BIT manuscript No.
(will be inserted by the editor)

Comparison of various methods for computing the action of the matrix exponential

Marco Caliari · Peter Kandolf · Alexander Ostermann · Stefan Rainer

Received: date / Accepted: date

Abstract The implementation of exponential integrators requires the action of the matrix exponential and related functions of a large matrix. There are various methods in the literature for carrying out this task. In this paper we describe a new implementation of a method based on interpolation at Leja points. We numerically compare this method with others from the literature. As we are interested in exponential intergrators we choose the test examples from spatial discretization of time dependent partial differential equations in two and three space dimensions. The test matrices thus have large eigenvalues and can be nonnormal.

Keywords Leja interpolation · action of matrix exponential · Krylov subspace method · Taylor series · exponential integrators

Mathematics Subject Classification (2000) 65F60 · 65D05 · 65L04

Peter Kandolf acknowledges the financial support by a scholarship of the University of Innsbruck.

Marco Caliari
Dipartimento di Informatica, Università di Verona, Ca Vignal 2, Strada Le Grazie 15, I-37134 Verona, Italy
E-mail: marco.caliari@univr.it

Peter Kandolf
Institut für Mathematik, Universität Innsbruck, Technikerstr. 13, A-6020 Innsbruck, Austria
E-mail: peter.kandolf@uibk.ac.at

Alexander Ostermann
Institut für Mathematik, Universität Innsbruck, Technikerstr. 13, A-6020 Innsbruck, Austria
E-mail: alexander.ostermann@uibk.ac.at

Stefan Rainer
Institut für Mathematik, Universität Innsbruck, Technikerstr. 13, A-6020 Innsbruck, Austria
E-mail: alexander.ostermann@uibk.ac.at

1 Introduction

In recent years, exponential integrators (see, e.g. [7]) became an attractive choice for the time integration of large stiff systems of differential equations. The efficient implementation heavily relies on the fast computation of the action of certain matrix functions on a given vector. Standard methods such as Padé approximation or diagonalization are only reasonable if the dimension of the system is small. For large scale problems other methods have to be considered.

In this article we are concerned with the comparison of four commonly used classes of methods for approximating the action of large scale matrix functions on vectors, namely Krylov subspace methods, Chebyshev methods, Taylor series methods and a new version of the so-called Leja point method. All used codes are implemented in Matlab and therefore easily comparable in efficiency.

Most exponential integrators make use of linear combinations of the exponential and the related φ functions (see, e.g. [7]). Moreover, since the computation of φ functions can be rewritten in terms of a single matrix exponential by considering a slightly augmented matrix (see, e.g. [2,10,11]), we will concentrate in this article on the numerical approximation of the matrix exponential applied to a vector. The comparisons are carried out in the following way. For a prescribed absolute or relative tolerance we measure the CPU time that the codes need to calculate their result. Furthermore we check whether the results meet the prescribed accuracy. To make a fair comparison in Matlab all codes are tested using a single CPU only.

The outline of the paper is as follows. In Section 2 we recall the idea of the Leja point method and introduce a new numerical realization. In Section 3 we present the competing methods used in our numerical experiments. Finally Section 4 is devoted to the comparison of the methods. As test examples we use spatial discretizations of linear operators arising from partial differential equations in either two or three space dimensions. We conclude in Section 5.

2 Leja approximation of the action of the matrix exponential

The Leja approximation of the matrix exponential $\exp(A)v$ is based on interpolation of the underlying scalar exponential function at Leja points. Their selection is governed by the spectral properties of A. The method was first proposed in [6] for the φ_1 function. In the following we briefly recall the method, some modifications will be discussed in Section 2.1.

In our applications the spectrum of A satisfies

$$\alpha \le \operatorname{Re}\sigma(A) \le \nu \le 0, \quad -\beta \le \operatorname{Im}\sigma(A) \le \beta.$$

We recover the values α (smallest real), ν (largest real) and β (largest imaginary part) by separately considering the symmetric and skew-symmetric part of A. By using Gershgorin's disk theorem we approximate their spectra. This

provides us with the interval $[\alpha, \nu]$ for the symmetric part and $\mathrm{i}[-\beta, \beta]$ for the skew-symmetric part. Therefore, the field of values $\mathcal{F}(A)$ of A (and hence its spectrum) is contained in the rectangle R with vertices of vertexes $(\alpha, -\beta)$, (α, β), (ν, β) and $(\nu, -\beta)$. The number ν represents the real part of the smallest eigenvalue in magnitude. This, or even a reasonable approximation of it, might be hard to obtain, e.g. in the case of *operator* functions where Gershgorin's disks cannot be computed. From our experience, in such cases ν can safely be set to 0. On the other hand, accurate enough approximations of the values α and β can be obtained by a few iterations using the direct power method (or more sophisticated methods, such as *implicitly restarted Arnoldi methods*, used by ARPACK [8] and `eigs` of Matlab) applied to the symmetric and the skew-symmetric part of the operator, respectively.

Now we can construct the ellipse $\mathcal{E}$ with semiaxes a and b circumscribing the rectangle R with smallest capacity $(a + b)/2$. We take Leja points on the focal interval (i.e. the interval between the foci) of this ellipse. From the maximal convergence properties of scalar interpolation at Leja points, we have superlinear convergence in the matrix case as well (see [6] and references therein).

For a compact set $K \subset \mathbb{C}$ a sequence of Leja points is defined recursively by

$$z_m \in \arg\max_{z \in K} \prod_{j=0}^{m-1} |z - z_j|, \quad z_0 \text{ given.}$$

They lie on ∂K by the maximum principle. In our application, K is the focal interval I of the ellipse $\mathcal{E}$. If $I \subset \mathbb{R}$ is horizontal (this happens when $\nu - \alpha \leq 2\beta$) it is possible to use a set of precomputed Leja points on the reference interval $[-2, 2]$. Now one interpolates the function $\exp(c + \gamma\xi)$, $\xi \in [-2, 2]$ in the Newton form, where c is the midpoint of the focal interval and γ a quarter of its length. The resulting scheme (for simplicity written for the approximation of the scalar function $\exp(z)$) is

$$\begin{aligned} p_m(z) &= p_{m-1}(z) + d_m r_{m-1}(z) \quad \text{for } m > 0, \\ r_m(z) &= \big((z - c)/\gamma - \xi_m\big) r_{m-1}(z,) \end{aligned}$$

with

$$p_0(z) = d_0, \quad r_0(z) = \big((z - c)/\gamma - \xi_0\big),$$

where $\{d_j\}_{j=0}^m$ are the divided differences of the function $\exp(c + \gamma\xi)$ at the Leja points $\{\xi_j\}_{j=1}^m$ of the interval $[-2, 2]$. On the other hand, if the focal interval I of the ellipse is parallel to the imaginary axis (this happens when $\nu - \alpha < 2\beta$), then the Leja points are complex and even the approximation of a real matrix function would be performed in complex arithmetic. Instead we consider *conjugate pairs* of Leja points which are symmetric, by construction.

