Electronic Transactions on Numerical Analysis.
Volume 47, pp. 57–72, 2017.

ISSN 1068–9613.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

INCREMENTAL COMPUTATION OF BLOCK TRIANGULAR MATRIX EXPONENTIALS WITH APPLICATION TO OPTION PRICING*

DANIEL KRESSNER†, ROBERT LUCE†, AND FRANCESCO STATTI†

Abstract. We study the problem of computing the matrix exponential of a block triangular matrix in a peculiar way: block column by block column, from left to right. The need for such an evaluation scheme arises naturally in the context of option pricing in polynomial diffusion models. In this setting, a discretization process produces a sequence of nested block triangular matrices, and their exponentials are to be computed at each stage until a dynamically evaluated criterion allows to stop. Our algorithm is based on scaling and squaring. By carefully reusing certain intermediate quantities from one step to the next, we can efficiently compute such a sequence of matrix exponentials.

AMS subject classifications. 15A16, 65F60, 91G20

Key words. matrix exponential, block triangular matrix, polynomial diffusion models, option pricing

1. Introduction.

We study the problem of computing the matrix exponential for a sequence of nested block triangular matrices. In order to give a precise problem formulation, consider a sequence of block upper triangular matrices $G_0, G_1, G_2, \ldots$ of the form

$$G_n = \begin{bmatrix} G_{0,0} & G_{0,1} & \cdots & G_{0,n} \\ & G_{1,1} & \cdots & G_{1,n} \\ & & \ddots & \vdots \\ & & & G_{n,n} \end{bmatrix} \in \mathbb{R}^{d_n \times d_n}, \tag{1.1}$$

where all diagonal blocks $G_{n,n}$ are square. In other words, the matrix G_i arises from G_{i-1} by appending a block column (and adjusting the size). We aim at computing the sequence of matrix exponentials

$$\exp(G_0),\ \exp(G_1),\ \exp(G_2),\ \ldots\ . \tag{1.2}$$

One could, of course, simply compute each of the exponentials (1.2) individually using standard techniques; see [11] for an overview. However, the sequence of matrix exponentials (1.2) inherits the nested structure from the matrices G_n in (1.1), i.e., $\exp(G_{n-1})$ is a leading principle submatrix of $\exp(G_n)$. In effect only the last block column of $\exp(G_n)$ needs to be computed, and the goal of this paper is to explain how this can be achieved in a numerically safe manner.

In the special case where the spectra of the diagonal blocks $G_{n,n}$ are separated, Parlett's method [13] yields—in principle—an efficient computational scheme: compute the exponentials $F_{0,0} := \exp(G_{0,0})$ and $F_{1,1} := \exp(G_{1,1})$ separately, then the missing (1,2) block of $\exp(G_1)$ is given as the unique solution X to the Sylvester equation

$$G_{0,0}X - XG_{1,1} = F_{0,0}G_{0,1} - G_{0,1}F_{1,1}.$$

Continuing in this manner, all the off-diagonal blocks required to compute (1.2) could be obtained from solving Sylvester equations. However, it is well known (see [8, Chapter 9]) that Parlett's method is numerically safe only when the spectra of the diagonal blocks are well

*Received February 28, 2017. Accepted June 29, 2017. Published online on September 18, 2017. Recommended by K. Jbilou. F. Statti's research was supported through the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 307465-POLYTE.

†École Polytechnique Fédérale de Lausanne, Station 8, 1015 Lausanne, Switzerland
({daniel.kressner, robert.luce, francesco.statti}@epfl.ch).

ETNA
Kent State University and
Johann Radon Institute (RICAM)

separated in the sense that all involved Sylvester equations are well-conditioned. Since we consider the block structure as fixed, imposing such a condition would severely limit the scope of applications; it is certainly not met by the application we discuss below.

A general class of applications for the described incremental computation of exponentials arises from the matrix representations of a linear operator $\mathcal{G} : V \to V$ restricted to a sequence of nested, finite-dimensional subspaces of a given infinite-dimensional vector space V. More precisely, one starts with a finite-dimensional subspace V_0 of V with a basis $\mathcal{B}_0$. Successively, the vector space V_0 is extended to $V_1 \subseteq V_2 \subseteq \cdots \subseteq V$ by generating a sequence of nested bases $\mathcal{B}_0 \subseteq \mathcal{B}_1 \subseteq \mathcal{B}_2 \subseteq \cdots$. Assume that $\mathcal{G}V_n \subseteq V_n$ for all $n = 0, 1, \ldots$, and consider the sequence of matrix representations G_n of $\mathcal{G}$ with respect to $\mathcal{B}_n$. Due to the nestedness of the bases, G_n is constructed from G_{n-1} by adding the columns representing the action of $\mathcal{G}$ to $\mathcal{B}_n \setminus \mathcal{B}_{n-1}$. As a result, we obtain a sequence of matrices structured as in (1.1).

A specific example for the scenario outlined above arises in computational finance, when pricing options are based on polynomial diffusion models; see [6]. As we explain in more detail in Section 3, in this setting $\mathcal{G}$ is the generator of a stochastic differential equation (SDE), and then V_n are nested subspaces of multivariate polynomials. Some pricing techniques require the computation of certain conditional moments that can be extracted from the matrix exponentials (1.2). While increasing n allows for a better approximation of the option price, the value of n required to attain a desired accuracy is usually not known a priori. Algorithms that choose n adaptively can be expected to rely on the incremental computation of the whole sequence (1.2).

Exponentials of block triangular matrices have also been studied in other contexts. For two-by-two block triangular matrices, Dieci and Papini study conditioning issues in [4] and discuss the choice of scaling parameters for using Padé approximants to exponential function in [3]. In the case where the matrix is also block-Toeplitz, a fast exponentiation algorithm is developed in [2].

The rest of this paper is organized as follows. In Section 2 we give a detailed description of our algorithm for incrementally computing exponentials of block triangular matrices as in (1.1). In Section 3 we discuss polynomial diffusion models and some pricing techniques which necessitate the use of such an incremental algorithm. Finally, numerical results are presented in Section 4.

2. Incremental scaling and squaring. Since the set of conformally partitioned block triangular matrices forms an algebra and $\exp(G_n)$ is a polynomial in G_n, the matrix $\exp(G_n)$ has the same block upper triangular structure as G_n, that is,

$$\exp(G_n) = \begin{bmatrix} \exp(G_{0,0}) & * & \cdots & * \\ & \exp(G_{1,1}) & \ddots & \vdots \\ & & \ddots & * \\ & & & \exp(G_{n,n}) \end{bmatrix} \in \mathbb{R}^{d_n \times d_n}.$$

As outlined in the introduction, we aim at computing $\exp(G_n)$ block column by block column, from left to right. Our algorithm is based on the scaling and squaring methodology, which we briefly summarize next.

2.1. Summary of the scaling and squaring method. The scaling and squaring method uses a rational function to approximate the exponential function and typically involves three steps. Denote by $r_{k,m}(z) = \frac{p_{k,m}(z)}{q_{k,m}(z)}$ the (k, m)-Padé approximant to the exponential function, meaning that the numerator is a polynomial of degree k and the denominator is a polynomial of degree m. These Padé approximants are very accurate close to the origin, and in a first step

ETNA
Kent State University and
Johann Radon Institute (RICAM)

the input matrix G is therefore scaled by a power of two, so that $\|2^{-s}G\|$ is small enough to guarantee an accurate approximation $r_{k,m}(2^{-s}G) \approx \exp(2^{-s}G)$.

