Preconditioning the Matrix Exponential Operator with Applications *

Paul Castillo and Yousef Saad

Department of Computer Science and Engineering,
University of Minnesota,
Minneapolis, MN 55455

November 24, 1997

Abstract

The idea of preconditioning is usually associated with solution techniques for solving linear systems or eigenvalue problems. It refers to a general method by which the original system is transformed into one which admits the same solution but which is easier to solve. Following this principle we consider in this paper techniques for preconditioning the matrix exponential operator, $e^A y_0$, using different approximations of the matrix A. These techniques are based on using generalized Runge Kutta type methods. Preconditioners based on the sparsity structure of the matrix, such as diagonal, block diagonal, and least-squares tensor sum approximations are presented. Numerical experiments are reported to compare the quality of the schemes introduced.

Keywords: exponential operator, preconditioner, generalized Runge Kutta methods

1 Introduction

In this paper we are concerned with the issue of deriving efficient techniques for performing the matrix exponential operation:

$$w = e^A y_0 \ . \tag{1}$$

Here, the vector y_0 and the matrix A are given and we seek to compute w. The matrix A is large and sparse. It is typical to assume that A is semi-definite negative. The operation (1) is at the heart of many important algorithms in numerical analysis. Its single most important application is probably in the design of solution techniques for systems of Ordinary Differential Equations or time-dependent Partial Differential Equations.

One way to perform the above operation is to explicitly compute the exponential of A and apply the result to the vector A. This would be comparable to computing $A^{-1}b$ by first computing the inverse of A and then applying the result to the right-hand side b, and it would be just as inefficient in general. Extending this parallel with the problem of solving linear systems, it is clear that an attractive algorithm is one which would require only a small number matrix-vector products with the matrix A to compute an approximation to $e^A y_0$. Krylov subspace

*This work was supported by NSF under grant CCR-9618827 and by the Minnesota Supercomputer Institute.

algorithms of this type have been considered in recent years, see [4, 5, 7, 8, 9, 14, 16]. These algorithms are based on computing $exp(A)y_0$ in the form of a polynomial of A times the vector y_0, in other words the approximation is of the form,

$$e^A y_0 \approx p(A)y_0$$

where p is some polynomial of low degree. This is comparable to approximating $A^{-1}y_0$ by $p(A)y_0$ as is done in Krylov subspace algorithms such as the Conjugate Gradient algorithm.

The next natural extension which arises from our parallel with linear systems is the use of preconditioning techniques. The main idea used in preconditioning linear systems is the following:

> *A 'preconditioning' matrix M close in some sense to A is found for which the 'preconditioning' operation $M^{-1}y_0$ is easy to evaluate for an arbitrary vector y_0. Then the desired result $A^{-1}y_0$ is computed by using several matrix-vector products and preconditioning operations.*

We may wish to use a similar principle for computing the left-hand side w in (1): Given A, find M for which $exp(M)y_0$ is easy to compute and then find algorithms which compute $exp(A)y_0$ by utilizing preconditioning operations $exp(M)x$ and matrix-vector products Ax. However, here the issue is not as simple because computing $exp(A)v$ from $exp(M)y_0$ is not an easy operation.

In order to see how this can be done, it is most natural to consider the computation of the vector $e^{-tA}y_0$, where t is a positive parameter, referring to time. Here it is assumed that the matrix A is definite positive. The desired vector $w = exp(-At)y_0$ is the solution at time t of the system of differential equations:

$$\begin{cases} y' &= -Ay, \quad \text{in } [0,t] \\ y(0) &= y_0 \end{cases} \tag{2}$$

If M is a preconditioner to A, then the vector $exp(-M)v$ is solution to the system,

$$\begin{cases} y' &= -My, \quad \text{in } [0,t] \\ y(0) &= y_0 \end{cases} \tag{3}$$

Proceeding similarly to linear systems, we can see that the correction equation, i.e., the equations which allow us to obtain the desired result from its approximation is the system of equations of the form $y' = -My + (M - A)y$ with the same initial condition $y(0) = y_0$. Letting $r(t) = (M - A)y(t)$ the above system is of the form

$$\begin{cases} y' &= -My + r(t), \quad \text{in } [0,t] \\ y(0) &= y_0 \end{cases} \tag{4}$$

Now, standard integration methods can be applied. In order to exploit preconditioning operations we will be particularly interested in methods which can take advantage of operations of the form $exp(-M)x$ or of the form $\phi(-M)x$ with $\phi(z) = (exp(z) - 1)/z$, see [2].

2 Algorithms

Integration schemes based on an arbitrary approximation of matrix A will now be presented. The 'preconditioning' matrix B will denote any approximation of A, and

$$E = A - B \tag{5}$$

the error of the approximation. We consider two approaches. The first approach consists of writing the ODE system (2) as a new linear ODE system and solving this new system by using simple quadrature rules. The second approach consists of transforming the linear system into a new one via the Lawson transformation [10], and solving the system by using generalized Runge Kutta methods. The underlying assumption is that $e^B v$, can be computed easily for an arbitrary vector v.

2.1 A Predictor-Corrector approach

The first approach consists of writing system (2) as

$$\begin{cases} y' + By &= -Ey, \quad \text{in } [0, t] \\ y(0) &= y_0 \end{cases} \tag{6}$$

The solution of the differential equation at time t can be written in the following form

$$y(t) = e^{-tB}\left(y_0 - \int_0^t e^{(s-t)B} E y(s) ds\right).$$

The solution at time $t + h$ can be expressed in terms of the solution at time t as follows,

$$\begin{aligned} y(t+h) &= e^{-(t+h)B}\left(y_0 - \int_0^{t+h} e^{(s-(t+h))B} E y(s) ds\right) \\ &= e^{-hB} y(t) - \int_t^{t+h} e^{(s-(t+h))B} E y(s) ds \\ &= e^{-hB} y(t) - \int_0^h e^{(s-h)B} E y(t+s) ds \end{aligned}$$

In order to avoid the computation of the exponential term $e^{-hB}y(t)$, of the previous expression, at each integration step, we can proceed as in [4] to further simplify the last equation by using the identity

$$e^{-hB} = I - B\int_0^h e^{(s-h)B} ds$$

Hence, the following general expression is obtained

$$y(t+h) = y(t) - \int_0^h e^{(s-h)B} v(s) ds$$

where

$$v(s) = Ey(t+s) + By(t) = E\left(y(t+s) - y(t)\right) + Ay(t), \ s \in [0\ h] \tag{7}$$

Approximating $y(t+h)$ has been reduced to the problem of approximating numerically an integral, which can be accomplished by using a quadrature rule of the form

$$\int_0^h e^{(s-h)B} v(s) ds \;=\; h \sum \alpha_j e^{(s_j - h)B} v(s_j)$$

where the coefficients α_j are the weights at the quadrature nodes s_j of the interval $[0, h]$.

