

Comparing Leja and Krylov approximations of large scale matrix exponentials^{*}

L. Bergamaschi¹, M. Caliari², A. Martínez², and M. Vianello²

¹ Dept. of Math. Methods and Models, University of Padova, berga@dmsa.unipd.it

² Dept. of Pure and Appl. Math., University of Padova,
{acalomar,mcaliari,marcov}@math.unipd.it

Abstract. We have implemented a numerical code (ReLPM, Real Leja Points Method) for polynomial interpolation of the matrix exponential propagators $\exp(\Delta t A) \mathbf{v}$ and $\varphi(\Delta t A) \mathbf{v}$, $\varphi(z) = (\exp(z) - 1)/z$. The ReLPM code is tested and compared with Krylov-based routines, on large scale sparse matrices arising from the spatial discretization of 2D and 3D advection-diffusion equations.

1 Introduction

The systematic study and application of the so-called “exponential integrators” began about two decades ago, but has received a strong impulse in recent years; see, e.g., [5, 7, 8] and references therein. A building-block of exponential integrators is the efficient evaluation of the underlying matrix exponential functions, like $\exp(\Delta t A) \mathbf{v}$ and $\varphi(\Delta t A) \mathbf{v}$, $\varphi(z) = (\exp(z) - 1)/z$ (here $A \in \mathbf{R}^{n \times n}$, $\mathbf{v} \in \mathbf{R}^n$, and $\Delta t > 0$ is a time step). To this respect, most authors regard Krylov-like (cf. e.g. [7, 14]) as the methods of choice. Nevertheless, an alternative class of polynomial methods has been developed since the beginning (cf., e.g., [6, 15, 13]), which are based on direct interpolation or approximation of the exponential functions on the spectrum (or the field of values) of the relevant matrix. Despite of a preprocessing stage needed to get an estimate of some marginal eigenvalues, the latter are competitive with Krylov-like methods in several instances, namely on large scale, sparse and in general nonsymmetric matrices, arising from the spatial discretization of parabolic PDEs; see, e.g., [4, 10, 11].

Among others, the ReLPM (Real Leja Points Method), proposed in [4] and applied to advection-diffusion models in [2], has shown very attractive computational features. It rests on Newton interpolation of the exponential functions at a sequence of Leja points on the real focal interval of a family of confocal ellipses in the complex plane. The use of Leja points is suggested by the fact that they guarantee maximal (and thus superlinear) convergence of the interpolant on every ellipse of the confocal family, and thus superlinear convergence of the corresponding matrix polynomials to the matrix exponential functions.

^{*} Work supported by the MIUR PRIN 2003 project “Dynamical systems on matrix manifolds: numerical methods and applications” (co-ordinator L. Lopez, University of Bari), by the ex-60% funds of the University of Padova, and by the GNCS-INdAM.

This feature is shared also by other set of interpolation points, like e.g. standard Chebyshev points, but differently from the latter, at the same time Leja points allow to increase the interpolation degree just by adding new nodes of the same sequence; see [1, 4] for the scalar and matrix features of interpolation at Leja points. A key step in the approximation procedure is given by estimating cheaply a real focal interval, say $[a, b]$, such that the “minimal” ellipse of the confocal family which contains the spectrum (or the field of values) of the matrix is not too “large” (the underlying theoretical notion is that of “capacity” of a compact complex set). The numerical experience with matrices arising from stable spatial discretizations of parabolic equations (which are the main target of the ReLPM code) has shown that good results can be obtained at a very low cost, simply by intersecting the Gershgorin’s circles of the matrix with the real axis. Indeed, it is worth stressing that the ReLPM method works well with “stiff” matrices, whose spectrum (or whose field of values) has a projection on the real axis which is nonpositive and much larger than the projection on the imaginary axis; cf. [4].