The second step consists of evaluating the rational approximation $r_{k,m}(2^{-s}G)$, and, finally, an approximation to $\exp(G)$ is obtained in a third step by repeatedly squaring the result, i.e.,

$$\exp(G) \approx r_{k,m}(2^{-s}G)^{2^s}.$$

Different choices of the scaling parameter s and of the approximation degrees k and m yield methods of different characteristics. The choice of these parameters is critical for the approximation quality and for the computational efficiency; see [8, Chapter 10].

In what follows we describe techniques that allow for an incremental evaluation of the matrix exponential of the block triangular matrix (1.1) using scaling and squaring. These techniques can be used with any choice for the actual underlying scaling and squaring method, defined through the parameters s, k, and m.

2.2. Tools for the incremental computation of exponentials. Before explaining the algorithm, we first introduce some notation that is used throughout. The matrix G_n from (1.1) can be written as

$$G_n = \left[\begin{array}{ccc|c} G_{0,0} & \cdots & G_{0,n-1} & G_{0,n} \\ & \ddots & \vdots & \vdots \\ & & G_{n-1,n-1} & G_{n-1,n} \\ \hline & & & G_{n,n} \end{array}\right] =: \begin{bmatrix} G_{n-1} & g_n \\ 0 & G_{n,n} \end{bmatrix}, \tag{2.1}$$

where $G_{n-1} \in \mathbb{R}^{d_{n-1}\times d_{n-1}}$, $G_{n,n} \in \mathbb{R}^{b_n \times b_n}$, so that $g_n \in \mathbb{R}^{d_{n-1}\times b_n}$. Let s be the scaling parameter and $r = \frac{p}{q}$ the rational function used in the approximation (for simplicity we will often omit the indices k and m). We denote the scaled matrix by $\tilde{G}_n := 2^{-s}G_n$, and we partition it as in (2.1).

The starting point of the algorithm consists in computing the Padé approximant of the exponential $\exp(G_0) = \exp(G_{0,0})$ using a scaling and squaring method. Then, the sequence of matrix exponentials (1.2) is incrementally computed by reusing at each step previously obtained quantities. So more generally, assume that $\exp(G_{n-1})$ has been approximated using a scaling and squaring method. The three main computational steps for obtaining the Padé approximant of $\exp(G_n)$ are

1. evaluating the polynomials $p\left(\tilde{G}_n\right)$, $q\left(\tilde{G}_n\right)$,
2. evaluating $p\left(\tilde{G}_n\right)^{-1} q(\tilde{G}_n)$, and
3. repeatedly squaring it.

We now discuss each of these steps separately noting the quantities to keep at every iteration.

2.2.1. Evaluating $p\left(\tilde{G}_n\right), q\left(\tilde{G}_n\right)$ from $p\left(\tilde{G}_{n-1}\right), q\left(\tilde{G}_{n-1}\right)$. Similarly to (2.1), we start by writing $P_n := p\left(\tilde{G}_n\right)$ and $Q_n := q\left(\tilde{G}_n\right)$ as

$$P_n = \begin{bmatrix} P_{n-1} & p_n \\ 0 & P_{n,n} \end{bmatrix}, \quad Q_n = \begin{bmatrix} Q_{n-1} & q_n \\ 0 & Q_{n,n} \end{bmatrix}.$$

In order to evaluate P_n, we first need to compute monomials of $\tilde{G}_n$ that can be written as

$$\tilde{G}_n^l = \begin{bmatrix} \tilde{G}_{n-1}^l & \sum_{j=0}^{l-1} \tilde{G}_{n-1}^j \tilde{g}_n \tilde{G}_{n,n}^{l-j-1} \\ & \tilde{G}_{n,n}^l \end{bmatrix}, \quad \text{for } l = 1, \ldots, k.$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Denote by $X_l := \sum_{j=0}^{l-1} \tilde{G}_{n-1}^j \tilde{g}_n \tilde{G}_{n,n}^{l-j-1}$ the upper off-diagonal block of $\tilde{G}_n^l$, then we have the relation

$$X_l = \tilde{G}_{n-1} X_{l-1} + \tilde{g}_n \tilde{G}_{n,n}^{l-1}, \qquad \text{for } l = 2, \cdots, k,$$

with $X_1 := \tilde{g}_n$ so that all the monomials $\tilde{G}_n^l$, $l = 1, \ldots, k$, can be computed in $\mathcal{O}(b_n^3 + d_{n-1} b_n^2 + d_{n-1}^2 b_n)$. Let $p(z) = \sum_{l=0}^k \alpha_l z^l$ be the numerator polynomial of r, then we have that

$$P_n = \begin{bmatrix} P_{n-1} & \sum_{l=0}^k \alpha_l X_l \\ & p(\tilde{G}_{n,n}) \end{bmatrix}, \tag{2.2}$$

which can be assembled in $\mathcal{O}(b_n^2 + d_{n-1} b_n)$ since only the last block column needs to be computed. The complete evaluation of P_n is summarized in Algorithm 1.

Algorithm 1 Evaluation of P_n, using P_{n-1}.

Input: $G_{n-1}, G_{n,n}, g_n, P_{n-1}$, Padé coefficients $\alpha_l, l = 0, \cdots, k$.
Output: P_n.

1: $\tilde{g}_n \leftarrow 2^{-s} g_n$, $\tilde{G}_{n,n} \leftarrow 2^{-s} G_{n,n}$, $\tilde{G}_{n-1} \leftarrow 2^{-s} G_{n-1}$
2: $X_1 \leftarrow \tilde{g}_n$
3: **for** $l = 2, 3, \cdots, k$ **do**
4: Compute $\tilde{G}_{n,n}^l$
5: $X_l = \tilde{G}_{n-1} X_{l-1} + \tilde{g}_n \tilde{G}_{n,n}^{l-1}$
6: **end for**
7: $X_0 \leftarrow \mathbf{0}_{d_{n-1} \times b_n}$
8: Compute off diagonal block of P_n: $\sum_{l=0}^k \alpha_l X_l$
9: Compute $p(\tilde{G}_{n,n}) = \sum_{l=0}^k \alpha_l \tilde{G}_{n,n}^l$
10: Assemble P_n as in (2.2)

Similarly, one computes Q_n from Q_{n-1}, using again the matrices X_l.

