Comparison of software for computing the action of the matrix exponential

Marco Caliari · Peter Kandolf ·
Alexander Ostermann · Stefan Rainer

Abstract The implementation of exponential integrators requires the action of the matrix exponential and related functions of a possibly 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 other codes from the literature. As we are interested in applications to exponential integrators 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 (2010) 65F60 · 65D05 · 65L04

M. Caliari
Dipartimento di Informatica, Università di Verona, Strada Le Grazie 15, 37134 Verona, Italy
e-mail: marco.caliari@univr.it

P. Kandolf · A. Ostermann (✉) · S. Rainer
Institut für Mathematik, Universität Innsbruck, Technikerstr. 13, 6020 Innsbruck, Austria
e-mail: alexander.ostermann@uibk.ac.at

P. Kandolf
e-mail: peter.kandolf@uibk.ac.at

S. Rainer
e-mail: stefan.rainer@uibk.ac.at

1 Introduction

In recent years, exponential integrators (see, e.g. [8]) 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 terms of efficiency.

Most exponential integrators make use of linear combinations of the exponential and the related φ functions (see, e.g. [8]). 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 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 Sect. 2 we recall the idea of the Leja point method and introduce a new numerical realization. In Sect. 3 we present the competing methods used in our numerical experiments. Finally Sect. 4 is devoted to the comparison of the codes. 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 Sect. 5.

2 Leja approximation of the action of the matrix exponential

The Leja approximation of the matrix exponential $\exp(A)b$ 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 are discussed in Sect. 2.1.

In our implementation of Leja interpolation, we include the spectrum of A in a box that lies *symmetrically* with respect to the real axis. We describe this box using the real numbers α, ν and β, i.e.

$$\alpha \leq \operatorname{Re}\sigma(A) \leq \nu \leq 0, \qquad -\beta \leq \operatorname{Im}\sigma(A) \leq \beta. \tag{2.1}$$

Note that by an appropriate shift of A we can always assume that $\nu \leq 0$. We numerically determine the bounds α, ν and β by separately considering the symmetric and skew-symmetric parts of A. With the help of Gershgorin's disk theorem we easily compute an inclusion to their spectra. This provides us with the interval $[\alpha, \nu]$ for

the symmetric part and $i[-\beta, \beta]$ for the skew-symmetric part. The field of values $\mathcal{F}(A)$ of A (and hence its spectrum) is thus contained in the rectangle R with vertices $(\alpha, -\beta)$, (α, β), (ν, β) and $(\nu, -\beta)$. The number ν is an upper bound to the real part of the smallest eigenvalue in magnitude.

Now we can construct the ellipse $\mathcal{E}$ with semiaxes a_1 and a_2 circumscribing the rectangle R with smallest capacity $(a_1 + a_2)/2$. 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 $\nu - \alpha \geq 2\beta$) it is possible to use a set of precomputed Leja points on the reference interval $[-2, 2]$. Note that good approximations can be obtained in a fast way as described in [3]. Now one interpolates the function $\exp(c + \gamma\xi)$, $\xi \in [-2, 2]$ with Newton's interpolation formula, where c is the midpoint of the focal interval and γ a quarter of its length. The resulting scheme is