We give now two formulas, which are the basis for practical implementation of the ReLPM method. The kernel of the ReLPM code is given by interpolation of $\varphi(h\lambda)$, for suitable $h \leq \Delta t$, at Leja points of the real focal interval $[a, b] = [c - 2\gamma, c + 2\gamma]$. Observe that once $\varphi(hA)\mathbf{v}$ is computed, then $\exp(hA)\mathbf{v} = h\varphi(hA)\mathbf{v} + \mathbf{v}$. In practice, it is numerically convenient to interpolate the function $\varphi(h(c + \gamma\xi))$ at Leja points $\{\xi_s\}$ of the reference interval $[-2, 2]$ (since it has capacity equal to 1, cf. [15]). Then, given the corresponding divided differences $\{d_i\}$ for such a function, the matrix Newton polynomial of degree m is

$$p_m(A) = \sum_{i=0}^m d_i \Omega_i \approx \varphi(hA), \quad \Omega_i = \prod_{s=0}^{i-1} ((A - cI)/\gamma - \xi_s I). \quad (1)$$

In general, it is not feasible to interpolate with the original time step Δt , which has to be fractionized. This happens, for example, when the expected degree for convergence is too large. The ReLPM code subdivides dynamically Δt into smaller substeps $h = h_k$, and recovers the required vector $\varphi(\Delta t A)\mathbf{v}$ according to the time marching scheme

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \varphi(h_k A)(A\mathbf{y}_k + \mathbf{v}), \quad k = 0, 1, \dots, k^*; \quad \mathbf{y}_0 = \mathbf{0}, \quad (2)$$

where $\sum h_k = \Delta t$. Here we use the fact that $\Delta t \varphi(\Delta t A)\mathbf{v}$ is the solution at $t = \Delta t$ of the differential system $\dot{\mathbf{y}}(t) = A\mathbf{y}(t) + \mathbf{v}$, $\mathbf{y}(0) = \mathbf{0}$.

2 The ReLPM code

In this section we present the pseudo-codes of the three subroutines which compose the ReLPM code. They are displayed in Tables 1–3, and accompanied by a detailed documentation.

Comments to Table 1. This is the main subroutine. It accepts a matrix $A \in \mathbf{R}^{n \times n}$, a vector $\mathbf{v} \in \mathbf{R}^n$, a time step $\Delta t > 0$, and the type of exponential

Table 1. SUBROUTINE RELPM

```

1. INPUT:  $A, \mathbf{v}, \Delta t, \text{exptype}, \text{tol}$ 
2. CONSTANTS:  $M = 124, (\xi_0, \dots, \xi_M)$  array of  $M + 1$  Leja points in  $[-2, 2]$ 
3.  $k := 0, \rho := \Delta t, \mathbf{p} := \mathbf{0}, \mathbf{w} := \mathbf{v}$ 
4.  $a, b :=$  “extrema of the real points in the Gersghorin’s circles of  $A$ ”
5.  $c := (a + b)/2, \gamma := (b - a)/4, \nu := 3\gamma, h := \min \{\Delta t, M/\nu\}, \text{oldh} := 0$ 
6. REPEAT
    7. IF  $h \neq \text{oldh}$  THEN
        8. CALL DIVDIFF( $h, c, \gamma, M, (\xi_0, \dots, \xi_M), (d_0, \dots, d_M)$ )
        9.  $\text{oldh} := h$ 
    10. ENDIF
    11. CALL INTERP( $A, \mathbf{w}, h, \text{tol}, c, \gamma, M, (\xi_0, \dots, \xi_M), (d_0, \dots, d_M), \mathbf{q}, \text{err}, m$ )
    12. IF  $m > M$  THEN  $h := h/2$ 
    13. ELSE
        14.  $\rho := \rho - h, \mathbf{p} := \mathbf{p} + h\mathbf{q}$ 
        15. IF  $\rho > 0$  THEN
            16.  $\mathbf{w} := A\mathbf{p}, \mathbf{w} := \mathbf{w} + \mathbf{v}, k := k + 1, \sigma := h\gamma/m$ 
            17. IF  $\sigma > 1$  THEN  $h := \min \{\sigma h, M/\gamma, \rho\}$ 
            18. ELSE  $h := \min \{h, \rho\}$ 
            19. ENDIF
        20. ENDIF
    21. ENDIF
22. UNTIL  $\rho = 0$ 
23.  $k^* := k$ , IF  $\text{exptype} = 0$  THEN  $\mathbf{w} := A\mathbf{p}, \mathbf{p} := \mathbf{w} + \mathbf{v}$  ELSE  $\mathbf{p} := \mathbf{p}/\Delta t$  ENDIF
24. OUTPUT: the vector  $\mathbf{p}$  such that  $\mathbf{p} \approx \exp(\Delta t A)\mathbf{v}$  ( $\text{exptype} = 0$ ), or  $\mathbf{p} \approx \varphi(\Delta t A)\mathbf{v}$ ,
     $\varphi(z) = (e^z - 1)/z$  ( $\text{exptype} = 1$ ); the total number of substeps  $k^*$ 