2.2.2. Evaluating $Q_n^{-1} P_n$. With the matrices P_n and Q_n at hand, we now need to compute the rational approximation $Q_n^{-1} P_n$. We assume that Q_n is well-conditioned, in particular, non-singular, which is ensured by the choice of the scaling parameter and of the Padé approximation; see, e.g., [9]. We focus on the computational cost. For simplicity, we introduce the notation

$$\tilde{F}_n = \begin{bmatrix} \tilde{F}_{0,0} & \cdots & \tilde{F}_{0,n} \\ & \ddots & \vdots \\ & & \tilde{F}_{n,n} \end{bmatrix} := Q_n^{-1} P_n, \qquad F_n = \begin{bmatrix} F_{0,0} & \cdots & F_{0,n} \\ & \ddots & \vdots \\ & & F_{n,n} \end{bmatrix} := \tilde{F}_n^{2^s},$$

and we see that

$$\begin{aligned} \tilde{F}_n = Q_n^{-1} P_n &= \begin{bmatrix} Q_{n-1}^{-1} & -Q_{n-1}^{-1} q_n Q_{n,n}^{-1} \\ 0 & Q_{n,n}^{-1} \end{bmatrix} \begin{bmatrix} P_{n-1} & p_n \\ 0 & P_{n,n} \end{bmatrix} \\ &= \begin{bmatrix} \tilde{F}_{n-1} & Q_{n-1}^{-1}(p_n - q_n Q_{n,n}^{-1} P_{n,n}) \\ 0 & Q_{n,n}^{-1} P_{n,n} \end{bmatrix}. \end{aligned} \tag{2.3}$$

To solve the linear system $Q_{n,n}^{-1} P_{n,n}$, we compute an LU decomposition with partial pivoting for $Q_{n,n}$, requiring $\mathcal{O}(b_n^3)$ operations. This LU decomposition is saved for future

ETNA
Kent State University and
Johann Radon Institute (RICAM)

use, and hence we may assume that we have available the LU decompositions for all diagonal blocks from previous computations:

$$\Pi_l Q_{l,l} = L_l U_l, \quad l = 0, \ldots, n-1. \tag{2.4}$$

Here, $\Pi_l \in \mathbb{R}^{b_l \times b_l}$, $l = 0, \ldots, n-1$, are permutation matrices, $L_l \in \mathbb{R}^{b_l \times b_l}$, $l = 0, \ldots, n-1$, are lower triangular matrices, and $U_l \in \mathbb{R}^{b_l \times b_l}$, $l = 0, \ldots, n-1$, are upper triangular matrices.

Set $Y_n := p_n - q_n Q_{n,n}^{-1} P_{n,n} \in \mathbb{R}^{d_{n-1} \times b_n}$, and partition it as

$$Y_n = \begin{bmatrix} Y_{0,n} \\ \vdots \\ Y_{n-1,n} \end{bmatrix}.$$

Then we compute $Q_{n-1}^{-1} Y_n$ by block backward substitution using the decompositions of the diagonal blocks. Hence this computation requires $\mathcal{O}(d_{n-1}^2 b_n + d_{n-1} b_n^2)$ operations, so that the number of operations for computing $\tilde{F}_n$ is $\mathcal{O}(b_n^3 + d_{n-1}^2 b_n + d_{n-1} b_n^2)$. Algorithm 2 describes the complete procedure to compute $\tilde{F}_n$.

Algorithm 2 Evaluation of $\tilde{F}_n = Q_n^{-1} P_n$.

Input: Q_n, P_n and quantities (2.4)
Output: $\tilde{F}_n = Q_n^{-1} P_n$ and LU decomposition of $Q_{n,n}$.

1: Compute $\Pi_n Q_{n,n} = L_n U_n$ and keep it for future use (2.4)
2: Compute $\tilde{F}_{n,n} := Q_{n,n}^{-1} P_{n,n}$
3: $Y_n = p_n - q_n Q_{n,n}^{-1} P_{n,n}$
4: $\tilde{F}_{n-1,n} = U_{n-1}^{-1} L_{n-1}^{-1} \Pi_{n-1} Y_{n-1,n}$
5: **for** $l = n-2, n-3, \cdots, 0$ **do**
6: $\quad \tilde{F}_{l,n} = U_l^{-1} L_l^{-1} \Pi_l (Y_{l,n} - \sum_{j=l+1}^{n-1} Q_{l,j} \tilde{F}_{j,n})$
7: **end for**
8: Assemble $\tilde{F}_n$ as in (2.3)

2.2.3. The squaring phase. Having computed $\tilde{F}_n$, which we write as

$$\tilde{F}_n = \begin{bmatrix} \tilde{F}_{n-1} & \tilde{f}_n \\ & \tilde{F}_{n,n} \end{bmatrix},$$

we now need to compute s repeated squares of that matrix, i.e.,

$$\tilde{F}_n^{2^l} = \begin{bmatrix} \tilde{F}_{n-1}^{2^l} & \sum_{j=0}^{l-1} \tilde{F}_{n-1}^{2^{l-1+j}} \tilde{f}_n \tilde{F}_{n,n}^{2^j} \\ & \tilde{F}_{n,n}^{2^l} \end{bmatrix}, \quad l = 1, \ldots, s, \tag{2.5}$$

so that $F_n = \tilde{F}_n^{2^s}$. Setting $Z_l := \sum_{j=0}^{l-1} \tilde{F}_{n-1}^{2^{l-1+j}} \tilde{f}_j \tilde{F}_{n,n}^{2^j}$, we have the recurrence

$$Z_l = \tilde{F}_{n-1}^{2^{l-1}} Z_{l-1} + Z_{l-1} \tilde{F}_{n,n}^{2^{l-1}},$$

with $Z_0 := \tilde{f}_n$. Hence, if we have stored the intermediate squares from the computation of F_{n-1}, i.e.,

$$\tilde{F}_{n-1}^{2^l}, \quad l = 1, \ldots, s, \tag{2.6}$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

then we can compute all the quantities Z_l, for $l = 1, \ldots, s$, in a complexity of $\mathcal{O}(d_{n-1}^2 b_n + d_{n-1} b_n^2)$ operations so that the total cost for computing F_n (and the intermediate squares of $\tilde{F}_n$) is $\mathcal{O}(d_{n-1}^2 b_n + d_{n-1} b_n^2 + b_n^3)$. Again, we summarize the squaring phase in the following algorithm.

Algorithm 3 Evaluation of $F_n = \tilde{F}_n^{2^s}$.

Input: $\tilde{F}_{n-1}, \tilde{f}_n, \tilde{F}_{n,n}$, quantities (2.6).
Output: F_n and updated intermediates.

1: $Z_0 \leftarrow \tilde{f}_n$
2: **for** $l = 1, 2, \cdots, s$ **do**
3: Compute $\tilde{F}_{n,n}^{2^l}$
4: $Z_l = \tilde{F}_{n-1}^{2^{l-1}} Z_{l-1} + Z_{l-1} \tilde{F}_{n,n}^{2^{l-1}}$
5: Assemble $\tilde{F}_n^{2^l}$ as in (2.5) and save it
6: **end for**
7: $F_n \leftarrow \tilde{F}_n^{2^s}$

2.3. Overall algorithm. Using the techniques from the previous section, we now give a concise description of the overall algorithm. We assume that the quantities listed in equations (2.4) and (2.6) are stored in memory, with a space requirement of $\mathcal{O}(d_{n-1}^2)$.

In view of this, we assume that F_{n-1} and the aforementioned intermediate quantities have been computed. Algorithm 4 describes the overall procedure to compute F_n and to update the intermediates; we continue to use the notation introduced in (2.1).

Algorithm 4 Computation of $F_n \approx \exp(G_n)$, using F_{n-1}.

Input: Block column g_n, diagonal block $G_{n,n}$, quantities (2.4), and (2.6).
Output: F_n, and updated intermediates.

1: Extend P_{n-1} to P_n using Algorithm 1, and form analogously Q_n
2: Compute $\tilde{F}_n$ using Algorithm 2
3: Evaluate $F_n = \tilde{F}_n^{2^s}$ using Algorithm 3

As explained in the previous section, the number of operations for each step in Algorithm 4 is $\mathcal{O}(d_{n-1}^2 b_n + d_{n-1} b_n^2 + b_n^3)$, using the notation introduced at the beginning of Section 2.2. If F_n were simply computed from scratch, without the use of the intermediates, the number of operations for scaling and squaring would be $\mathcal{O}((d_{n-1} + b_n)^3)$. In the typical situation where $d_{n-1} \gg b_n$, the dominant term in the latter complexity bound is d_{n-1}^3, which is absent from the complexity bound of Algorithm 4.