They are defined by $z_0 = c \in \mathbb{R}$ and

$$z_m \in \arg\max_{z \in I} \prod_{i=0}^{m-1} |z - z_i|, \quad z_{m+1} = \bar{z}_m \quad \text{for } m \text{ odd}.$$

They can also be precomputed on the reference interval $\mathrm{i}[-2,2]$, and the Newton interpolation can be written in *real* arithmetic (if the argument z is real), i.e.

$$\begin{aligned} p_m(z) &= p_{m-2}(z) + \mathrm{Re}(d_{m-1})\, r_{m-2}(z) + d_m q_m(z), \qquad \text{for } m > 0 \text{ even}, \\ r_m(z) &= \tfrac{1}{\gamma}(z-c) q_m + \mathrm{Im}(\xi_{m-1})^2 r_{m-2}(z), \\ q_m(z) &= \tfrac{1}{\gamma}(z-c) r_{m-2}(z) \end{aligned}$$

with

$$p_0(z) = d_0, \quad r_0(z) = (z-c)/\gamma,$$

where now $\{d_j\}_{j=0}^m$ are the divided differences (real for even j) of the function $\exp(c+\gamma\xi)$ at the conjugate complex Leja points $\{\xi_j\}_{j=1}^m$ of the interval $\mathrm{i}[-2,2]$. In Figure 2.1 we see an example of Newton interpolation at complex Leja points

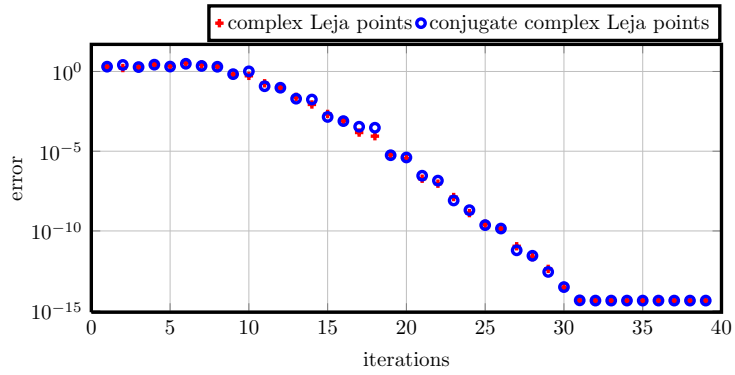

Fig. 2.1 Convergence rates for the interpolation of $\exp(z)$, $z \in \mathrm{i}[-8,8]$ at complex Leja points and conjugate complex Leja points,respectively.

and conjugate complex Leja points, respectively, with no evident difference in the convergence rates. In the practical implementation in the matrix case it is sufficient to use two (three) vectors $p = p_m$ and $r = r_m$ ($q = q_m$) and to update them at each iteration. Moreover, a quite good *a posteriori* estimate e_m of the interpolation error is given by the difference of two successive approximations (see [5])

$$\exp(z) - p_{m-1}(z) \approx e_m = d_m r_{m-1}(z), \tag{2.1a}$$

for interpolation at Leja points and

$$\exp(z) - p_{m-2}(z) \approx e_m = \mathrm{Re}(d_{m-1})\, r_{m-2}(z) + d_m q_m(z), \tag{2.1b}$$

for interpolation at conjugate complex Leja points.

In the notion of the ε-pseudospectrum of A

$$\Lambda_\varepsilon(A) = \{z \in \mathbb{C}\colon \left\|(zI - A)^{-1}\right\|_2 \geq \varepsilon^{-1}\}$$

it is possible to derive an *a priori* error estimate for the convergence rate in the matrix case (see [6]). It is essentially based on the scalar convergence rate. Unfortunately, this estimate leads in practice to a considerable overestimate, since the ellipse containing $\Lambda_\varepsilon(A)$ is usually much larger than $\Lambda_\varepsilon(A)$ itself.

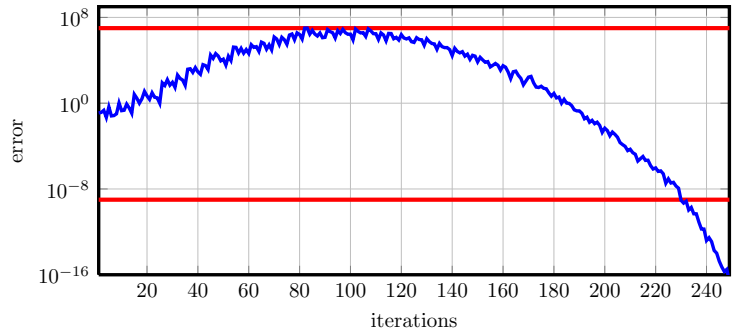

Fig. 2.2 Convergence rate (a posteriori error estimate) for a 2D advection-diffusion operator discretized by finite differences (see Example 1 below).

In Figure 2.2 a typical behavior for interpolation of a matrix function can be observed. We display the *a posteriori* error estimate (norm of e_m in (2.1a)) for the evaluation of $\exp(\tau A)v$ in Example 1 ($N = 100, \mathrm{Pe} = 0.495, \tau = 5 \cdot 10^{-3}$). The large hump is produced by the globally (not monotonically) decreasing divided differences and the increasing magnitude of the matrix polynomial r_{m-1}. This behavior is hardly seen in the scalar case but in the context of matrix functions referred to as the *hump effect*, see [9, 11].

A second observation, drawn from Figure 2.2, is that even if the terms e_m decrease eventually (this is true as the divided differences are computed as described in [4] and not in the standard way) they very more than 16 orders of magnitude. Therefore it is not possible to reach an error below 10^{-9} in double precision (bottom line). A remedy for this problem is to use the functional equation of the exponential and introduce a sub stepping procedure to approximate $\exp(A)v$ by

$$\exp(A)v = \underbrace{\exp(A/s)\ldots\exp(A/s)}_{s \text{ inner steps}}\, v,$$

where each of the s inner steps is computed by the above algorithm.

2.1 The new strategy for the Leja interpolation

Our aim is to speed up the Leja interpolation in terms of CPU time by keeping a simple user interface. For large matrices the most resource consuming part is the Newton interpolation, in particular the computation of the matrix-vector products and the error estimation. In this context we start from the reasonable assumption that the underlying `LAPACK` and `BLAS` routines are implemented in an optimal way. Therefore we can not influence the computational cost of the matrix-vector products. Consequently we concentrate on implementing the Newton interpolation in the most efficient way using the smallest possible number of interpolation points. If the amount of inner steps and the amount of interpolations points per step are known a priory, the interpolation can be optimized.

As mentioned above the Leja interpolation highly depends on the smallest real eigenvalue α as well as the largest imaginary eigenvalue β of the matrix A. These values define the confocal interval of the used Leja points. The smallest matrix that has the same spectral properties is the 2×2 matrix

$$S = \begin{bmatrix} \alpha & \beta \\ -\beta & 0 \end{bmatrix}.$$

If we perform an interpolation with S to compute $\exp(S)w$, the convergence rate should roughly be the same as for interpolating $\exp(A)v$. We can then, once and for all, compute the convergence range of the Leja interpolation for varying α and β and use this as an a priory estimate of our unknown properties. There are some constraints, the most important one is the acceptable number of Leja points for one inner step. Tests show that an upper limit of 150 points is reasonable for one inner step. The selection procedure is then rather simple. If we are in the convergence range, i.e. the interpolation for S converged in $\ell \leq 150$ iterations, we use ℓ iterations for the interpolation of A as well. Otherwise we introduce additional inner steps, reduce α and β accordingly and make a new guess on the interpolation points. It is necessary to compute convergence ranges for various tolerances and distinguish between real and complex conjugate Leja points.