```

function (\exp or φ). The output is a vector which approximates the corresponding function of the matrix $\Delta t A$, applied to the vector \mathbf{v} . The underlying method is the time-marching scheme (2), with a dynamical managing of the variable substeps $h = h_k$, and Newton interpolation as in (1) of $\varphi(hA)$ at real Leja points related to spectral estimates for A .

- 1.) *exptype*: type of exponential function (0 for \exp , 1 for φ); *tol*: the relative error tolerated for the result $\exp(\Delta t A)\mathbf{v}$ or $\varphi(\Delta t A)\mathbf{v}$.
- 2.) M : maximum interpolation degree allowed “a priori” in the Newton interpolation of $\varphi(hA)$, $h \leq \Delta t$; (ξ_0, \dots, ξ_M) is an array of Leja interpolation points, like e.g. the Fast Leja points in [1]. It is worth stressing that the default $M = 124$ is tuned on spatial discretization matrices of linear advection-diffusion models, in order to guarantee that all the divided differences computed by the subroutine DIVDIFF are accurate (see [3]).
- 3.) Initializations: ρ is the portion of Δt still to be covered; $\mathbf{p} = \mathbf{y}_0$ and $\mathbf{w} = A\mathbf{y}_0 + \mathbf{v}$, cf. (2).
4. - 5.) Approximation of the real focal interval of a “minimal” ellipse which

contains the numerical range of the underlying matrix, and the associated parameters: c is the center and γ the capacity (length/4) of the interval. Hereafter, h and $oldh$ are the current and the previous (sub)steps. For any given substep h , theoretical estimates show that superlinear convergence of the matrix interpolation polynomial should start at a degree between $h\gamma$ and $2h\gamma$, cf. [10, 4], provided that the capacity of the minimal ellipse above is relatively close to γ . Convergence at reasonable tolerances of the matrix interpolation polynomial is expected for a degree lower than $h\nu = 3h\gamma$ (the factor 3 is an “empirical” choice, based on numerical experience). Hence, the input step Δt is possibly reduced in such a way that $[h\nu] \leq M$ (here $[\cdot]$ denotes the integer part).

6. - 22.) Main loop: implements the time marching scheme (2) with dynamical managing of the substeps $h = h_k$.

7. - 10.) When the current and the previous substeps are different the divided differences (d_0, \dots, d_M) are (re)computed.

11.) Computes $\mathbf{q} = p_m(A)\mathbf{w} \approx \varphi(hA)\mathbf{w}$, where $\mathbf{w} = A\mathbf{p} + \mathbf{v}$, $\mathbf{p} \approx \mathbf{y}_k$.

12. - 21.) If the interpolation process has not converged, the substep is halved and control returns to point 6, otherwise the current substep has been successful.

14.) The remaining portion ρ of Δt and \mathbf{p} are updated: now $\mathbf{p} \approx \mathbf{y}_{k+1}$.

15. - 20.) If Δt has not been completed, prepares the next substep.

16.) Computes $\mathbf{w} \approx A\mathbf{y}_{k+1} + \mathbf{v}$; σ is a parameter used to detect fast convergence, i.e. convergence degree smaller than $h\gamma$.

17. - 19.) When the actual convergence degree m is smaller than $h\gamma$ (fast convergence), the next substep h is increased but in such a way that $h\gamma \leq M$ (since convergence is expected again at a degree lower than $h\gamma$).

23.) Now $\mathbf{p} \approx \mathbf{y}_{k^*} = \Delta t \varphi(\Delta t A)\mathbf{v}$: computes the right type of matrix exponential function according to *exptype*.