In order to solve our original problem, i.e., the computation of the sequence $\exp(G_0)$, $\exp(G_1)$, $\exp(G_2)$, $\ldots$, we use Algorithm 4 repeatedly; the resulting procedure is shown in Algorithm 5.

We now derive a complexity bound for the number of operations spent in Algorithm 5. For simplicity of notation we consider the case where all diagonal blocks are of equal size, i.e., $b_k \equiv b \in \mathbb{N}$ so that $d_k = (k+1)b$. At iteration k the number of operations spent within Algorithm 4 is thus $\mathcal{O}(k^2 b^3)$. Assume that the termination criterion used in Algorithm 5 effects to stop the procedure after the computation of F_n. The overall complexity bound for the number of operations until termination is $\mathcal{O}(\sum_{k=0}^{n} k^2 b^3) = \mathcal{O}(n^3 b^3)$, which matches the

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Algorithm 5 Approximation of $\exp(G_0), \exp(G_1), \ldots$

Input: Padé approximation parameters k, m, and s
Output: $F_0 \approx \exp(G_0)$, $F_1 \approx \exp(G_1), \ldots$

```
Compute F_0 using scaling and squaring, store intermediates for Algorithm 4
for n = 1, 2, ... do
    Compute F_n from F_{n-1} using Algorithm 4
    if termination criterion is satisfied then
        return
    end if
end for
```

complexity bound of applying scaling and squaring only to $G_n \in \mathbb{R}^{(n+1)b\times(n+1)b}$, which is also $\mathcal{O}((nb)^3)$.

In summary the number of operations needed to compute F_n by Algorithm 5 is asymptotically the same as applying the same scaling and squaring setting *only* to compute $\exp(G_n)$, while Algorithm 5 incrementally reveals *all* exponentials $\exp(G_0), \ldots, \exp(G_n)$ in the course of the iteration, satisfying our requirements outlined in the introduction.

2.4. Adaptive scaling. In Algorithms 4 and 5 we have assumed that the scaling power s is given as input parameter and that it is fixed throughout the computation of the exponentials $\exp(G_0), \ldots, \exp(G_n)$. This is in contrast to what is usually intended in the scaling and squaring method; see Section 2.1. On the one hand s must be sufficiently large so that $r_{k,m}(2^{-s}G_l) \approx \exp(2^{-s}G_l)$, for $0 \le l \le n$. If, on the other hand, s is chosen *too large*, then the evaluation of $r_{k,m}(2^{-s}G_l)$ may become inaccurate due to *overscaling*. So if s is fixed and the norms $\|G_l\|$ grow with increasing l, as one would normally expect, then an accurate approximation cannot be guaranteed for all l.

Most scaling and squaring designs hence choose s in dependence of the norm of the input matrix [11, 7, 9]. For example, in the algorithm by Higham described in [9], it is the smallest integer satisfying

$$\|2^{-s}G_l\|_1 \le \theta \approx 5.37\ldots. \tag{2.7}$$

In order to combine our incremental evaluation techniques with this scaling and squaring design, the scaling power s must thus be chosen dynamically in the course of the evaluation. Assume that s satisfies the criterion (2.7) at step $l-1$ but not at step l. We then simply discard all accumulated data structures from Algorithm 4, increase s to match the bound (2.7) for G_l, and start Algorithm 5 anew with the *repartitioned* input matrix

$$G_n = \left[\begin{array}{ccc|c|c|c} G_{0,0} & \cdots & G_{0,l} & G_{0,l+1} & \cdots & G_{0,n} \\ & \ddots & \vdots & \vdots & & \vdots \\ & & G_{l,l} & G_{l,l+1} & \cdots & G_{l,n} \\ \hline & & & G_{l+1,l+1} & \cdots & G_{l+1,n} \\ & & & & \ddots & \vdots \\ & & & & & G_{n,n} \end{array}\right] = \underbrace{\left[\begin{array}{c|c|c|c} \hat{G}_{0,0} & \hat{G}_{0,1} & \cdots & \hat{G}_{0,n-l} \\ \hline & \hat{G}_{1,1} & \cdots & \hat{G}_{1,n-l} \\ & & \ddots & \vdots \\ & & & \hat{G}_{n-l,n-l} \end{array}\right]}_{=:\hat{G}_{n-l}}. \tag{2.8}$$

The procedure is summarized in Algorithm 6.

It turns out that the computational overhead induced by this restarting procedure is quite modest. In the notation introduced for the complexity discussion in Section 2.3, the number of operations for computing $\exp(G_n)$ by Higham's scaling and squaring method is $\mathcal{O}(\log(\|G_n\|_1)(nb)^3)$. Since there are at most $\log(\|G_n\|_1)$ restarts in Algorithm 6, the total

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Algorithm 6 Approximation of $\exp(G_0), \exp(G_1), \ldots$ with adaptive scaling.

Input: Padé approximation parameters k, m, norm bound θ.
Output: $F_0 \approx \exp(G_0)$, $F_1 \approx \exp(G_1), \ldots$

$s \leftarrow \max\{0, \log(\|G_0\|_1)\}$
Compute F_0 using scaling and squaring, store intermediates for Algorithm 4
for $l = 1, 2, \ldots$ **do**
  if $\|G_l\|_1 > \theta$ **then**
    Repartition $G_n = \hat{G}_{n-l}$ as in (2.8)
    Restart algorithm with $\hat{G}_{n-l}$
  end if
  Compute F_l from F_{l-1} using Algorithm 4
  if termination criterion is satisfied **then**
    return
  end if
end for

number of operations for incrementally computing all exponentials $\exp(G_0), \ldots, \exp(G_n)$ can be bounded by a function in $\mathcal{O}(\log(\|G_n\|_1)^2(nb)^3)$. We assess the actual performance of Algorithm 6 in Section 4.

In our application from option pricing it turns out that the norms of the matrices G_l do not grow dramatically (see Sections 3.2 and 3.3), and quite accurate approximations to all the matrix exponentials can be computed even if the scaling factor is fixed (see Section 4.2).

3. Option pricing in polynomial models. The main purpose of this section is to explain how certain option pricing techniques require the sequential computation of matrix exponentials for block triangular matrices. The description will necessarily be rather brief; we refer, e.g., to the textbook [5] for more details.

Because we are evaluating at initial time $t = 0$, the price of a certain option expiring at time $\tau > 0$ consists of computing an expression of the form

$$e^{-r\tau}\mathbb{E}[f(X_\tau)], \tag{3.1}$$

where $(X)_{0\le t\le\tau}$ is a d-dimensional stochastic process modeling the price of financial assets over the time interval $[0, \tau]$, $f : \mathbb{R}^d \to \mathbb{R}$ is the so-called payoff function, and r represents a fixed interest rate. In the following, we consider stochastic processes described by an SDE of the form

$$dX_t = b(X_t)dt + \Sigma(X_t)dW_t, \tag{3.2}$$

where W denotes a d-dimensional Brownian motion, $b : \mathbb{R}^d \mapsto \mathbb{R}^d$, and $\Sigma : \mathbb{R}^d \mapsto \mathbb{R}^{d\times d}$.