This procedure is working well for normal matrices, however, it gives problems for nonnormal matrices. In the latter case the obtained amount of interpolation points from S is to low, in general. Nevertheless by analyzing the promising results for normal matrices we where able to define an upper and a lower estimate for the number of iterations. For each of the sampling tolerances $10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}$, and for real and complex conjugate Leja points independently we define the following procedure with the help of those two estimates.

(a) Compute the values α and β of the rectangle R surrounding $\sigma(A)$.
(b) Predict the amount of inner steps and compute upper and lower limits of required iterations per step (s, m, n). Here the sampling tolerance is chosen smaller than the prescribed tolerance.

(c) Compute m divided differences for the step size $1/s$.
(d) Perform a Newton interpolation for s inner steps. The error is checked the first time after n iterations and then after every fifth iteration.

The error estimate is based on (2.1). Henceforth this code will be called `Leja`. The code computes the relative error scaled to the computed solution. Due to the selection of n, the superlinear convergence and the discrete range of sampling tolerances, it is possible that the obtained results are *too* accurate. We also note that this approach is extendable to allow a dense output for several values of τ (time step size) as described in [2].

3 Further methods for computing the action of a matrix function

There are a large number of codes for computing the action of matrix function on a given vector. However, many of these codes are described only in a paper, implemented in other program languages (such as FORTRAN or C), or not freely accessible. Such codes will not be considered here. There is also a large class of algorithms that compute the full matrix function and not its action on a given vector. Whereas such an approach works fine for small-scale problems it is too costly for situations that we have in mind. Finally we will not consider rational approximations, such us rational Krylov methods or contour integrals, since they need to solve linear systems which is too expensive in general.

We will now briefly describe the other codes that we considered in the numerical tests. We divided them into three classes.

3.1 Krylov subspace methods

The computation of $\exp(A)v$ with a Krylov method basically consists of two steps. The first step finds an appropriate Krylov subspace and the second computes the matrix exponential of a smaller matrix using standard methods.

We consider two codes based on Krylov subspace methods, namely the implementation used in [11] (`Expokit`) and an implementation by Hochbruck[1] (`Krylov2`). Both methods compute the Krylov subspace using the Arnoldi method. Given the matrix $A \in \mathbb{C}^{n\times n}$ and the vector $b \in \mathbb{C}^n$ the Arnoldi method computes an orthonormal basis $V_m \in \mathbb{C}^{n\times m}$ of the Krylov subspace $\mathcal{K}_m(A, b)$ and an upper Hessenberg matrix $H_m \in \mathbb{C}^{m\times m}$. Here the mth Krylov subspace is defined by

$$\mathcal{K}_m(A, b) = \operatorname{span}\{b, Ab, \ldots, A^{m-1}b\}.$$

For this basis and the Hessenberg matrix one gets the relation

$$(\lambda I - A)V_m = V_m(\lambda I - H_m) + h_{m+1,m}v_{m+1}e_m^{\mathrm{T}},$$

where e_m^T denotes the mth unit vector in $\mathbb{R}^m$.

[1] We thank M. Hochbruck, T. Pazur and A. Demirel for providing us with a Matlab code.

Using the fact that $V_m(\lambda I - H_m)^{-1}e_1$ is a Galerkin approximation to $(\lambda I - A)^{-1}b$ one then gets, by Cauchy's integral formula

$$\begin{aligned}\exp(A)b &= \frac{1}{2\pi\mathrm{i}}\int_\Gamma \mathrm{e}^\lambda(\lambda I - A)^{-1}b\,\mathrm{d}\lambda \\ &\approx \frac{1}{2\pi\mathrm{i}}\int_\Gamma \mathrm{e}^\lambda V_m(\lambda I - H_m)^{-1}e_1\,\mathrm{d}\lambda = V_m\exp(H_m)e_1,\end{aligned}$$

where the curve Γ surrounds the field of values of the matrix A.

Both codes use the matlab routine `expm` to compute $\exp(H_m)$. A crucial parameter is the maximally allowed dimension of the Krylov subspace. Whereas `Expokit` requires $m \le 30$, `Krylov2` allows $m \le 1000$. If `Krylov2` is not able to find an reliable solution in $\mathcal{K}_{1000}(A, b)$ it stops with an error, `Expokit` performs a substep strategy to find an acceptable approximation.

3.2 Taylor series methods

The code `expmv` (see [2]) approximates the exponential of a given matrix $A \in \mathbb{C}^{n\times n}$ acting on $v \in \mathbb{C}^n$ by

$$\exp(A)v \approx \left(T_m(s^{-1}A)^s\right)v,$$

with T_m denoting the truncated Taylor series expansion of $\exp(x)$. For a fixed integer m, the parameter s has to guarantee that

$$s^{-1}A \in \left\{M \in \mathbb{C}^{n\times n} : \rho(\mathrm{e}^{-M}T_m(M) - I) < 1\right\}$$

with $\rho(M)$ denoting the spectral radius of M. For each m, it is possible to choose the optimal value of the integer s in such a way that $\|\Delta A\| \le \mathrm{TOL}\,\|A\|$ for any matrix norm and given tolerance TOL, where $\|\Delta A\|$ is implicitly defined by $T_m(s^{-1}A)^s = \exp(A + \Delta A)$. The value of m is chosen in such a way to minimize the computational cost of the algorithm, taking into account that the evaluation of a too long Taylor expansions for matrices $s^{-1}A$ with large norm could lead to numerical instability. Therefore, it is required that $m \le m_{\max} = 55$.

The code `expmv` allows only three different sizes of tolerances TOL, namely half, single, and double precision.

3.3 Chebychev methods

The idea to use a Chebychev polynomial for approximating a matrix function is quite established (see [1,3]). Chebychev methods work efficiently if the matrix A is Hermitian or skew-Hermitian. The code `Cheb` by Güttel[2] is designed for Hermitian matrices. For a given Hermitian matrix $A \in \mathbb{C}^{n\times n}$ with eigenvalues in $[\alpha, \nu] \subset \mathbb{R}$ the action of the exponential of A on v is approximated by a

[2] We thank S. Güttel for providing us with a Matlab code.

Chebychev expansions with points in $[\alpha, \nu]$. In the code `Cheb` the necessary Chebychev coefficients are computed with `fft`, even though they are values of known Bessel functions. A similar approach is used for solving ordinary differential equations (e.g. [1]) but is not considered in this paper.

4 Numerical comparisons

In this section we outline some differences between the different methods described above as well as our new implementation on the basis of four test examples. All of the following numerical comparisons are computed with Matlab 2012a on a 64bit (glnxa64) Fedora 16 workstation with a 3GHz Intel Core 2 vPro and 8GB RAM. Matlab is restricted to a single computational thread, on a single core, by the `-singleCompThreat` command and in addition the JVM is deactivated with the `-nojvm` command. This configuration allows a comparison without any parallel computations. We are looking at these computations from the exponential integrator point of view. Therefore we are not interested in results that are as accurate as possible but up to a typical accuracy of 6 digits. We used `expmv` from [2] with maximal precision to compute a reference solution.

In the following we will briefly describe each of the examples and the configuration for the respective numerical experiment. In each of the test cases we compute $\exp(\tau A)v$ for a prescribed tolerance TOL. Each of the codes is only provided with τ, A, v and TOL to keep the user interface simple. The time step size τ allows us to control the magnitude of the eigenvalues of the example matrices and therefore to vary between stiff and nonstiff situations.