Table 2. SUBROUTINE DIVDIFF

-
1. INPUT: $h, c, \gamma, M, (\xi_0, \dots, \xi_M)$
 2. “computes (d_0, \dots, d_M) , the divided differences of $\varphi(h(c + \gamma\xi))$, $\xi \in [-2, 2]$, at the Leja points (ξ_0, \dots, ξ_M) , by the accurate matrix algorithm in [3]”
 3. OUTPUT: (d_0, \dots, d_M)
-

Comments to Table 2. This subroutine accepts a time step h , two parameters related to the spectral features of an external matrix A , a maximum interpolation degree M and a corresponding array of Leja interpolation points. It returns the divided differences for the Newton polynomial interpolation of $\varphi(hA)$ up to degree M . The subroutine is thought to work in double precision.

1.) h : time step; c and γ : see point 4 in the comments to the subroutine RELPM; M and (ξ_0, \dots, ξ_M) : maximum interpolation degree and corresponding Leja in-

terpolation points, see the comment to point 2 of the subroutine RELPM.

2.) Computes the $M+1$ divided differences in double precision as the first column of $\varphi(h(c + \gamma \Xi_M))$, where Ξ_M is the $(M+1) \times (M+1)$ bidiagonal matrix with the Leja points (ξ_0, \dots, ξ_M) on the main diagonal and $(1, \dots, 1)$ on the diagonal immediately below. The matrix $\varphi(h(c + \gamma \Xi_M))$ is approximated via 16-term Taylor expansions by the scheme proposed in [3], on the basis of [9]. Differently from the standard divided differences table, this algorithm computes accurately the divided differences even when their size goes below machine precision, provided that M is not too large for the given h , c and γ (to avoid the underflow of some intermediate quantities in the computation); see the comment to point 2 of the subroutine RELPM. A more sophisticated implementation could discard the possible tail of divided differences that are not sufficiently accurate (see [3]).

3.) Given (d_0, \dots, d_M) , the Newton interpolation polynomial of degree $m \leq M$ for $\varphi(h\lambda)$ is $p_m(\lambda) = \sum_{i=0}^m d_i \prod_{s=0}^{i-1} ((\lambda - c)/\gamma - \xi_s)$, $\lambda \in [c - 2\gamma, c + 2\gamma]$.

Table 3. SUBROUTINE INTERP

1. INPUT: A , \mathbf{w} , h , tol , c , γ , M , (ξ_0, \dots, ξ_M) , (d_0, \dots, d_M)

2. CONSTANTS: $\ell = 5$

3. $\mathbf{u} := \mathbf{w}$, $\mathbf{q} := d_0 \mathbf{w}$, $e_0 := \|\mathbf{q}\|_2$, $\beta := \|\mathbf{w}\|_2$, $m := 0$

4. REPEAT

5. $\mathbf{z} := (A\mathbf{u})/\gamma$, $\mathbf{u} := \mathbf{z} - (c/\gamma + \xi_m)\mathbf{u}$

6. $m := m + 1$, $e_m := |d_m| \|\mathbf{u}\|_2$

7. $\mathbf{q} := \mathbf{q} + d_m \mathbf{u}$

8. IF $m \geq \ell - 1$ THEN $err := (e_m + \dots + e_{m-\ell})/\ell$ ENDIF

9. UNTIL $err \leq \beta tol$ or $m \geq M$

10. IF $err > \beta tol$ THEN $m := M + 1$ ENDIF

11. OUTPUT: the vector $\mathbf{q} = p_m(A)\mathbf{w}$, the estimated error err , and the interpolation degree m , such that $\|\mathbf{q} - \varphi(hA)\mathbf{w}\|_2 \approx err$, with $err \leq \|\mathbf{w}\|_2 tol$ when $m \leq M$, whereas a convergence failure occurred when $m > M$.

Comments to Table 3. This subroutine tries to compute an approximation $\mathbf{q} = p_m(A)\mathbf{w} \approx \varphi(hA)\mathbf{w}$, cf. (1), up to an error (relative to $\|\mathbf{w}\|_2$) less than a given tolerance, where $A \in \mathbf{R}^{n \times n}$, $\mathbf{w} \in \mathbf{R}^n$ and $\Delta t \geq h > 0$.