We can derive a predictor-corrector type scheme by using the midpoint rule to approximate the integral. The scheme consists of two steps. First an approximation of $y(t+h/2)$ is computed by assuming that $y(t)$ is constant in the interval $[t, t+h]$. Hence, from (7) the value of v at the mid-point, i.e., $v(\frac{1}{2}h)$ is approximated by $Ay(t)$. Second, we use the resulting approximation of $y(t+h/2)$ to compute y at time $t+h$. Again, we assume that $v(s)$ is constant in the interval $[0, h]$ and equal to the new value of $v(\frac{1}{2}h) = Ey(t+\frac{h}{2}) + By(t)$.

Define the operator

$$\phi_h(A) = \int_0^h e^{(s-h)A} ds \tag{8}$$

which has been used in Krylov subspace methods to approximate the exponential operator, [5, 7, 8, 16]. Note that

$$\phi_h(z) = \frac{1 - e^{-hz}}{z}.$$

Then, denoting by $y_{k+\alpha}$ the approximation of $y(t+\alpha h)$, for $\alpha = 0, \frac{1}{2}, 1$ the algorithm just described can be summarized by the following equations

$$y_{k+1/2} \;=\; y_k - \phi_{\frac{h}{2}}(B) A y_k \tag{9}$$

$$y_{k+1} \;=\; y_k - \phi_h(B) \left[E y_{k+1/2} + B y_k \right] \tag{10}$$

$$\;=\; y_k - \phi_h(B) \left[A y_k + E(y_{k+1/2} - y_k) \right] \tag{11}$$

These equations define the first algorithm called PC-EXPO, to compute $e^A y_o$.

ALGORITHM **2.1 Algorithm PC-EXPO**$(A, B, y_0, nsteps)$

1. *For* $k = 1 : nsteps$ *Do:*
2. $\quad v = Ay_k$
3. $\quad f_{h/2} = \phi_{\frac{h}{2}}(B) v$
4. $\quad v_{1/2} = v - E f_{h/2}$
5. $\quad y_{k+1} = y_k - \phi_h(B) v_{1/2}$
6. *EndDo*

2.1.1 Order of PC-EXPO

To determine the order of the algorithm PC-EXPO we start by merging equations (9) and (10) into one equation and making the assumption that $y_k = y(t)$ is exact. This yields,

$$y_{k+1} = y(t) - \phi_h(B) \left[I - E \phi_{\frac{h}{2}}(B) \right] A y(t)$$

Next, we note that the exact solution $y(t+h)$ takes the form:

$$\begin{aligned} y(t+h) &= e^{-hA}y(t) = [I - A\phi_h(A)]y(t) \\ &= y(t) - \phi_h(A)Ay(t) \end{aligned}$$

As a result,

$$y(t+h) - y_{k+1} = \left[\phi_h(B)\left(I - E\phi_{\frac{h}{2}}(B)\right) - \phi_h(A)\right] Ay(t)$$

The order of the term in between the brackets can now be estimated by using Taylor series expansions. The error equation involves various terms of the form $\phi_h(X)$ for different values of X. The expansion of $\phi_h(X)$ for an arbitrary X is

$$\phi_h(X) = hI - \frac{h^2}{2!}X + \frac{h^3}{3!}X^2 - \dots$$

The error matrix $\phi_h(B)\left(I - E\phi_{\frac{h}{2}}(B)\right) - \phi_h(A)$ can be grouped in two pieces as:

$$(\phi_h(B) - \phi_h(A)) - \phi_h(B)E\phi_{\frac{h}{2}}(B)$$

For the first term we have

$$\phi_h(B) - \phi_h(A) = \frac{h^2}{2}E - \frac{h^3}{6}((B+E)^2 - B^2) + \cdots$$

For the second part we can write

$$\begin{aligned} \phi_h(B)E\phi_{\frac{h}{2}}(B) &= \left(hI - \frac{h^2}{2}B + \frac{h^3}{6}B^2 - \dots\right) E \left(\frac{h}{2}I - \frac{h^2}{8}B + \frac{h^3}{48}B^2 - \dots\right) \\ &= \frac{h^2}{2}E - \frac{h^3}{8}EB - \frac{h^3}{4}BE + \cdots \end{aligned}$$

Finally the local error is of the form,

$$\begin{aligned} E_h(y(t)) &= h^3\left[\frac{1}{8}EB + \frac{1}{4}BE - \frac{1}{6}((B+E)^2 - B^2)\right] Ay(t) + O(h^4) \\ &= \frac{h^3}{24}\left[2BE - EB - 4E^2\right] Ay + O(h^4) \end{aligned}$$

The method is therefore of second order as expected.

Nothing was said so far about the expression of the error with respect to the type of approximation used for A, i.e., with respect to the term E. An interesting question that is left open at this point is whether or not it is possible to find predictor-corrector formulas which have the same order, but whose resulting error is quadratic with respect to E. In the above error formula, the local error is linear in E. It may be possible that a certain combination of PC integration yields an error in which the terms EB and EB are all cancelled out, resulting in an error of order $O(h^3\|E^2\|) + O(h^4\|E\|)$.

2.2 Runge Kutta methods

It is possible to develop generalized Runge Kutta - type schemes by applying Runge Kutta methods to a modified system as was first suggested by Lawson in a series of paper [3, 10, 11, 12, 13]. Consider the change of variable,

$$z(t) = e^{(t-t_0)B} y(t), \tag{12}$$

in the interval $[t_0, t_1]$. The essential idea of 'generalized Runge-Kutta methods' as given in [10] consists of performing a local transformation of the problem in order to reduce the magnitude of the eigenvalues. The ordinary differential equation $y' = -Ay$, becomes

$$\begin{cases} z'(t) &= g(t,z) \\ z(t_0) &= y_0 \end{cases} \tag{13}$$

where $g(t,z) = e^{(t-t_0)B}(B-A)e^{-(t-t_0)B}z(t)$.

The rationale for the local transformation (12) is that the matrices $e^{(t-t_0)B}(B-A)e^{-(t-t_0)B}$ and $B-A$, being similar, have the same spectrum. If B is a good approximation of A, the eigenvalues of $B-A$, hence of $e^{(t-t_0)B}(B-A)e^{-(t-t_0)B}$, will be small.

2.2.1 Runge Kutta 2

The simplest Runge Kutta type algorithm can be derived by using the implicit trapezoidal quadrature rule, at each integration step. This scheme is of order 2. One step of the method, in the variable z, reads

$$\begin{cases} Z_1 &= z_0 \\ Z_2 &= z_0 + hg(t_0, Z_1) \\ z_1 &= z_0 + \frac{h}{2}\left(g(t_0, Z_1) + g(t_0+h, Z_2)\right) \end{cases}$$

Using the change of variable $z(t) = e^{(t-t_0)B}y(t)$, and noticing that z_1 is an approximation of $z(t_0+h)$ we obtain the following Runge Kutta scheme in the original variable y

$$\begin{cases} Y_1 &= y_0 \\ Y_2 &= e^{-hB}y_0 + he^{-hB}(EY_1) \\ y_1 &= e^{-hB}y_0 + \frac{h}{2}\left(e^{-hB}(EY_1) + EY_2\right) \end{cases}$$

The corresponding algorithm is as follows

ALGORITHM **2.2 RK2-EXPO step**

1. $Y_1 = y_0$
2. $w = e^{-hB}Y_1$
3. $q = e^{-hB}EY_1$
4. $Y_2 = w + hq$
5. $y_1 = w + \frac{h}{2}(q + EY_2)$

Only two matrix exponential evaluations per time step are required by the algorithm.