Example 1 (advection-diffusion equation) We start to investigate the behaviour of the methods by an example that allows us easily to vary the spectral properties of the discretization matrix. We consider the advection-diffusion equation

$$\partial_t u = \varepsilon \Delta u + c \nabla u$$

on the domain $\Omega = [0, 1]^2$ with homogeneous Dirichlet boundary conditions. This problem is discretized by finite differences with grid size $\Delta x = \frac{1}{N+1}, N \geq 1$. This results in a sparse $N \times N$ matrix A and a problem with $\mathrm{df} = N^2$ degrees of freedom. We define the grid Péclet number

$$\mathrm{Pe} = \frac{c\Delta x}{2\varepsilon}$$

as the ratio of advection to diffusion, scaled by Δx. By increasing Pe the normality of the discretization matrix can be controlled. For the computations displayed in Figures 4.1–4.3 the parameters are chosen as follows: $\varepsilon = 1$ and $c = \frac{2\varepsilon \mathrm{Pe}}{\Delta x}$. As initial function we use $u_0(x, y) = 256 \cdot x^2(1-x)^2 y^2(1-y)^2$.

In Figure 4.1 the matrix changes from normal ($\beta = 0$) to nonnormal ($\beta = -0.45\alpha$). The Chebyshev method is only working reliable for normal

matrices whereas the other methods work fine for all cases. The methods `Leja` and `expmv` have approximately the same computational effort, independently of the normality of the input matrix. For `Expokit`, the computational cost slightly increase with increasing grid Péclet number, whereas `Krylov2` shows a considerable increase of the computational cost. In Figure 4.1 we used the maximum norm. We note that the discrete L^2 norm gave almost identical pictures.

In Figure 4.2 we take a closer look for $\mathrm{Pe} = 0.5$ and also include the actual error for each of the methods (except `Cheb` for the above described reasons). For this choice of Pe, the exact solution becomes very small and the distinction between relative and absolute error gets important. We can see that `Expokit` and `Krylov2` effectively only consider the absolute error, whereas `expmv` and `Leja` match the specified relative tolerance.

In our third test case for this example, see Figure 4.3, we fix the dimension of A but vary the grid Péclet number between 0 and 1. The results are somewhat similar to those in Figure 4.1. One observes again that the computational cost increase for higher degrees of freedom but is almost constant, for the methods `Leja`, `expmv` and `Expokit`, for varying Pe. These three method also have about the same computational cost (varying by a factor 3). The `Krylov2` method experiences more and more difficulties for increasing nonnormality of A and in the worst case has a almost 40 times higher cost than `Leja`. The errors in this example are measured in a discrete L^2 norm but again the pictures stay almost the same for the maximum norm.

Throughout this experiments one could see that for small matrices the overhead produced by the computation of the divided differences in `Leja` is slowing down the method. For higher and higher dimensions however, the percentage of this (almost) fixed cost decrease in comparison to the overall cost and the method improves with respect to the other methods, see Figure 4.1. The difference between `expmv` and `Leja` can be explained by the amount of matrix-vector products needed by each of the methods.

Example 2 (reactive transport in heterogeneous porous media [12]) The advection-diffusion-reaction equation

$$\phi(x)\partial_t u = \nabla \cdot (D\nabla u) - \nabla \cdot (q(x)u) + R(x,u), \quad D = \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix}$$

is used to model many applications in geo-engineering. A finite volume discretization with $N \times N$ points in the domain $\Omega = [0,1]^2$ results in a sparse discretization matrix with a more complex structure than the one in the previous example. In particular, we are using the matrix L of Eg. (6) from [12, p. 3959].[3] For the computation displayed in Figure 4.4 the diffusion parameters are set to $D_1 = 10^{-3}, D_2 = 10^{-4}$, and the dynamical viscosity is chosen $\mu = 1$. As initial function we use $u_0(x,y) = 256 \cdot x^2(1-x)^2y^2(1-y)^2$.

[3] We thank A. Tambue for providing us with a Matlab code.

In Figure 4.4 we investigate the behavior of the considered methods for increasing matrix dimensions. In this experiment we prescribed a higher accuracy but but due to the super-linear convergence this does not result in a considerably higher cost. The spectral properties of L are similar to Example 1 with $\mathrm{Pe} = 0.9$. As soon as the overhead of `Leja` is compensated by the reduced amount of matrix-vector products `Expokit`, `expmv` and `Leja` have roughly the same computational cost with an almost negligible disadvantage for `Expokit`. The `Krylov2` implementation on the other hand has a considerable increase in computational cost. Even though the computational cost are high, `Krylov2` produces the largest errors of all methods. the absolute error, however, is still below the prescribed tolerances. For this example we do not have a monotonic growth of computational cost, due to the definition of the permeability and the porosity $\phi(x)$ of the heterogeneous media, which includes random variables. The experiments show however that all the methods are effected almost in the same way.

Example 3 (Schrödinger equation with harmonic potential in 3D) Our next example is the 3D Schrödinger equation

$$\partial_t u = \frac{\mathrm{i}}{2} \left(\Delta - \varepsilon |x|^2 \right) u$$

with harmonic potential. We discretize this problem with finite differences in N^3 points on the domain $\Omega = [0,1]^3$. This results in a discretization matrix with pure imaginary spectrum and allows us to test the stability of the methods on the imaginary axis. Like described in Section 2 this forces the `Leja` implementation to use complex Leja points. Although the spectrum lies on the negative imaginary axis, we decided to use complex conjugate Leja points. As initial function we use $u_0(x,y,z) = 4096 \cdot x^2(1-x)^2 y^2(1-y)^2 z^2(1-z)^2$.

In Figure 4.5 `Leja`, `expmv` and `Expokit` show a nearly parallel linear growth of computational cost, this time with a slight advantage for `Leja`, even though the implementation achieves about 3 digits more accuracy than required. This example shows that the procedure described in Section 2.1 might not always result in the least possible computational cost. However, it is still fast. The CPU time of `Leja`, `expmv` and `Expokit` is varying only by a factor of about 2, whereas for `Krylov2` the CPU time increases by a factor of almost 3000. Already for this moderately stiff example `Krylov2` has problems. This seems curious as `Expokit` and `Krylov2` are both Krylov subspace methods that rely on the `expm` method of Matlab. The main difference seems to be that `Expokit` only allows a Krylov subspace of up to dimension 30, whereas `Krylov2` uses 1000 as a default upper limit. From our experiments `expm` is only feasible for low dimensional problems. For a 1000×1000 matrix one call of `expm` almost needs as much time as the whole computation `Leja` needs to solve the above problem for $N = 24$. This might have a weakened effect when the computation is done on a multi-core machine but for the single core computation the differences are considerable.