- 1.) (d_0, \dots, d_M) are the divided differences for the Newton interpolation up to degree M of the function $\varphi(h(c + \gamma\xi))$ at the Leja points $(\xi_0, \dots, \xi_M) \subset [-2, 2]$.
- 2.) ℓ : number of consecutive error estimates to be averaged in order to filter error oscillations (the default $\ell = 5$ has shown a good numerical behavior).
- 3.) Initializations: $\mathbf{u} = \Omega_0 \mathbf{w}$, $\mathbf{q} = p_0(A)\mathbf{w}$ (cf. (1)).
4. - 9.) Main loop: Newton interpolation of the matrix operator $\varphi(hA)\mathbf{w}$.
- 5.) Computes the vector $((A - cI)/\gamma - \xi_m I)\mathbf{u} = A\mathbf{u}/\gamma - (c/\gamma + \xi_m)\mathbf{u} = \Omega_{m+1}\mathbf{w}$, avoiding to shift and scale the matrix.

6. - 7.) Computes e_m , the norm of the new term in the Newton interpolation, and the vector $\mathbf{q} = p_m(A)\mathbf{w}$.
- 8.) Averages the last ℓ values of e_i to get the actual interpolation error estimate.
- 9.) Exits the loop as soon as the estimated error is below the relative tolerance, or the maximum interpolation degree M has been reached.
- 10.) Sets the output degree m to a value $> M$ in case that convergence has not been attained.

3 Numerical tests and comparisons

In this section we present some numerical examples, where we have tested the ReLPM code on large scale matrices arising from spatial discretizations of 2D and 3D advection-diffusion equations. The model problem is

$$\frac{\partial u}{\partial t} = \operatorname{div}(D\nabla u) - \langle \vec{v}, \nabla u \rangle, \quad x \in \Omega, \quad t > 0, \quad (3)$$

with initial condition $u(x, t) = u_0(x)$, $x \in \Omega$, and mixed Dirichlet and Neumann boundary conditions on $\Gamma_D \cup \Gamma_N = \partial\Omega$, $\Omega \subset \mathbf{R}^d$, $d = 2, 3$. In (3), D is a $d \times d$ diffusion matrix, and $\vec{v} \in \mathbf{R}^d$ a constant velocity field. Finite Difference (FD) or Finite Elements (FE) discretizations produce a system of ODEs like $\dot{\mathbf{y}}(t) = A\mathbf{y}(t) + \mathbf{b}$, $\mathbf{y}(0) = \mathbf{u}_0$, where A is large, sparse and nonsymmetric. In the FE case it is obtained cheaply by left-multiplying the stiffness matrix with the inverse of a diagonal mass matrix, via the mass-lumping technique (cf. [2]).

In all the examples we have computed $\varphi(\Delta t A)\mathbf{v}$ with $\mathbf{v} = (1, \dots, 1)^t$ (corresponding to $u_0 \equiv 1$), for two values $(\Delta t)_1$ and $(\Delta t)_2$ of the time step Δt , depending on the specific matrix. The tests have been performed in double precision by a Fortran version of the ReLPM code, on an IBM Power5 processor with 1.8Gb of RAM. We also give the comparisons with the PHIPRO Fortran code by Y. Saad (again in double precision), which is based on Krylov subspace approximations (cf. [12]). The results are collected in Table 4. We report, for both methods, the number of substeps (steps), the number of total iterations (i.e., of matrix-vector products), and the CPU time. In addition, for PHIPRO we show also the chosen dimension for the Krylov subspace. This is a delicate choice, for which a simple criterion does not seem to be available; in any case, the default $m = 60$ is not suitable in the examples. The tolerances for both methods have been tuned in order to have an error, relative to the “exact” result of the exponential operator, of about 10^{-6} in the 2-norm (which is compatible with the underlying spatial discretization error). It is worth observing that even using smaller tolerances the performances of both methods would not change significantly, since superlinear convergence has already been reached.

Example 1 (FE-2D). We have taken a 2D equation like (3), with $\Omega = (0, 1)^2$, $D = I$, $\vec{v} = (60, 60)$, and homogeneous Dirichlet boundary conditions. We have adopted a standard Galerkin FE discretization with linear basis functions, on a uniform mesh with $n = 490\,000$ nodes and 977 202 triangular elements, which

produces a matrix with 3 424 402 nonzeros (average nonzeros per row ≈ 7). Here $(\Delta t)_1 = 0.001$ and $(\Delta t)_2 = 0.01$.

