Doctoral School in Mathematics

Backward error accurate methods for computing the matrix exponential and its action

Franco Zivcovich

UNIVERSITÀ DEGLI STUDI DI TRENTO
Dipartimento di Matematica

UNIVERSITÀ DEGLI STUDI DI VERONA
Dipartimento di Informatica

Thesis presented for the degree of
Doctor of Philosophy

Supervisor: Prof. Marco Caliari

2020

Doctoral thesis in Mathematics.
Joint doctoral programme in Mathematics, **XXXII cycle**.

Department of Mathematics,
University of Trento.

Department of Computer Science,
University of Verona.

Supervisor: **Prof. Marco Caliari**.

Preface

The theory of partial differential equations constitutes today one of the most important topics of scientific understanding for they can easily model a great number of problems. In the 50s, the arrival of computers allowed for the first time to calculate quantities that before could only be estimated very approximately. This offered to researchers and engineers the possibility to use the numerical results for the modification or adaptation of their scientific arguments and experiments.

In this context, the so-called *exponential integrators*, a class of numerical methods for the time integration of differential equations, were first developed. Since then, research on exponential integrators made important steps ahead and new, refined exponential integrators have been developed. Among others, we can mention exponential Runge–Kutta, exponential Rosenbrock, or Magnus integrators.

An aspect in common to every exponential integrator is that their efficient implementation depends on how quickly and accurately the action of a matrix exponential on a vector is approximated. Such a feature places exponential integrators at the junction between the fields of scientific computing and numerical linear algebra. While this duality makes research on exponential integrators eclectic and open to scientists with different backgrounds, it also constitutes a source of inefficiencies.

In fact, despite several algorithms have been perfected for computing the action of the matrix exponential, less than a handful are actually employed when it comes to solving problems originated in the real applications. On the one hand, this is because numerical linear algebraists sometimes develop very involved and elegant algorithms that in turn poorly fit the computations typical of the applications. On the other hand, engineers, but even researchers in the field of scientific computing, are usually reluctant to adopt new and complex methods for computing the matrix functions underlying their methods, often blaming their own side of the implementation when inefficiencies arise.

The goal I set myself, maybe naïvely, when I first started my thesis work, was to build routines that are sophisticated and elegant but that also perform well on those problems coming from the applications. While I may have lost my resolve on elegance, I made sure I could claim that I developed routines that perform well on real problems. For this reason, every chapter of my thesis is accompanied by rigorous, fair and extensive tests.

After a long and slightly technical introduction to the core problem motivating my research, I will outline the structure of my thesis.

Then, in Chapter 2, I will thoroughly analyze the relationship existing between the divided differences of an analytic function and the interpolation set. This analysis leads to novel representations of the divided differences that play a crucial role in the successive phases of this thesis. The main result of this chapter is a routine, `dd_phi`, for the fast and accurate computation of the divided differences of the exponential and closely related

functions.

I will then proceed, in Chapter 3, to study the problem of computing the matrix exponential. The reason that initially pushed me to study this problem is the high precision computation of the exponential of the Hessenberg matrices arising from the Krylov methods. However, the resulting routine, called `expkptotf`, stands out even when compared with routines expressly designed for computations in standard double precision arithmetic.

In Chapter 4, I will treat the problem of computing the action of the matrix exponential on a vector. The links with the other chapters are surprisingly many, showing how strongly interconnected are the various aspects of the problem of computing the matrix exponential. As a result I will present two routines, `explhe` and `pkryexp`, based on polynomial interpolations of the exponential function, which elicit in innovative ways information on the spectrum of the input matrices and fine-tune the approximation parameters accordingly.

Finally, I will present a brief chapter of conclusion, where I resume the experience I collected writing these pages and I outline the future work.

Acknowledgements

It is my pleasure to express my gratitude to my supervisor, Marco Caliari, for his guidance, his scientific contribution to my doctoral project, and his precious feedback on my work. I really appreciated the freedom to pursue, without pressure, any research topic I found interesting. Part of this work was carried out while I was visiting other institutions: I thank Nicholas J. Higham, Frédéric Hecht, Jorge Sastre, Javier Ibáñez and Emilio Defez for their great hospitality and for many fruitful discussions. I am very grateful to Fabio Cassini for reading the preliminary draft of this thesis and providing much useful feedback that improved the quality and coherence of the presentation. I also wish to thank all my current and former colleagues for the scientific and personal exchange, and for providing happy distractions from work.

Contents

1 Introduction **11**

1.1 Methods for computing the action of the matrix exponential 13

1.1.1 Truncated Taylor series 14

1.1.2 Polynomial interpolation 15

1.1.3 Krylov method 17

1.2 Thesis outline 22

2 Computing the divided differences of analytic functions **25**

2.1 Introduction 25

2.2 The change of basis operator 26

2.2.1 Factorization of the change of basis operator 29

2.3 The divided differences of the exponential and closely related functions . . 34

2.3.1 Numerical experiments 39

2.4 Conclusions 43

3 Computing the matrix exponential **49**

3.1 Introduction 49

3.2 Approximation technique and polynomial evaluation scheme 52

3.2.1 Polynomial Evaluation Scheme 53

3.3 Analysis of the backward error 55

3.3.1 Maximum approximation degree 58

3.3.2 Krylov subspace projection of the backward error 60

3.3.3 Accuracy check: on-the-fly backward error estimate 62

3.4 Numerical experiments 65

3.4.1 Tests in double precision arithmetic 67

3.4.2 Tests in multiple precision arithmetic 69

3.5 Conclusions 70

4 Computing the action of the matrix exponential **76**
4.1 Introduction 76
4.1.1 On the reordering of an interpolation set 79
4.2 Matrices with skinny field of values 80
4.2.1 Contour integral approximation of the backward error matrix 82
4.2.2 Final selection of the polynomial interpolation 89
4.3 Extended Ritz's values interpolation 91
4.3.1 Backward error analysis 92
4.3.2 Final selection of the polynomial interpolation 99
4.4 Numerical Experiments 100
4.4.1 Tests over the test set $\mathcal{A}$ 101
4.4.2 Tests over the test set $\mathcal{S}$ 104
4.5 Conclusions 111

5 Conclusions and future work **113**

Chapter 1

Introduction

In nature there exist many examples of a quantity $u(t)$ which continuously grows or decays over time by an amount proportional to $u(t)$, with a continuous rate given by the scalar a. Such an $u(t)$ obeys the differential equation

$$\dot{u}(t) = au(t),$$

which shows an exponential solution. Here the dot denotes the derivative with respect to t. In fact, provided that $u_0 = u(t_0)$, by using the technique of separation of the variables we have that the solution exists and it is uniquely determined by

$$u(t) = \mathrm{e}^{(t-t_0)a}u_0.$$

More in general, we can consider a system of differential equations of the type

$$\dot{u}(t) = \mathbf{A}u(t) \tag{1.1}$$

where now $u(t)$ is a vector function, $u : \mathbb{R} \to \mathbb{C}^N$ and $\mathbf{A}$ is the square matrix of size N representing a constant bounded linear operator acting over $\mathbb{C}^N$. Here $\mathbf{A}$ may be the Jacobian of a certain function or an approximation of it, and it can be large and sparse. By sparse we mean that the number of non-zero elements in $\mathbf{A}$ is a multiple of the size N rather than of the number N^2 of the total entries of $\mathbf{A}$. The exponential of the matrix $\mathbf{A}$ can be defined in many ways, one of those is by means of the exponential series

$$\mathrm{e}^{\mathbf{A}} := \sum_{k=0}^{\infty} \frac{\mathbf{A}^k}{k!}.$$

If we differentiate term by term the exponential series of $\mathrm{e}^{(t-t_0)\mathbf{A}}$ with respect to t we find that

$$\frac{d}{dt}\mathrm{e}^{(t-t_0)\mathbf{A}} = \mathbf{A}\mathrm{e}^{(t-t_0)\mathbf{A}}.$$

It is therefore evident that the system of differential equations (1.1) admits too the exponential solution

$$u(t) = \mathrm{e}^{(t-t_0)\mathbf{A}}u_0.$$

Suppose now that we want to solve the slightly more complicated system of differential equations

$$\dot{u}(t) = \mathbf{A}u(t) + b(t, u(t))$$

The solution in this case is given by the corresponding Volterra integral equation

$$u(t) = \mathrm{e}^{(t-t_0)\mathbf{A}}u_0 + \int_{t_0}^{t} \mathrm{e}^{(t-s)\mathbf{A}}b(s, u(s))ds \tag{1.2}$$

that is also called Lagrange's *variation-of-constants formula*. If the function b is constant with respect to t we can pull it out of the integral sign. Then, we can easily find an analytic expression for the solution:

$$u(t) = \mathrm{e}^{(t-t_0)\mathbf{A}}u_0 + (t - t_0)\varphi_1((t - t_0)\mathbf{A})b$$

where φ_1 is defined as

$$\varphi_1(\mathbf{A}) := \sum_{k=0}^{\infty} \frac{\mathbf{A}^k}{(k+1)!}.$$

If instead the function b is not constant with respect to t but it is analytic, we can expand it in a Taylor series about t_0 and write the solution as

$$u(t) = \mathrm{e}^{(t-t_0)\mathbf{A}}u_0 + \sum_{\ell=1}^{\infty}(t - t_0)^{\ell}\varphi_\ell((t - t_0)\mathbf{A})b_\ell, \tag{1.3}$$

where

$$b_\ell := \frac{d^{\ell-1}}{dt^{\ell-1}}b(t, u(t))|_{t=t_0}$$

and

$$\varphi_\ell(\mathbf{A}) := \sum_{k=0}^{\infty} \frac{\mathbf{A}^k}{(k+\ell)!}.$$

Clearly, if there exists a q such that b_q is the zero function we have that the truncation to degree q of the series on the right hand side of (1.3) leads to an exact solution. Otherwise this series gives the approximation

$$\hat{u}(t) = \mathrm{e}^{(t-t_0)\mathbf{A}}u_0 + \sum_{\ell=1}^{q}(t - t_0)^{\ell}\varphi_\ell((t - t_0)\mathbf{A})b_\ell \tag{1.4}$$

of the analytic solution (1.3). A wide class of exponential integrator methods is obtained by employing suitable approximations to the vectors b_ℓ in (1.4), and more methods can be obtained by differently approximating the integral in (1.2) (see [30]).

The problem of efficiently evaluate the approximation $\hat{u}(t)$ of $u(t)$ for a certain order q can be reduced to the computation of just a single matrix exponential of a singular, slightly larger matrix $\tilde{\mathbf{A}}$. It was in fact shown in [2, Section 2] that if we set

$$\tilde{\mathbf{A}} := \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ 0 & \mathbf{J} \end{pmatrix}, \quad \mathbf{J} := \begin{pmatrix} 0 & \mathbf{I}_{q-1} \\ 0 & 0 \end{pmatrix},$$

where $\mathbf{B}$ is the matrix whose columns are $b_q, b_{q-1}, \ldots, b_1$ and $\mathbf{I}_{q-1}$ is the identity matrix of size $q-1$, then we have that the desired approximation can be formed as

$$\hat{u}(t) = \begin{pmatrix} \mathbf{I}_N & 0 \end{pmatrix} \mathrm{e}^{(t-t_0)\tilde{\mathbf{A}}} \begin{pmatrix} u_0 \\ e_q \end{pmatrix}$$

where e_q is the qth column of $\mathbf{I}_q$.

Exponential integrators, sometimes used in combination with splitting methods (for a detailed dissertation on splitting methods the reader could refer to [40]), play a key role in the field of the applications due to their effectiveness when applied to stiff or highly oscillatory problems. Therefore, we can see today exponential integrators applied to quantum dynamics, chemistry, mathematical finance and many other branches of the field of applications. For a detailed survey on the exponential integrators and their applications see Hochbruck and Ostermann [30].

1.1 Methods for computing the action of the matrix exponential

The efficiency of this class of methods strongly depends on the fast and accurate approximation of the action of the matrix exponential on a vector. This task, that can be reduced without loss of generality to the task of computing

$$\mathrm{e}^{\mathbf{A}} v,$$

where $\mathbf{A}$ is a complex-valued N sized square matrix and v is a vector of compatible dimension, constitutes the central topic of this manuscript.

Which approach is best to compute $\mathrm{e}^{\mathbf{A}} v$ is not clear. Generally, since $\mathbf{A}$ may be large and sparse while its exponential is usually dense, it is not convenient to form $\mathrm{e}^{\mathbf{A}}$ and then multiply it into v. On the contrary it is usually preferable to compute *tout court* the vector $\mathrm{e}^{\mathbf{A}} v$.

To do so, recall that the minimal polynomial of a vector v is the non-zero monic polynomial p of lowest degree such that $p(\mathbf{A})v = 0$. The degree ν of the minimal polynomial of v with respect to $\mathbf{A}$ is often called the grade of v with respect to $\mathbf{A}$ (see [53, Section 6.2]). A consequence of the Cayley–Hamilton theorem is that the grade of v does not exceed N. Therefore, we can form $\mathrm{e}^{\mathbf{A}} v$ as a linear combination of vectors from

$$\{v, \mathbf{A}v, \mathbf{A}^2 v, \ldots, \mathbf{A}^{\nu-1} v\}.$$

Of course, not every vector from such a basis is cheaply available to us since ν could be of same order N of the matrix while we surely wish to perform fewer matrix-vector products. Once again thanks to the Cayley–Hamilton theorem, we have that for a suitable γ the vector $(\mathbf{I} + \gamma\mathbf{A})^{-1} v$ is a polynomial in $\mathbf{A}$ of degree $\nu - 1$ applied to v. Therefore we know we can rapidly invade $\mathbb{C}^\nu$ considering rational approximations to $\mathrm{e}^{\mathbf{A}} v$.

As a consequence, a great deal of routines based on rational approximations techniques have been designed. Among others, we can mention the Padé approximation usually employed for approximating the matrix $\mathrm{e}^{\mathbf{A}}$, [1, 18], the massively parallel in time REXI

method (see [63]), the best rational L_∞ approximation (see [67]), and all those methods that are based on rational Krylov approximations (see for more details [14, 22, 41, 43, 49]).

The main drawback of this approach is that rational methods require to solve linear systems. In fact, if it is not a problem to solve linear systems involving $\mathbf{I}+\gamma\mathbf{A}$, then other well-established implicit methods are available for solving stiff differential equations. If, on the other hand, the matrix is large and/or ill-conditioned, suitable preconditioners and iterative methods for solving linear systems have to be used. But such methods lie their foundations on sequences of matrix-vector products. Therefore at this point, one finds out that the vectors $\mathbf{A}^k v$ for high powers k may be neglected anyway making sometimes the effort of solving linear systems to be unjustified.

In this work, we do not consider rational approximations to $\mathrm{e}^{\mathbf{A}}v$, we focus on methods which do not require to solve linear systems involving the matrix $\mathbf{A}$.