3.1. Polynomial diffusion models. During the last years, polynomial diffusion models have become a versatile tool in financial applications, including option pricing. In the following, we provide a short summary and refer to the paper by Filipović and Larsson [6] for the mathematical foundations.

For a polynomial diffusion process, one assumes that the coefficients of the vector b in (3.2) and the matrix $A := \Sigma\Sigma^T$ satisfy

$$A_{ij} \in \mathrm{Pol}_2(\mathbb{R}^d), \qquad b_i \in \mathrm{Pol}_1(\mathbb{R}^d) \qquad \text{for } i, j = 1, \ldots, d. \tag{3.3}$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Here, $\mathrm{Pol}_n(\mathbb{R}^d)$ represents the set of d-variate polynomials of total degree at most n, that is,

$$\mathrm{Pol}_n(\mathbb{R}^d) := \left\{ \sum_{0 \le |\mathbf{k}| \le n} \alpha_{\mathbf{k}} x^{\mathbf{k}} \mid x \in \mathbb{R}^d, \alpha_{\mathbf{k}} \in \mathbb{R} \right\},$$

where we use multi-index notation: $\mathbf{k} = (k_1, \dots, k_d) \in \mathbb{N}_0^d$, $|\mathbf{k}| := k_1 + \cdots + k_d$, and $x^{\mathbf{k}} := x_1^{k_1} \dots x_d^{k_d}$. In the following, $\mathrm{Pol}(\mathbb{R}^d)$ represents the set of all multivariate polynomials on $\mathbb{R}^d$.

Associated with A and b we define the partial differential operator $\mathcal{G}$ by

$$\mathcal{G}f = \frac{1}{2} \mathrm{Tr}(A \nabla^2 f) + b^T \nabla f, \tag{3.4}$$

which represents the so called generator for (3.2); see [12]. It can be directly verified that (3.3) implies that $\mathrm{Pol}_n(\mathbb{R}^d)$ is invariant under $\mathcal{G}$ for any $n \in \mathbb{N}$, that is,

$$\mathcal{G}\mathrm{Pol}_n(\mathbb{R}^d) \subseteq \mathrm{Pol}_n(\mathbb{R}^d). \tag{3.5}$$

REMARK 3.1. In many applications, one is interested in solutions to (3.2) that lie on a state space $E \subseteq \mathbb{R}^d$ to incorporate, for example, nonnegativity. This problem is largely studied in [6], where existence and uniqueness of solutions to (3.2) on several types of state spaces $E \subseteq \mathbb{R}^d$ and for large classes of A and b is shown. Let us now fix a basis of polynomials $\mathcal{H}_n = \{h_1, \dots, h_N\}$ for $\mathrm{Pol}_n(\mathbb{R}^d)$, where $N = \dim \mathrm{Pol}_n(\mathbb{R}^d) = \binom{n+d}{n}$, and write

$$H_n(x) = (h_1(x), \dots, h_N(x))^T.$$

Let G_n denote the matrix representation with respect to $\mathcal{H}$ of the linear operator $\mathcal{G}$ restricted to $\mathrm{Pol}_n(\mathbb{R}^d)$. By definition,

$$\mathcal{G}p(x) = H_n(x)^T G_n \vec{p}$$

for any $p \in \mathrm{Pol}_n(\mathbb{R}^d)$ with coordinate vector $\vec{p} \in \mathbb{R}^N$ with respect to $\mathcal{H}_n$. By [6, Theorem 3.1], the corresponding polynomial moment can be computed from

$$\mathbb{E}[p(X_\tau)] = H_n(X_0)^T e^{\tau G_n} \vec{p}. \tag{3.6}$$

The setting discussed above corresponds to the scenario described in the introduction. We have a sequence of subspaces

$$\mathrm{Pol}_0(\mathbb{R}^d) \subseteq \mathrm{Pol}_1(\mathbb{R}^d) \subseteq \mathrm{Pol}_2(\mathbb{R}^d) \subseteq \cdots \subseteq \mathrm{Pol}(\mathbb{R}^d),$$

and the polynomial preserving property (3.5) implies that the matrix representation G_n is block upper triangular with $n+1$ square diagonal blocks of size

$$1, d, \binom{1+d}{2}, \dots, \binom{n+d-1}{n}.$$

In the rest of this section we introduce two different pricing techniques that require the incremental computation of polynomial moments of the form (3.6).

ETNA
Kent State University and
Johann Radon Institute (RICAM)

3.2. Moment-based option pricing for Jacobi models. The Jacobi stochastic volatility model is a special case of a polynomial diffusion model and it is characterized by the SDE

$$dY_t = (r - V_t/2)dt + \rho\sqrt{Q(V_t)}dW_{1t} + \sqrt{V_t - \rho^2 Q(V_t)}dW_{2t},$$
$$dV_t = \kappa(\theta - V_t)dt + \sigma\sqrt{Q(V_t)}dW_{1t},$$

where

$$Q(v) = \frac{(v - v_{\min})(v_{\max} - v)}{(\sqrt{v_{\max}} - \sqrt{v_{\min}})^2},$$

for some $0 \leq v_{\min} < v_{\max}$. Here, W_{1t} and W_{2t} are independent standard Brownian motions and the model parameters satisfy the conditions $\kappa \geq 0$, $\theta \in [v_{\min}, v_{\max}]$, $\sigma > 0$, $r \geq 0$, $\rho \in [-1, 1]$. In their paper, Ackerer et al. [1] use this model in the context of option pricing where the price of the asset is specified by $S_t := e^{Y_t}$ and V_t represents the squared stochastic volatility. In the following, we briefly introduce the pricing technique they propose and explain how it involves the incremental computation of polynomial moments.

Under the Jacobi model with the discounted payoff function f of an European claim, the option price (3.1) at initial time $t = 0$ can be expressed as

$$\sum_{n \geq 0} f_n l_n, \tag{3.7}$$

where $\{f_n, n \geq 0\}$ are the Fourier coefficients of f and $\{l_n, n \geq 0\}$ are Hermite moments. As explained in [1], the Fourier coefficients can be conveniently computed in a recursive manner. The Hermite moments are computed using (3.6). Specifically, consider the monomial basis of $\mathrm{Pol}_n(\mathbb{R}^2)$:

$$H_n(y, v) := (1, y, v, y^2, yv, v^2, \ldots, y^n, y^{n-1}v, \ldots, v^n)^T. \tag{3.8}$$

Then

$$l_n = H_n(Y_0, V_0)^T e^{\tau G_n} \vec{h}_n, \tag{3.9}$$

where $\vec{h}_n$ contains the coordinates with respect to (3.8) of

$$\frac{1}{\sqrt{n!}} h_n\left(\frac{y - \mu_w}{\sigma_w}\right),$$

with real parameters σ_w, μ_w and the nth Hermite polynomial h_n.

Truncating the sum (3.7) after a finite number of terms allows us to obtain an approximation of the option price. Algorithm 7 describes a heuristic to selecting the truncation based on the absolute value of the summands using Algorithm 5 for computing the required moments incrementally.

As discussed in Section 2, a norm estimate for G_n is instrumental for choosing a priori the scaling parameter in the scaling and squaring method. The following lemma provides such an estimate for the model under consideration.