$$p_m(z) = p_{m-1}(z) + d_m r_{m-1}(z) \quad \text{for } m > 0,$$

$$r_m(z) = \left(\frac{1}{\gamma}(z - c) - \xi_m\right) r_{m-1}(z)$$

with

$$p_0(z) = d_0, \qquad r_0(z) = \left(\frac{1}{\gamma}(z - c) - \xi_0\right),$$
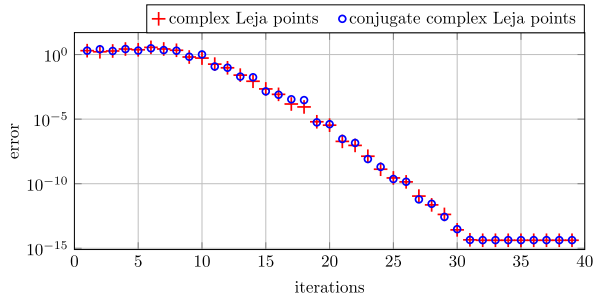
where $\{d_j\}_{j=0}^m$ are the divided differences of the function $\exp(c + \gamma\xi)$ at the Leja points $\{\xi_j\}_{j=1}^m$ on the interval $[-2, 2]$. On the other hand, if the focal interval I of the ellipse is parallel to the imaginary axis (this happens when $\nu - \alpha < 2\beta$), then the 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

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

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

$$p_m(z) = p_{m-2}(z) + \text{Re}(d_{m-1})r_{m-2}(z) + d_m q_m(z) \quad \text{for } m > 0 \text{ even,}$$

Fig. 1 Convergence rates for
the interpolation of $\exp(z)$,
$z \in i[-8, 8]$ at complex Leja
points and conjugate complex
Leja points, respectively

$$r_m(z) = \frac{1}{\gamma}(z - c)q_m + \text{Im}(\xi_{m-1})^2 r_{m-2}(z),$$

$$q_m(z) = \frac{1}{\gamma}(z - c)r_{m-2}(z)$$

with

$$p_0(z) = d_0, \qquad r_0(z) = \frac{1}{\gamma}(z - c),$$

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 $\{\xi_j\}_{j=1}^m$ on the interval $i[-2, 2]$. In Fig. 1 we see an example of Newton interpolation at complex Leja points and conjugate complex Leja points, respectively, with no evident difference in the convergence rates. In the practical implementation of 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, quite a good *a posteriori* estimate e_m of the interpolation error is given by the difference of two successive approximations (see [7])

$$\exp(z) - p_{m-1}(z) \approx e_m = d_m r_{m-1}(z) \tag{2.2a}$$

for interpolation at Leja points, and

$$\exp(z) - p_{m-2}(z) \approx e_m = \text{Re}(d_{m-1})r_{m-2}(z) + d_m q_m(z) \tag{2.2b}$$

for interpolation at conjugate complex Leja points, respectively.

With the notion of the ε-pseudospectrum of A

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

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 the estimate of $\Lambda_\varepsilon(A)$ is usually much larger than $\Lambda_\varepsilon(A)$ itself.

In Fig. 2 a typical behavior for the interpolation of a matrix function can be observed. We display the *a posteriori* error estimate (norm of e_m in (2.2a)) for the evaluation of $\exp(\tau A)b$ in Example 1 below ($N = 100$, Pe $= 0.495$, $\tau = 5 \cdot 10^{-3}$). The

Fig. 2 Convergence rate (a posteriori error estimate) for a 2D advection-diffusion operator discretized by finite differences (see Example 1 below). The *upper* (*red*) *line* indicates the maximum error during the recursion. Therefore, numbers *below the lower* (*red*) *line* are not significant in `IEEE` double precision. (Color figure online)
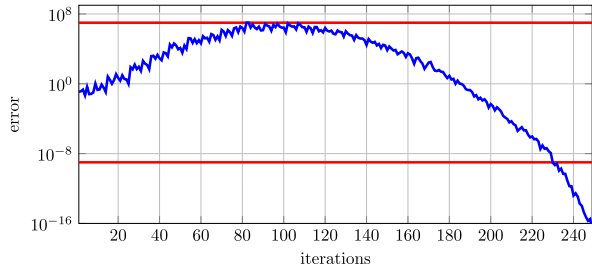
large hump is produced by the eventually (not monotonically) decreasing divided differences and the increasing magnitude of the matrix polynomial r_{m-1}. This behavior is referred to as the *hump effect* in the context of matrix functions, see [9, 11].