2.2.2 Runge Kutta 3

The Heun algorithm is a well known Runge Kutta type method of order 3, [6]. It is derived from the two point Radau quadrature formula,

$$\int_{t_0}^{t_1} f(t)dt \;=\; \frac{h}{4}\left(f(t_0) + 3f\left(t_0 + \frac{2}{3}h\right)\right)$$

One step of the Heun scheme in the z variable is as follows, [6]:

$$\begin{cases} Z_1 &= z_0 \\ Z_2 &= z_0 + \frac{1}{3}hg(t_0, Z_1) \\ Z_3 &= z_0 + \frac{2}{3}h\left(g(t_0 + \frac{1}{3}h, Z_2)\right) \\ z_1 &= z_0 + \frac{1}{4}h\left(g(t_0, Z_1) + 3g(t_0 + \frac{2}{3}h, Z_3)\right) \end{cases}$$

The corresponding algorithm translated into the y variable reads

$$\begin{cases} Y_1 &= y_0 \\ Y_2 &= e^{-\frac{1}{3}hB}Y_1 + \frac{1}{3}he^{-\frac{1}{3}hB}EY_1 \\ Y_3 &= e^{-\frac{2}{3}hB}Y_1 + \frac{2}{3}he^{-\frac{1}{3}hB}EY_2 \\ y_1 &= e^{-hB}y_0 + \frac{1}{4}h\left(e^{-hB}EY_1 + 3e^{-\frac{1}{3}hB}EY_3\right) \end{cases}$$

An economical procedure associated with the above sequence would be as follows:

$$\begin{aligned} Y_1 &= y_0 \\ q &= e^{-\frac{h}{3}B}Y_1 \\ s &= e^{-\frac{h}{3}B}EY_1 \\ Y_2 &= q + \frac{h}{3}s \\ Y_3 &= e^{-\frac{h}{3}B}(q + 2\frac{h}{3}EY_2) \\ y_1 &= e^{-\frac{2}{3}hB}(q + \frac{h}{4}s) + \frac{3h}{4}e^{-\frac{h}{3}B}EY_3 \end{aligned}$$

Five exponential products are required in the above procedure. Note that if we were to compute the matrix $C = exp(-\frac{h}{3}B)$ explicitly as would be done if B is dense, then only this matrix need be computed since the products $exp(-\frac{2h}{3}B)$ and $exp(-hB)$ can be achieved with 2 or 3 successive products with the matrix C. This and other implementation issues will be discussed in section 4.

It is possible to reduce the number of matrix exponentials from 5 to 3. For this we can derive a variant of a Runge Kutta type method which is also of order 3 by modifying Heun's scheme. In Heun's scheme, Z_3 approximates the value of the function at the quadrature node, $t_0 + \frac{2}{3}h$, by integrating g on the interval $[0, \frac{2}{3}h]$ using the midpoint quadrature rule. The value of the function z, Z_2, at the midpoint is obtained by using an explicit Euler step. The variant is based on the same two point Radau quadrature rule. However, we can approximate $g(t + \frac{2}{3}h, Z)$ by using the trapezoidal rule instead. For the values at the quadrature nodes, we use Z_1 and Z_2

which are approximations of the function at node $t_0 + \frac{2}{3}h$ using an explicit Euler step. This gives the following method

$$\begin{cases} Z_1 &= z_0 \\ Z_2 &= z_0 + \frac{2h}{3} g(t_0, Z_1) \\ Z_3 &= z_0 + \frac{h}{3}\Big(g(t_0, Z_1) + g(t_0 + \frac{2}{3}h, Z_2)\Big) \\ z_1 &= z_0 + \frac{h}{4}\Big(g(t_0, Z_1) + 3g(t_0 + \frac{2}{3}h, Z_3)\Big) \end{cases}$$

It can easily bee seen that this Runge Kutta algorithm is of order three by showing that the coefficients of its Butcher tableau satisfy the order conditions of theorem 2.1 pp. 145, in [6] The Butcher tableau of a general three stage Runge Kutta method is of the form

c_1	a_{11}	a_{12}	a_{13}
c_2	a_{21}	a_{22}	a_{23}
c_3	a_{31}	a_{32}	a_{33}
	b_1	b_2	b_3

The corresponding Butcher tableau of the above modified Heun scheme is

0	0	0	0
2/3	2/3	0	0
2/3	1/3	1/3	0
	1/4	0	3/4

Simple calculations show that these coefficients satisfy

$$\sum_j b_j = 1 \qquad \sum_{j,k} b_j a_{jk} = \tfrac{1}{2}$$

$$\sum_{j,k,l} b_j a_{jk} a_{jl} = \tfrac{1}{3} \quad \sum_{j,k,l} b_j a_{jk} a_{kl} = \tfrac{1}{6}$$

which are the conditions required for the scheme to be of order three [6].

Translating the above algorithm into the y variable, we obtain the following scheme

$$\begin{cases} Y_1 &= y_0 \\ Y_2 &= e^{-\frac{2h}{3}B} y_0 + \frac{2h}{3} e^{-\frac{2h}{3}B} E Y_1 \\ Y_3 &= e^{-\frac{2h}{3}B} y_0 + \frac{h}{3}\Big(e^{-\frac{2h}{3}B} E Y_1 + E Y_2\Big) \\ y_1 &= e^{-hB} y_0 + \frac{h}{4}\Big(e^{-hB} E Y_1 + 3 e^{-\frac{h}{3}B} E Y_3\Big) \end{cases}$$

We can further minimize the number of matrix exponentials, by proceeding as follows

ALGORITHM **2.3 RK3-EXPO step**

1. $Y_1 = y_0$
2. $w = e^{-\frac{2h}{3}B} Y_1$
3. $q = e^{-\frac{2h}{3}B} E Y_1$
4. $Y_2 = w + \frac{2h}{3} q$
5. $Y_3 = w + \frac{h}{3}(q + E Y_2)$
6. $y_1 = e^{-\frac{h}{3}B}\Big(w + \frac{h}{4} q + \frac{3h}{4} E Y_3\Big)$

Therefore the scheme requires 3 matrix exponential evaluations per time step.