LEMMA 3.2. *Let G_n be the matrix representation of the operator $\mathcal{G}$ defined in (3.4) with respect to the basis (3.8) of $\mathrm{Pol}_n(\mathbb{R}^2)$. Define*

$$\alpha := \frac{\sigma(1 + v_{\min}v_{\max} + v_{\max} + v_{\min})}{2(\sqrt{v_{\max}} - \sqrt{v_{\min}})^2}.$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Algorithm 7 Option pricing for the European call option under the Jacobi stochastic volatility model.

Input: Model and payoff parameters, tolerance ϵ
Output: Approximate option price

1: $n = 0$
2: Compute l_0, f_0; set Price $= l_0 f_0$.
3: **while** $|l_n f_n| > \epsilon \cdot$ Price **do**
4: $\quad n = n + 1$
5: $\quad$ Compute $\exp(\tau G_n)$ using Algorithm 4.
6: $\quad$ Compute Hermite moment l_n using (3.9).
7: $\quad$ Compute Fourier coefficient f_n as described in [1].
8: $\quad$ Price $=$ Price $+ l_n f_n$;
9: **end while**

Then the matrix 1-norm of G_n is bounded by

$$n(r + \kappa + \kappa\theta - \sigma\alpha) + \frac{1}{2}n^2(1 + |\rho|\alpha + 2\sigma\alpha).$$

Proof. The operator $\mathcal{G}$ in the Jacobi model takes the form

$$\mathcal{G}f(y,v) = \frac{1}{2}\operatorname{Tr}(A(v)\nabla^2 f(y,v)) + b(v)^\top \nabla f(y,v),$$

where

$$b(v) = \begin{bmatrix} r - v/2 \\ \kappa(\theta - v) \end{bmatrix}, \quad A(v) = \begin{bmatrix} v & \rho\sigma Q(v) \\ \rho\sigma Q(v) & \sigma^2 Q(v) \end{bmatrix}.$$

Setting $S := (\sqrt{v_{\max}} - \sqrt{v_{\min}})^2$, we consider the action of the generator $\mathcal{G}$ on a basis element $y^p v^q$:

$$\begin{aligned}\mathcal{G}y^p v^q =& y^{p-2}v^{q+1}p\frac{p-1}{2} - y^{p-1}v^{q+1}p\Big(\frac{1}{2} + \frac{q\rho\sigma}{S}\Big) + y^{p-1}v^q p\Big(r + q\rho\sigma\frac{v_{\max}+v_{\min}}{S}\Big)\\ &- y^{p-1}v^{q-1}\frac{pq\rho\sigma v_{\max}v_{\min}}{S} - y^p v^q q\Big(\kappa + \frac{q-1}{2}\frac{\sigma^2}{S}\Big)\\ &- y^p v^{q-2}q\frac{q-1}{2}\frac{\sigma^2 v_{\max}v_{\min}}{S} + y^p v^{q-1}q\Big(\kappa\theta + \frac{q-1}{2}\sigma^2\frac{v_{\max}+v_{\min}}{S}\Big).\end{aligned}$$

For the matrix 1-norm of G_n, one needs to determine the values $(p,q) \in \mathcal{M}$, where we define $\mathcal{M} := \{(p,q) \in \mathbb{N}_0 \times \mathbb{N}_0 | p+q \le n\}$, for which the 1-norm of the coordinate vector of $\mathcal{G}y^p v^q$ becomes maximal. Taking into account the nonnegativity of the involved model parameters and replacing ρ by $|\rho|$, we obtain an upper bound as follows:

$$\begin{aligned}&p\frac{p-1}{2} + p\Big(\frac{1}{2} + \frac{q|\rho|\sigma}{S}\Big) + p\Big(r + q|\rho|\sigma\frac{v_{\max}+v_{\min}}{S}\Big) + \frac{pq|\rho|\sigma v_{\max}v_{\min}}{S}\\ &\quad + q\Big(\kappa + \frac{q-1}{2}\frac{\sigma^2}{S}\Big) + q\frac{q-1}{2}\frac{\sigma^2 v_{\max}v_{\min}}{S} + q\Big(\kappa\theta + \frac{q-1}{2}\sigma^2\frac{v_{\max}+v_{\min}}{S}\Big)\\ &= pr + q\kappa(\theta+1) + \frac{1}{2}p^2 + 2pq|\rho|\alpha + q(q-1)\sigma\alpha\\ &\le n(r + \kappa + \kappa\theta) + \frac{1}{2}n^2 + 2pq|\rho|\alpha + n(n-1)\sigma\alpha.\end{aligned}$$

This completes the proof, noting that the maximum of pq on $\mathcal{M}$ is bounded by $n^2/4$ over $\mathcal{M}$. ∎

ETNA
Kent State University and
Johann Radon Institute (RICAM)

The result of Lemma 3.2 predicts that the norm of G_n grows, in general, quadratically. This prediction is confirmed numerically for parameter settings of practical relevance.

3.3. Moment-based option pricing for Heston models. The Heston model is another special case of a polynomial diffusion model characterized by the SDE

$$\begin{aligned} dY_t &= (r - V_t/2)d_t + \rho\sqrt{V_t}dW_{1t} + \sqrt{V_t}\sqrt{1-\rho^2}dW_{2t}, \\ dV_t &= \kappa(\theta - V_t)dt + \sigma\sqrt{V_t}dW_{1t}, \end{aligned}$$

with model parameters satisfying the conditions $\kappa \geq 0$, $\theta \geq 0$, $\sigma > 0$, $r \geq 0$, $\rho \in [-1, 1]$. As before, the asset price is modeled via $S_t := e^{Y_t}$, while V_t represents the squared stochastic volatility.

Lasserre et al. [10] developed a general option pricing technique based on moments and semidefinite programming (SDP). In the following we briefly explain the main steps and in which context an incremental computation of moments is needed. In doing so, we restrict ourselves to the specific case of the Heston model and European call options.

Consider the payoff function $f(y) := (e^y - e^K)^+$ for a certain log strike value K. Let $\nu(dy)$ be the Y_τ-marginal distribution of the joint distribution of the random variable (Y_τ, V_τ). Define the restricted measures ν_1 and ν_2 as $\nu_1 = \nu|_{(-\infty,K]}$ and $\nu_2 = \nu|_{[K,\infty)}$. By approximating the exponential in the payoff function with a Taylor series truncated after n terms, the option price (3.1) can be written as a certain linear function L in the moments of ν_1 and ν_2, i.e.,

$$\mathbb{E}[f(Y_\tau)] = L(n, \nu_1^0, \cdots, \nu_1^n, \nu_2^0, \cdots, \nu_2^n),$$

where ν_i^m represents the mth moment of the ith measure.

A lower / upper bound of the option price can then be computed by solving the optimization problems

$$SDP_n := \begin{cases} \min / \max & L(n, \nu_1^0, \cdots, \nu_1^n, \nu_2^0, \cdots, \nu_2^n) \\ \text{subject to} & \nu_1^j + \nu_2^j = \nu^j, \quad j = 0, \cdots, n \\ & \nu_1 \text{ is a Borel measure on } (-\infty, K], \\ & \nu_2 \text{ is a Borel measure on } [K, \infty). \end{cases} \tag{3.10}$$

Two SDPs arise when writing the last two conditions in (3.10) via moment and localizing matrices corresponding to the so-called truncated Stieltjes moment problem.