A second observation, made from Fig. 2, is that even if the terms e_m decrease eventually (this is true as the divided differences are computed as described in [5] and not in the standard way) they vary more than 16 orders of magnitude. Therefore, it is not possible to reach here 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 substepping procedure to approximate $\exp(A)b$ by

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

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

2.1 The new strategy for the Leja point method

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 cannot 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 interpolation points per step are known a priori, the interpolation can be optimized.

As mentioned above the Leja interpolation highly depends on the spectral bounds α, β for the matrix A, see (2.1). 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 be roughly the same as for interpolating $\exp(A)b$. 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 priori estimate of our unknown properties. There are some constraints, the most important one is the maximally allowed 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 too low, in general. Nevertheless by analyzing the promising results for normal matrices we were 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 independently for real and complex conjugate Leja points, we define the following procedure with the help of the upper and lower estimates.

(a) Compute the values α and β of the rectangle R enclosing $\sigma(A)$.
(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 Newton interpolation for each of the s inner steps. In each step the error is checked the first time after n iterations and then after every fifth iteration.

Henceforth this code will be called `Leja`. The error estimate is based on (2.2a) and (2.2b). It 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 our 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 is a large number of methods for computing the action of a matrix function on a given vector. However, many of these methods were never implemented or were implemented in other program languages than Matlab (such as FORTRAN or C). Some of them are even 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.

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

3.1 Krylov subspace methods

The computation of $\exp(A)b$ with a Krylov method basically consists in two steps. In the first step one determines an appropriate Krylov subspace, in the second step one computes the matrix exponential of a smaller matrix using standard methods.

The code `Expokit` (see [11]) is a program that computes the Krylov subspace by 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

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

For this basis and the Hessenberg matrix one gets the relation

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

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

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,

$$\exp(A)b = \frac{1}{2\pi i} \int_\Gamma e^\lambda (\lambda I - A)^{-1}b \, d\lambda$$

$$\approx \frac{1}{2\pi i} \int_\Gamma e^\lambda V_m(\lambda I - H_m)^{-1}e_1 \, d\lambda = V_m \exp(H_m)e_1,$$

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

The code uses the Matlab routine `expm` to compute $\exp(H_m)$. A crucial parameter is the maximally allowed dimension of the Krylov subspace. In `Expokit` this is set to $m_{\max} = 30$. If a higher dimension is required, `Expokit` uses 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 $b \in \mathbb{C}^n$ by

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

with T_m denoting the truncated Taylor series expansion of order m of the exponential function at zero. For a fixed integer m, the parameter s is chosen in such a way that $T_m(s^{-1}A)^s = \exp(A + \Delta A)$ with $\|\Delta A\| \leq \text{TOL}$, where ΔA is the backward error of the computation with respect to truncation errors (see [2]). Note that `expmv` is the only method in our comparison that bounds the backward error. The value of m is chosen such that the computational cost of the algorithm is minimized, taking into account that the evaluation of a too long Taylor expansion for matrices $s^{-1}A$ with large norm could lead to numerical instability. Therefore, it is required that $m \leq m_{\max} = 55$.

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

3.3 Chebychev methods

The idea of using a Chebychev polynomials for approximating a matrix function is quite established (see [1, 4]). Chebychev methods work efficiently if the matrix A is Hermitian or skew-Hermitian. The code Cheb by Güttel[1] 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 b is approximated by a Chebychev expansion with interpolation 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 can be used for solving ordinary differential equations (see, e.g. [1]) but is not considered in this paper.

4 Numerical comparisons

In this section we outline some differences between the 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 64 bit (glnxa64) Fedora 16 workstation with a 3 GHz Intel Core 2 vPro and 8 GB RAM. Matlab is restricted to a single computational thread on a single core by using the -singleCompThread command flag. In addition the JVM is deactivated with the -nojvm command flag. This configuration, which limits the computations to one CPU or processor core, allows us to make a fair comparison where the specific computer architecture has less influence. We are looking at these computations from the exponential integrator point of view. Therefore, we are typical interested in accuracies of 4 to 8 digits, say. However, we will also present a comparison where we prescribe 12 digits accuracy. We include this example to show that our conclusions presented below do not change for higher accuracy requirements. In all cases we used expmv from [2] with maximal precision to compute a reference solution.

