimized routines written in a low-level language, whereas all the computations in `phipm` are done in the native MATLAB language. We have noticed that the Crank–Nicolson and the ADI methods performance has improved relative to `phipm` in newer version of MATLAB. We intend to release C++ and CUDA versions of this software in the near future.

4.2 Comparisons between `phipm` and the functions in EXPOKIT

The University of Florida sparse matrix collection, compiled by Davis [2007], is an excellent, well-maintained website containing various classes of sparse matrices. All the sparse matrices that we use in this subsection are available from that website.

Experiment 1. We compute the action of the matrix exponential of four different sparse matrices described below on certain vectors b_0. We use the `expv` and `phiv` routines from EXPOKIT and the `phipm` routine described in this paper to compute $\mathrm{e}^{tA}b_0$ (the `phiv` routine computes $\mathrm{e}^{tA}b_0 + \varphi_1(tA)b_1$, so by setting $b_1 = 0$ this code can also be used to compute $\mathrm{e}^{tA}b_0$). The `expv` and `phiv` routines are implemented in MATLAB like the `phipm` routine, but they do not use the Lanczos algorithm when A is symmetric. Therefore, we implemented a variant of `phipm` which computes (12) using a Krylov subspace of fixed dimension $m = 30$. This variant is called `phip`. It can be considered as an extension of the `phiv` code to symmetric matrices and $p \geq 1$ and similar to the code developed by Sofroniou and Spaletta [2007].

The four matrices we consider are as follows:

—The first matrix, `orani678` from the Harwell–Boeing collection [Duff et al. 1989], is an unsymmetric sparse matrix of order $n = 2,529$ with $nnz = 90,158$ nonzero

elements. We choose $t = 10$, $b_0 = [1, 1, \ldots, 1]^T$ and Tol $= \sqrt{e}$, where $e = 2^{-52}$ denotes the machine epsilon.

—The second example is `bcspwr10`, also from the Harwell–Boeing collection. This is a symmetric Hermitian sparse matrix of order $n = 5,300$ with $nnz = 21,842$. We set $t = 2$, $b_0 = [1, 0, \ldots, 0, 1]^T$ and Tol $= 10^{-5}$.

—The third example, `gr_30_30`, again of the Harwell–Boeing collection, is a symmetric matrix arising when discretizing the Laplacian operator using a nine-point stencil on a 30×30 grid. This yields an order $n = 900$ sparse matrix with $nnz = 7,744$ nonzero elements. Here, we choose $t = 2$ and compute $\mathrm{e}^{-tA}\mathrm{e}^{tA}b_0$, where $b_0 = [1, 1, \ldots, 1]^T$, in two steps: first the forward step computes $w = e^{tA}b_0$ and then we use the result w as the operand vector for the reverse part $e^{-tA}w$. The result should approximate b_0 with Tol $= 10^{-14}$.

—The final example uses `helm2d03` from the `GHS_indef` collection (see [Davis 2007] for more details), which describes the Helmholtz equation $-\Delta^T\nabla u - 10000u = 1$ on a unit square with Dirichlet $u = 0$ boundary conditions. The resulting symmetric sparse matrix of order $n = 392,257$ has $nnz = 2,741,935$ nonzero elements. We compute $\mathrm{e}^{tA}b_0 + t\varphi_1(tA)b_1$, where $t = 2$ and $b_0 = b_1 = [1, 1, \ldots, 1]^T$. In this test, only the codes `phiv`, `phip` and `phipm` are compared, with Tol $= \sqrt{e}$.

Only the third example has a known exact solution. To compute the exact solutions for the other three examples we use `phiv`, `phipm` and `expmv` from [Al-Mohy and Higham 2010], with a small tolerance so that all methods agree to a suitable level of accuracy. We report relative errors computed at the using

$$\text{error} = \left\| \frac{u_{\text{exact}} - u_{\text{approx}}}{u_{\text{exact}}} \right\|,$$

where u_{exact} and u_{approx} are the exact and approximate solutions. When the exact solution has components which are zero they are removed from the relative error calculations. In the first three comparisons we measure the average speedup and error of each of the codes relative to `expv`; in the final comparison we measure relative to `phiv`. The `tic` and `toc` functions from MATLAB are used to compute the timings. We ran the comparisons 100 times to compute the average speedup. We summarize our findings in Table I.

Table I. Comparisons of the average speedup of `phiv`, `phip` and `phipm` relative to `expv` and the relative errors on four matrices taken from the University of Florida sparse matrix collection.