Example 2 (FE-3D). Here we have a 3D advection-dispersion equation, where D is the hydrodynamic dispersion tensor, $D_{ij} = \alpha_T |\vec{v}| \delta_{ij} + (\alpha_L - \alpha_T) v_i v_j / |\vec{v}|$, $1 \leq i, j \leq d$ (α_L and α_T being the longitudinal and transverse dispersivity, respectively). The domain is $\Omega = [0, 1] \times [0.0.5] \times [0, 1]$, discretized by a regular mesh of $n = 161 \times 81 \times 41 = 524\,681$ nodes and 3 072 000 tetrahedral elements. The boundary conditions are homogeneous Dirichlet on $\Gamma_{\mathcal{D}} = \{0\} \times [0.2, 0.3] \times [0, 1]$, whereas homogeneous Neumann conditions are imposed on $\Gamma_{\mathcal{N}} = \partial\Omega \setminus \Gamma_{\mathcal{D}}$. The velocity is $\vec{v} = (1, 0, 0)$, the transmissivity coefficients are piecewise constant and vary by an order of magnitude depending on the elevation of the domain, $\alpha_L(x_3) = \alpha_T(x_3) \in \{0.0025, 0.025\}$. The resulting FE matrix has 7 837 641 nonzeros (average per row ≈ 14). Here $(\Delta t)_1 = 1$ and $(\Delta t)_2 = 10$.

Example 3 (FD-2D). Again a 2D model, with $\Omega = (0, 10)^2$, $D = I$, $\vec{v} = (100, 100)$, and homogeneous Dirichlet boundary conditions. We have adopted a second order central FD discretization on a uniform grid with stepsize 0.01 ($n = 1\,002\,001$ nodes), generating a pentadiagonal matrix with 5 006 001 nonzeros. Here $(\Delta t)_1 = 0.01$ and $(\Delta t)_2 = 0.1$.

Example 4 (FD-3D). Here a 3D equation, with $\Omega = (0, 1)^3$, $D = I$, $\vec{v} = (200, 200, 200)$, and homogeneous Dirichlet boundary conditions. Here we have adopted a second order central FD discretization on a uniform grid with stepsize 0.005 ($n = 8\,120\,601$ nodes), generating an eptadiagonal matrix with 56 601 801 nonzeros. Here $(\Delta t)_1 = 10^{-3}$ and $(\Delta t)_2 = 5.2 \times 10^{-3}$.

Comments on the results. First, notice that RELPM performs better than PHIPRO in all the tests, even with an optimal choice of the Krylov subspace dimension (boxed CPU times), in spite of a smaller number of total Krylov iterations. Indeed, it is worth stressing that RELPM computes only 1 matrix-vector product, 2 daxpys, 1 vector scaling and 1 scalar product per iteration inside INTERP. Moreover, it allocates only the matrix and 6 vectors, whereas PHIPRO has to allocate, besides the matrix, all the m Krylov subspace generators and 4 vectors (neglecting a matrix and some vectors of dimension m). In addition, when m increases, PHIPRO decreases the total number of iterations, but is penalized by the long-term recurrence in the orthogonalization process. For small m , it is penalized by a larger number of substeps. The difference in storage requirements produces remarkable consequences in the last example. There, the matrix is extremely large, and PHIPRO can work only with $m \leq 10$ due to the memory limitations (1.8Gb), becoming about 2.5 times slower than RELPM.

References

1. Baglama, J., Calvetti, D., Reichel, L.: Fast Leja points. *Electron. Trans. Numer. Anal.* **7** (1998) 124–140
2. Bergamaschi, L., Caliari, M., Vianello, M.: The ReLPM exponential integrator for FE discretizations of advection-diffusion equations. Springer LNCS vol. 3039 - part IV, 2004 (Proc. of ICCS 2004) 434–442

Table 4. Comparing RELPM and PHIPRO on the advection-diffusion discretization matrices in Examples 1–4 (the CPU times are in seconds).