In the following we briefly describe each of the examples and the configuration for the respective numerical experiment. In each of the test cases we compute $\exp(\tau A)b$ for a prescribed tolerance TOL. Each of the codes is only provided with τ, A, b 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.

The various codes treat the prescribed tolerance TOL in a different way. Expokit normalizes the vector b and uses TOL as a stopping criterion in the Arnoldi iteration. If the norm of $\exp(\tau A)b$ is small, the relative error is badly controlled (see, e.g., Fig. 4 below). The code expmv controls the backward error by requiring $\|\Delta A\| \leq \text{TOL}\|A\|$ for any given norm. Cheb relates TOL to the 1-norm of the last 20 Chebychev coefficients employed. Finally, Leja controls with TOL the relative error of the result (in any given norm).

---

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

Fig. 3 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 numbers Pe = 0, 0.1, 0.9 and $\tau = 10^{-2}$. The error is measured in the maximum norm for the 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
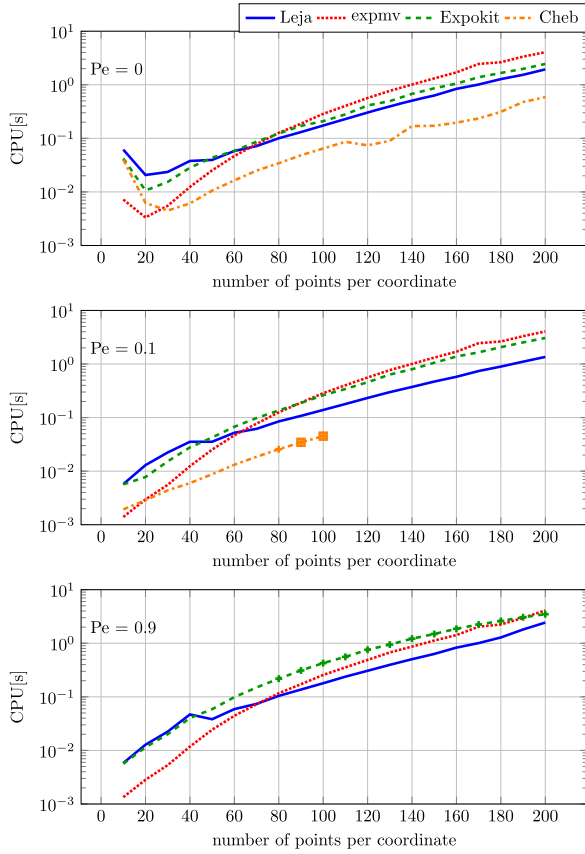
Example 1 (advection-diffusion equation) We start to investigate the behavior of the methods by an example that allows us easily to vary the spectral properties of the discretization matrix. We consider the advection-diffusion equation

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

on the domain $\Omega = [0, 1]^2$ with homogeneous Dirichlet boundary conditions. This problem is discretized in space 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 df $= N^2$ degrees of freedom. We define the grid Péclet number