2.2.3 Runge Kutta 4

Similarly, a classical Runge Kutta method of order 4 can be adapted to derive a generalized Runge Kutta scheme. The scheme for the z variable is:

$$\left\{\begin{array}{ll} Z_1 & = z_0 \\ Z_2 & = z_0 + \frac{h}{2} g(t_0, Z_1) \\ Z_3 & = z_0 + \frac{h}{2} g(t_0 + \frac{1}{2}h, Z_2) \\ Z_4 & = z_0 + h g(t_0 + \frac{1}{2}h, Z_3) \\ z_1 & = z_0 + \frac{h}{6}\left(g(t_0, Z_1) + 2g(t_0 + \frac{1}{2}h, Z_2) + 2g(t_0 + \frac{1}{2}h, Z_3) + g(t_0 + h, Z_4)\right) \end{array}\right.$$

The method can be rewritten in terms of the original variable y,

$$\left\{\begin{array}{ll} Y_1 & = y_0 \\ Y_2 & = e^{-\frac{h}{2}B} y_0 + \frac{h}{2} e^{-\frac{h}{2}B} E Y_1 \\ Y_3 & = e^{-\frac{h}{2}B} y_0 + \frac{h}{2} E Y_2 \\ Y_4 & = e^{-hB} y_0 + h e^{-\frac{h}{2}B} E Y_3 \\ y_1 & = e^{-hB} y_0 + \frac{h}{6}\left(e^{-hB} E Y_1 + 2e^{-\frac{h}{2}B} E Y_2 + 2e^{-\frac{h}{2}B} E Y_3 + E Y_4\right) \end{array}\right.$$

The corresponding computational algorithm is as follows,

ALGORITHM **2.4 RK4-EXPO step**

1. $Y_1 = y_0$
2. $w = e^{-\frac{h}{2}B} Y_1$
3. $q = e^{-\frac{h}{2}B} E Y_1$
4. $Y_2 = w + \frac{h}{2} q$
5. $Y_3 = w + \frac{h}{2} E Y_2$
6. $Y_4 = e^{-\frac{h}{2}B}(w + h E Y_3)$
7. $y_1 = e^{-\frac{h}{2}B}\left(w + \frac{h}{6} q + \frac{h}{3} E(Y_2 + Y_3)\right) + \frac{h}{6} E Y_4$

Therefore the scheme requires 4 matrix exponential evaluations per time step.

It is clear that the order of each of the previous generalized Runge Kutta methods is the same as its corresponding classical Runge Kutta scheme, a direct consequence of the local error bound of a classical Runge Kutta method applied to the ordinary differential equation (13) and the local transformation (12). Let z_h be the approximation of the function $z(t)$ at time $t + h$, after applying one step of a Runge Kutta method of order p. The local error, $z(t + h) - z_h$ is of order $O(h^{p+1})$. From the Lawson transformation we obtain

$$y(t + h) - y_h \;=\; e^{-hB}\left(z(t + h) - z_h\right) \;=\; O(h^{p+1})$$

Therefore, the algorithms RK2-EXPO, RK3-EXPO and RK4-EXPO are of order 2, 3 and 4 respectively.

3 Approximations of A

Three different choices for the preconditioning matrix B are now considered. Implementations issues will be discussed in the next section. In some cases the matrix A arises from the discretization of elliptic and parabolic partial differential equations, on a rectangular domain. However, we also consider the general case when A is a general sparse irregularly structured matrix.

3.1 Diagonal and Block-Diagonal approximations

The simplest choice for the matrix B is to take it to be diagonal. For example, a natural choice is to let B be the main diagonal of A. The exponential operator, e^B, is again a diagonal matrix, and is composed by the exponential of the diagonal entries of B. The computational complexity of matrix $e^{\alpha hB}$ and the matrix-vector multiplication $e^{\alpha hB}y$ are both of order $O(n)$.

The approximation of A by a diagonal matrix is not very accurate, and the next simplest alternative is to take B to be a block diagonal matrix consisting of some diagonal blocks of A. In this case the matrix $e^{\alpha hB}$ is also a block diagonal matrix where each block in the diagonal is the exponential of a block.

3.2 Domain Decomposition techniques

An attractive idea used for solving linear systems in parallel is that additive Schwarz techniques in domain decomposition methods. In these methods, the preconditioning used can be viewed as a block diagonal of (a reordered version of) the original matrix but each block corresponds to the variables of a physical subdomain. The above block techniques can be generalized to define Schwarz preconditioners for the matrix-exponential operator. From the point of view of the implementation, the only difference with the block preconditioners just described is that the blocks can be general sparse matrices.

3.3 Tensor Product approximations

There are a number of applications where the matrix is naturally approximated by the tensor sum of two small matrices. The best known example is that corresponding to matrices arising from the discretization of elliptic and parabolic partial differential equations in two dimensions. In the separable case, A can be decomposed into a sum of two tensor products,

$$A = L_y \otimes I + I \otimes L_x$$

where L_x and L_y are discretizations of the original operator in the x and y directions respectively, We have recently proposed a preconditioner for linear systems whose matrix can be approximated by a sum of tensor products, [1]. This approximation is attractive mainly for two reasons. First, the tensor sum approximation captures the sparsity pattern of the discretization matrix, better than the block diagonal approximation. This approximation is more compact if we compare it with the block diagonal approximation since it requires the storage of two matrices of small order.

4 Implementation and cost considerations

The dominant cost in all the RK-like schemes presented in the earlier sections is in the successive computations of the vectors $e^{\alpha hB}y$ where α is a scalar.

4.1 Dense and dense block-diagonal case

It is interesting to start by considering the situation when B is a small dense matrices. In such situations efficient and stable methods for computing matrix exponentials can be found in the literature, see for example the classical paper of Moler and Van Loan, [15]. In this case, it is

important to note that since most of the approximations are expressed in terms of powers of $e^{\alpha hB}$ we can compute this matrix only once. For example in Algorithm 2.4, all such products are expressed in terms of $exp(-\frac{h}{2}B)$. This matrix can be computed and all four operations with it (lines 2,3,5,6) can be done as one standard matrix-vector product. Similarly in Algorithm 2.3, it is clear that only $exp(-\frac{h}{3}B)$ must be computed. Then the vectors of the form $exp(-\frac{h}{3}B)v$ can be computed as one matrix-vector product while vectors of the form $exp(-2\frac{h}{3}B)$ can be computed two matrix-vector products since $exp(-2\frac{h}{3}B)$ is the square of $exp(-\frac{h}{3}B)$.

The situation is similar when B is block diagonal and each block is a small dense block. It is then more economical to compute the exponentials of these blocks explicitly and proceed similarly to the dense case just explained. If each block is of size p, the cost of this approach is essentially dominated by the computations of the small exponential matrices, which is $O(p^3)$ each. Since there are N/p such blocks, the total cost per step is $O(Np^2)$. As can be seen this approach may become ineffective when p is large. Note that these issues are identical with the ones encountered for preconditioning linear systems by block diagonal preconditioners.

4.2 Use of Krylov methods

The approach outlined above for dense or block matrices cannot be exploited in the algorithm PC-EXPO or in some other instances. In PC-EXPO the computations of $\phi_{\alpha h}(B)y$, where $\phi_h(z)\frac{1-e^{-hz}}{z}$ depends on the current value of the vector y.