	`orani678`		`bcspwr10`		`gr_30_30`		`helm2d03`	
code	speed	error	speed	error	speed	error	speed	error
`expv`	1	3.1×10^{-9}	1	5.8×10^{-14}	1	1.2×10^{-7}		
`phiv`	0.96	1.6×10^{-7}	0.97	1.0×10^{-14}	0.94	2.1×10^{-7}	1	4.3×10^{-8}
`phip`	0.95	3.5×10^{-11}	3.94	8.0×10^{-13}	2.69	1.8×10^{-6}	2.59	1.9×10^{-7}
`phipm`	1.35	2.4×10^{-11}	6.10	5.7×10^{-5}	3.59	3.9×10^{-6}	4.63	1.9×10^{-7}

Experiment 2. In these computations we evaluate $\varphi_0(tA)b_0 + t\varphi_1(tA)b_1 + \cdots + t^4\varphi_4(tA)b_4$, where $b_0 = \cdots = b_4 = [1, 1, \ldots, 1]^T$, with the codes `phip` and `phipm`. This comparison gauges the efficiency gains achieved by allowing the Krylov subspace size to vary. The implementations are identical except for the fact that `phipm` can vary m as well. We use the four sparse matrices described above, with the same values of t and Tol, except for the sparse matrix `gr_30_30` with Tol $= \sqrt{e}$, is used and we only compute the forward part of the problem. We summarize our findings in Table II.

Table II. Comparisons of the average speedup of `phipm` relative to `phip` and the relative errors on four large sparse matrices taken from the University of Florida sparse matrix collection.

	`orani678`		`bcspwr10`		`gr_30_30`		`helm2d03`	
code	speed	error	speed	error	speed	error	speed	error
`phip`	1	8.7×10^{-13}	1	2.5×10^{-10}	1	4.6×10^{-13}	1	5.2×10^{-8}
`phipm`	1.37	2.1×10^{-12}	1.35	4.2×10^{-5}	1.16	6.0×10^{-13}	1.87	5.2×10^{-8}

Discussion of the results. These comparisons show that in all cases the `phipm` code is more efficient, in some cases by a considerable margin. Summarizing, adapting both the dimension of Krylov subspace as well as the length of the time steps significantly increases overall efficiency. Given that in an implementation of an exponential integrator `phipm` would be called several times in a step over many steps during the integration, this increase in efficiency can often lead to very large overall computational gains.

5. CONCLUSION AND FUTURE WORK

The `phipm` function is an efficient routine which computes the action of linear combinations of φ-functions on operand vectors. The implementation combines time stepping with a procedure to adapt the Krylov subspace size. It can be considered as an extension of the codes provided in EXPOKIT and MATHEMATICA.

The φ-functions are the building blocks of exponential integrators. An implementation of the algorithm in a lower-level language will be useful in this context; this is work in progress. We are also working on the implementation of exponential integrators which use the `phipm` routine described in this paper and hope to report on this shortly.

We intend to improve the code over time. Some issues which we plan to investigate have already been mentioned. One of them is the issue of stability, especially in view of the error estimates in Figure 3 which might point to stability problems. Our choice to compute the φ-function of the reduced matrix H_m by adding some rows and columns and then computing the matrix exponential may exacerbate any instabilities. Perhaps it is better to compute the φ-function of H_m directly. This also allows us to exploit the fact that H_m is symmetric if the matrix L in the original differential equation is symmetric, for instance by using rational Chebyshev approximants. In any case, regardless of whether we augment the matrix H_m or not, we do not need to compute the matrix function to full precision. It should also be possible to exploit the fact that H_m is Hessenberg.

We also intend to modify `phipm` to take advantage of recent advances in parallel processing technology, specifically the use of graphics cards accessed using programming languages such as CUDA, which provide promise of significant computational improvements. We are currently investigating the application of Krylov-based methods to option pricing problems, where the governing PDEs are often linear. The corresponding discretized ODEs often take the form of Equation (4), which can naturally be computed using `phipm`. Finally, as mentioned in the introduction, there are alternatives to the (polynomial) Krylov method considered in this paper. We plan to study other methods and compare them against the method introduced here. Competitive methods can be added to the code, because we expect that the performance of the various methods depends strongly on the characteristics of both the problem and the exponential integrator.

ACKNOWLEDGMENTS

The authors thank Nick Higham, Karel in 't Hout, Brynjulf Owren, Roger Sidje and the anonymous referee for helpful discussions.

REFERENCES

Al-Mohy, A. H. and Higham, N. J. 2010. Computing the action of the matrix exponential, with an application to exponential integrators. Tech. Rep. 30, Manchester Institute for Mathematical Sciences, Manchester, England.

Butcher, J. C. 2008. *Numerical methods for ordinary differential equations*, Second ed. John Wiley & Sons Ltd., Chichester.

Caliari, M. and Ostermann, A. 2009. Implementation of exponential Rosenbrock-type integrators. *Appl. Numer. Math. 59*, 568–581.

Crank, J. and Nicolson, P. 1947. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *Proc. Cambridge Philos. Soc. 43*, 50–67.

Davis, T. A. 2007. The University of Florida sparse matrix collection. Tech. Rep. REP-2007-298, CISE Dept, University of Florida.