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

as the ratio of advection to diffusion, scaled by Δx. By increasing Pe the nonnormality of the discretization matrix can be controlled. An appropriate measure for the nonnormality of a matrix A is the number

$$\kappa = \frac{\|AA^* - A^*A\|}{\|A\|^2}.$$

Fig. 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 1 for $\text{Pe} = 0.5$ and $\tau = 10^{-2}$. The error is measured in the maximum norm for the prescribed tolerance $\text{TOL} = 10^{-4}$. The + indicates that the achieved relative error is larger than TOL. The *gray line* indicates the prescribed tolerance
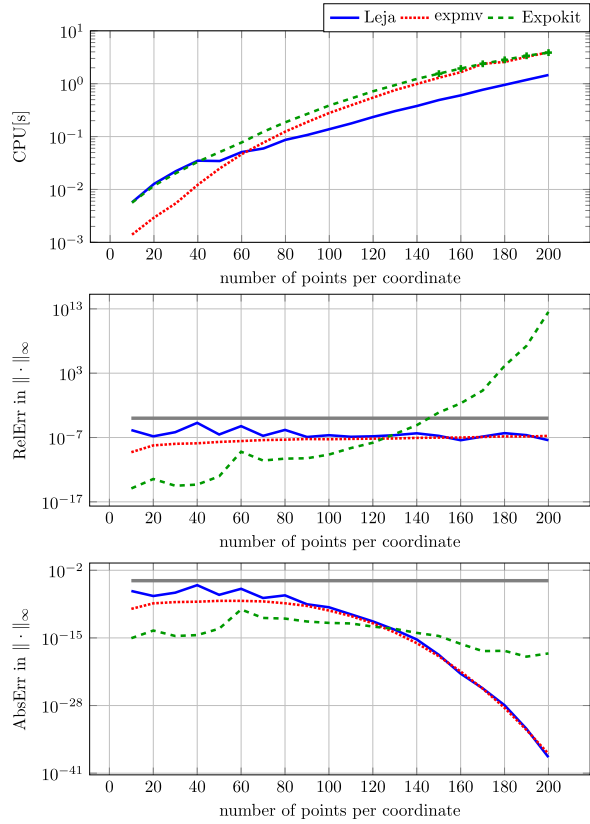

In our example, we found the relation $\text{Pe} = 8\kappa$ in the maximum norm. For the computations displayed in Figs. 3–6 the parameters are chosen as follows: $\varepsilon = 1$ and $c = \frac{2\varepsilon \text{Pe}}{\Delta x}$. As initial value we use the function $u_0(x, y) = 256 \cdot x^2(1 - x)^2 y^2(1 - y)^2$.

In Fig. 3 the matrix changes from normal ($\text{Pe} = 0$) to nonnormal ($\text{Pe} = 1$). In this range, our bound β in (2.1) changes accordingly from $\beta = 0$ to $\beta = -0.45\alpha$. The Chebyshev method is only working reliably 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 nonnormality of the input matrix. For `Expokit`, the computational cost slightly increases with increasing grid Péclet number. In Fig. 3 we use the maximum norm. We note that the discrete L^2 norm gives almost identical results.

In Fig. 4 we take a closer look for $\text{Pe} = 0.5$ 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` effectively only considers the absolute error, whereas `expmv` and `Leja` match the specified relative tolerance.

In our third test case for this example, see Fig. 5, we fix the dimension of A but vary the grid Péclet number between 0 and 1. The results are somewhat similar to those in Fig. 3. One observes again that the computational cost increases for higher