For the domain decomposition algorithms as well as other cases in which the blocks are large and sparse, the 3 or 4 matrix exponentials required in Algorithms 2.3 and 2.4 must be performed separately. However, it is sometimes possible to take advantage of the computation of one of the terms to facilitate the other ones. For example, if Krylov methods are used, it is possible to exploit block Krylov techniques to compute terms such as

$$\begin{aligned} w &= e^{-2\frac{h}{3}B}Y_1 \\ q &= e^{-2\frac{h}{3}B}EY_1 \end{aligned}$$

which arise in lines 2 and 3 of Algorithm 2.3 and the similar computations which arise in lines 2-3 and lines 5-6 of Algorithm 2.4. With such techniques, the computations of the exponentiations is reduced to two block-exponential calculations for both algorithms. Further, an interesting observation can be made. In [5], subspaces Krylov subspaces used for a certain vector were effectively used to compute matrix-exponentials with other vectors. Note that when h is small the Krylov subpaces needed for the computation of the terms

$$\begin{aligned} & e^{-\frac{h}{2}B}(w + hY_3) \\ & e^{-\frac{h}{2}B}\left(w + \frac{h}{3}(Y_2 + Y_3)\right) \end{aligned}$$

in Algorithm 2.4 will be very close to the Krylov subspace used for the computation of

$$e^{-\frac{h}{2}B}w$$

in earlier steps. A projection method with this subspace can therefore be quite effective.

4.3 The tensor product case

In the case when

$$\alpha hB = X \oplus Y \equiv X \otimes I + I \otimes Y$$

then $e^{\alpha hB}v$ can be computed as

$$e^{\alpha hB}v = \; e^{X\otimes I + I\otimes Y}v = e^{X\otimes I}\left(e^{I\otimes Y}v\right)$$

For this purpose we can store the exponentials of X and Y which are typically small, represented in dense form or in some other compact form. For example a factored form involving the eigenvalue decomposition of B, which can be easily deduced from the eigenvalue decomposition of the matrices X and Y, can be used. Thus, in the PDE case with constant coefficients, X and Y are tridiagonal matrices which have constant elements on the diagonals. In this particular situation, matrix products with $exp(X)$ and $exp(Y)$ can be performed with one-dimensional FFTs and only a diagonal matrix of eigenvalues need to be stored.

Consider the computation of the first term $w = e^{I\otimes Y}v$. The matrix involved in this product is of the form

$$e^{I\otimes Y} = \begin{pmatrix} e^Y & & & & & & \\ & e^Y & & & & & \\ & & e^Y & & & & \\ & & & \ddots & & & \\ & & & & \ddots & & \\ & & & & & e^Y & \\ & & & & & & e^Y \end{pmatrix} \tag{14}$$

As a result the same matrix exponential of Y is applied to different pieces of the vector v to obtain the corresponding pieces of the result w. Similarly, if $e^X = \{f_{ij}\}_{i,j=1,m}$ then

$$e^{X\otimes I} = \begin{pmatrix} f_{11}I & f_{12}I & \cdots & \cdots & f_{1m}I \\ f_{21}I & f_{22}I & \cdots & \cdots & f_{2m}I \\ f_{31}I & f_{32}I & \cdots & \cdots & f_{3m}I \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ f_{m1}I & f_{m2}I & \cdots & \cdots & f_{mm}I \end{pmatrix} \tag{15}$$

As can be easily seen a matrix product operation with this matrix is virtually identical with a product with the matrix (14). Indeed, if $z = e^{X\otimes I}w$ the components

$$z_1, z_{n+1}, z_{2n+1}, \ldots, z_{kn+1}, \ldots z_{(m-1)n+1}$$

are the result of applying e^X to the vector

$$w_1, w_{n+1}, w_{2n+1}, \ldots, w_{kn+1}, \ldots w_{(m-1)n+1}$$

In general the vectors

$$z_i, z_{n+i}, z_{2n+i}, \ldots, z_{kn+i}, \ldots z_{(m-1)n+i}$$

are the result of applying e^X to the vector

$$w_i, w_{n+i}, w_{2n+i}, \ldots, w_{kn+i}, \ldots w_{(m-1)n+i}$$

The result of this observation is that the same matrix exponential, namely e^X is now applied to a permuted version of the vector w and the result is a permuted subvector of z. In fact, if w and z are stored in $n \times m$ arrays column-wise, then to obtain the rows of the matrix representing z we must apply the same matrix exponential to the corresponding rows of the matrix representing w. This is a well-known strategy when tensor products are used.

4.4 FFT methods

In the previous section we have discussed implementation issues of the tensor sum preconditioner for the general case (no knowledge of the eigenvalue decomposition). In that case we have to compute the eigenvalue decomposition. In some special cases, such as preconditioning the matrix with a Laplacean operator on a regular grid, we can obtain faster implementations by using Fourier transforms, as in the fast Poisson solvers. The advantage with this type of preconditioner is that the eigenvalue decomposition is already known and we can use Fast Fourier Transforms to compute matrix-vector multiplications.

Let B be the discretization of the Laplacean operator, be of the form $B = X \otimes I_p + I_{N/p} \otimes Y$. In this case we know the eigenvalue decomposition of X and Y. The computation of the term $e^{hI \otimes Y} v$ can be done as follows, let $Y = QDQ^T$ the eigenvalue decomposition of Y.

ALGORITHM **4.1 Exponential using FFT**

1. *For k = 1,N/p Do*
2. $\quad \hat{v}_k = Q^T v_k$
3. $\quad \hat{v}_k = e^{hD} \hat{v}_k$
4. $\quad \hat{u}_k = Q \hat{v}_k$
5. *End For*

where $v_k = v(1 + p(k-1) : pk)$. Steps 2 and 4 can be computed by using Fast Fourier Transform techniques. This matrix-vector operation has a complexity of order $O(p \log(p))$, hence the total complexity of performing $e^{hI \otimes Y} v$ is of order $O(N \log(p))$, while the complexity of the method described for the general case is of order $O(Np^2)$.

5 Experiments

We have performed several tests to compare the quality of each preconditioner. Matrices comes from the discretization of a PDE using a centered finite difference scheme. We have discretized the unit square using a uniform, 34×34 grid. The order of the test matrices is 1024, using Dirichlet boundary conditions. The matrices were generated by SPARSKIT2 [17] and have been stored using the diagonal format. The diagonal preconditioner we have used consists of the main diagonal and the block diagonal consists of a tridiagonal matrix formed by the diagonals of offsets -1,0 and 1. We have integrated the differential system on the interval [0, 1], using several number of uniform steps,

$$10,\ 50,\ 100,\ 250,\ 500,\ 750,\ 1000,\ 1250,\ 1500,\ 2000,\ 4000,\ 8000,\ 16000.$$

The code was run until the norm of the difference between the computed solution at time $t = 1$ and the "true" solution was less than a tolerance TOL, usually TOL $= 10^{-9}$ or when the number of steps reached the maximum number of uniform steps, 16000.