1.1.1 Truncated Taylor series

We prefer to adopt methods that form the approximation of $\mathrm{e}^{\mathbf{A}}v$ combining vectors from

$$\{v, \mathbf{A}v, \mathbf{A}^2v, \ldots, \mathbf{A}^{m-1}v\} \tag{1.5}$$

which, provided without loss of generality that $m < \nu$, is a basis spanning K_m, called *Krylov subspace* of dimension m. The simplest method is constituted by the truncated Taylor series, i.e. the approximation of $\mathrm{e}^{\mathbf{A}}v$ by

$$T_{m-1}(\mathbf{A})v := \sum_{k=0}^{m-1} \frac{\mathbf{A}^k}{k!}v$$

coupled with a sub-stepping strategy. The sub-stepping strategy consists in determining s positive scalars $\tau_1, \tau_2, \ldots, \tau_s$ such that

$$\sum_{l=0}^{s-1} \tau_{l+1} = 1$$

so that we can set $v^{(0)} := v$, then march as

$$v^{(l+1)} := T_{m-1}(\tau_{l+1}\mathbf{A})v^{(l)}, \quad l = 0, 1, \ldots, s-1$$

and recover the desired approximation $v^{(s)}$. In order to shift the eigenvalues of $\mathbf{A}$ to a more favorable location for $T_{m-1}(x)$, we consider to work with the shifted version of the input matrix

$$\mathbf{B} := \mathbf{A} - \mu\mathbf{I},$$

instead of $\mathbf{A}$, for some scalar μ. Popular choices of μ all aim to drag the spectrum in a neighborhood of the origin. Due to numerical stability reasons, if the real part of μ is positive we recover the desired approximation as $\mathrm{e}^{\mu}v^{(s)}$, otherwise we multiply $\mathrm{e}^{\tau_{l+1}\mu}$ into $v^{(l+1)}$ at each sub-step l.

This method is also equipped with an early termination criterion. Suppose that at

sub-step l we encounter a positive integer i smaller than $m-1$ such that

$$\left\| \frac{\tau_{l+1}\mathbf{B}^{i-1}}{(i-1)!}v \right\|_\infty + \left\| \frac{\tau_{l+1}\mathbf{B}^{i}}{i!}v \right\|_\infty$$

is not larger than

$$\mathrm{tol} \cdot \left\| \sum_{k=0}^{i} \frac{\tau_{l+1}\mathbf{B}^{k}}{k!}v \right\|_\infty,$$

for a certain tolerance tol prescribed by the user. Then we stop the computations, i.e. we set

$$v^{(l+1)} := \sum_{k=0}^{i} \frac{\tau_{l+1}\mathbf{B}^{k}}{k!}v^{(l)}$$

and we proceed to the successive sub-steps.

To the best of our knowledge, the most prominent implementation of this technique was recently developed in the manuscript [2].

The greatest vulnerability of the truncated Taylor series technique is caused by the so-called *hump phenomenon*, that is the possibility that the norm of the partial sums $T_i(\mathbf{B})v$, with $i < m$, grows very large before it converges to $\left\|\mathrm{e}^{\mathbf{B}}v\right\|$, that in turn could be small. Bearing in mind that the truncated Taylor series is equivalent to an interpolation in the Hermite sense of the exponential function at the origin, it was shown in [6] that the hump phenomenon is likely to happen when the interpolation points lie far from the eigenvalues of the matrix $\mathbf{B}$. In finite precision arithmetic, such phenomenon kicks in destructively due to severe cancellation errors.

1.1.2 Polynomial interpolation

In order to avert the onset of the hump phenomenon, we take into account a slightly more involved class of methods: the polynomial interpolation in Newton form. If we consider the interpolation sequence $\sigma_{m-1} := (z_0, z_1, \ldots, z_{m-1})$ and the Newton polynomials defined as

$$\pi_{k,\sigma_{m-1}}(x) := \prod_{j=0}^{k-1}(x - z_j),$$

we can rewrite the basis (1.5) as

$$\{v, \pi_{1,\sigma_{m-1}}(\mathbf{A})v, \pi_{2,\sigma_{m-1}}(\mathbf{A})v, \ldots, \pi_{m-1,\sigma_{m-1}}(\mathbf{A})v\} \tag{1.6}$$

that spans over the same Krylov subspace K_m. This basis is called Newton basis. The polynomial methods in Newton form approximate $\mathrm{e}^{\mathbf{A}}v$ by

$$p_{m-1}(\mathbf{A})v := \sum_{k=0}^{m-1} d[z_0, z_1, \ldots, z_k]\pi_{k,\sigma_{m-1}}(\mathbf{A})v,$$

where $d[z_0], d[z_0, z_1], \ldots, d[z_0, z_1, \ldots, z_{m-1}]$ are the divided differences of the exponential function at the interpolation sequence. These methods are coupled with a sub-stepping

strategy too, so that we can set $v^{(0)} := v$, then march as

$$v^{(l+1)} := p_{m-1}(\tau_{l+1}\mathbf{A})v^{(l)}, \quad l = 0, 1, \ldots, s-1$$

and recover the desired approximation $v^{(s)}$. In order to shift the eigenvalues of $\mathbf{A}$ to a more favorable location for $p_{m-1}(x)$, we consider to work with the shifted version of the input matrix

$$\mathbf{B} := \mathbf{A} - \mu\mathbf{I},$$

instead of $\mathbf{A}$, for some scalar μ. Popular choices of μ all aim to drag the spectrum in a neighborhood of the origin. Due to numerical stability reasons, if the real part of μ is positive we recover the desired approximation as $\mathrm{e}^{\mu}v^{(s)}$, otherwise we multiply $\mathrm{e}^{\tau_{l+1}\mu}$ into $v^{(l+1)}$ at each sub-step l.

This class of methods is also equipped with an early termination criterion. Suppose that at sub-step l we encounter a positive integer i smaller than $m-1$ such that

$$\left\| d[z_0, z_1, \ldots, z_{i-1}] \prod_{j=0}^{i-2} (\tau_{l+1}\mathbf{B} - z_j\mathbf{I})v \right\|_\infty + \left\| d[z_0, z_1, \ldots, z_i] \prod_{j=0}^{i-1} (\tau_{l+1}\mathbf{B} - z_j\mathbf{I})v \right\|_\infty$$

is not larger than

$$\mathrm{tol} \cdot \left\| \sum_{k=0}^{i} d[z_0, z_1, \ldots, z_k] \prod_{j=0}^{k-1} (\tau_{l+1}\mathbf{B} - z_j\mathbf{I})v \right\|_\infty .$$

Then we stop the computations, i.e. we set

$$v^{(l+1)} := \sum_{k=0}^{i} d[z_0, z_1, \ldots, z_k] \prod_{j=0}^{k-1} (\tau_{l+1}\mathbf{B} - z_j\mathbf{I})v^{(l)}$$

and we proceed to the successive sub-step.

Several polynomial interpolation methods were proposed in the literature. We mention among others [6, 7, 8, 36, 44, 66]. A slightly different class of methods is constituted by the polynomial approximations that are non-interpolatory, such as [4, 12, 45, 62].

If the interpolation points lie close (or even coincide) with the eigenvalues of largest magnitude, the hump phenomenon is greatly mitigated. This is because the norm of the vectors forming the basis in (1.6) have a reduced norm with respect to those forming the basis in (1.5). It is a basic linear algebra fact that the ith eigenvalue of $\pi_{k,\sigma_{m-1}}(\mathbf{B})$ equals $\pi_{k,\sigma_{m-1}}(\lambda_i)$, where $\lambda_1, \lambda_2, \ldots, \lambda_N$ are the eigenvalues of $\mathbf{B}$. To be persuaded of this, consider to write $\pi_{k,\sigma_{m-1}}(\mathbf{B})$ as an explicit polynomial in $\mathbf{B}$

$$\pi_{k,\sigma_{m-1}}(\mathbf{B}) = \mathbf{B}^k - (\sum_{j=0}^{k} z_j)\mathbf{B}^{k-1} + \ldots + (-1)^k \prod_{j=0}^{k} z_j\mathbf{I}$$

and to obtain the eigenvalues of $\pi_{k,\sigma_{m-1}}(\mathbf{B})$ as the algebraic sum of the eigenvalues of the monomials in $\mathbf{B}$ that form $\pi_{k,\sigma_{m-1}}(\mathbf{B})$.

Let us clarify this concept with the aid of an example: suppose we have to compute

the exponential of the matrix

$$\bar{\mathbf{B}} = \begin{pmatrix} -100 & 0 \\ 0 & \mathbf{B} \end{pmatrix}$$

and that the matrix $\mathbf{B}$ has eigenvalues in a unit circle centered at the origin and henceforth it should not represent a source of humping problems. Ignoring for a moment the sub-stepping strategy, with a truncated Taylor method it would take k to reach at least the value 100 before the norm of the first entry of $T_k(\bar{\mathbf{B}})v$ starts converging to its actual, very small, value $\mathrm{e}^{-100}e_1^T v$, where e_1 is the first column of $\mathbf{I}_N$. On the other hand, using the Hermite interpolation at $(-100, 0, \ldots, 0)$ we immediately "switch off" the dangerous component of $\bar{\mathbf{B}}$ burdening the algorithm for computing the divided differences of the task of accurately computing $d[-100]$ and the remaining divided differences.

On the convenient ordering of the interpolation sequence

Clearly, the mere presence of -100 among the interpolation points does not grant that the hump phenomenon will not kick in. We should in fact also make sure that -100 appears early on in the interpolation sequence. Therefore, given an interpolation set made out of points in $\mathbb{C}$ lying close to the eigenvalues of $\mathbf{A}$, a good ordering plays a crucial role. This is the reason why we refer to interpolation sequences rather than interpolation sets. Consider the set $P = \{x_0, x_1, \ldots, x_k\}$ of k distinct interpolation nodes such that $m_j + 1$ is the multiplicity of the node x_j and suppose $m_0 + m_1 + \ldots + m_k + k + 1 = m$. In [50] Reichel suggests an ordering of P, called *Leja ordering*, that returns an interpolation sequence $(z_0, z_1, \ldots, z_{m-1})$. For a fixed initial point $y_0 \in P$, one recursively chooses

$$y_{i+1} \in \arg\max_{x \in P} \prod_{j=0}^{i} |x - y_j|^{m_j+1}, \quad i = 0, 1, \ldots, k-1,$$

the ordered sequence of interpolation points is given by the selected nodes y_i repeated according to their multiplicity in P

$$(z_0, z_1, \ldots, z_{m-1}) = (\underbrace{y_0, \ldots, y_0}_{m_0+1}, \underbrace{y_1, \ldots, y_1}_{m_1+1}, \ldots, \underbrace{y_k, \ldots, y_k}_{m_k+1}).$$

When $z_0, z_1, \ldots, z_i$ are all distinct, since at step i the interpolation error is proportional to $\pi_{i,\sigma_i}(x)$, the procedure above selects at each step the point $z_{i+1} \in P$ where the interpolation error is likely to be the largest, forcing the interpolation error to be now 0 at z_{i+1}. This should greedily reduce the interpolation error. When instead $z_0, z_1, \ldots, z_i$ are not distinct, this reordering algorithm slightly diverges from this idea. In Section 4 we propose a new ordering algorithm that handles rigorously this case.

1.1.3 Krylov method

On the other hand, if the interpolation sequence is not carefully selected, the destructive hump phenomenon could even be enhanced. This is likely to be the case when the user does not have any effective information about the spectrum of $\mathbf{A}$.

Therefore, alternative ways for avoiding the hump phenomenon were designed. The most straightforward and intuitive solution consists in searching for a proper orthonormal

basis

$$\{v_1, v_2, \ldots, v_m\} \tag{1.7}$$

of the Krylov subspace K_m. While one could simply apply the Gram–Schmidt orthogonalization process to the basis in (1.5) (or equivalently (1.6)), it is less straightforward but more efficient to build this basis iteratively. To do so, as first step set $v_1 = v/\|v\|_2$. Suppose that $\{v_1, v_2, \ldots, v_j\}$ is an orthonormal basis of the Krylov subspace of dimension j, we obtain v_{j+1} normalizing the vector

$$r_j := \mathbf{A}v_j - \sum_{i=1}^{j} v_i(v_i^*\mathbf{A}v_j).$$

Since v_{j+1} and r_j are aligned, we have

$$v_{j+1}^* r_j = \|r_j\| = v_{j+1}^*\mathbf{A}v_j$$

and therefore, if we define

$$h_{i,j} := v_i^*\mathbf{A}v_j,$$

we can rewrite the relation above as

$$\mathbf{A}v_j = \sum_{i=1}^{j+1} v_i h_{i,j}.$$

Set now $\mathbf{V}_m$ to be the matrix whose jth column is v_j, clearly $\mathbf{V}_m^*\mathbf{V}_m$ is the m sized identity matrix but, more importantly, we can write

$$\mathbf{A}\mathbf{V}_m = \mathbf{V}_m\mathbf{H}_m + h_{m+1,m}v_{m+1}e_m^T, \tag{1.8}$$

where e_m is the mth column of $\mathbf{I}_m$ and $\mathbf{H}_m$, the projection of the matrix $\mathbf{A}$ over the Krylov subspace of dimension m, is the upper Hessenberg square matrix of size m whose (i, j) entry is $h_{i,j}$. The structure of this factorization is highlighted in Figure 1.1. In

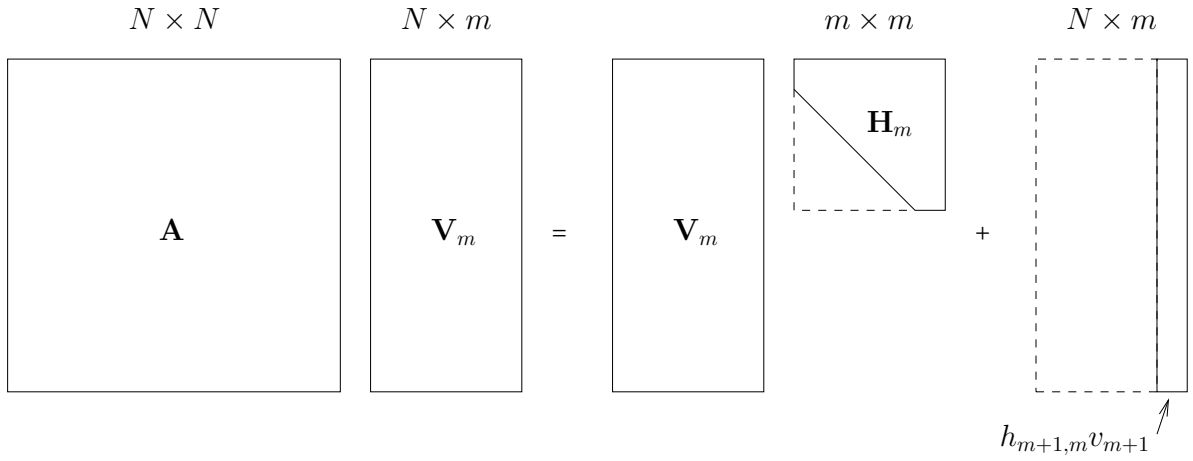


Figure 1.1: Arnoldi decomposition structure.

case the matrix $\mathbf{A}$ turns out to be Hermitian or skew-Hermitian, it is possible to speed

up the Arnoldi process by employing a simpler algorithm, called Lanczos process. In fact we know from (1.8) that if the matrix $\mathbf{A}$ is (skew-)Hermitian then the matrix $\mathbf{H}_m$ is (skew-)Hermitian too

$$\mathbf{H}_m^* = (\mathbf{V}_m^* \mathbf{A} \mathbf{V}_m)^* = \mathbf{V}_m^* \mathbf{A}^* \mathbf{V}_m = \mathbf{V}_m^* \mathbf{A} \mathbf{V}_m = \mathbf{H}_m,$$

and in particular tridiagonal since $\mathbf{H}_m$ shows in the first place a Hessenberg structure. Therefore, in case $\mathbf{A}$ is (skew-)Hermitian we can avoid to compute $h_{i,j}$ for those indexes i such that $|i-j| > 1$. This is particularly important because, while in the Arnoldi process the cost of an additional step gets higher as the number of iteration increases, the cost of an additional step of Lanczos is independent from the number of iterations previously performed. In fact the cost of Arnoldi process is quadratic in m, while the cost associated to Lanczos process is linear in m.