Example 4 (Molenkamp–Crowley in 2D with radial basis functions) In order to include an example with a dense matrix we consider the Molenkamp–Crowley equation

$$\partial_t u = \partial_x a(x,y)u + \partial_y b(x,y)u.$$

with $a(x,y) = 2\pi x$ and $b(x,y) = -2\pi y$ in $[-1,1]^2$. This problem is discretized on a regular $N \times N$ grid with Gaussian radial basis functions. The form factor is chosen in such a way that the approximation error in space is smaller than TOL. For a further discussion of the example and some literature on radial basis functions, see [5]. This kind of discretization results in a matrix where more than 90% of the entries are nonzero. As initial function we use

$$u_0(x,y) = 1.1734 \cdot \mathrm{e}^{-100\left((x-0.2)^2+(y-0.2)^2\right)}(x^2-1)^2(y^2-1)^2$$

The results of this example are summarized in Figure 4.6. For a dense matrix the matrix-vector products are more expensive and therefore `Leja` and `expmv` need significantly more CPU time, compared to the sparse situation in the previous examples. The cost for `Krylov2` and `Expokit` are increased as well. Altogether the methods vary only by a factor of about 3. In comparison to the previous examples our implementation of `Leja` needs more matrix-vector products than `expmv` and therefore consumes slightly more CPU time. Due to the storage limitations it is not feasible to include higher dimensional matrices in this experiment as it can not be guaranteed that Matlab is not starting to swap storage to the hard drive. The results in Figure 4.6 are given for the maximum norm. The corresponding results in a discrete L^2 norm look very similar, only `Krylov2` has some problems to keep the relative error requirements.

5 Concluding remarks

In this paper a new method for computing the action of the matrix exponential is proposed. It is based on a polynomial interpolation at Leja points and provides an a priory estimate of the required degree. This enables us to determine the required number of inner steps for an efficient computation.

From the numerical comparisons we can draw the following conclusions. All methods with the very exception of `Cheb` work well for the considered examples. The later gives satisfactory results only for nearly self-adjoint problems. For Krylov subspace methods, the maximal dimension of the subspace should not be too large since the exponential of a Hessenberg matrix of that dimension has to be computed, which can be expensive. A large Krylov dimension and a high degree in the interpolation methods can be avoided by choosing more inner steps which results in a smaller field of values of the matrix. For small step sizes τ, all methods are comparable and work well, but our interest lies in stiff problems with τ not too small as considered in the examples. A clear advantage of our interpolation at Leja points is the low demand of storage due to the employed short-term recursion.

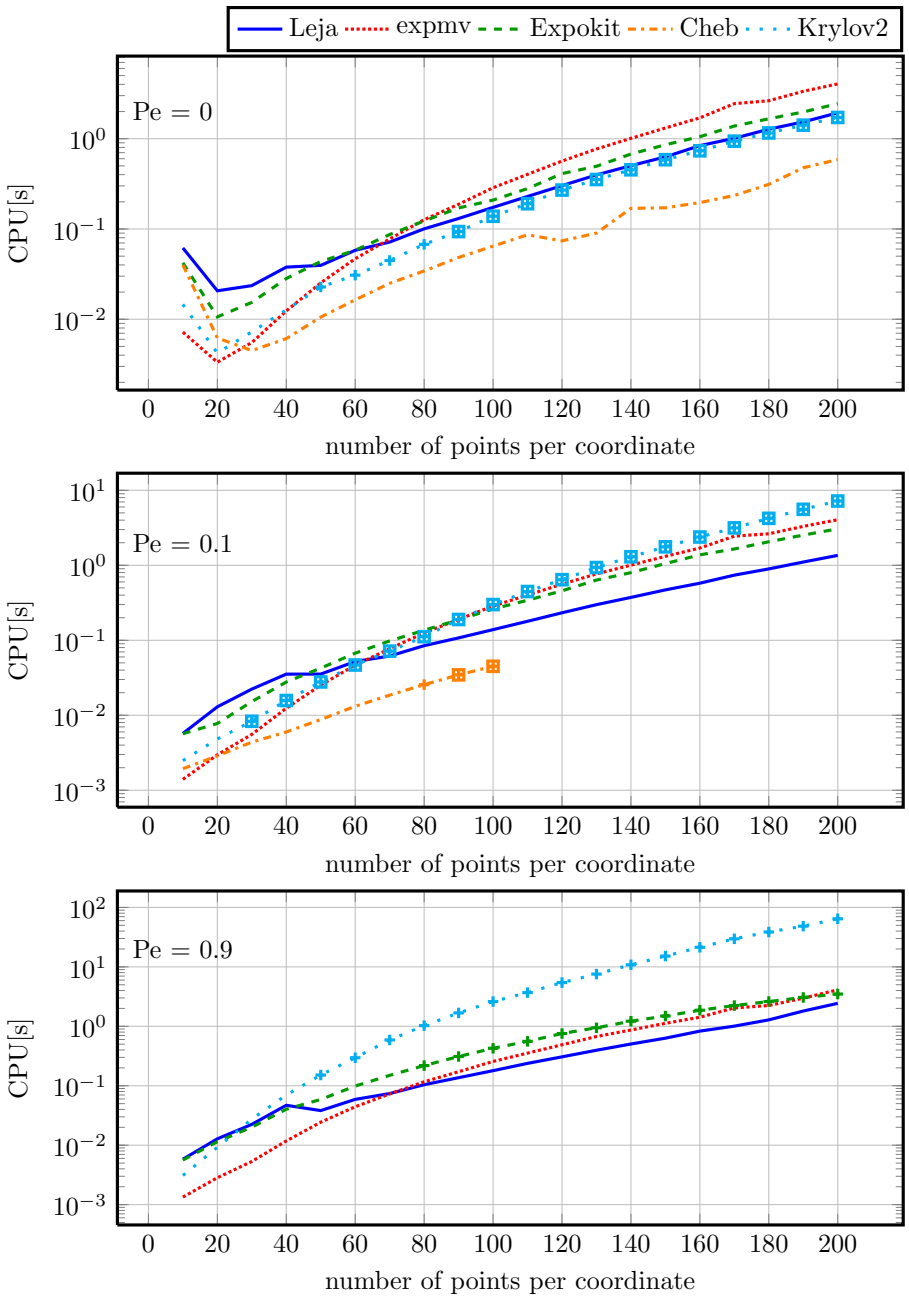

Fig. 4.1 Computational cost of various methods vs. number of points N per coordinate for the evaluation of $\exp(\tau A)u_0$ in Example 1 for grid Péclet number Pe $= 0, 0.1$ and 0.9 and $\tau = 10^{-2}$. The error is measured in the maximum norm for prescribed tolerance TOL $= 10^{-4}$. The $+$ indicates that the relative error is larger than TOL, the $\square$ indicates that the absolute error requirement is slightly violated.