Algorithm	Time (sec)	Error
PC-EXPO, k = 5	15.91	6.015840153596073D-08
PC-EXPO, k = 10	32.75	6.015840122709288D-08
PC-EXPO, k = 20	75.39	6.015840122709288D-08
RK2, diagonal	1.27	3.973377823609744D-04
RK2, Block diagonal	1.76	5.182567141728659D-05
RK2, tensor sum	2.85	7.857540791158436D-07

Table 1: Performance for step size 1/1000

5.1 A symmetric example

As a symmetric example we have consider a perturbed Heat equation problem

$$\frac{\partial u(x,y,t)}{\partial t} = \alpha \left(\frac{\partial}{\partial x} \left(a \frac{\partial u(x,y,t)}{\partial x} \right) + \frac{\partial}{\partial y} \left(b \frac{\partial u(x,y,t)}{\partial y} \right) \right)$$

where $a(x,y) = 1 + \epsilon_1(x,y)$ and $b(x,y) = 1 + \epsilon_2(x,y)$. The functions $\epsilon_1 = \gamma \cos(\omega(x^2+y^2))$ and $\epsilon_2 = \gamma \sin(\omega(x^2+y^2))$ represents perturbations, which do not exceed 30% of unity. In order to compare the accuracy of the solution given by our algorithms, we have used the routine DOPRI5 [6] to generate the "true" solution. The error norm is the norm of the difference between the computed solution and the solution given by the DOPRI5 routine.

In figures 2 to 4 we show the steps - error norm diagrams using a $\log_{10}$ scale, for the algorithms RK2-EXPO, RK3-EXPO and RK4-EXPO. As expected, the tensor sum preconditioner has the best performance, followed by the block diagonal preconditioner and the least accurate is the diagonal preconditioner.

In Figure 1 we show the performance of the algorithm PC-EXPO. We have included the plot of results with RK2-EXPO for reference. Both algorithms are preconditioned with the tensor sum approximation. This algorithm computes $e^{\alpha h B}$ using Krylov subspace approximations. Since the choice of the dimension of the Krylov subspace is a crucial aspect in terms of accuracy, we have used three different values for the dimension of the Krylov subspace 5,10,20. For larger step sizes a small dimension will degrade the accuracy of the scheme. This is an important aspect to consider if we want to achieve high accuracy, especially when using variable steps size methods.

Table 5.1 gives the time performance of the algorithms of order 2, PC-EXPO and RK2-EXPO, for a fixed step size, 1/1000. The PC-EXPO algorithm is more accurate but slower than the RK2-EXPO algorithm. These results were obtained on an SGI Challenge XL machine, using only one processor.

5.2 A nonsymmetric example

We consider the nonsymmetric model problem

$$\frac{\partial u(x,y,t)}{\partial t} = \alpha \left(\Delta u(x,y,t) + \beta \frac{\partial u(x,y,t)}{\partial x} + \gamma \frac{\partial u(x,y,t)}{\partial y} \right)$$

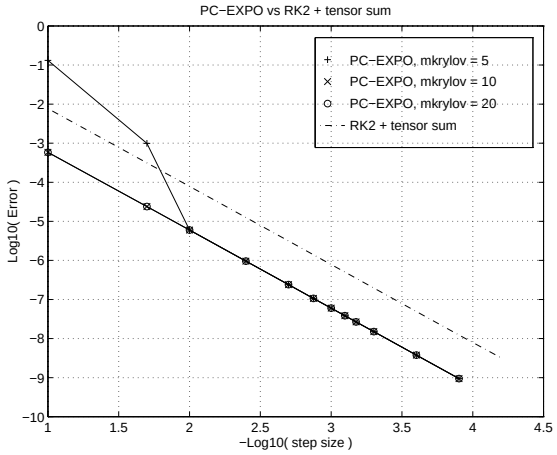

Figure 1: PC-EXPO

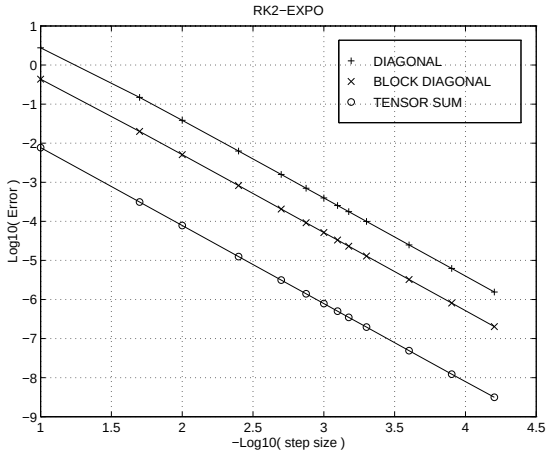

Figure 2: RK2-EXPO

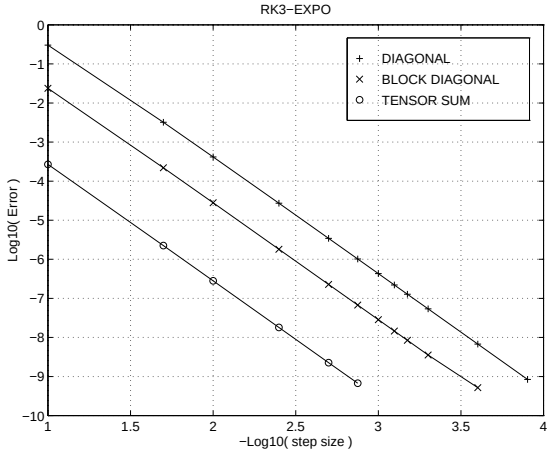

Figure 3: RK3-EXPO

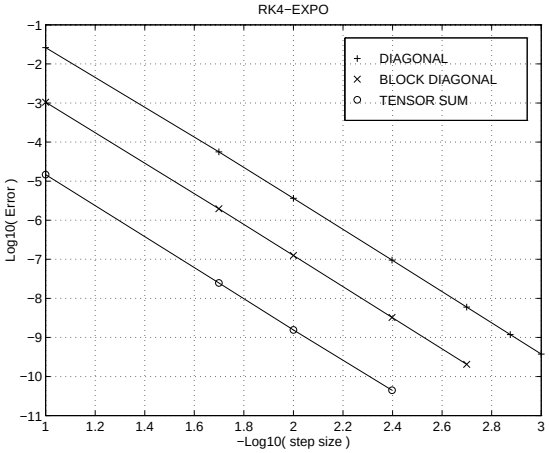

Figure 4: RK4-EXPO

in the the unit square, where the parameter α controls the degree of stiffness of the problem. The exact solution of this problem can be easily computed from the explicit knowledge of the eigenvalues and eigenvectors of the discretization matrix, A. In fact, the matrix A can be decomposed into a sum of tensor products of the form

$$A = L_y \otimes I + I \otimes L_x$$

where L_x and L_y are tridiagonal matrices. As a preconditioner we used the discretization of the Laplacean operator $\alpha \Delta u$ in tensor sum form. We present numerical experiments with three values of α, namely $\alpha = 0.001$, $\alpha = 0.01$ and $\alpha = 0.1$. This is meant to represent a transition from a nonstiff problem to a stiff problem. In Figures 5 to 13, we illustrate the quality of each preconditioner. For these experiments we took $\beta = 1.0$ and $\gamma = 0.0$. As the problem becomes more stiff the diagonal and block diagonal preconditioner become less efficient. However the tensor preconditioner is less affected.