Now we report verbatim a simple routine that can be used to perform the Arnoldi or Lanczos process given $\mathbf{A}$, v and the parameter `skewness` that it is equal to 1 if $\mathbf{A}$ is Hermitian, to -1 if $\mathbf{A}$ is skew-Hermitian while it is 0 otherwise.

```
 1. function [ V, H, n2, hm1 ] = my_krylov( A, v, m, skewness )
 2. % If skewness == 0 perform m steps of the ARNOLDI process on A and v.
 3. % For m < N produce V s.t. cjt( V ) * V = eye( m ) and H Hessenberg s.t.
 4. %    A * V( :,1:m ) = V( :,1:m ) * H + hm1 * V( :,m+1 ) * e_m',
 5. % where e_m is the m-th column of eye( m ).
 6. % If skewness == 1 ( or skewness == -1 ) then H is ( skew )Hermitian
 7. % and the steps are the equivalent but cheaper LANCZOS.
 8.  n2 = norm( v ); V = [ v / n2, zeros( size( v,1 ),m ) ];
 9.  H = zeros( m+1 );
10.  for j = 1:m
11.     z = A * V( :,j );
12.     for k = j:-1:max( ( j - 1 ) * abs( skewness ),1 )
13.       H( k,j ) = ctranspose( V( :,k ) ) * z;
14.       z = z - H( k,j ) * V( :,k );
15.     end
16.     H( j+1,j ) = norm( z );
17.     if ( H( j+1,j ) == 0 ) % 'happy breakdown'
18.       break,
19.     end
20.     V( :,j+1 ) = z / H( j+1,j );
21.   end
22.   hm1 = H( j+1,j ); H = H( 1:j,1:j ); V = V( :,1:j+1 );
23.   if ( skewness )
24.     H( j+1:j+1:power( j,2 ) ) = skewness * H( 2:j+1:power( j,2 ) );
25.     if ( skewness == -1 )
26.       H( 1:j+1:power( j,2 ) ) = 1i * imag( H( 1:j+1:power( j,2 ) ) );
27.     end
28.   end
29. end
```

Lines `17.` to `19.` contain the so-called "happy breakdown" of the Arnoldi process that

allows to early terminate the Krylov projection.

The desired approximation of $\mathrm{e}^{\mathbf{A}}v$, called *Krylov approximation*, is then obtained by forming

$$k_m(\mathbf{A}, v) := \|v\|_2 \, \mathbf{V}_m \mathrm{e}^{\mathbf{H}_m} e_1,$$

coupled with a sub-stepping strategy, so that we can set $v^{(0)} := v$, then march as

$$v^{(l+1)} := k_m(\tau_{l+1}\mathbf{A}, v^{(l)}), \quad l = 0, 1, \ldots, s-1$$

and recover the desired approximation $v^{(s)}$. In order to shift the eigenvalues of $\mathbf{A}$ to a more favorable location for the Krylov approximant, we consider to work with the shifted version of the input matrix

$$\mathbf{B} := \mathbf{A} - \mu\mathbf{I},$$

instead of $\mathbf{A}$, for some scalar μ. Popular choices of μ all aim to drag the spectrum in a neighborhood of the origin. Due to numerical stability reasons, if the real part of μ is positive we recover the desired approximation as $\mathrm{e}^{\mu}v^{(s)}$, otherwise we multiply $\mathrm{e}^{\tau_{l+1}\mu}$ into $v^{(l+1)}$ at each sub-step l.

We mention among others the manuscripts on Krylov approximation of the matrix exponential [27, 28, 46, 52].

Krylov method is a polynomial method

In the following, we report some results, taken from [52], which show that the Krylov method is indeed a polynomial method .

Lemma 1. *Let $\mathbf{A}$ be any matrix while $\mathbf{V}_m$ and $\mathbf{H}_m$ are the results of m steps of the Arnoldi (or Lanczos) method applied to $\mathbf{A}$. Then for any polynomial $p_j(x)$ of degree $j \leq m-1$ the following equality holds:*

$$p_j(\mathbf{A})v_1 = \mathbf{V}_m p_j(\mathbf{H}_m)e_1.$$

Proof. The goal is to prove by induction that

$$\mathbf{A}^j v_1 = \mathbf{V}_m \mathbf{H}_m^j e_1$$

which is clearly true for $j = 0$. Let $\mathbf{W}_m = \mathbf{V}_m\mathbf{V}_m^*$ be the orthogonal projector onto K_m as represented in the original basis. Assume that is true for $j \leq m-2$. Since the vectors $\mathbf{A}^j v_1$ and $\mathbf{A}^{j+1}v_1$ belong to K_m we have

$$\mathbf{A}^{j+1}v_1 = \mathbf{W}_m\mathbf{A}^{j+1}v_1 = \mathbf{W}_m\mathbf{A}\mathbf{A}^j v_1 = \mathbf{W}_m\mathbf{A}\mathbf{W}_m\mathbf{A}^j v_1.$$

The relation $\mathbf{H}_m = \mathbf{V}_m\mathbf{A}\mathbf{V}_m^*$ yields $\mathbf{W}_m\mathbf{A}\mathbf{W}_m = \mathbf{V}_m^*\mathbf{H}_m\mathbf{V}_m$. Using the induction hypothesis, we get

$$\mathbf{A}^{j+1}v_1 = \mathbf{V}_m\mathbf{H}_m\mathbf{V}_m^*\mathbf{V}_m\mathbf{H}_m^j v_1 = \mathbf{V}_m\mathbf{H}_m^{j+1}v_1$$

proving the Lemma. □

Now let $\psi_\eta(x)$ be the minimal polynomial of $\mathbf{A}$, where η is the degree of $\psi_\eta(x)$. We know, as a consequence of the Cayley–Hamilton theorem, that any power of $\mathbf{A}$ can be

expressed in terms of a polynomial in $\mathbf{A}$ of degree at most $\nu - 1$. Let us recall that $\eta \geq \nu$ where ν is the grade of v in $\mathbf{A}$. An immediate consequence is that if $f(x)$ is an entire function then $f(\mathbf{A}) = p_{\eta-1}(\mathbf{A})$ for a certain polynomial of degree at most $\eta - 1$. The next lemma characterizes such polynomial.

Lemma 2. *Let $\mathbf{A}$ be any matrix whose minimal polynomial $\psi_\eta(x)$ is of degree η and $f(x)$ a function in the complex plane which is analytic in an open set containing the spectrum of $\mathbf{A}$. Moreover, let $p_{\eta-1}(x)$ be the interpolating polynomial of the function $f(x)$, in the Hermite sense, at the roots of the minimal polynomial of $\mathbf{A}$, repeated according to their multiplicities. Then*

$$f(\mathbf{A}) = p_{\eta-1}(\mathbf{A}).$$

Proof. See [20] for a proof. □

We observe that it is not necessary to work with polynomials of the same degree of the minimal polynomial of $\mathbf{A}$. Suppose $q(x)$ is any polynomial interpolating $f(x)$, in the Hermite sense, at the roots of $\psi_\eta(x)$ and at some other arbitrary interpolation set. Clearly, $q(x)$ complies with the assumptions of the previous lemma for it is analytic over the whole complex plane. Therefore we have that $q(\mathbf{A}) = p_{\eta-1}(\mathbf{A})$, thus $f(\mathbf{A}) = q(\mathbf{A})$.

Therefore, going back to the Hessenberg matrix provided by the Arnoldi process, we can state that

$$\mathrm{e}^{\mathbf{H}_m} = p_{m-1}(\mathbf{H}_m)$$

where the polynomial p_{m-1} interpolates the exponential function at the set of the eigenvalues $\{\rho_1, \rho_2, \dots \rho_m\}$ of $\mathbf{H}_m$, called *Ritz's values*, in the Hermite sense. The set of Ritz's values, in fact, contains for sure the roots of the minimal polynomial of $\mathbf{H}_m$ taken with their multiplicities. We are now ready to state the main characterization theorem.

Theorem 1. *The approximation $k_m(\mathbf{A}, v)$ is mathematically equivalent to approximating $\mathrm{e}^{\mathbf{A}}v$ by $p_{m-1}(\mathbf{A})v$, where $p_{m-1}(\mathbf{A})v$ is the (unique) polynomial of degree $m-1$ which interpolates the exponential function in the Hermite sense on the set $\{\rho_1, \rho_2, \dots, \rho_m\}$ of Ritz's values.*

Proof. Using Lemma 2 we have

$$\|v\|_2 \, \mathbf{V}_m \mathrm{e}^{\mathbf{H}_m} e_1 = \|v\|_2 \, \mathbf{V}_m p_{m-1}(\mathbf{H}_m) e_1$$

where p_{m-1} is the polynomial defined in the statement of the theorem. Using Lemma 1, this becomes

$$\|v\|_2 \, \mathbf{V}_m \mathrm{e}^{\mathbf{H}_m} e_1 = \|v\|_2 \, p_{m-1}(\mathbf{A}) v_1 = p_{m-1}(\mathbf{A}) v.$$

□

Therefore the Krylov method coincides with a polynomial interpolation at the Ritz's values. The question is: are the Ritz's values any good for avoiding the onset of the hump phenomenon? The answer is positive: assume that $(\rho_i^{(m)}, s_i^{(m)})$ is an eigenpair of $\mathbf{H}_m$ where we temporarily added the apex m for sake of clarity. Then we have

$$\mathbf{A}\mathbf{V}_m s_i^{(m)} = \mathbf{V}_m \mathbf{H}_m s_i^{(m)} + h_{m+1,m} v_{m+1} e_m^T s_i^{(m)} = \rho_i^{(m)} \mathbf{V}_m s_i^{(m)} + h_{m+1,m} v_{m+1} e_m^T s_i^{(m)}$$

and clearly

$$\mathbf{A}\mathbf{V}_m s_i^{(m)} - \rho_i^{(m)} \mathbf{V}_m s_i^{(m)} = h_{m+1,m} v_{m+1} e_m^T s_i^{(m)}$$

from which, applying the triangular inequality, we get that

$$\left\| \mathbf{A}(\mathbf{V}_m s_i^{(m)}) - \rho_i^{(m)} (\mathbf{V}_m s_i^{(m)}) \right\|_2 = h_{m+1,m} |e_m^T s_i^{(m)}|.$$

Therefore we know that the Ritz's values approximate the eigenvalues of $\mathbf{A}$, generally starting from the external one, more and more precisely as m grows. This makes the Krylov method an excellent way to avert the risk of humping.

On the other hand, the Krylov method suffers from certain vulnerabilities, first of all, the computational cost of enlarging the Krylov subspace with the Arnoldi algorithm grows quadratically with the subspace dimension. Also, for large problems the storage of the basis vectors alone becomes burdensome. Furthermore, it is a well-known issue that the Arnoldi process suffers from loss of orthogonality, undermining the accuracy of the approximation. For further details see [53, Section 6.3.2]. In considering these issues, the user should keep in mind that each of them can be encountered at every sub-step. Proposed solutions involve restarting the Arnoldi method (see, for instance, [13, 17]) or modifications of the Arnoldi process based on an incomplete orthogonalization (see [21]).

Finally, differently from other polynomial methods whose parameters are set a priori in finitely many variants, the Krylov approximants depend on $\mathbf{A}$ and v. Hence for Krylov methods it is not possible to know, at the beginning of the computation, parameters such as the optimal m and the sub-stepping strategy.

These vulnerabilities divided the researchers, that during the years polarized into the two seemingly competing groups: those interested by the simplicity and elasticity of Taylor and Newton interpolation methods and those that instead are attracted by the great effectiveness of the Krylov method.

1.2 Thesis outline

It is exactly in this context that this thesis work came into play. We now proceed to briefly sketch how this manuscript is organized.

- Chapter 2: *Computing the divided differences of analytic functions.*

  In Chapter 2 we treat the problem of computing quickly and accurately the divided differences of analytic functions. Special attention will be dedicated to the computation of the divided differences of the functions φ_ℓ. Thanks to Opitz's theorem (see [66]) we know that the divided differences of a function f at the sequence $(z_0, z_1, \ldots, z_m)$ can be obtained as $f(\mathbf{Z}(z_0, z_1, \ldots, z_m))$ where

$$\mathbf{Z}(z_0, z_1, \ldots, z_m) = \begin{pmatrix} z_0 & & & \\ 1 & z_1 & & \\ & \ddots & \ddots & \\ & & 1 & z_m \end{pmatrix}$$

  reducing the problem of computing the divided differences for the exponential func-

tion to the accurate computation of the first column of the matrix exponential of $\mathbf{Z}(z_0, z_1, \ldots, z_m)$. Although $\mathbf{Z}(z_0, z_1, \ldots, z_m)$ is small and structured as a bidiagonal matrix, the most known routines for the computation of the matrix exponential miserably fail in obtaining accurate values of $d[z_0], d[z_0, z_1], \ldots, d[z_0, z_1, \ldots, z_m]$. This is because the component of the first column of

$$\mathrm{e}^{\mathbf{Z}(z_0,z_1,\ldots,z_m)}$$

greatly decays in norm. Therefore the routines from the literature, that return results that are accurate in norm, fail to appreciate errors that may be relatively large. To understand why this constitutes a problem, consider the case where we approximate the vector $w^T = [P,\, 0]$ by $\tilde{w}^T = [P,\, \epsilon]$ with $P \gg 1$. The relative error

$$\frac{\|w - \tilde{w}\|}{\|w\|}$$

can be several orders of magnitude smaller than the agreed tolerance while the 0 component is so wrongly approximated that if the relative error was computed componentwise then it would be going to infinite. Therefore, the computation of the divided differences of the exponential function requires to be carried on by a routine that is extremely accurate.

As a result, we designed a routine, `dd_phi`, for the computation of the divided differences of the exponential function that is more accurate than its competitors and, at the same time, it is an order of magnitude faster.

The results are reported in Chapter 1, that is based on the publication

[68] F. Zivcovich, *Fast and accurate computation of divided differences for analytic functions, with an application to the exponential function*, Dolomites Res. Notes Approx, Vol. 12, pp 28–42, 2019.

- Chapter 3: *Computing the matrix exponential.*

  The technical difficulties encountered in the computation of the exponential of the matrix $\mathbf{Z}(z_0, z_1, \ldots, z_m)$ led us to wonder if this problem affects other related fields of numerical analysis. We found out this is the case of the Krylov approximations of matrix functions, that form the approximation by combining orthogonal vectors using as coefficients the first column of the matrix $f(\mathbf{H}_m)$, that shares the Hessenberg structure with $\mathbf{Z}(z_0, z_1, \ldots, z_m)$. In fact, as it was shown in [9, Table 6], the state-of-the-arts routines for computing the matrix exponential sometimes fail to return an accurate approximation of the exponential of $\mathbf{H}_m$. This is because, similarly to the divided differences' case, such routines return results that are accurate in norm but fail to appreciate errors that may be component-wise relatively large although absolutely small. This is a problem since, in some circumstances, it is crucial to have very good approximations in order to preserve certain quantities, for example for large linear Hamiltonian systems [32]. Therefore we investigated the possibility to build a routine that is as accurate as the user wishes.