Fig. 5 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 the prescribed tolerance $TOL = 10^{-6}$. The + indicates that the achieved relative error is larger than TOL
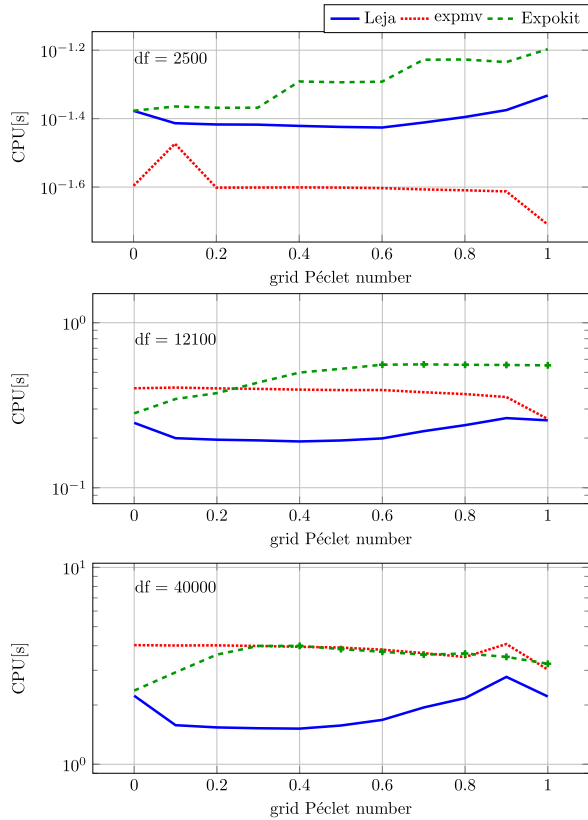
degrees of freedom but is almost constant for the methods `Leja`, `expmv` and `Expokit`, for varying Pe. These three methods also have about the same computational cost (varying by a factor of 3). The errors in this example are measured in a discrete L^2 norm but again the pictures stay almost the same for the maximum norm. In Fig. 6 we repeat the above experiment with $TOL = 10^{-12}$ to show that our conclusions stay the same for more stringent tolerances.

Throughout these experiments one could see that for small matrices the overhead produced by the accurate evaluation of the divided differences [5] in `Leja` is slowing down the method. For higher dimensions, however, the percentage of this (almost) fixed cost decreases in comparison to the overall cost and the method improves with respect to the other methods, see Fig. 3. The difference in CPU time between `expmv` and `Leja` can be explained by the different 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

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

Fig. 6 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 the prescribed tolerance TOL $= 10^{-12}$. The + indicates that the achieved relative error is larger than TOL
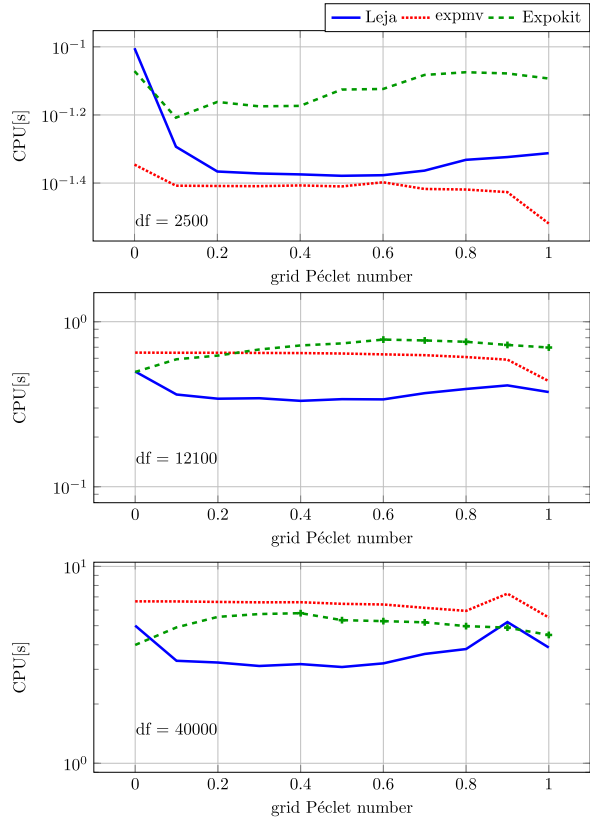

is used to model many applications in geo-engineering. A finite volume discretization with $N \times N$ points in the domain $\Omega = [0, 1]^2$ results in a sparse discretization matrix with a more complex structure than the one in the previous example. In particular, we are using the matrix L of Eq. (6) from [12, p. 3959].[2] For the computations displayed in Fig. 7 the diffusion parameters are set to $D_1 = 10^{-3}$, $D_2 = 10^{-4}$, and the dynamical viscosity is chosen $\mu = 1$. As initial value we use $u_0(x, y) = 256 \cdot x^2(1 - x)^2 y^2(1 - y)^2$.