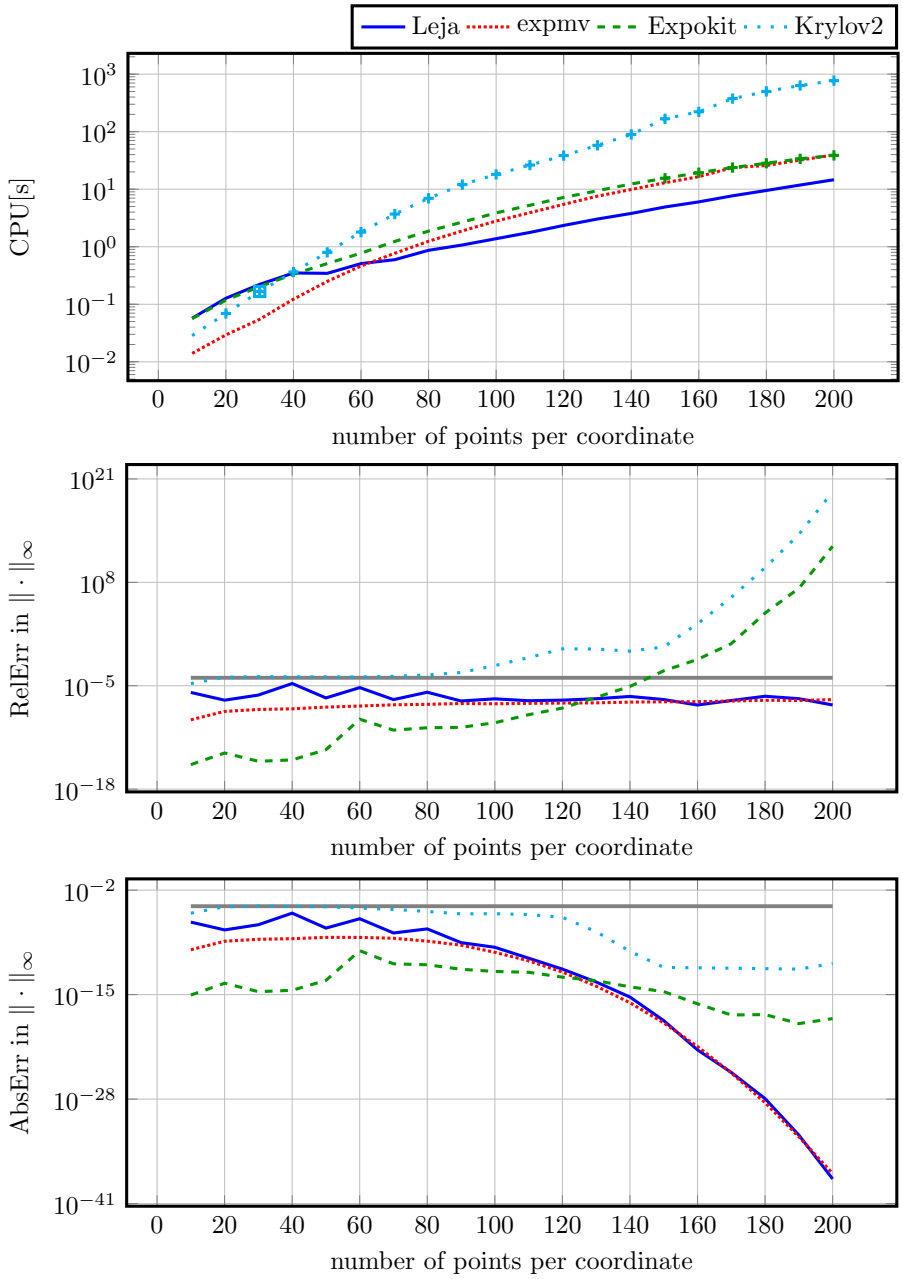

Fig. 4.2 Computational cost and achieved accuracy of various methods vs. number of points per N coordinate for the evaluation of $\exp(\tau A)u_0$ in Example 1 for $\mathrm{Pe} = 0.5$ and $\tau = 10^{-2}$. The error is measured in the maximum norm for prescribed tolerance $\mathrm{TOL} = 10^{-4}$. The $+$ indicates that the achieved relative error is larger than TOL, the $\square$ indicates that the absolute error requirement is slightly violated.

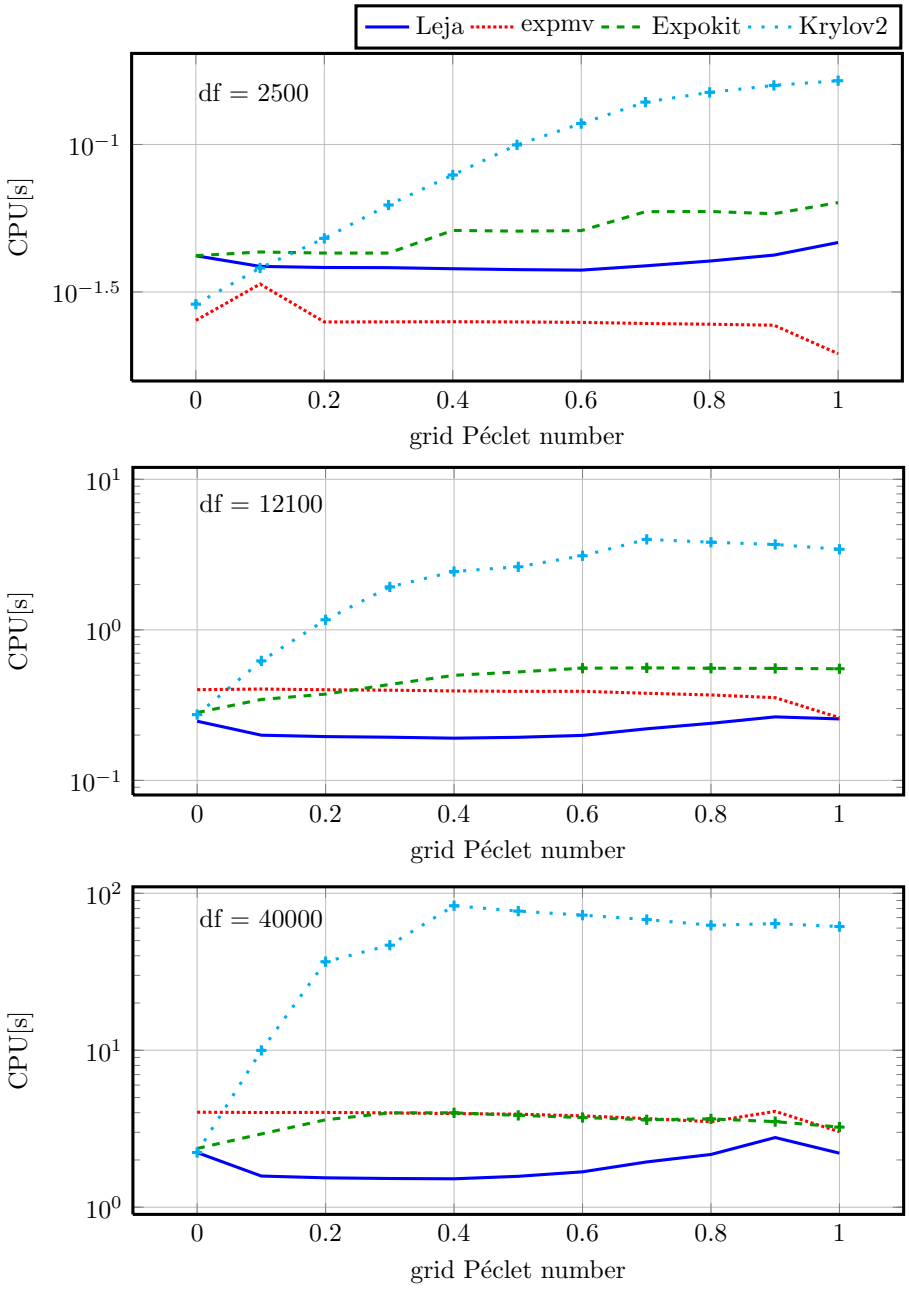

Fig. 4.3 Computational cost of various methods vs. grid Péclet number for the evaluation of $\exp(\tau A)u_0$ in Example 1 with df $= 2500, 12100$ and 40000 degrees of freedom and $\tau = 10^{-2}$. The error is measured in a discrete L^2 norm for prescribed tolerance TOL $= 10^{-6}$. The $+$ indicates that the achieved relative error is larger than TOL.