  As a result, we developed and perfected a routine, `expkptotf`, for computing the

matrix exponential in arbitrary precision arithmetic for any given tolerance that shows accuracy and performances that are superior to existing alternatives.

The results are collected in Chapter 3, that is based on the publication and preprint

[9] M. Caliari, F. Zivcovich, *On-the-fly backward error estimate for matrix exponential approximation by Taylor algorithm*, Journal of Computational and Applied Mathematics, 532–548, 2018;

[60] J. Sastre, J. Ibañez, E. Defez, F. Zivcovich, *On-the-fly backward error Krylov projection for arbitrary precision high performance Taylor approximation of the matrix exponential*, submitted, 2019.

- Chapter 4: *Computing the action of the matrix exponential.*

  Finally, we studied in the main chapter of this manuscript new polynomial approximations to the vector $e^{\mathbf{A}}v$. Acknowledged the great effectiveness of the Krylov method, we tried to replicate its characteristics while avoiding its vulnerabilities using polynomial interpolations at set of points carefully determined according to $\mathbf{A}$. To do so, we studied new interpolation sets whose elements fall closer to the estimated location of the eigenvalues of the input matrices, increasing in this way the efficiency of the polynomial methods. Furthermore, we developed an innovative reordering algorithm of the interpolation sets aimed to increase accuracy and performances of the polynomial methods, improving and substituting the Leja ordering that we introduced at the end of Section 1.1.2. In addition to that, we managed to engineer a procedure for computing the coefficients of the backward error polynomial without recovering to computations in higher precision arithmetic. This allowed for runtime estimates of the norm of backward error matrix for certain interpolation sets determined "on the go" when needed.

  As a result we built two MATLAB routines, `explhe` and `pkryexp`, that proved to be faster and more accurate than many other routines in the literature for the approximation of the action of the matrix exponential.

  The results we obtained are reported in Chapter 4, that led to the production of the following three manuscripts

  [8] M. Caliari, P. Kandolf, F. Zivcovich, *Backward error analysis of polynomial approximations for computing the action of the matrix exponential*, Bit Numer Math 58:907, 2018;

  [10] M. Caliari, F. Cassini, F. Zivcovich, *Approximation of the matrix exponential for matrices with skinny fields of values*, submitted, 2019;

  [11] M. Caliari, F. Zivcovich, *Extended Ritz interpolation for computing the action of the matrix exponential*, in preparation.

Chapter 2

Computing the divided differences of analytic functions

This chapter contains an adaptation of the paper [68] where we analyze in-depth the structure of the matrix representing the operator mapping the coefficients of a polynomial $p(x)$ in the monomial basis $\mathcal{M}$ into those in the Newton basis $\mathcal{N}$. A factorization in minimal terms of the said matrix has been obtained and as a consequence, a factorization of its inverse is available too. As a result, a new high performances routine for the computation of the divided differences of the exponential and the closely related φ_l functions have been produced with satisfying results.

2.1 Introduction

Given a sequence of $n+1$ interpolation points $\sigma_n = (z_0, z_1, \ldots, z_n)$ and the corresponding function values $f(z_0), f(z_1), \ldots, f(z_n)$, there exists a unique polynomial $p(x)$ of degree n interpolating $f(x)$ in the Hermite sense at σ_n. There are many polynomial basis under which terms $p(x)$ can be written, in particular we are going to focus on the monomial and the Newton basis. The *monomial basis* $\mathcal{M} = \{1, x, \ldots, x^n\}$ is such that the $n+1$ scalars $a_0, a_1, \ldots, a_n$ identify $p(x)$ in terms of elements from $\mathcal{M}$ as

$$p(x) = \sum_{k=0}^{n} a_k x^k$$

or, in vector form, as

$$p(x) = m(x,n)^T a(n)$$

where $m(x,n) := (1, x, \ldots, x^n)^T$ and $a(n)$ is the vector having on its kth component the coefficient a_k. The *Newton basis* $\mathcal{N} = \{\pi_{0,\sigma_n}(x), \pi_{1,\sigma_n}(x), \ldots, \pi_{n,\sigma_n}(x)\}$ where

$$\pi_{0,\sigma_n}(x) \equiv 1, \quad \pi_{k,\sigma_n}(x) = \prod_{j=0}^{k-1}(x - z_j)$$

is such that the $n+1$ scalars $d[z_0], d[z_0, z_1], \ldots, d[z_0, z_1, \ldots, z_n]$, called divided differences, identify $p(x)$ in terms of elements from $\mathcal{N}$ as

$$p(x) = \sum_{k=0}^{n} d[z_0, z_1, \ldots, z_k]\pi_{k,z}(x)$$

or, in vector form, as

$$p(x) = \pi(x, n)^T d(n)$$

where $\pi(x, n) = (\pi_{0,\sigma_n}(x), \pi_{1,\sigma_n}(x), \ldots, \pi_{n,\sigma_n}(x))^T$ and $d(n)$ is the vector having on its kth component the coefficient $d[z_0, z_1, \ldots, z_k]$. In this work we are going to analyze in depth the structure of the matrix representing the operator that maps $a(n)$ into $d(n)$ and its inverse.

As a result we will come up with several simple algorithms for switching between $a(n)$ and $d(n)$ each one of them showing different numerical and analytic properties. In particular one of these algorithms fits especially well the task of computing the table of divided differences of a given function $f(x)$, i.e. the matrix having in the $n-k+1$ entries of its $(k+1)$st column the divided differences of the function $f(x)$, i.e. the matrix

$$\begin{pmatrix} d[z_0] & d[z_0, z_1] & \cdots & d[z_0, z_1, \ldots, z_n] \\ & d[z_1] & \cdots & d[z_1, z_2, \ldots, z_n] \\ & & \ddots & \vdots \\ & & & d[z_n] \end{pmatrix}$$

Thanks to this feature, together with the properties of the exponential function, we will show how to build a new routine for the fast and accurate computation of the divided differences of the exponential function and of the closely related φ_ℓ functions. Each of the algorithms we will report is meant to work as it is in MATLAB.

2.2 The change of basis operator

To easily define the matrix representing the change of basis operator and its inverse we need to introduce two important kinds of symmetric polynomials: the complete homogeneous symmetric polynomials and the elementary symmetric polynomials. We recall that a symmetric polynomial is a polynomial such that, if any of the variables are interchanged, remains unchanged.

The *complete homogeneous symmetric polynomial* of degree $k-j$ over the variables $z_0, z_1, \ldots, z_j$ is:

$$c_{k-j}(z_0, z_1, \ldots, z_j) = \sum_{0 \le i_1 \le \ldots \le i_{k-j} \le j} z_{i_1} \cdots z_{i_{k-j}}$$

with $c_{k-j}(z_0, z_1, \ldots, z_j)$ equal to 0 if $k-j < 0$ and to 1 if $k-j = 0$. In other words, $c_{k-j}(z_0, z_1, \ldots, z_j)$ is the sum of all monomials of total degree $k-j$ in the variables.

The *elementary symmetric polynomial* of degree $k-j$ over the variables $z_0, z_1, \ldots, z_{k-1}$ is:

$$e_{k-j}(z_0, z_1, \ldots, z_{k-1}) = \sum_{0 \le i_1 < \ldots < i_{k-j} \le k-1} z_{i_1} \cdots z_{i_{k-j}}$$

with $e_{k-j}(z_0, z_1, \ldots, z_{k-1})$ equal to 0 if $k - j < 0$ and to 1 if $k - j = 0$. In other words, $e_{k-j}(z_0, z_1, \ldots, z_{k-1})$ is the sum of all monomials with $k - j$ factors and of total degree $k - j$ in the variables.

In the literature (see [19], Theorem p. 776) it has been analytically derived that the matrix $\mathbf{C}(z_0, z_1, \ldots, z_n)$ representing the operator that maps $a(n)$ into $d(n)$, i.e.:

$$d(n) = \mathbf{C}(z_0, z_1, \ldots, z_n)a(n)$$

is defined by means of complete homogeneous symmetric polynomials as the matrix having

$$c_{k-j}(z_0, z_1, \ldots, z_j)$$

in its $(j+1, k+1)$ position. Since the complete homogeneous symmetric polynomials are such that $c_{k-j}(z_0, z_1, \ldots, z_j) = 0$ if $k-j < 0$ and $c_{k-j}(z_0, z_1, \ldots, z_j) = 1$ if $k-j = 0$ we have that $\mathbf{C}(z_0, z_1, \ldots, z_n)$ is an upper triangular matrix with all ones on the main diagonal independently from the instance of the sequence $\sigma_n = (z_0, z_1, \ldots, z_n)$. Therefore we know that $\mathbf{C}(z_0, z_1, \ldots, z_n)$ is invertible with inverse $\mathbf{E}(z_0, z_1, \ldots, z_n) := \mathbf{C}(z_0, z_1, \ldots, z_n)^{-1}$. As an example we show in the following the matrix $\mathbf{C}(z_0, z_1, z_2, z_3)$ mapping $a(3)$ into $d(3)$:

$$\mathbf{C}(z_0, z_1, z_2, z_3) = \begin{pmatrix} 1 & z_0 & z_0^2 & z_0^3 \\ 0 & 1 & z_0 + z_1 & z_0^2 + z_0 z_1 + z_1^2 \\ 0 & 0 & 1 & z_0 + z_1 + z_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Notice that z_3 does not appear in $\mathbf{C}(z_0, z_1, z_2, z_3)$.

The next step is to determine analytically the inverse $\mathbf{E}(z_0, z_1, \ldots, z_n)$. To do so, we expand the Newton basis polynomials $\pi_{k,\sigma_n}(x)$ in their explicit form applying the so-called *Vieta's formulas*, which express the coefficients of a polynomial with respect to the monomial basis as symmetric functions of its roots. Briefly, such formulas tell us that the coefficient of the jth power of the kth Newton basis polynomial is such that

$$\text{coef}(j, \pi_{k,\sigma_n}(x)) = (-1)^{k-j} e_{k-j}(z_0, z_1, \ldots, z_{k-1})$$

and hence we can rewrite the kth Newton basis polynomial $\pi_{k,z}(x)$ in the form

$$\pi_{k,\sigma_n}(x) = \sum_{j=0}^{k} (-1)^{k-j} e_{k-j}(z_0, z_1, \ldots, z_{k-1}) x^j.$$

From here it follows that the matrix $\mathbf{E}(z_0, z_1, \ldots, z_n)$ representing the operator that maps $d(n)$ into $a(n)$ is defined by means of elementary symmetric polynomials as the matrix having

$$(-1)^{k-j} e_{k-j}(z_0, z_1, \ldots, z_{k-1})$$

in its $(j+1, k+1)$ position. In fact, for such $\mathbf{E}(z_0, z_1, \ldots, z_n)$ we have that

$$\pi(x, n)^T = m(x, n)^T \mathbf{E}(z_0, z_1, \ldots, z_n)$$

and hence from

$$p(x) = m(x,n)^T \mathbf{E}(z_0, z_1, \ldots, z_n) d(n)$$

we can derive by comparison the identity:

$$a(n) = \mathbf{E}(z_0, z_1, \ldots, z_n) d(n).$$

As an example, we show in the following the matrix $\mathbf{E}(z_0, z_1, z_2, z_3)$ mapping $d(3)$ into $a(3)$:

$$\mathbf{E}(z_0, z_1, z_2, z_3) = \begin{pmatrix} 1 & -z_0 & z_0 z_1 & -z_0 z_1 z_2 \\ 0 & 1 & -z_0 - z_1 & z_0 z_1 + z_0 z_2 + z_1 z_2 \\ 0 & 0 & 1 & -z_0 - z_1 - z_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Notice that z_3 does not appear in $\mathbf{E}(z_0, z_1, z_2, z_3)$.

We conclude this section by writing two MATLAB routines that can be used to build the matrices $\mathbf{C}(z_0, z_1, \ldots, z_n)$ and $\mathbf{E}(z_0, z_1, \ldots, z_n)$ but first we need to give some recurrence properties of the complete homogeneous symmetric polynomials and the elementary symmetric polynomials.

Claim 1. *The complete homogeneous symmetric polynomial over $z_0, z_1, \ldots, z_j$ of degree $k-j$ for any $p \in \{0, 1, \ldots, j\}$ can be decomposed as follows:*

$$\begin{aligned} c_{k-j}(z_0, z_1, \ldots, z_j) = & \, c_{k-j}(z_0, z_1, \ldots, z_{p-1}, z_{p+1}, \ldots, z_j) \\ & + z_p c_{k-j-1}(z_0, z_1, \ldots, z_j). \end{aligned} \tag{2.1}$$

Proof. The claim follows if we consider the alternative representation of the complete homogeneous symmetric polynomials

$$c_{k-j}(z_0, z_1, \ldots, z_j) = \sum_{i_0 + i_1 + \ldots + i_j = k-j} z_0^{i_0} z_1^{i_1} \cdots z_j^{i_j}$$

where $i_0, i_1, \ldots, i_j$ are positive integers. □

Claim 2. *The elementary symmetric polynomial over $z_0, z_1, \ldots, z_{k-1}$ of degree $k-j$ for any $p \in \{0, 1, \ldots, k-1\}$ can be decomposed as follows:*

$$\begin{aligned} e_{k-j}(z_0, z_1, \ldots, z_{k-1}) = & \, e_{k-j}(z_0, z_1, \ldots, z_{p-1}, z_{p+1}, \ldots, z_{k-1}) \\ & + z_p e_{k-j-1}(z_0, z_1, \ldots, z_{p-1}, z_{p+1}, \ldots, z_{k-1}). \end{aligned} \tag{2.2}$$

Proof. Consider the Newton basis polynomial $\pi_{k,\sigma_n}(x)$. For any $p \in \{0, 1, \ldots, k-1\}$ the coefficient of its term of degree j can be computed as

$$\begin{aligned} \mathrm{coef}(j, \pi_{k,\sigma_n}(x)) &= \mathrm{coef}(j, x(x-z_p)^{-1}\pi_{k,z}(x)) + \mathrm{coef}(j, -z_p(x-z_p)^{-1}\pi_{k,z}(x)) \\ &= \mathrm{coef}(j-1, (x-z_p)^{-1}\pi_{k,z}(x)) - z_p \mathrm{coef}(j, (x-z_p)^{-1}\pi_{k,z}(x)) \end{aligned}$$

The claim follows by applying Vieta's formulas to each side of this identity. □

Thanks to the decomposition (2.1) with $p = j$ we can express the entry $(j+1, k+1)$ of $\mathbf{C}(z_0, z_1, \ldots, z_n)$ by means of just z_j and the entries (j, k) and $(j+1, k)$. This can be exploited to build the following code:

```
 1. function C = build_C( z )
 2. % Return the matrix C mapping a(n) into d(n).
 3. % Input:
 4. % -   z, interpolation sequence
 5. % Output:
 6. % -   C, matrix mapping a(n) into d(n)
 7. n = length( z ) - 1;
 8. C = [ cumprod([ 1 ones(1,n)*z(1) ]); zeros(n,n+1) ];
 9. for j = 1:n
10.  for k = j:n
11.    C( j+1,k+1 ) = C( j,k ) + z( j+1 ) * C( j+1,k );
12.  end
13. end
```

Given the interpolation sequence $\sigma_n = (z_0, z_1, \ldots, z_n)$, the routine `build_C` returns the matrix $\mathbf{C}(z_0, z_1, \ldots, z_n)$.