| Δt | Code | FE-2D | | | FE-3D | | | FD-2D | | | FD-3D | | |
|----------------|----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|--------|
| | PHIPRO | steps | iter | CPU | steps | iter | CPU | steps | iter | CPU | steps | iter | CPU |
| $(\Delta t)_1$ | $m = 10$ | 126 | 1386 | 57.1 | 81 | 891 | 59.8 | 54 | 594 | 44.2 | 35 | 385 | 349.3 |
| | $m = 20$ | 40 | 840 | 45.5 | 28 | 588 | 50.2 | 21 | 441 | 43.1 | † | † | † |
| | $m = 25$ | 29 | 754 | 43.2 | 21 | 546 | 45.2 | 16 | 416 | 45.6 | † | † | † |
| | $m = 30$ | 23 | 713 | 45.4 | 17 | 527 | 48.1 | 13 | 403 | 55.4 | † | † | † |
| | $m = 50$ | 13 | 663 | 58.4 | 10 | 510 | 58.9 | 8 | 408 | 67.5 | † | † | † |
| | RELPM | 17 | 857 | 27.0 | 12 | 585 | 32.0 | 5 | 392 | 20.6 | 3 | 234 | 133.0 |
| $(\Delta t)_2$ | PHIPRO | steps | iter | CPU | steps | iter | CPU | steps | iter | CPU | steps | iter | CPU |
| | $m = 10$ | 868 | 9548 | 414.3 | 428 | 4708 | 302.1 | 431 | 4741 | 375.4 | 158 | 1738 | 1593.2 |
| | $m = 20$ | 287 | 6027 | 318.8 | 152 | 3192 | 242.1 | 166 | 3486 | 323.4 | † | † | † |
| | $m = 25$ | 190 | 4940 | 280.9 | 117 | 3042 | 250.4 | 126 | 3276 | 353.2 | † | † | † |
| | $m = 30$ | 149 | 4619 | 306.6 | 94 | 2914 | 268.0 | 103 | 3193 | 415.6 | † | † | † |
| | $m = 50$ | 73 | 3723 | 343.3 | 53 | 2703 | 301.0 | 58 | 2958 | 516.6 | † | † | † |
| RELPM | 131 | 7720 | 227.3 | 74 | 4335 | 231.7 | 49 | 3617 | 186.3 | 16 | 1094 | 633.4 | |

3. Caliari, M.: Accurate evaluation of divided differences for polynomial interpolation of exponential propagators. Draft (2005)
4. Caliari, M., Vianello, M., Bergamaschi, L.: Interpolating discrete advection-diffusion propagators at Leja sequences. *J. Comput. Appl. Math.* **172** (2004) 79–99
5. Cox, S. M., Matthews, P. C.: Exponential time differencing for stiff systems. *J. Comput. Phys.* **176** (2002) 430–455
6. Druskin, V. L., Knizhnerman, L. A.: Two polynomial methods for calculating functions of symmetric matrices. *U.S.S.R. Comput. Math. and Math. Phys.* **29** (1989), 112–121
7. Hochbruck, M., Lubich, C., Selhofer, H.: Exponential integrators for large systems of differential equations. *SIAM J. Sci. Comput.* **19** (1998) 1552–1574
8. Hochbruck, M., Ostermann, A.: Exponential Runge-Kutta methods for parabolic problems. *Appl. Numer. Math.* **53** (2005) 323–339
9. McCurdy, A., Ng, C., Parlett, B. N.: Accurate Computation of Divided Differences of the Exponential Function. *Math. Comp.* **43** (1984) 501–528
10. Moret, I., Novati, P.: The computation of functions of matrices by truncated Faber series. *Numer. Funct. Anal. Optim.* **22** (2001) 697–719
11. Novati, P.: A polynomial method based on Feiér points for the computation of functions of unsymmetric matrices. *Appl. Numer. Math.* **44** (2003) 201–224
12. Saad, Y.: SPARSKIT: a basic tool kit for sparse matrix computations. Dept. of Computer Science and Engineering, University of Minnesota. Version 2 (2005)
13. Schaefer, M. J.: A polynomial based iterative method for linear parabolic equations. *J. Comput. Appl. Math.* **29** (1990) 35–50
14. Sidje, R. B.: Expokit. A software package for computing matrix exponentials. *ACM Trans. Math. Software* **24** (1998) 130–156
15. Tal-Ezer, H.: Polynomial approximation of functions of matrices and applications. *J. Sci. Comput.* **4** (1989), 25–60