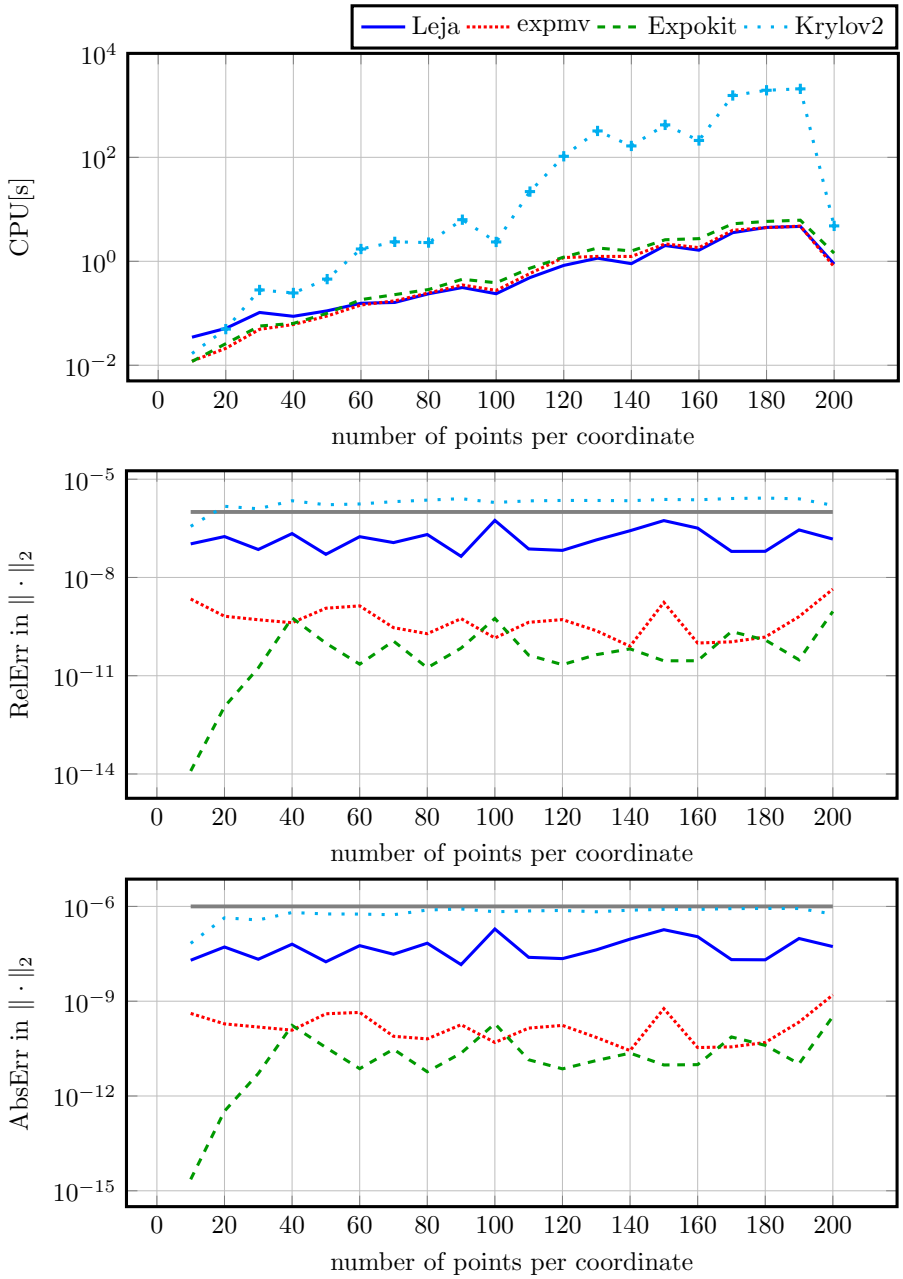

Fig. 4.4 Computational cost and achieved accuracy of various methods vs. number of points N per coordinate for the evaluation of $\exp(\tau A)u_0$ in Example 2 for $\tau = 10^{-1}$. The error is measured in a discrete L^2 norm for prescribed tolerance TOL $= 10^{-6}$. The $+$ indicates that the achieved relative error is larger than TOL.