Thanks to the decomposition (2.2) with $p = k-1$ we can express the entry $(j+1, k+1)$ of $\mathbf{E}(z_0, z_1, \ldots, z_n)$ by means of just z_{k-1} and the entries (j, k) and $(j+1, k)$. This can be exploited to build the following code:

```
 1. function E = build_E( z )
 2. % Return the matrix E mapping d(n) into a(n).
 3. % Input:
 4. % -   z, interpolation sequence
 5. % Output:
 6. % -   E, matrix mapping d(n) into a(n)
 7. n = length( z ) - 1;
 8. E = [ cumprod([ 1,-z(1:n) ]); zeros(n,n+1) ];
 9. for j = 1:n
10.  for k = j:n
11.    E( j+1,k+1 ) = E( j,k ) - z( k ) * E( j+1,k );
12.  end
13. end
```

Given the interpolation sequence $\sigma_n = (z_0, z_1, \ldots, z_n)$, the routine `build_E` returns the matrix $\mathbf{E}(z_0, z_1, \ldots, z_n)$.

2.2.1 Factorization of the change of basis operator

Although the two algorithms with which we concluded the past section can be used to switch back and forth between $d(n)$ and $a(n)$, it may not be a good idea to employ them for this purpose. Much faster algorithms can be created and we will do so in this section after we will have shown some factorization properties of the matrices $\mathbf{E}(z_0, z_1, \ldots, z_n)$ and $\mathbf{C}(z_0, z_1, \ldots, z_n)$.

Consider now again the decomposition of the elementary symmetric polynomials given in equation (2.2), this time with $p = 0$. We have

$$\begin{aligned} e_{k-j}(z_0, z_1, \ldots, z_{k-1}) &= e_{k-j}(z_1, \ldots, z_{k-1}) + z_0 e_{k-j-1}(z_1, \ldots, z_{k-1}) \\ &= e_{k-j}(0, z_1, \ldots, z_{k-1}) + z_0 e_{k-j-1}(0, z_1, \ldots, z_{k-1}) \end{aligned} \tag{2.3}$$

where the last equality comes from intrinsic properties of the elementary symmetric polynomials.

We know from the rule that we followed to build the matrix $\mathbf{E}(z_0, z_1, \ldots, z_n)$ that the left hand side of (2.3) equals $(-1)^{k-j}$ times the entry $(j+1, k+1)$ of $\mathbf{E}(z_0, z_1, \ldots, z_n)$. Analogously, on the right hand side of (2.3) we have that $e_{k-j}(0, z_1, \ldots, z_{k-1})$ equals $(-1)^{k-j}$ times the entry $(j+1, k+1)$ of $\mathbf{E}(0, z_1, \ldots, z_n)$ while we have that $e_{k-j-1}(0, z_1, \ldots, z_{k-1})$ equals $(-1)^{k-j-1}$ times the entry $(j+2, k+1)$ of $\mathbf{E}(0, z_1, \ldots, z_n)$.

What we can deduce from this is that for $j, k \in \{0, 1, \ldots, n-1\}$ the entry $(j+1, k+1)$ of $\mathbf{E}(z_0, z_1, \ldots, z_n)$ equals the entry $(j+1, k+1)$ of $\mathbf{E}(0, z_1, \ldots, z_n)$ minus z_0 times the entry $(j+2, k+1)$ of $\mathbf{E}(0, z_1, \ldots, z_n)$. In matrix form this can be represented by

$$\mathbf{E}(z_0, z_1, \ldots, z_n) = (\mathbf{I}_{n+1} - z_0 \mathbf{J}_{n+1}) \mathbf{E}(0, z_1, \ldots, z_n) \tag{2.4}$$

where $\mathbf{I}_{n+1}$ and $\mathbf{J}_{n+1}$ are respectively the identity matrix of size $n+1$ and the zero matrix of size $n+1$ having all ones on its first superdiagonal. Bearing in mind how the matrix representing the inverse change of basis operator is built, we know that

$$(\mathbf{I}_{n+1} - z_0 \mathbf{J}_{n+1}) = \mathbf{E}(z_0, 0, \ldots, 0)$$

and hence

$$\mathbf{E}(z_0, z_1, \ldots, z_n) = \mathbf{E}(z_0, 0, \ldots, 0) \mathbf{E}(0, z_1, \ldots, z_n).$$

Again, by using the definition of the elementary symmetric polynomials we know that

$$\mathbf{E}(0, z_1, \ldots, z_n) = \begin{pmatrix} 1 & \\ & \mathbf{E}(z_1, \ldots, z_n) \end{pmatrix} \tag{2.5}$$

and therefore we can iterate the factorization step after step over the sub-matrices until we get to the full factorization

$$\mathbf{E}(z_0, z_1, \ldots, z_n) = \mathbf{E}(z_0, 0, \ldots, 0) \mathbf{E}(0, z_1, 0, \ldots, 0) \cdots \mathbf{E}(0, \ldots, 0, z_n). \tag{2.6}$$

There is now the possibility to obtain an even more complete factorization for which the matrix $\mathbf{E}(z_0, z_1, \ldots, z_n)$ is factored into the product of $n(n+1)/2$ matrices. Set the matrix $\mathbf{E}_i(z_p)$ to be the $n+1$ square identity matrix with $-z_p$ in the entry $(i, i+1)$, i.e.

$$\mathbf{E}_i(z_p) = \begin{pmatrix} \mathbf{I}_i & & & \\ & 1 & -z_p & \\ & & 1 & \\ & & & \mathbf{I}_{n-i-1} \end{pmatrix}.$$

If we now left multiply $\mathbf{E}_{i+1}(z_p)$ into $\mathbf{E}_i(z_p)$ it is clear that we are merely copying and pasting the bottom right corner of $\mathbf{E}_{i+1}(z_p)$ into the bottom right corner of $\mathbf{E}_i(z_p)$, since the latter consists of an identity matrix. Following this simple principle we can factor $\mathbf{E}(0, \ldots, z_p, \ldots, 0)$ with $p < n$ into the $n - p$ matrices

$$\mathbf{E}_n(z_p) \mathbf{E}_{n-1}(z_p) \cdots \mathbf{E}_{p+1}(z_p)$$

where $\mathbf{E}(0, \ldots, 0, z_n)$ is already the identity matrix. It is possible hence to rewrite the

matrix $\mathbf{E}(z_0, z_1, \ldots, z_n)$ as the product of $n(n+1)/2$ factors

$$\mathbf{E}(z_0, z_1, \ldots, z_n) = \overset{\curvearrowleft}{\prod_{k=0}^{n-1}} (\overset{\curvearrowleft}{\prod_{j=0}^{n-k-1}} \mathbf{E}_{n-j}(z_k)) \tag{2.7}$$

where the curved arrow pointing left indicates that the product sign has to be understood as juxtaposing the upcoming matrices on the left side.

The factorizations of $\mathbf{E}(z_0, z_1, \ldots, z_n)$ shown in (2.6) and (2.7) offer the possibility to factor as well the matrix $\mathbf{C}(z_0, z_1, \ldots, z_n)$ too. In fact we know that the matrix $\mathbf{C}(z_0, z_1, \ldots, z_n)$ equals $\mathbf{E}^{-1}(z_0, z_1, \ldots, z_n)$ and hence we know that, analogously to equation (2.6), $\mathbf{C}(z_0, z_1, \ldots, z_n)$ equals

$$\mathbf{E}(0, \ldots, 0, z_n)^{-1} \mathbf{E}(0, \ldots, 0, z_{n-1}, 0)^{-1} \cdots \mathbf{E}(z_0, 0, \ldots, 0)^{-1}. \tag{2.8}$$

Following the same reasoning we can rewrite, analogously to equation (2.7), the matrix $\mathbf{C}(z_0, z_1, \ldots, z_n)$ as the product of $n(n+1)/2$ factors

$$\mathbf{C}(z_0, z_1, \ldots, z_n) = \overset{\curvearrowright}{\prod_{k=0}^{n-1}} (\overset{\curvearrowright}{\prod_{j=0}^{n-k-1}} \mathbf{E}^{-1}_{n-j}(z_k)).$$

As a side note if we define $\mathbf{C}_i(z_p)$ to be the inverse of the matrix $\mathbf{E}_i(z_p)$, it can be easily seen that

$$\mathbf{C}_i(z_p) = \mathbf{E}_i(-z_p) \tag{2.9}$$

and thus we can rewrite the new factorizaton in a nicer form as

$$\mathbf{C}(z_0, z_1, \ldots, z_n) = \overset{\curvearrowright}{\prod_{k=0}^{n-1}} (\overset{\curvearrowright}{\prod_{j=0}^{n-k-1}} \mathbf{C}_{n-j}(z_k)). \tag{2.10}$$

We now list some highly efficient routines for switching between the monomial and Newton's basis coefficients. Such routines will be based on the factorizations given in equations (2.7) and (2.10). Part of the efficiency of such an algorithm lies in the fact of being in-place, this means that they are algorithms that transform input using no auxiliary data structure. As a consequence, at first sight, it may be confusing that the vector $d(n)$ will be given in input under the name `a_n` and vice-versa $a(n)$ will be given in input under the name `d_n`.

We start by listing the code that given $d(n)$ and the interpolation sequence $\sigma_n = (z_0, z_1, \ldots, z_n)$ returns the coefficients $a(n)$, i.e. it performs the mapping of the operator represented by the matrix $\mathbf{E}(z_0, z_1, \ldots, z_n)$.

```
1. function a_n = Ed_n( z,a_n )
2. % Compute a(n) given d(n) and z.
3. % Input:
4. % -    z, interpolation sequence
5. % -  a_n, vector d(n)
6. % Output:
```

```
 7. % -  a_n, vector a(n)
 8. n = length( z ) - 1;
 9. for j = n:-1:1
10.   for k = j:n
11.      a_n( k ) =  a_n( k ) - z( j ) * a_n( k+1 );
12.   end
13. end
```

Notice that we can get rid of the inner loop by substituting it with the line

```
10. a_n( k:n ) = a_n( k:n ) - z( k ) * a_n( k+1:n+1 );
```

that is a vectorised and therefore parallelizable version of the code. This is equivalent to implementing the factorization of (2.6).

We proceed by listing the inverse algorithm that, given $a(n)$ and the interpolation sequence $\sigma_n = (z_0, z_1, \ldots, z_n)$, returns the coefficients $d(n)$, i.e. it performs the mapping of the operator represented by the matrix $\mathbf{C}(z_0, z_1, \ldots, z_n)$.

```
 1. function d_n = Ca_n( z,d_n )
 2. % Compute d(n) given a(n) and z.
 3. % Input:
 4. % -    z, interpolation sequence
 5. % -  d_n, vector a(n)
 6. % Output:
 7. % -  d_n, vector d(n)
 8. n = length( z ) - 1;
 9. for j = 0:n-1
10.   for k = n:-1:j+1
11.     d_n( k ) = d_n( k ) + z( j+1 ) * d_n( k+1 );
12.   end
13. end
```

Notice that we can't get rid of the inner loop as we could do with the previous code.

In order to do so, we have to reorder somehow the matrix multiplications appearing in equation (2.10). Thanks to their special structure, the matrices $\mathbf{C}_{i_1}(z_{p_1})$ and $\mathbf{C}_{i_2}(z_{p_2})$ commute if and only if $|i_1 - i_2| \neq 1$. Bearing that in mind, consider the matrices in equation (2.10) written down in their extended form, that is without using the product notation. Considering the matrices from right to left we pick the first one that can commute with its left neighbor, that is $\mathbf{C}_1(z_0)$, and we are going to swap it with its left neighbors until we encounter $\mathbf{C}_2(z_1)$ with which it can't commute. At this point we are going to pick the matrix $\mathbf{C}_2(z_1)\mathbf{C}_1(z_0)$ and swap it with its left neighbors until we encounter $\mathbf{C}_3(z_2)$, and so on until the moment in which we obtain

$$\mathbf{C}_n(z_{n-1})\mathbf{C}_{n-1}(z_{n-2})\cdots\mathbf{C}_1(z_0)(\overset{\curvearrowright}{\prod_{k=0}^{n-2}}(\overset{\curvearrowright}{\prod_{j=0}^{n-k-2}}\mathbf{C}_{n-j}(z_k))). \tag{2.11}$$

We can iterate the commuting operations on and on over the matrices that are still nested into the product sign. In this way we obtain the following reordering of the original

factorization:

$$\mathbf{C}(z_0, z_1, \ldots, z_n) = \prod_{k=0}^{\overset{\curvearrowleft}{n-1}} \Big(\prod_{j=0}^{\overset{\curvearrowleft}{n-k-1}} \mathbf{C}_{n-j}(z_{n-k-j-1}) \Big). \tag{2.12}$$

From this we obtain also a reordering for the factorization of the inverse matrix

$$\mathbf{E}(z_0, z_1, \ldots, z_n) = \prod_{k=0}^{\overset{\curvearrowright}{n-1}} \Big(\prod_{j=0}^{\overset{\curvearrowright}{n-k-1}} \mathbf{E}_{n-j}(z_{n-k-j-1}) \Big). \tag{2.13}$$

Let us see how the changes in the arrangement of the matrices shown in formula (2.13) are reflected in the code that given $d(n)$ and the interpolation sequence $\sigma_n = (z_0, z_1, \ldots, z_n)$ returns $a(n)$.

```
 1. function a_n = Ed_n_res( z,a_n )
 2. % Compute a(n) given d(n) and z. Resorted factorization.
 3. % Input:
 4. % -    z, interpolation sequence
 5. % -  a_n, vector d(n)
 6. % Output:
 7. % -  a_n, vector d(n)
 8. n = length( z ) - 1;
 9. for k = 0:n-1
10.   for j = n:-1:k+1
11.     a_n( j ) = a_n( j ) - z( j-k ) * a_n( j+1 );
12.   end
13. end
```

Differently from before, this algorithm is not vectorizable hence we cannot get rid of the inner loop nor we can parallelise the calculations.

Let us see instead how the changes in the arrangement of the matrices shown in formula (2.12) reflect into the code that, given $a(n)$ and the interpolation sequence $\sigma_n = (z_0, z_1, \ldots, z_n)$, returns $d(n)$.

```
 1. function d_n = Ca_n_res( z,d_n )
 2. % Compute d(n) given a(n) and z. Resorted factorization.
 3. % Input:
 4. % -    z, interpolation sequence
 5. % -  d_n, vector a(n)
 6. % Output:
 7. % -  d_n, vector d(n)
 8. n = length( z ) - 1;
 9. for k = 0:n-1
10.   for j = n-k:n
11.     d_n( j ) = d_n( j ) + z( j-n+k+1 ) * d_n( j+1 );
12.   end
13. end
```

This algorithm can be successfully vectorised. In fact, we can get rid of the inner loop replacing it with the line

```
10. d_n( n-k:n ) = d_n( n-k:n ) + z( 1:k+1 ) .* d_n( n-k+1:n+1 );
```

making it possible to parallelise the calculations.