Douglas, Jr., J. and Rachford, Jr., H. H. 1956. On the numerical solution of heat conduction problems in two and three space variables. *Trans. Amer. Math. Soc. 82*, 421–439.

Duff, I. S., Grimes, R. G., and Lewis, J. G. 1989. Sparse matrix test problems. *ACM Transactions on Mathematical Software 15,* 1 (Mar.), 1–14.

Friesner, R. A., Tuckerman, L. S., Dornblaser, B. C., and Russo, T. V. 1989. A method for the exponential propagation of large stiff nonlinear differential equations. *J. Sci. Comput. 4,* 4, 327–354.

Gallopoulos, E. and Saad, Y. 1992. Efficient solution of parabolic equations by Krylov approximation methods. *SIAM J. Sci. Statist. Comput. 13*, 1236–1264.

Hairer, E., Nørsett, S. P., and Wanner, G. 1993. *Solving ordinary differential equations. I*, Second ed. Springer Series in Computational Mathematics, vol. 8. Springer-Verlag, Berlin. Nonstiff problems.

Heston, S. L. 1993. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *Review of Financial Studies 6,* 2, 327–43.

Higham, N. J. 2005. The scaling and squaring method for the matrix exponential revisited. *SIAM J. Matrix Anal. Appl. 26,* 4, 1179–1193 (electronic).

Higham, N. J. 2008. *Functions of matrices*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA. Theory and computation.

Hochbruck, M. and Lubich, C. 1997. On Krylov subspace approximations to the matrix exponential operator. *SIAM J. Numer. Anal. 34,* 5, 1911–1925.

Hochbruck, M., Lubich, C., and Selhofer, H. 1998. Exponential integrators for large systems of differential equations. *SIAM J. Sci. Comput. 19,* 5, 1552–1574.

Horn, R. A. and Johnson, C. R. 1991. *Topics in matrix analysis.* Cambridge University Press, Cambridge.

Hundsdorfer, W. and Verwer, J. 2003. *Numerical solution of time-dependent advection-diffusion-reaction equations.* Springer Series in Computational Mathematics, vol. 33. Springer-Verlag, Berlin.

In 't Hout, K. J. 2007. ADI schemes in the numerical solution of the Heston PDE. *Numerical Analysis and Applied Mathematics 936*, 10–14.

In 't Hout, K. J. and Foulon, S. 2010. ADI finite difference schemes for option pricing in the Heston model with correlation. *Int. J. Numer. Anal. Model. 7,* 2, 303–320.

In 't Hout, K. J. and Welfert, B. D. 2009. Unconditional stability of second-order ADI schemes applied to multi-dimensional diffusion equations with mixed derivative terms. *Appl. Numer. Math. 59,* 4, 677–692.

Koikari, S. 2007. An error analysis of the modified scaling and squaring method. *Computers Math. Applic. 53,* 8, 1293–1305.

Koikari, S. 2009. On a block Schur–Parlett algorithm for φ-functions based on the sep-inverse estimate. *ACM Trans. on Math. Soft. 38,* 2, Article 12.

López-Fernández, M. 2010. A quadrature based method for evaluating exponential-type functions for exponential methods. *BIT 50,* 3, 631–655.

Minchev, B. and Wright, W. M. 2005. A review of exponential integrators for semilinear problems. Tech. Rep. 2/05, Department of Mathematical Sciences, NTNU, Norway. `http://www.math.ntnu.no/preprint/`.

Moler, C. B. and Van Loan, C. F. 2003. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review 45,* 1, 3–49.

Moret, I. 2007. On RD-rational Krylov approximations to the core-functions of exponential integrators. *Numer. Linear Algebra Appl. 14,* 5, 445–457.

Niesen, J. and Wright, W. M. 2011. Krylov subspace methods applied to option pricing problems. In preparation.

Saad, Y. 1992. Analysis of some Krylov subspace approximations to the matrix exponential operator. *SIAM J. Numer. Anal. 29,* 1, 209–228.

Schmelzer, T. and Trefethen, L. N. 2007. Evaluating matrix functions for exponential integrators via Carathéodory–Fejér approximation and contour integrals. *Electronic Transactions on Numerical Analysis 29*, 1–18.

Sidje, R. 1998. EXPOKIT: Software package for computing matrix exponentials. *ACM Trans. on Math. Soft. 24,* 1, 130–156. See `http://www.maths.uq.edu.au/expokit/` for latest software.

Skaflestad, B. and Wright, W. M. 2009. The scaling and modified squaring method for matrix functions related to the exponential. *Appl. Numer. Math. 59,* 3-4, 783–799.

Sofroniou, M. and Spaletta, G. 2007. Efficient computation of Krylov approximations to the matrix exponential and related functions. Preprint.

Tavella, D. and Randall, C. 2000. *Pricing financial instruments.* John Wiley & Sons Ltd., New York.

Trefethen, L. N. and Bau, III, D. 1997. *Numerical linear algebra.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

Received Month Year; revised Month Year; accepted Month Year