Table 5.2 shows the time performance of the generalized Runge Kutta algorithms, for $\alpha = 0.001$. The column of time represents the time required by a given algorithm and its preconditioner to achieve certain precision (or at least close to the desired precision), in our case we have chosen $10^{-6}, 10^{-8}, 10^{-10}$ for RK2,RK3 and RK4, respectively. In all the cases the best times correspond to the tensor sum preconditioner. Better timing results can be obtained by using FFT techniques, in the case of a tensor preconditioner such as the Laplacean operator. In our experiments we did not use FFT transforms.

Algorithm	Preconditioner	Time (sec)	Error
RK2-EXPO	diagonal	20.37	1.5376D-06
	block diagonal	13.17	7.7080D-07
	tensor sum	0.29	8.5654D-07
RK3-EXPO	diagonal	3.92	5.2967D-08
	block diagonal	4.16	1.3604D-08
	tensor sum	0.28	1.0100D-08
RK4-EXPO	diagonal	2.99	3.6491D-10
	block diagonal	1.86	1.8122D-10
	tensor sum	0.30	4.5165D-11

Table 2: Time performance

6 Conclusion

Two different classes of techniques for preconditioning the exponential operator were presented in this paper. In the first approach the computation of the exponential operator has been reduced to the problem of approximating an integral which involves the exponential of a preconditioner for A. The integral is approximated by means of Krylov subspace approximations and of simple quadrature rules. These methods have the advantage of using matrix-vector operations as basic operation and not requiring any knowledge of the eigenvalue decomposition of the preconditioner. The second approach is based on Generalized Runge Kutta methods. This class of methods consists of applying the Lawson transformation using a preconditioner B of A, and solving the resulting system by explicit Runge Kutta methods. Operations of the form $e^{\alpha h B}v$ are computed by using matrix-vector multiplications involving the explicit knowledge of one matrix exponential, making these methods more efficient than the methods based on Krylov subspace approximations. We have numerically studied the performance of three types of preconditioners: diagonal, block diagonal, and tensor sum approximation. In our experiments tensor sum preconditioners were the most efficient and tended to be less sensitive to stiffness, because the related approximations are of better quality. In this paper we have only considered Generalized Runge Kutta methods with fixed step size, more efficient algorithms could be derived using variable step size.