Formula (3.6) is used in this setting to compute the moments ν^j. Increasing the relaxation order n iteratively allows us to find sharper bounds (this is trivial because increasing n adds more constraints). One stops as soon as the bounds are sufficiently close. Algorithm 8 summarizes the resulting pricing algorithm.

The following lemma extends the result of Lemma 3.2 to the Heston model.

LEMMA 3.3. *Let G_n be the matrix representation of the operator $\mathcal{G}$ introduced above with respect to the basis (3.8) of $\mathrm{Pol}_n(\mathbb{R}^2)$. Then the matrix 1-norm of G_n is bounded by*

$$n(r + \kappa + \kappa\theta - \frac{\sigma^2}{2}) + \frac{1}{2}n^2(1 + |\rho|\frac{\sigma}{2} + \sigma^2).$$

Proof. Similar to the proof of Lemma 3.2. ∎

4. Numerical experiments. We have implemented the algorithms described in this paper in MATLAB and compare them with Higham's scaling and squaring method from [9], which typically employs a diagonal Padé approximation of degree 13 and is referred to as

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Algorithm 8 Option pricing for European options based on SDP and moments relaxation.

Input: Model and payoff parameters, tolerance ϵ
Output: Approximate option price

1: $n = 1$, gap $= 1$
2: **while** gap $> \epsilon$ **do**
3: Compute $\exp(\tau G_n)$ using Algorithm 4
4: Compute moments of order n using (3.6)
5: Solve corresponding SDP_n to get $LowerBound$ and $UpperBound$
6: gap $= |UpperBound - LowerBound|$
7: $n = n + 1$
8: **end while**

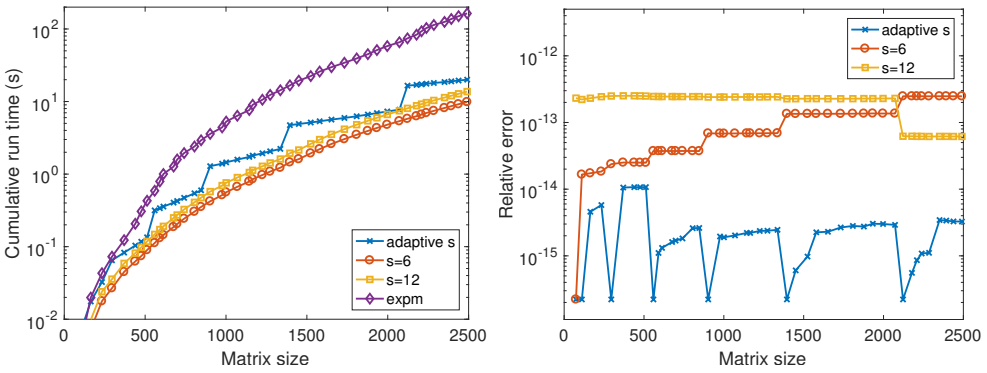

FIG. 4.1. *Comparison of* `incexpm` *and* `expm` *for a random block triangular matrix.* Left: *Cumulative run time for computing the leading portions.* Right: *Relative error of* `incexpm` *with respect to* `expm`.

"`expm`" in the following. The implementations of our algorithms for block triangular matrices, Algorithm 5 (fixed scaling parameter) and Algorithm 6 (adaptive scaling parameter), are based on the same scaling and squaring design and are referred to as "`incexpm`' in the following. All experiments were run on a standard laptop (Intel Core i5, 2 cores, 256kB/4MB L2/L3 cache) using a single computational thread.

4.1. Random block triangular matrices. We first assess run time and accuracy on a randomly generated block upper triangular matrix $G_n \in \mathbb{R}^{2491\times 2491}$. There are 46 diagonal blocks of size varying between 20 and 80. The matrix is generated to have a spectrum contained in the interval $[-80, -0.5]$, and a well conditioned eigenbasis X ($\kappa_2(X) \approx 100$).

Figure 4.1 (left) shows the wall clock time for the incremental computation of all the leading exponentials. Specifically each data point shows the time vs. $d_l = b_0 + \cdots + b_l$ needed for computing the $l+1$ matrix exponentials $\exp(G_0), \exp(G_1), \ldots, \exp(G_l)$, given $0 \le l \le n$, when using

- `expm` (by simply applying it to each matrix separately),
- `incexpm` with the adaptive scaling strategy from Algorithm 6,
- `incexpm` with fixed scaling power 6 (scaling used by `expm` for G_0),
- `incexpm` with fixed scaling power 12 (scaling used by `expm` for G_n).

As expected, `incexpm` is much faster than naively applying `expm` to each matrix separately; the total times for $l = n$ are also displayed in Table 4.1. For reference we remark that the run time of MATLAB's `expm`, which applies only the final matrix G_n, is 13.65s, which is very close to the run time of `incexpm` with scaling parameter set to 12 (see Section 2.3 for

ETNA
Kent State University and
Johann Radon Institute (RICAM)

TABLE 4.1
Run time and relative error attained by `expm` *and* `incexpm` *on a random block triangular matrix of size* 2491.

Algorithm	Time (s)	Rel. error
`expm`	163.60	
`incexpm` (adaptive)	20.01	3.27e-15
`incexpm` ($s = 6$)	9.85	2.48e-13
`incexpm` ($s = 12$)	13.70	6.17e-14

a discussion of the asymptotic complexity). Indeed, a closer look at the runtime profile of `incexpm` reveals that the computational overhead induced by the more complicated data structures is largely compensated in the squaring phase by taking advantage of the block triangular matrix structure, from which MATLAB's `expm` does not profit automatically. It is also interesting to note that the run time of the adaptive scaling strategy is roughly only twice the run time for running the algorithm with a fixed scaling parameter 6, despite its worse asymptotic complexity.

The accuracy of the approximations obtained by `incexpm` is shown on the right in Figure 4.1. We assume `expm` as a reference, and measure the relative distance between these two approximations, i.e.,

$$\frac{\|\texttt{expm}(G_l) - \texttt{incexpm}(G_l)\|_F}{\|\texttt{expm}(G_l)\|_F},$$

at each iteration l (quantities smaller than the machine precision are set to u in Figure 4.1 for plotting purpose). One notes that the approximations of the adaptive strategy remain close to `expm` throughout the sequence of computations. An observed drop of the error down to u for this strategy corresponds to a restart in Algorithm 6; the approximation at this step is *exactly* the same as the one of `expm`. Even for the fixed scaling parameters 6 and 12, the obtained approximations are quite accurate.

4.2. Application to option pricing. We now show results for computing option prices using Algorithm 7 for the set of parameters

$$v_0 = 0.04, \quad x_0 = 0, \quad \sigma_w = 0.5, \quad \mu_w = 0, \quad \kappa = 0.5, \quad \theta = 0.04, \quad \sigma = 0.15,$$
$$\rho = -0.5, \quad v_{\min} = 0.01, \quad v_{\max} = 1, \quad r = 0, \quad \tau = 1/4, \quad k = \log(1.1).$$

We use the tolerance $\epsilon = 10^{-3}$ for stopping Algorithm 7.

We explore the use of different algorithms for the computation of the matrix exponentials in line 5 of Algorithm 7: `incexpm` with adaptive scaling, `incexpm` with fixed scaling parameter $s = 7$ (corresponding to the upper bound from Lemma 3.2 for $n = 60$), and `expm`. Similar to Figure 4.1, the observed cumulative run times and errors are shown in Figure 4.2. Again, `incexpm` is observed to be significantly faster than `expm` (except for small matrix sizes) while delivering the same level of accuracy. Both `incexpm` run times are also close to the run time of MATLAB's `expm` applied only to the final matrix τG_n (4.64s).