We conclude this section with a remark reconnecting this work to the literature. At the beginning of Section 2 we mentioned [19, Theorem p. 776], which states that if $\mathbf{V}(z_0, z_1, \ldots, z_n)$ is the Vandermonde matrix over the interpolation sequence then the standard LU-decomposition of its transpose is

$$\mathbf{V}(z_0, z_1, \ldots, z_n)^T = \mathbf{C}(z_0, z_1, \ldots, z_n)^T \mathbf{U}(z_0, z_1, \ldots, z_n)$$

where $\mathbf{U}(z_0, z_1, \ldots, z_n)$ is upper triangular. Since the inverse of $\mathbf{C}(z_0, z_1, \ldots, z_n)^T$ is $\mathbf{E}(z_0, z_1, \ldots, z_n)^T$ we can also write

$$\mathbf{E}(z_0, z_1, \ldots, z_n)^T \mathbf{V}(z_0, z_1, \ldots, z_n)^T = \mathbf{U}(z_0, z_1, \ldots, z_n)$$

and therefore any transpose decomposition of $\mathbf{E}(z_0, z_1, \ldots, z_n)$ can be seen as the concatenation of the Gaussian elimination steps.

2.3 The divided differences of the exponential and closely related functions

In the recent years in the field of applications the problem of computing functions of matrices has attracted a rising interest. Among such functions it can be highlighted the exponential and more in general the φ_ℓ functions, that we recall that are defined as:

$$\varphi_\ell(x) = \sum_{i=0}^{\infty} \frac{x^i}{(i+\ell)!} \tag{2.14}$$

and that include the exponential function, since clearly $\varphi_0(x)$ coincides with e^x. When it comes to the approximation of functions of matrices it is very often convenient to interpolate at some sequence $\sigma_n = (z_0, z_1, \ldots, z_n)$ in order to try and exploit spectral properties of the input matrix. Hence a fast and accurate computation of divided differences plays a crucial role in executing such tasks. In this section we are going to introduce a new algorithm for the efficient computation of the divided differences for the φ_ℓ functions at the interpolation sequence $\sigma_n = (z_0, z_1, \ldots, z_n)$, namely

$$d[z_0], d[z_0, z_1], \ldots, d[z_0, z_1, \ldots, z_n].$$

We start by presenting the state-of-the-arts algorithm for the computation of the divided differences of the exponential function at σ_n, developed in [38], since our routine will inherit its structure. This algorithm is based on the so called Opitz's theorem, that has been originally introduced in [47]. According to the Opitz theorem, given any $k \in \{0, 1, \ldots, n\}$, the divided differences

$$d[z_k], d[z_k, z_{k+1}], \ldots, d[z_k, z_{k+1}, \ldots, z_n]$$

of a function $f(x)$ are in order in the last $n-k+1$ entries of the $(k+1)$th column of $f(\mathbf{Z}(z_0, z_1, \ldots, z_n))$, where

$$\mathbf{Z}(z_0, z_1, \ldots, z_n) = \begin{pmatrix} z_0 & & & \\ 1 & z_1 & & \\ & \ddots & \ddots & \\ & & 1 & z_n \end{pmatrix},$$

and hence the desired divided differences can be derived as

$$\exp(\mathbf{Z}(z_0, z_1, \ldots, z_n))e_1$$

where e_1 is the first column of the matrix $\mathbf{I}_{n+1}$.

At this point, one could think that it is enough to compute this vector by using one of the many routines developed to approximate the action of the matrix exponential. The reality is that such routines are only relatively accurate in norm, while we are required to approximate in componentwise relatively high precision each one of the divided differences that, for the exponential function, are rapidly decreasing. Furthermore, general-purpose routines are not designed for this case of interest, which presents a pattern that should be exploited.

Returning to the routine developed in [38], a crucial step that must be performed in order to handle accurately arbitrarily large inputs, is to apply a scaling technique. It is possible in fact to recover the wanted divided differences by just powering the matrix of the divided differences on a scaled sequence. In practice, for a positive integer s, set

$$\mathbf{F}_{(0)} := \exp(\mathbf{Z}(2^{-s}z_0, 2^{-s}z_1, \ldots, 2^{-s}z_n))$$

and set

$$\mathbf{F}_{(i+1)} := \mathbf{R}\mathbf{F}_{(i)}^2\mathbf{R}^{-1}$$

with

$$\mathbf{R} = \begin{pmatrix} 1 & & & \\ & 2 & & \\ & & \ddots & \\ & & & 2^{n-1} \end{pmatrix}$$

then, thanks to the Lemma from ([38], pp.509), we know that for $0 \leq i \leq s$ the following holds:

$$\mathbf{F}_{(i)} = \exp(\mathbf{Z}(2^{i-s}z_0, 2^{i-s}z_1, \ldots, 2^{i-s}z_n))$$

and in particular

$$\mathbf{F}_{(s)} = \exp(\mathbf{Z}(z_0, z_1, \ldots, z_m))$$

that is the table of divided differences at σ_n. Since we are not interested in the whole divided differences matrix but just in the first column, the powering part can be carried on applied to the vector e_1 and hence performing just matrix-vector products. In addition to that, the algorithm from [38] exploited the work of [48] that investigates the relation between the triangular matrix $\mathbf{T}$ and $f(\mathbf{T})$ when $f(x)$ is an analytic function, as the exponential. As a result, $\mathbf{F}_{(0)}$ is computed in an extremely efficient way and the resulting

routine is characterized by very high performances.

What we aim to do in this section is to develop a new routine that outperforms the one from [38] by exploiting the tools we built in this work. In particular, we will focus on the fast computation of $\mathbf{F}_{(0)}$ while we will maintain the powering part untouched.

One idea would be to adapt the algorithms derived in the previous section to the purpose of approximating the divided differences of a power series. In fact the matrix $\mathbf{C}(z_0, z_1, \ldots, z_n)$ has 1 as its last bottom-right entry, hence it is assigns the value a_n to $d[z_0, z_1, \ldots, z_n]$. This is exact if a_n is the highest nonzero coefficient of $p(x)$ while it merely is an approximation otherwise. To fix this problem, we just have to add sufficiently many additional interpolation points $z_{n+1}, z_{n+2}, \ldots, z_N$ in order to recover the lost accuracy. Unfortunately, this approach would require computing each column of $\mathbf{F}_{(0)}$ separately with increased computational complexity due to the additional points. Therefore it may be too slow when it comes to competing with an algorithm as optimized as the one developed in [38].

In order to tackle this inconvenient, we now derive another factorization of the matrix $\mathbf{C}(z_0, z_1, \ldots, z_n)$ that we will show to have an interesting property. Consider the following manipulation of formula (2.4)

$$\begin{aligned}\mathbf{E}(z_0, z_1, \ldots, z_n) &= (\mathbf{I}_{n+1} - z_0\mathbf{J}_{n+1})\mathbf{E}(0, z_1, \ldots, z_n) \\ &= \mathbf{E}(0, z_1, \ldots, z_n) - z_0\mathbf{J}_{n+1}\mathbf{E}(0, z_1, \ldots, z_n)\end{aligned}$$

that, together with the trivial identity

$$\mathbf{J}_{n+1}\mathbf{E}(0, z_1, \ldots, z_n) = \mathbf{E}(z_1, \ldots, z_n, 0)\mathbf{J}_{n+1},$$

that is due to the peculiar structure of $\mathbf{J}_{n+1}$, gives that $\mathbf{E}(z_0, z_1, \ldots, z_n)$ equals

$$\mathbf{E}(z_1, \ldots, z_n, 0)(\mathbf{C}(z_1, \ldots, z_n, 0)\mathbf{E}(0, z_1, \ldots, z_n) - z_0\mathbf{J}_{n+1}).$$

Let us study now the structure of the matrix $\mathbf{C}(z_1, \ldots, z_n, 0)\mathbf{E}(0, z_1, \ldots, z_n)$. To do so we employ formula (2.11) over $z_1, z_2, \ldots, z_n, 0$ instead of $z_0, z_1, \ldots, z_n$, showing that the matrix $\mathbf{C}(z_1, \ldots, z_n, 0)$ can be factorized as

$$\mathbf{C}_n(z_n)\mathbf{C}_{n-1}(z_{n-1})\cdots\mathbf{C}_1(z_1)(\overset{\curvearrowright}{\prod_{k=0}^{n-2}}(\overset{\curvearrowright}{\prod_{j=0}^{n-k-2}}\mathbf{C}_{n-j}(z_{k+1}))).$$

We know already from the past section that

$$\overset{\curvearrowright}{\prod_{k=0}^{n-2}}(\overset{\curvearrowright}{\prod_{j=0}^{n-k-2}}\mathbf{C}_{n-j}(z_{k+1})) = \overset{\curvearrowleft}{\prod_{k=0}^{n-2}}(\overset{\curvearrowleft}{\prod_{j=0}^{n-k-2}}\mathbf{E}^{-1}_{n-j}(z_{k+1})) = (\overset{\curvearrowleft}{\prod_{k=0}^{n-2}}(\overset{\curvearrowleft}{\prod_{j=0}^{n-k-2}}\mathbf{E}_{n-j}(z_{k+1})))^{-1}$$

and hence, by comparison with (2.7), we obtain

$$(\overset{\curvearrowleft}{\prod_{k=0}^{n-2}}(\overset{\curvearrowleft}{\prod_{j=0}^{n-k-2}}\mathbf{E}_{n-j}(z_{k+1})))^{-1} = \mathbf{E}^{-1}(0, z_1, \ldots, z_n),$$

from which we can deduce that

$$\mathbf{C}(z_1,\ldots,z_n,0)\mathbf{E}(0,z_1,\ldots,z_n)=\mathbf{C}_n(z_n)\mathbf{C}_{n-1}(z_{n-1})\cdots\mathbf{C}_1(z_1)$$

which is the $n+1$ sized identity matrix with $z_1,z_2,\ldots,z_n$ on its first superdiagonal. In conclusion, we have that the matrix $\mathbf{E}(z_0,z_1,\ldots,z_n)$ equals

$$\mathbf{E}(z_1,z_2,\ldots,z_n,0)\mathbf{C}_n(z_n-z_0)\mathbf{C}_{n-1}(z_{n-1}-z_0)\cdots\mathbf{C}_1(z_1-z_0)$$

that, thanks to (2.9), we know to be also equal to

$$\mathbf{E}(z_1,z_2,\ldots,z_n,0)\mathbf{E}_n(z_0-z_n)\mathbf{E}_{n-1}(z_0-z_{n-1})\cdots\mathbf{E}_1(z_0-z_1).$$

We can proceed analogously in factorizing the matrix $\mathbf{E}(z_1,z_2,\ldots,z_n,0)$ and then the matrix $\mathbf{E}(z_2,z_3,\ldots,z_n,0,0)$ and so on until we get

$$\mathbf{E}(z_0,z_1,\ldots,z_n)=\prod_{k=0}^{\overset{\curvearrowright}{n}}(\prod_{j=0}^{\overset{\curvearrowleft}{n-1}}\mathbf{E}_{n-j}(z_k-z_{n-j+k}),) \tag{2.15}$$

provided that we set $z_k=0$ for every $k\geq n+1$. Hence

$$\mathbf{C}(z_0,z_1,\ldots,z_n)=\prod_{k=0}^{\overset{\curvearrowleft}{n}}(\prod_{j=0}^{\overset{\curvearrowright}{n-1}}\mathbf{C}_{n-j}(z_k-z_{n-j+k})) \tag{2.16}$$

provided, again, that $z_k=0$ for every $k\geq n+1$. We now have the possibility to write a new algorithm for switching between $a(n)$ and $d(n)$.

This algorithm enjoys a very useful property for our purposes when we use it to compute $d(n)$ from $a(n)$. In fact, at each stage $n-k$ of the outer product sign we are computing the divided differences over $z_k,z_{k+1},\ldots,z_{n+k}$ starting from the divided differences at $z_{k+1},z_{k+2},\ldots,z_{n+k+1}$. To make it evident consider as an example the first step we made toward this factorization

$$\mathbf{E}(z_1,z_2,\ldots,z_n,0)\mathbf{E}_n(z_0-z_n)\mathbf{E}_{n-1}(z_0-z_{n-1})\cdots\mathbf{E}_1(z_0-z_1).$$

When it is inverted and applied to $a(n)$ we have

$$\mathbf{C}_1(z_0-z_1)\mathbf{C}_2(z_0-z_2)\cdots\mathbf{C}_n(z_0-z_n)\mathbf{C}(z_1,z_2,\ldots,z_n,0)a(n)$$

that clearly equals

$$\mathbf{C}_1(z_0-z_1)\mathbf{C}_2(z_0-z_2)\cdots\mathbf{C}_n(z_0-z_n)d'(n)$$

where $d'(n)$ is the vector of the divided differences over $z_1,z_2,\ldots,z_n,0$. Hence it is possible to fill up the columns of the matrix $\mathbf{F}_{(0)}$ while computing the divided differences over $z_0,z_1,\ldots,z_N$ from the coefficients $a_0,a_1,\ldots,a_N$, reducing drastically the required CPU time.

For our particular purpose, we picked N to be equal to $n+30$ and the scaling parameter

s to be a positive integer such that the interpolation points are not spread too far apart, namely $s^{-1}|z_i - z_j| \leq 3.5$ and $s^{-1}|z_k| \leq 3.5$ for any choice of the positive integers $i, j, k \leq n + 1$. This is due to the fact that in the worst case scenario, i.e. $n = 0$, we want to approximate accurately $d[s^{-1}z_0] = \mathrm{e}^{s^{-1}z_0}$ and, in [2, 8] was shown that double precision is attainable by means of the Taylor series truncated to degree 30 provided that $s^{-1}|z_0| \leq 3.5$.

To enforce the condition on the interpolation points σ_m and to sensibly simplify the task, we apply a preconditioning to the interpolation sequence by computing the divided differences of the exponential function at $z_0 - \mu, z_1 - \mu, \ldots, z_n - \mu$ with

$$\mu = n^{-1} \sum_{i=0}^{n} z_i.$$

In this way $s^{-1}|z_i - z_j| \leq 3.5$ clearly implies $s^{-1}|z_k - \mu| \leq 3.5$ for any $i, j, k \leq n + 1$. In addition to that, such a shifting of the interpolation sequence may lead to smaller scaling parameters leading to improve the overall efficiency. Then, by exploiting the properties of the exponential function, we recover the divided differences over the original interpolation sequence by multiplying e^{μ} into the divided differences computed over the shifted points.