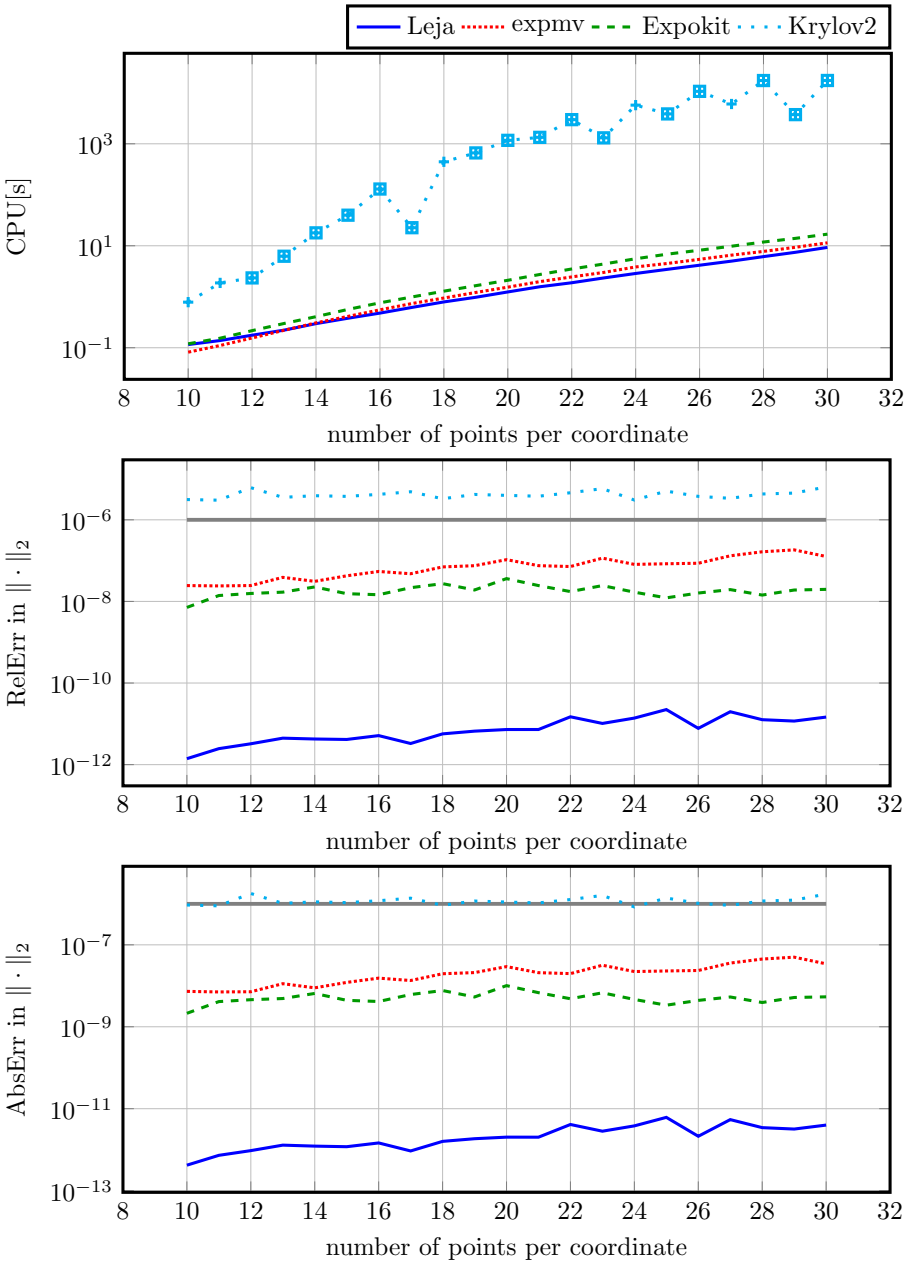

Fig. 4.5 Computational cost and achieved accuracy of various methods vs. number of points N per coordinate for the evaluation of $\exp(\tau A)u_0$ in Example 3 for $\varepsilon = 0.5$ and $\tau = 0.5$. The error is measured in a discrete L^2 norm for prescribed tolerance TOL $= 10^{-6}$. The $+$ indicates that the achieved relative error is larger than TOL, the $\square$ indicates that the absolute error requirement is slightly violated.

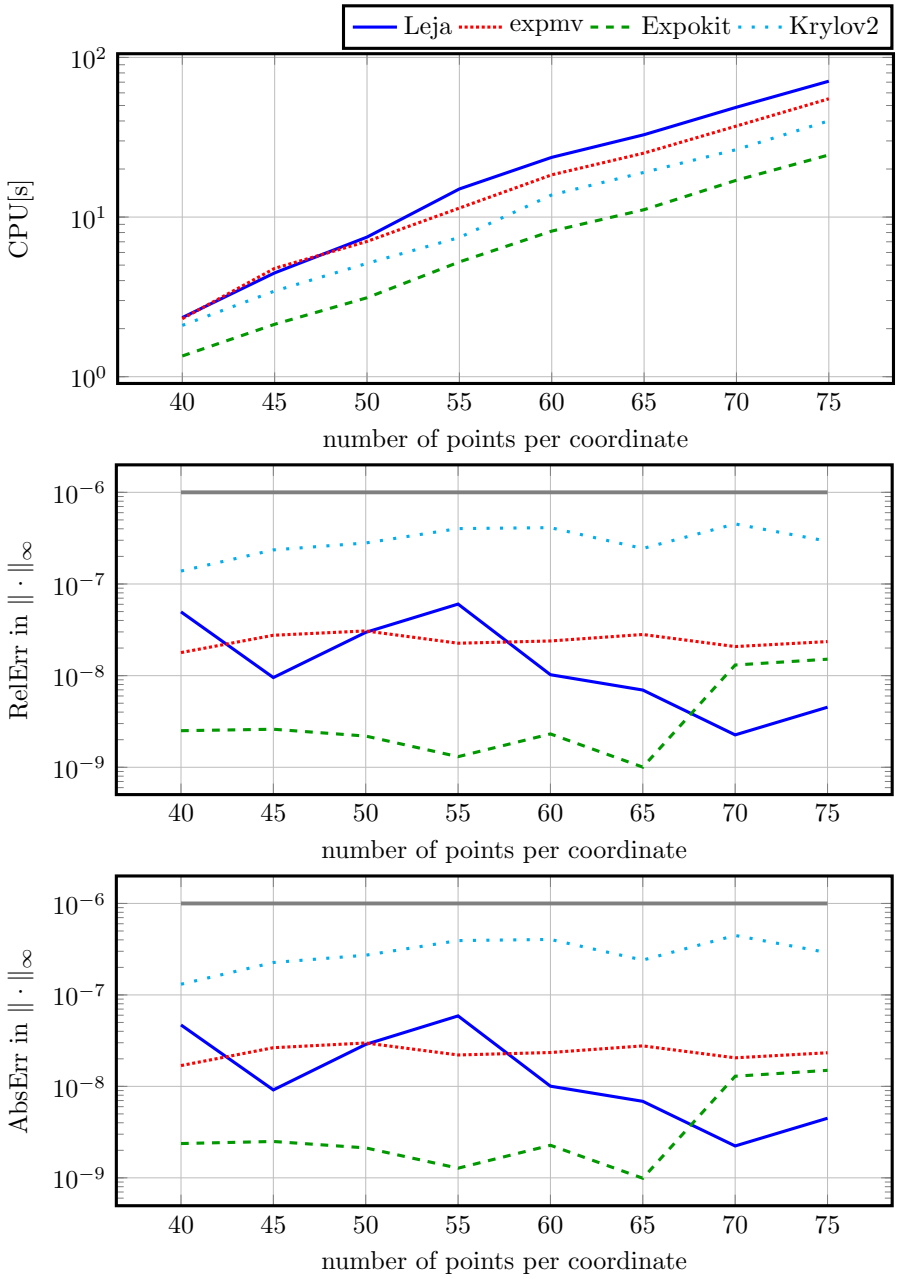

Fig. 4.6 Computational cost and achieved accuracy of various methods vs. number of points N per coordinate for the evaluation of $\exp(\tau A)u_0$ in Example 4 for $\tau = 0.5$. The error is measured in the maximum norm for prescribed tolerance $\mathrm{TOL} = 10^{-6}$.

References

1. Abdulle, A.: Fourth order Chebyshev methods with recurrence relation, SIAM J. Sci. Comput., 23(6), 2042–2055 (2002)
2. Al-Mohy, A.H., Higham, N.J.: Computing the action of the matrix exponential, with an application to exponential integrators, SIAM J. Sci. Comput., 33(2) (2011)
3. Bergamaschi, L., Caliari, M., Vianello M.: Efficient approximation of the exponential operator for discrete 2D advection-diffusion problems, Numer. Linear Algebra Appl., 10(3), 271–289 (2003)
4. Caliari, M.: Accurate evaluation of divided differences for polynomial interpolation of exponential propagators, Computing, 80(2), 189–201 (2007)
5. Caliari, M., Ostermann, A., Rainer, S.: Meshfree exponential integrators, SIAM J. Sci. Comput., Accepted for publication (2013)
6. Caliari M., Vianello, M., Bergamaschi,L.: Interpolating discrete advection-diffusion propagators at Leja sequences, J. Comput. Appl. Math. 172 (1), 79–99 (2004)
7. Hochbruck, M., Ostermann, A.: Exponential integrators, Acta Numer., 19, 209–286 (2010)
8. Lehoucq, R.B., Sorensen, D.C., Yang, C.: ARPACK Users' Guide: Solution of Large Scale Eigenvalue Problems by Implicity Restarted Arnoldi Methods, Software, Environments, and Tools, 6 (1997)
9. Moler, C., Van Loan, C.: Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later, SIAM Rev., 45(1), 3–49 (2003)
10. Saad, Y., Analysis of some Krylov subspace approximations to the matrix exponential operator, SIAM J. Numer. Anal. 29(1), 209–228 (1992).
11. Sidje, R.B.: Expokit. A software package for computing matrix exponentials, ACM Trans. Math. Softw.,1, 130–156 (1998)
12. Tambue, A., Lord, G.J., Geiger, S.: An exponential integrator for advection-dominated reactive transport in heterogeneous porous media, J. Comput. Phys., 229, 3957–3969 (2010)