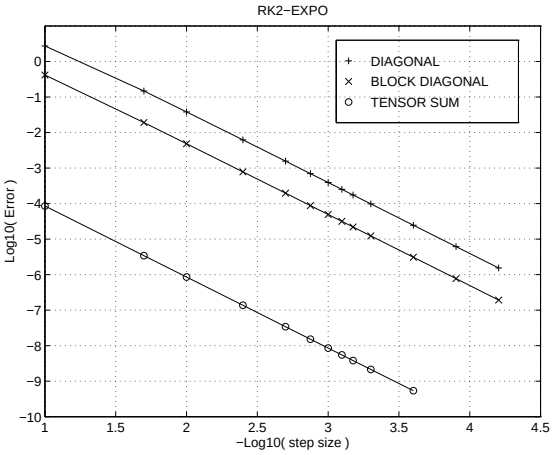

Figure 5: $\alpha = 0.001$

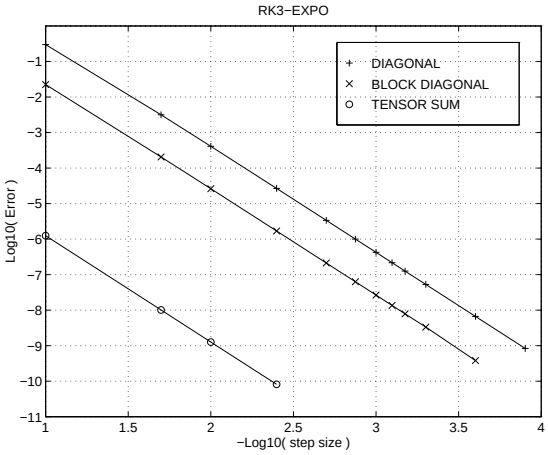

Figure 6: $\alpha = 0.001$

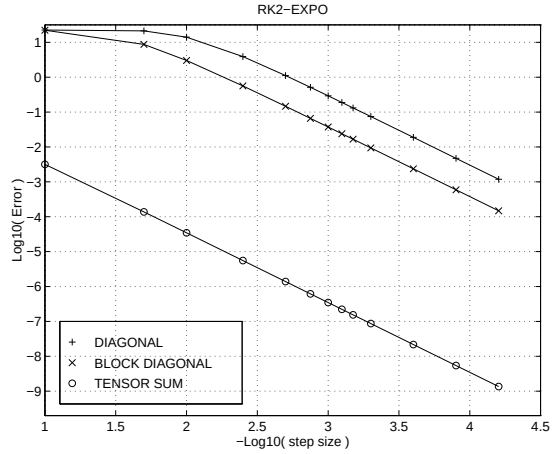

Figure 7: $\alpha = 0.01$

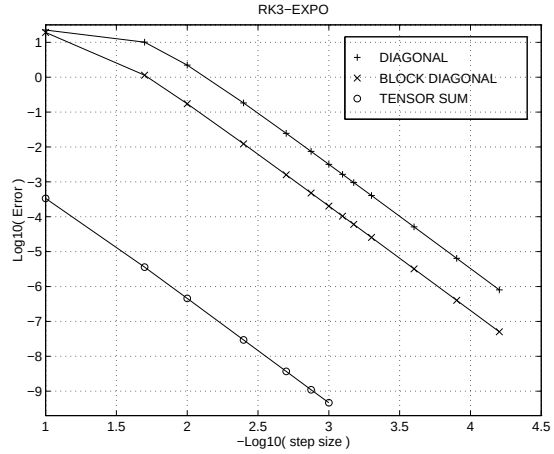

Figure 8: $\alpha = 0.01$

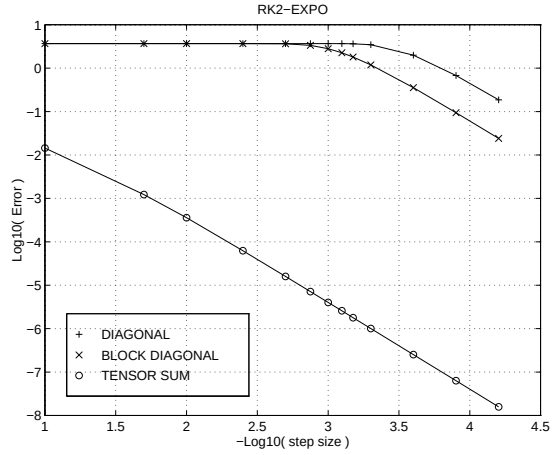

Figure 9: $\alpha = 0.1$

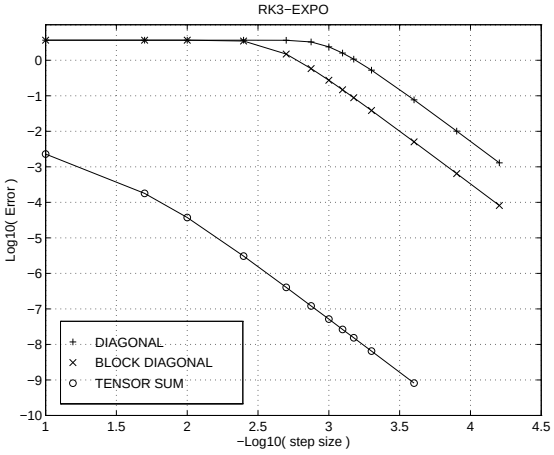

Figure 10: $\alpha = 0.1$

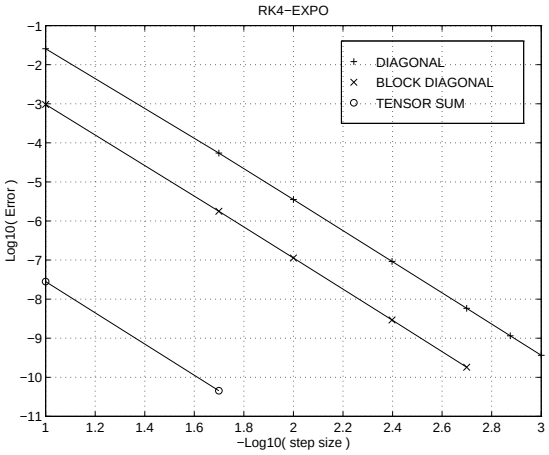

Figure 11: $\alpha = 0.001$

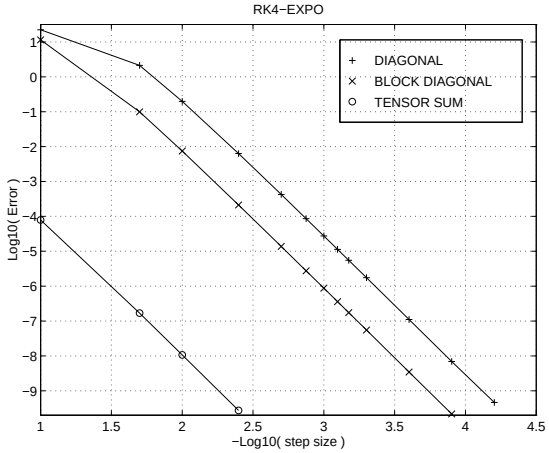

Figure 12: $\alpha = 0.01$

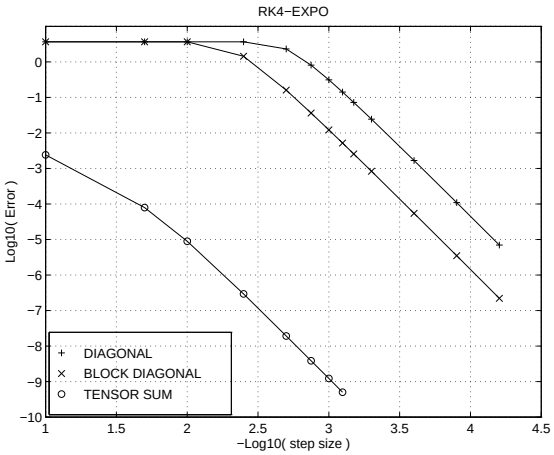

Figure 13: $\alpha = 0.1$

References

[1] P. Castillo and Y. Saad, *Tensor sum approximation preconditioners*, Proc. Eighth SIAM Conference on Parallel Processing for Scientific Computing, Minneapolis, March 1997.

[2] P. Chartier and B. Philippe, *Solution of Markov processes by waveform relaxation methods*, Tech. Report, IRISA, University of Rennes, France, 1997, In preparation.

[3] B.L. Ehle and J.D. Lawson, *Generalized Runge-Kutta processes for stiff initial-value problems.* J. Inst. Maths. Applics. 16 1975, pp. 11-21.

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

[5] E. Gallopoulos and Y. Saad, *Efficient solution of parabolic equations by Krylov approximation methods.* SIAM J. Sci Stat. Comput. Vol 13, No. 5, 1992, pp. 1236-1264.

[6] E. Hairer, S. P. Norsett and G. Wanner, *Solving Ordinary Differential Equations I.* Springer-Verlag, Berlin, 2nd edition, 1993.

[7] M. Hochbruck and C. Lubich, *On Krylov subspace approximations to the matrix exponential operator.* To appear in SIAM J. Numer. Anal.

[8] M. Hochbruck, C. Lubich and H. Selhofer, *Exponential integrators for large systems of differential equations.* To appear in SIAM J. Sci. Comp.

[9] L.A. Knizhnerman, *Computations of functions of unsymmetric matrices by means of Arnoldi's method.* J. Comput. Math. and Math. Phys. Vol 31, No. 1, 1991, pp. 5-16.

[10] J.D. Lawson, *Generalized Runge-Kutta processes for stable systems with large Lipschitz constants.* SIAM J. Numer. Anal. Vol 4, No 3, 1967, pp. 372-380.

[11] J.D. Lawson, *Some numerical methods for stiff ordinary and partial differential equations.* Proc. Second Manitoba Conference on Numerical Mathematics., 1972, pp. 27-34.

[12] J.D. Lawson and D.A. Swayne, *High-order near best uniform approximations to the solution of heat conduction problems.* Proc. IFIP Congress 80, 1980, pp. 741-746.

[13] J.D. Lawson and D.A. Swayne, *Reduction of matrix factorizations in solvers for stiff ordinary differential equations.* Congressus Numerantium 52, 1986, pp. 147-152.

[14] B. Nour-Omid, *Applications of the Lanczos algorithm.* Comput. Phys. Comm., 53, 1989, pp. 153-168.

[15] C.B. Moler and C.F. VanLoan, *Nineteen dubious ways to compute the exponential of a matrix.* SIAM Review, Vol 20, No. 4, 1978, pp. 801-836.

[16] Y. Saad, *Analysis of some Krylov subspace approximations to the matrix exponential operator.* SIAM J. Numer. Anal. Vol 29, No. 1, 1992, pp. 209-228.

[17] Y. Saad, *SPARSKIT: A basic tool kit for sparse matrix computations.* Technical Report 90-20, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffet Field, CA, 1990.