In the following, we list the algorithm we developed starting by the factorization we have just outlined in combination with the scaling and squaring algorithm.

```
 1. function dd = dd_phi( z,l )
 2. % Compute phi_l(x)'s divided differences.
 3. % Input:
 4. % -    z, interpolation points
 5. % Output:
 6. % -   dd, divided differences
 7. z = [ zeros( l,1 ); z( : ) ]; mu = mean( z ); z = z - mu;
 8. n = length( z ) - 1; N = n + 30;
 9. F = zeros( n+1 );
10. for i = 1:n
11.   F( i+1:n+1,i ) = z( i ) - z( i+1:n+1 );
12. end
13. s = max( ceil( max( max( abs( F ) ) ) / 3.5 ),1 );
14. % Compute F_0
15. dd = [ 1 1./cumprod( (1:N)*s ) ];
16. for j = n:-1:0
17.   for k = N:-1:(n-j+1)
18.     dd( k ) = dd( k ) + z( j+1 ) * dd( k+1 );
19.   end
20.   for k = (n-j):-1:1
21.     dd( k ) = dd( k ) + F( k+j+1,j+1 ) * dd( k+1 );
22.   end
23.   F( j+1,j+1:n+1 ) = dd( 1:n-j+1 );
24. end
25. F( 1:n+2:(n+1)^2 ) = exp( z/s );
26. F = triu( F );
27. % Squaring Part
```

```
28. dd = F( 1,: );
29. for k = 1:s-1
30.     dd = dd * F;
31. end
32. dd = exp( mu ) * transp(dd(l+1:n+1));
```

We now explain how we can obtain the divided differences of the functions $\varphi_l(x)$ performing just the two small modifications of lines 7 and 32. From formula (2.14) and from the representation of the divided differences shown in the first part of this work, the approximation of the divided differences of $\varphi_\ell(x)$ over the interpolation sequence $(z_0, z_1, \ldots, z_n)$ equals

$$d'(n) = \mathbf{C}(z_0, z_1, \ldots, z_n)a'(n).$$

Here $a'(n)$ is the vector such that the kth entry equals the $(k+l)$th entry of the vector of the coefficients of the exponential function in the monomial basis $m(x, n)$, namely: $a_{k+l} = 1/(k+l-1)!$. As a consequence, one can obtain $d'(n)$ by simply computing the divided differences of the exponential function over the interpolation sequence modified by adding l interpolation points at zero at the beginning $0, \ldots, 0, z_0, z_1, \ldots, z_n$ (executed at line 7) and then discarding the first l (executed at line 32). In fact from (2.5) we know that

$$\mathbf{C}(0, \ldots, 0, z_0, z_1, \ldots, z_n)$$

is the matrix having an l sized identity matrix in its upper left corner.

As a final note, we highlight that with minor modifications of the algorithm in [38] it is possible to obtain the whole divided difference table over an interpolation sequence. In order to apply those minor modifications to our routine it is enough to substitute line 13. with

```
13. s = max( 2^ceil( log2( max( max( abs( F ) ) ) / 3.5 ) ),1 );
```

and lines from 28. to 32. with

```
28. for k = 1:log2( s )
29.   F = F^2;
30. end
31. F = exp( mu ) * F( l+1:n+1,l+1:n+1 );
```

and of course demanding the output to be F and not dd.

2.3.1 Numerical experiments

In this section, we are going to run some numerical tests in order to test thoroughly the performances of our new algorithm, dd_phi, with respect to the most advanced competitors. The competitors are:

- dd_ts, this routine is based on the scaling and squaring algorithm applied to Opitz's theorem for triangular matrices developed in [38] and that we outlined in the past section. The MATLAB implementation we use is due to the author of [5] that also extended this algorithm to the high performance computation of the φ_l functions.

- `exptayotf` from [9], is the only existing routine able to compute the matrix exponential in double precision arithmetic for any given tolerance in a backward stable way. We will use this routine with tolerance set to `realmin` ≈2.23e-308 in combination with the Opitz's theorem in order to compute the divided differences required for running the tests.

On the other hand, we do not compare with the so-called Standard Recurrence algorithm that is the algorithm stemming from the recurrence defining the divided difference, i.e.

$$d[z_k, z_{k+1}, \ldots, z_{k+j}] = \frac{d[z_k, z_{k+1}, \ldots, z_{k+j-1}] - d[z_{k+1}, z_{k+2}, \ldots, z_{k+j}]}{z_k - z_{k+j}}$$

where $d[z_k] = \mathrm{e}^{z_k}$. The reason for this is that when it comes to the numerical implementation it is well known that the resulting routine is a very unstable algorithm, prone to huge accuracy loss. Evidence supporting this can be found in [5, Table 3], where a catastrophic propagation of the error through the steps of the Standard Recurrence algorithm is made evident. To stress this point, in Table 2.1 we report the approximation of some of the divided differences for the exponential function over the first Leja points distributed in the interval $[-2, 2]$. The Leja points over $[-2, 2]$ are defined as the sequence that starts with $z_0 = 0$ and that continues as

$$z_{i+1} \in \arg \max_{x \in [-2,2]} \prod_{j=0}^{i} |x - z_j|, \tag{2.17}$$

this set of points constitutes a particularly good set of interpolations points expressly meant for reducing the numerical instabilities of the polynomial approximations. In ad-

i	`exact`	Standard Recurrence
0	1.000000000000000e00	1.000000000000000e00
1	3.194528049465325e00	3.194528049465325e00
2	6.905489227709076e-01	6.905489227709078e-01
3	2.733266029381669e-01	2.733266029381672e-01
4	4.841100451817702e-02	4.841100451817710e-02
5	1.250375676884083e-02	1.250375676884073e-02
...	...	...
10	2.969328503472984e-07	2.969328505292328e-07
...	...	...
15	8.652201142961257e-13	8.650408252305874e-13
...	...	...
20	4.359767314729199e-19	-3.420012541100294e-17
...	...	...
25	6.335065785389162e-26	3.467430190505354e-16
...	...	...
30	3.912862814239202e-33	1.290495466811578e-16

Table 2.1: Divided differences $d[z_0, z_1, \ldots, z_i]$ at the Leja points $z_0, z_1, \ldots, z_{30}$ for e^x, comparison between the Standard Recurrence algorithm and exact data.

dition to that, we are not going to perform any comparison with the algorithm from [33] based on the computation of divided differences via their representation as a contour integral. In fact, although such an algorithm is an excellent tool for computing accurately the divided differences of certain classes of analytic functions, it is not designed expressly for those of the exponential function and therefore a comparison would turn out to be unfair, to say the least.

The first test we perform consists in testing the speed performances of the three routines under examination. In order to accurately measure the timing of each computation, we run the tests by using just one processor (MATLAB's option `-singleCompThread`). Moreover, we run each routine on each sequence 20 times and we register the average CPU time taken to process each input. The experiments were performed using the 64-bit (glxna64) version of MATLAB® 9.2 (R2017a) on a machine equipped with 16Gb of RAM and four Intel Core i7 processors running at 3.30GHz. The set of sequences over which we run the three routines are:

s1. $\gamma \cdot (z_0, z_1, \ldots, z_n)$ where $n \in \{1, 3, 5, \ldots, 99\}$, $\gamma = 8$ and the interpolation points z_i are randomly chosen following a normal distribution of mean 0 and variance 1;

s2. $\gamma \cdot (z_0, z_1, \ldots, z_n)$ where $n \in \{1, 3, 5, \ldots, 99\}$, $\gamma = 8$ and the interpolation points z_i have real and imaginary parts chosen following a normal distribution of mean 0 and variance 1.

We then report the results in two different graphs (see Figure 2.1) to compare the performances over real and complex inputs. What we can observe is that, over both real and complex inputs, `dd_phi` performs best by at least one order of magnitude. The surprising data is that the routine `exptayotf` is faster than `dd_ts` in the complex case even though it is not optimized for this particular task. We repeat this test but we ask, instead of the vector of the divided differences, the whole divided differences table. The results, reported in Figure 2.2, show that also in this case `dd_phi` stands out as the fastest routine.

The next test we perform is meant to establish how accurate is our routine `dd_phi` with respect to `dd_ts` and `exptayotf`. The interpolation sequences that we use to test the accuracy of the three algorithms are:

a1. $\gamma \cdot (z_0, z_1, \ldots, z_n)$ where $n \in \{10, 25, 50, 100\}$, $\gamma \in \{2, 4, \ldots, 512\}$ and the interpolation points z_i are randomly chosen following a normal distribution of mean 0 and variance 1;

a2. $\gamma \cdot (z_0, z_1, \ldots, z_n)$ where $n \in \{10, 25, 50, 100\}$, $\gamma \in \{2, 4, \ldots, 512\}$ and the interpolation points z_i have real and imaginary parts chosen following a normal distribution of mean 0 and variance 1;

a3. $\gamma \cdot (z_0, z_1, \ldots, z_n)$ where $n \in \{10, 25, 50, 100\}$, $\gamma \in \{2, 4, \ldots, 512\}$ and the interpolation points z_i are the n Chebyshev points over the interval $[-1, 1]$;

a4. $\gamma \cdot (z_0, z_1, \ldots, z_n)$ where $n \in \{10, 25, 50, 100\}$, $\gamma \in \{2, 4, \ldots, 512\}$ and the interpolation points z_i are the n Leja points (see [6, 8, 31, 50]) over the interval $[-1, 1]$;

a5. $\gamma \cdot (z_0, z_1, \ldots, z_n)$ where $n \in \{10, 25, 50, 100\}$, $\gamma \in \{2, 4, \ldots, 512\}$ and the interpolation points z_i are the n Leja points over the closed unit disk in the complex plane (see [50, Example 1.3]);

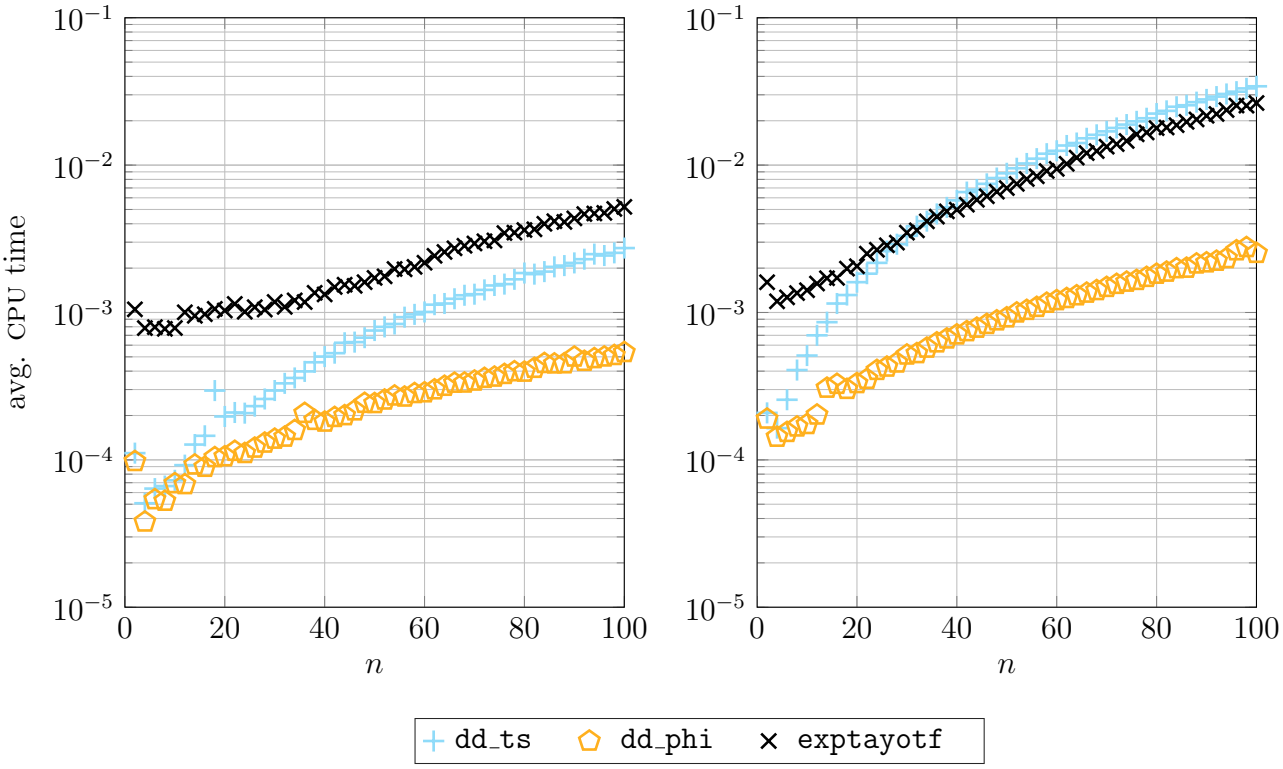


Figure 2.1: Average elapsed CPU time (y-axis) to compute the divided differences at the sequences of length n (x-axis) described in s1. (left) and s2. (right).

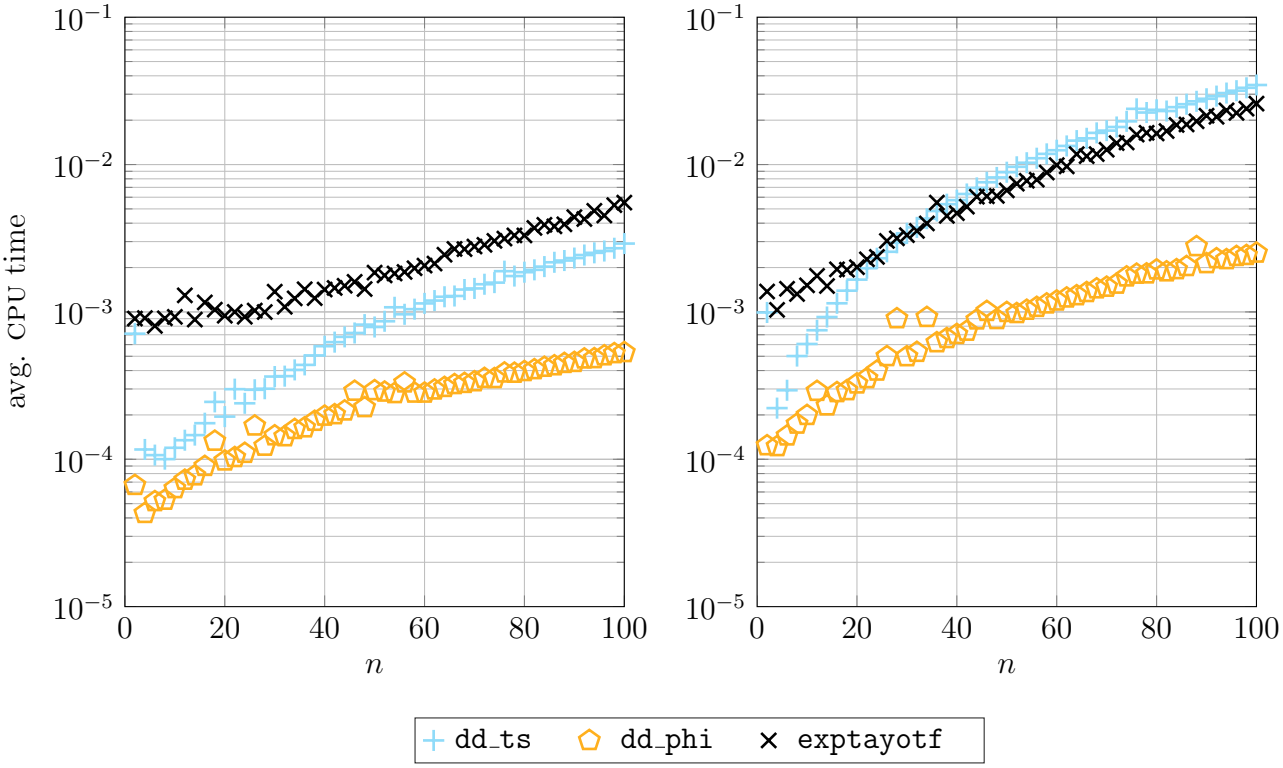


Figure 2.2: Average elapsed CPU time (y-axis) to compute the divided differences table at the sequences of length n (x-axis) described in s1. (left) and s2. (right).

a6. $\gamma \cdot (z_0, z_1, \ldots, z_n)$ where $n \in \{10, 25, 50, 100\}$, $\gamma \in \{2, 4, \ldots, 512\}$ and the interpolation points z_i are coalescing, i.e. $z_i = 2^{-i}$.