Table 4.2 displays the impact of the different algorithm on the overall Algorithm 7 in terms of execution time and accuracy. Concerning accuracy, we computed the relative error with respect to a reference option price computed by considering a truncation order $n = 100$. It can be observed that there is no difference in accuracy for the three algorithms.

REMARK 4.1. The block triangular matrices G_n arising from the generator in the Jacobi model actually exhibit additional structure. They are quite sparse and the diagonal blocks are in fact permuted triangular matrices (this does not hold for polynomial diffusion models in

ETNA
Kent State University and
Johann Radon Institute (RICAM)

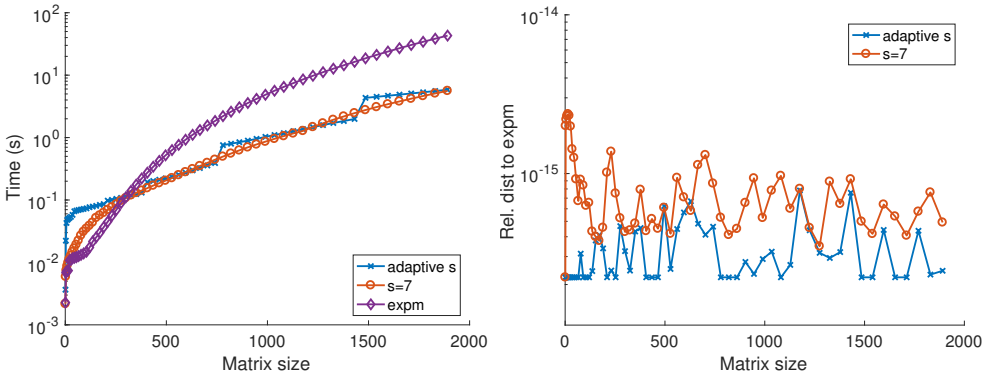

FIG. 4.2. *Comparison of* `incexpm` *and* `expm` *for the block upper triangular matrices arising in the context of the Jacobi model in Algorithm 7.* Left: *Cumulative run time for computing the leading portions.* Right: *Relative error of* `incexpm` *with respect to* `expm`.

TABLE 4.2
Total run time and option price errors for the Jacobi model for $n = 61$.

Algorithm	Time (s)	Rel. price error
`expm`	42.97	1.840e-03
`incexpm` (adaptive)	5.84	1.840e-03
`incexpm` ($s = 7$)	5.60	1.840e-03

general, though). For example, for $n = 2$ the matrix G_2 in the Jacobi model is explicitly given by

$$G_2 = \left[\begin{array}{ccc|ccc} 0 & r & \kappa\theta & 0 & -\frac{\rho\sigma v_{\max} v_{\min}}{S} & -\frac{\sigma^2 v_{\max} v_{\min}}{S} \\ & 0 & 0 & 2r & \kappa\theta & 0 \\ & -\frac{1}{2} & -\kappa & 1 & r + \frac{\rho\sigma(v_{\max}+v_{\min})}{S} & 2\kappa\theta + \frac{\sigma^2(v_{\max}+v_{\min})}{S} \\ \hline & & & 0 & 0 & 0 \\ & & & -1 & -\kappa & 0 \\ & & & 0 & -\frac{1}{2} - \frac{\rho\sigma}{S} & -2\kappa - \frac{\sigma^2}{S} \end{array}\right],$$

for $S := (\sqrt{v_{\max}} - \sqrt{v_{\min}})^2$.

While the particular structure of the diagonal blocks is taken into account automatically by `expm` and `incexpm` when computing the LU decompositions of the diagonal blocks, it is not so easy to benefit from the sparsity. Starting from sparse matrix arithmetic, the matrix quickly becomes denser during the evaluation of the initial rational approximation and in particular during the squaring phase. In all our numerical experiments we used a dense matrix representation throughout.

We repeated the experiments above for the Heston instead of the Jacobi model, that is, we investigated the impact of using our algorithms for computing the matrix exponentials in Algorithm 8. We found that the results for computing the matrix exponentials themselves look very similar to those for the Jacobi model (Figure 4.2) both in terms of run time and accuracy, so we refrain from giving further details here. There is, however, a notable difference. The evaluation of the stopping criterion requires the solution of two SDPs, which quickly becomes a computational challenge, eventually completely dominating the time needed for the computation of the matrix exponentials.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

5. Summary and future work. We have presented techniques for scaling and squaring algorithms that allow for the incremental computation of block triangular matrix exponentials. We combined these techniques with an adaptive scaling strategy that allows for both fast and accurate computation of each matrix exponential in this sequence (Algorithm 6). For our application in polynomial diffusion models, the run time can be further reduced by using fixed scaling parameter determined through the estimation techniques in Lemmas 3.2 and 3.3.

We observed in our numerical experiments that accurate approximations to these matrix exponentials can be obtained even for quite small, fixed scaling parameters. For the case of two-by-two block triangular matrices, the results of Dieci and Papini [3, 4] support this finding, but an extension of these results to cover a more general setting would be appreciable.

REFERENCES

[1] D. ACKERER, D. FILIPOVIĆ, AND S. PULIDO, *The Jacobi stochastic volatility model*, Swiss Finance Institute Research Paper No. 16-35, Swiss Finance Institute, Geneva, 2016.
[2] D. A. BINI, S. DENDIEVEL, G. LATOUCHE, AND B. MEINI, *Computing the exponential of large block-triangular block-Toeplitz matrices encountered in fluid queues*, Linear Algebra Appl., 502 (2016), pp. 387–419.
[3] L. DIECI AND A. PAPINI, *Padé approximation for the exponential of a block triangular matrix*, Linear Algebra Appl., 308 (2000), pp. 183–202.
[4] ———, *Conditioning of the exponential of a block triangular matrix*, Numer. Algorithms, 28 (2001), pp. 137–150.
[5] R. J. ELLIOTT AND P. E. KOPP, *Mathematics of Financial Markets*, 2nd ed., Springer, New York, 2005.
[6] D. FILIPOVIĆ AND M. LARSSON, *Polynomial diffusions and applications in finance*, Finance Stoch., 20 (2016), pp. 931–972.
[7] S. GÜTTEL AND Y. NAKATSUKASA, *Scaled and squared subdiagonal Padé approximation for the matrix exponential*, SIAM J. Matrix Anal. Appl., 37 (2016), pp. 145–170.
[8] N. J. HIGHAM, *Functions of Matrices*, SIAM, Philadelphia, 2008.
[9] ———, *The scaling and squaring method for the matrix exponential revisited*, SIAM Rev., 51 (2009), pp. 747–764.
[10] J. B. LASSERRE, T. PRIETO-RUMEAU, AND M. ZERVOS, *Pricing a class of exotic options via moments and SDP relaxations*, Math. Finance, 16 (2006), pp. 469–494.
[11] C. MOLER AND C. VAN LOAN, *Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later*, SIAM Rev., 45 (2003), pp. 3–49.
[12] B. ØKSENDAL, *Stochastic Differential Equations*, 6th ed., Springer, Berlin, 2003.
[13] B. N. PARLETT, *A recurrence among the elements of functions of triangular matrices*, Linear Algebra and Appl., 14 (1976), pp. 117–121.