In Fig. 7 we investigate the behavior of the considered methods for increasing matrix dimensions. In this experiment we prescribed a higher accuracy but due to the superlinear convergence this does not result in a considerably higher cost. The spectral properties of L are similar to that of A in Example 1 with 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`. In 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 include random variables. The experiments show, however, that all the methods are affected almost in the same way.

---

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

Fig. 7 Computational cost and achieved accuracy of various methods vs. number of points N per coordinate for the evaluation of $\exp(\tau L)u_0$ in Example 2 for $\tau = 10^{-1}$. The error is measured in a discrete L^2 norm for the prescribed tolerance $\text{TOL} = 10^{-6}$. The *gray line* indicates the prescribed tolerance
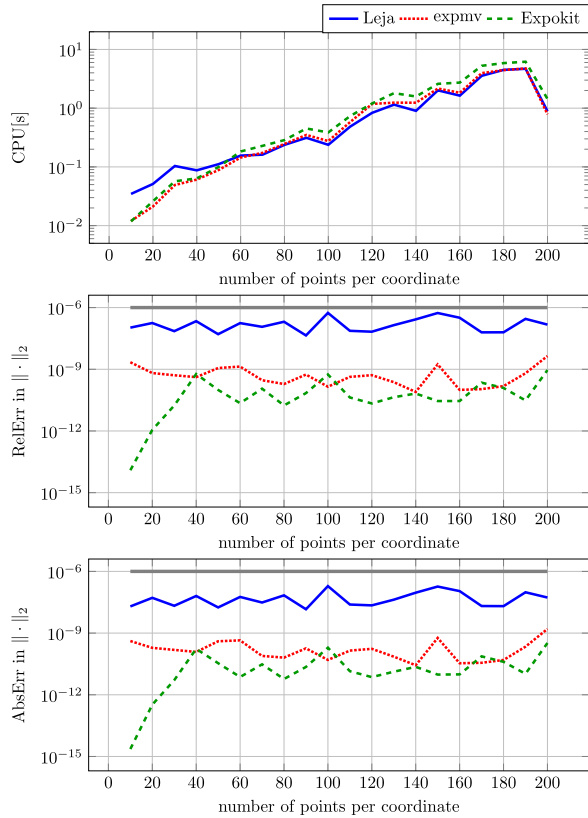
Example 3 (Schrödinger equation with harmonic potential in 3D) Our next example is the 3D Schrödinger equation

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

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. As described in Sect. 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 value 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 Fig. 8, `Leja`, `expmv` and `Expokit` show a nearly parallel linear growth of the 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 Sect. 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.

Fig. 8 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 the prescribed tolerance $\mathrm{TOL} = 10^{-6}$. The *gray line* indicates the prescribed tolerance
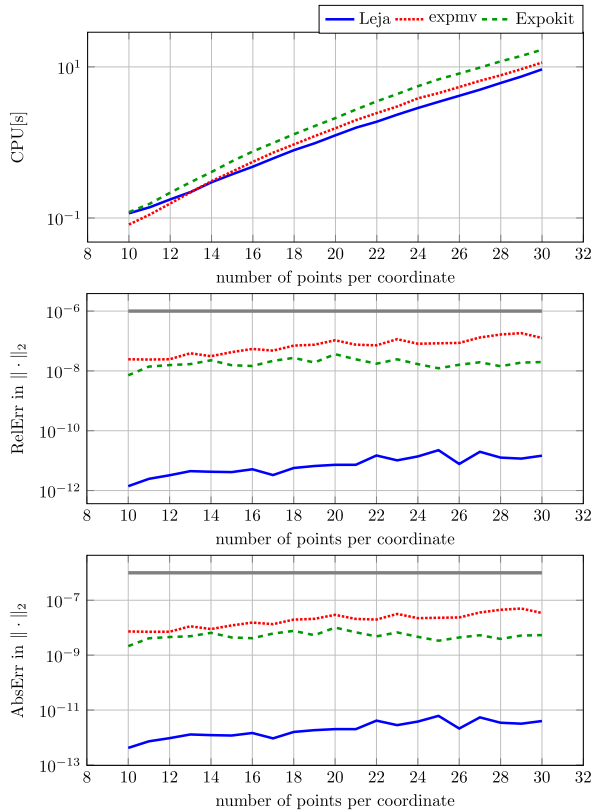
Example 4 (Molenkamp–Crowley in 2D with radial basis functions) In order to include an example with a dense matrix we consider the Molenkamp–Crowley equation

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

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 [7]. This kind of discretization results in a matrix where more than 90% of the entries are nonzero. As initial value we use

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

The results of this example are summarized in Fig. 9. 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 costs for Expokit have 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

Fig. 9 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 $\text{TOL} = 10^{-6}$. The *gray line* indicates the prescribed tolerance
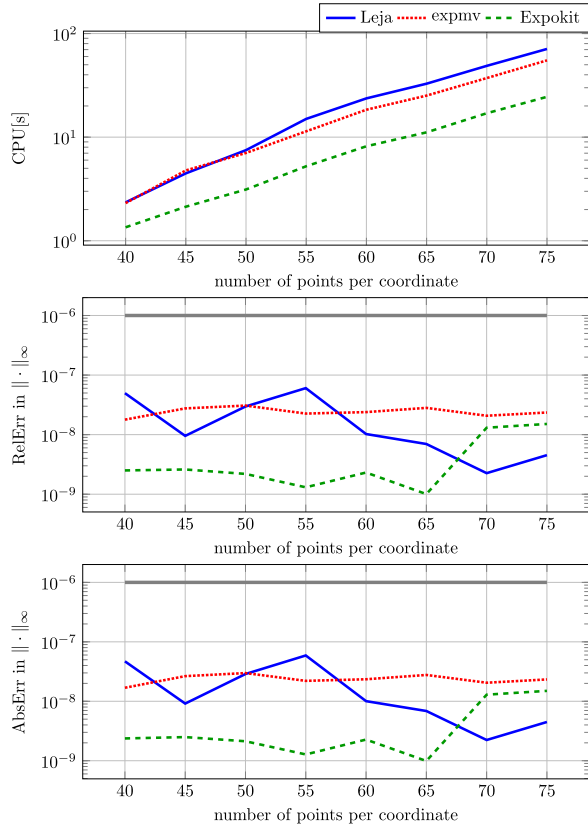
slightly more CPU time. Due to the storage limitations it is not feasible to include higher dimensional matrices in this experiment as it can not be guaranteed that Matlab is not starting to swap storage to the hard drive. The results in Fig. 9 are given for the maximum norm. The corresponding results in a discrete L^2 norm look very similar.

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 priori 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 latter gives satisfactory results only for nearly self-adjoint problems. For Krylov subspace methods, the maximal dimension of the subspace should not be too large. This 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 time

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 expmv and our interpolation at Leja points is the low demand of storage due to the employed short-term recursion.

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), 488–511 (2011)
3. Baglama, J., Calvetti, D., Reichel, L.: Fast Leja points. Electron. Trans. Numer. Anal. **7**, 124–140 (1998)
4. 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)
5. Caliari, M.: Accurate evaluation of divided differences for polynomial interpolation of exponential propagators. Computing **80**(2), 189–201 (2007)
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. Caliari, M., Ostermann, A., Rainer, S.: Meshfree exponential integrators. SIAM J. Sci. Comput. **35**(1), A431–A452 (2013)
8. Hochbruck, M., Ostermann, A.: Exponential integrators. Acta Numer. **19**, 209–286 (2010)
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. **24**(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**(10), 3957–3969 (2010)