The sequences in a1.–a3. are then reordered à la Leja (see [8, Section 3.2] and [50, Formula (1.5)]) so that they can be meaningful from a numerical point of view. Such a reordering is shown to lead to higher stability when it comes to Newton interpolation (see [50, Example 4.1]). On the other hand the sequences in a4.–a5. are by construction already ordered à la Leja while we do not reorder those in a6. since they are designed to stress the three routines.

For each sequence $\gamma \cdot (z_0, z_1, \ldots, z_n)$ we compute the mean relative error committed by each routine taking as trusted reference the divided differences computed using `exptayotf` with tolerance set to `realmin` ≈2.23e-308 and the input converted in a 200 digits `vpa` data type from the variable precision arithmetic toolbox of MATLAB. We plot in Figures from 2.3 to 2.6 the mean relative error committed by each routine as the scale parameter γ increases for the sequences described in a3.–a6., the most significant from a numerical point of view. We deduce from Figures 2.3 to 2.6 that, while there exists a clear relation between the choice of the scale parameter γ and the mean relative error, it does not exist between the length of a sequence and the mean relative error.

What is more important is that it appears evident that our routine, `dd_phi`, at least as accurate as its main competitors. Now, in order to rigorously determine which routine is the most accurate, we present the data from Figures 2.3 to 2.6 together with the data relative to the families of sequences described in a1. and a2. in Figure 2.7 as a performance profile. Figure 2.7 is such that a point (γ, ρ) on a curve related to a method represents the fraction of computed divided differences for which the corresponding error is bounded by ρ times the "unit" error, that we set to `eps` ≈ 2.22e-16.

It is clearly shown in Figure 2.7 that, although all the three routines are very accurate, the performance profile held by `dd_phi` stands out as the most favorable. It is followed by the one of `exptayotf` and then `dd_ts`.

In fact, data suggest that `dd_phi` commits an error smaller than 20 times `eps` over the 95.7% of the total amount of divided differences computed. This fraction drops to respectively 89.7% and 85.3% for the routines `exptayotf` and `dd_ts`.

When instead we consider an error large at most 50 times `eps`, we notice that `dd_phi` approximate the 100% of the divided differences committing a smaller error. As for `exptayotf` and `dd_ts` this figure drops to 95.7% and 97.4% respectively.

A less meaningful, but still significant, data is about the mean error committed over the 9990 divided differences approximated by each routine. The figures are: 4.42e-15 for `dd_phi`, 7.46e-15 for `dd_ts`, and 7.76e-15 for `exptayotf`. The maximum error committed by each routine over the 9990 divided differences approximated equals instead 6.68e-14 for `dd_phi`, 1.20e-13 for `dd_ts`, and 2.97e-13 for `exptayotf`.

2.4 Conclusions

From the numerical tests, we conclude that the new routine `dd_phi` is more stable, accurate and fast than the competitors existing nowadays. This is true both for the computation of the divided differences over an interpolation sequence $\sigma_n = (z_0, z_1, \ldots, z_n)$ and for the computation of the whole divided difference table.

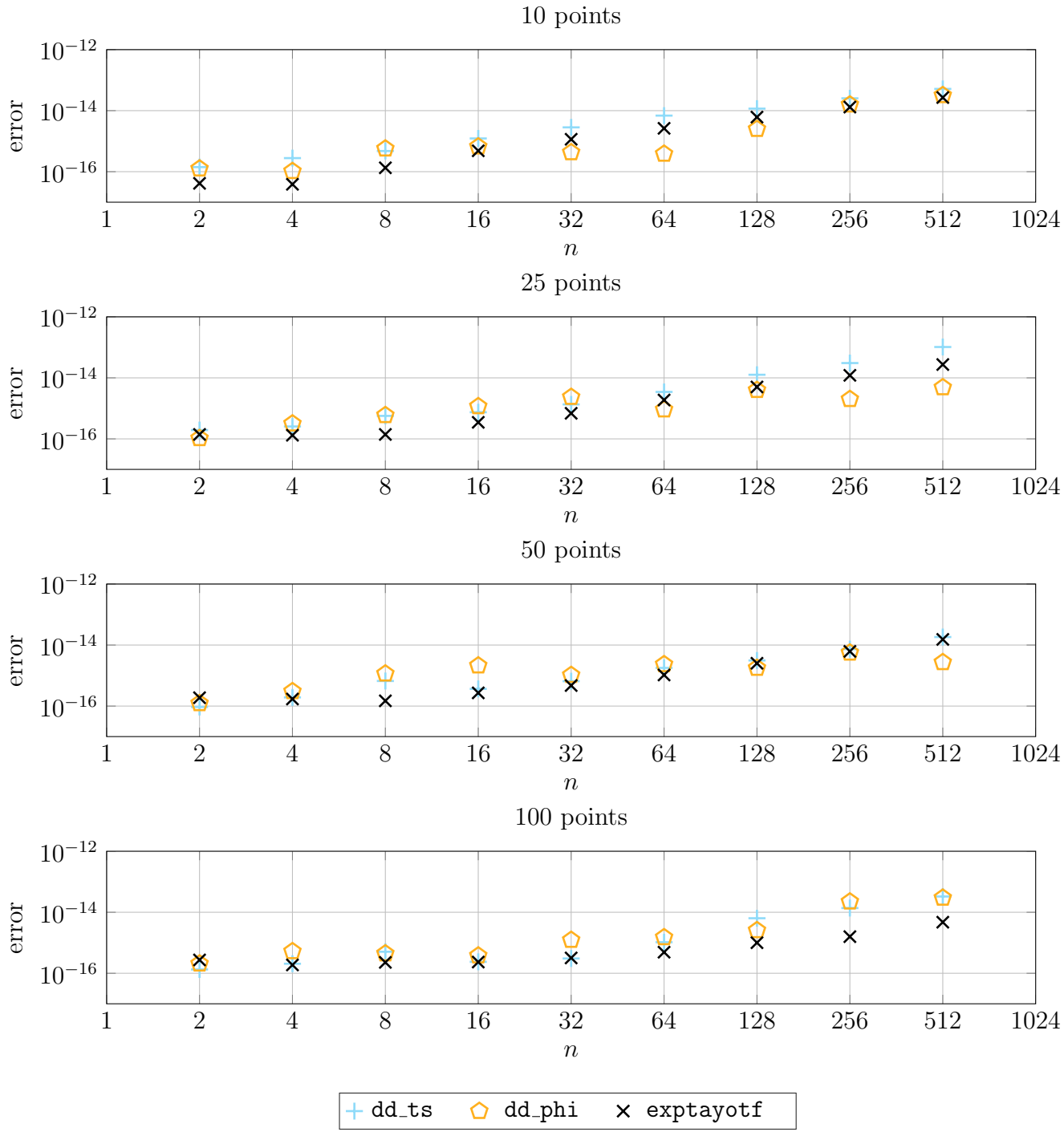


Figure 2.3: Mean average error (y-axis) as the scale parameter γ (x-axis) increases over the sequences described in a3., i.e. n Chebyshev points over $[-1, 1]$ and reordered à la Leja.

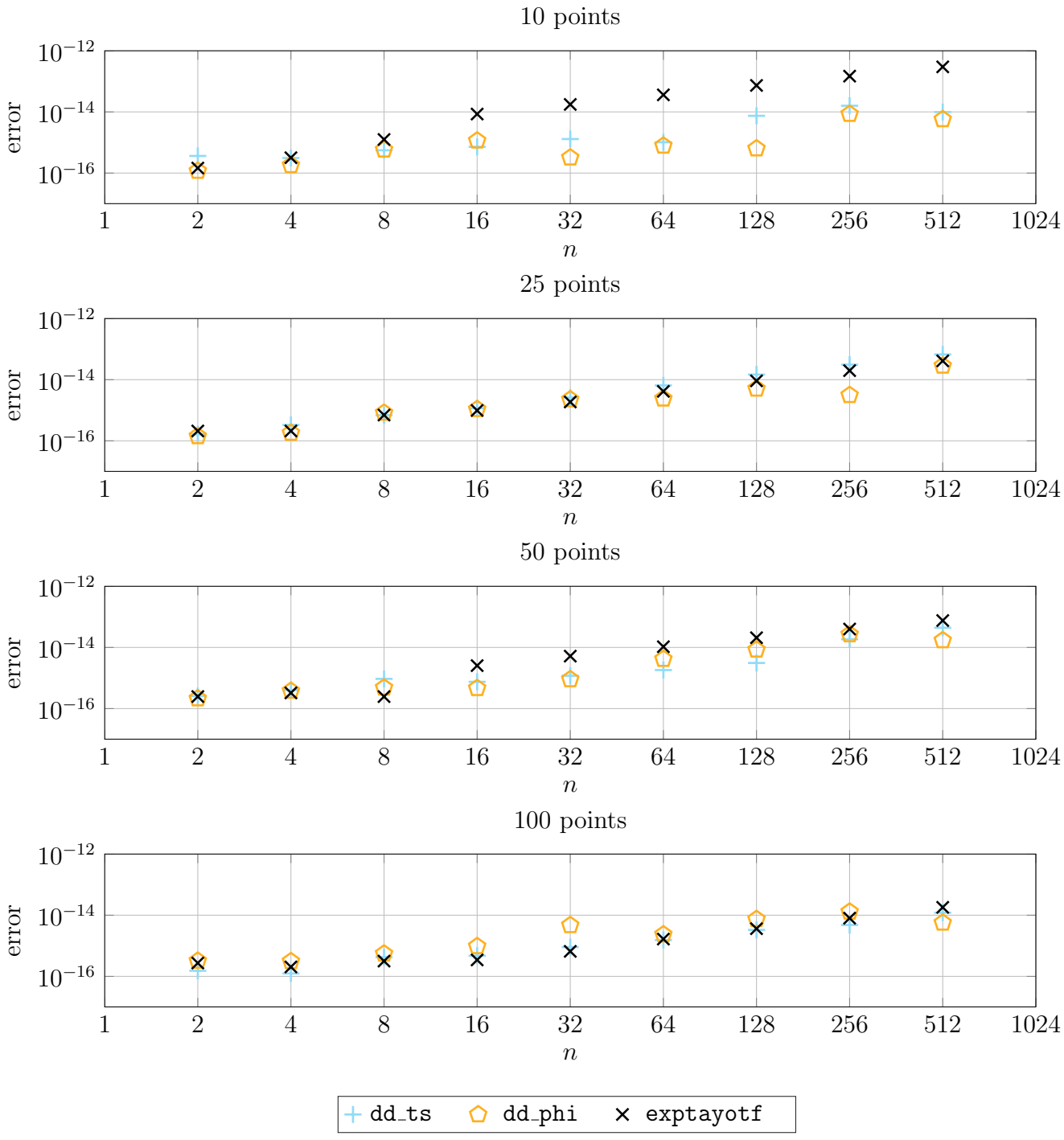


Figure 2.4: Mean average error (y-axis) as the scale parameter γ (x-axis) increases over the sequences described in a4., i.e. n Leja points over the interval $[-1, 1]$.

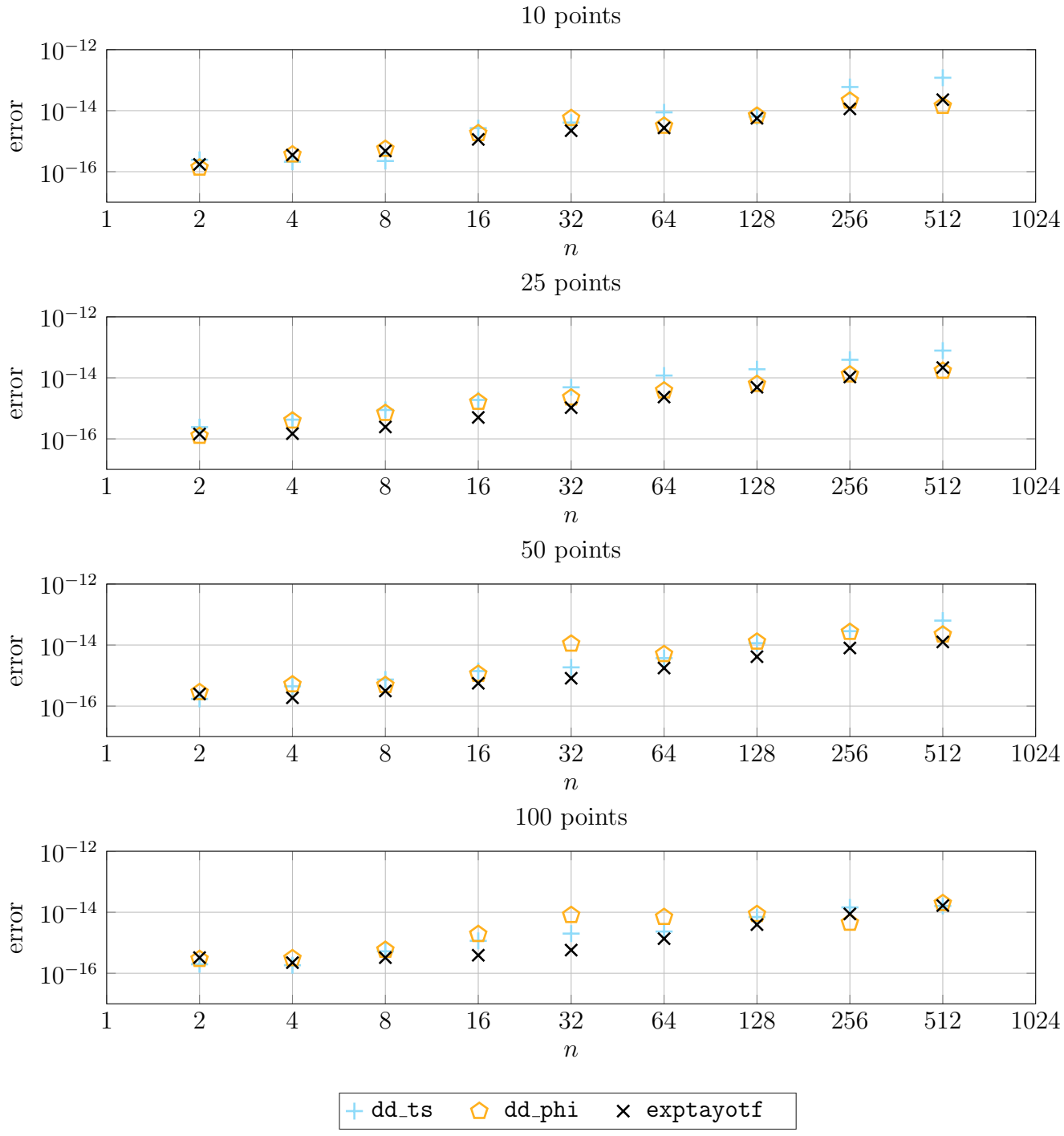


Figure 2.5: Mean average error (y-axis) as the scale parameter γ (x-axis) increases over the sequences described in a5., i.e. n Leja points over the closed complex unit disk.

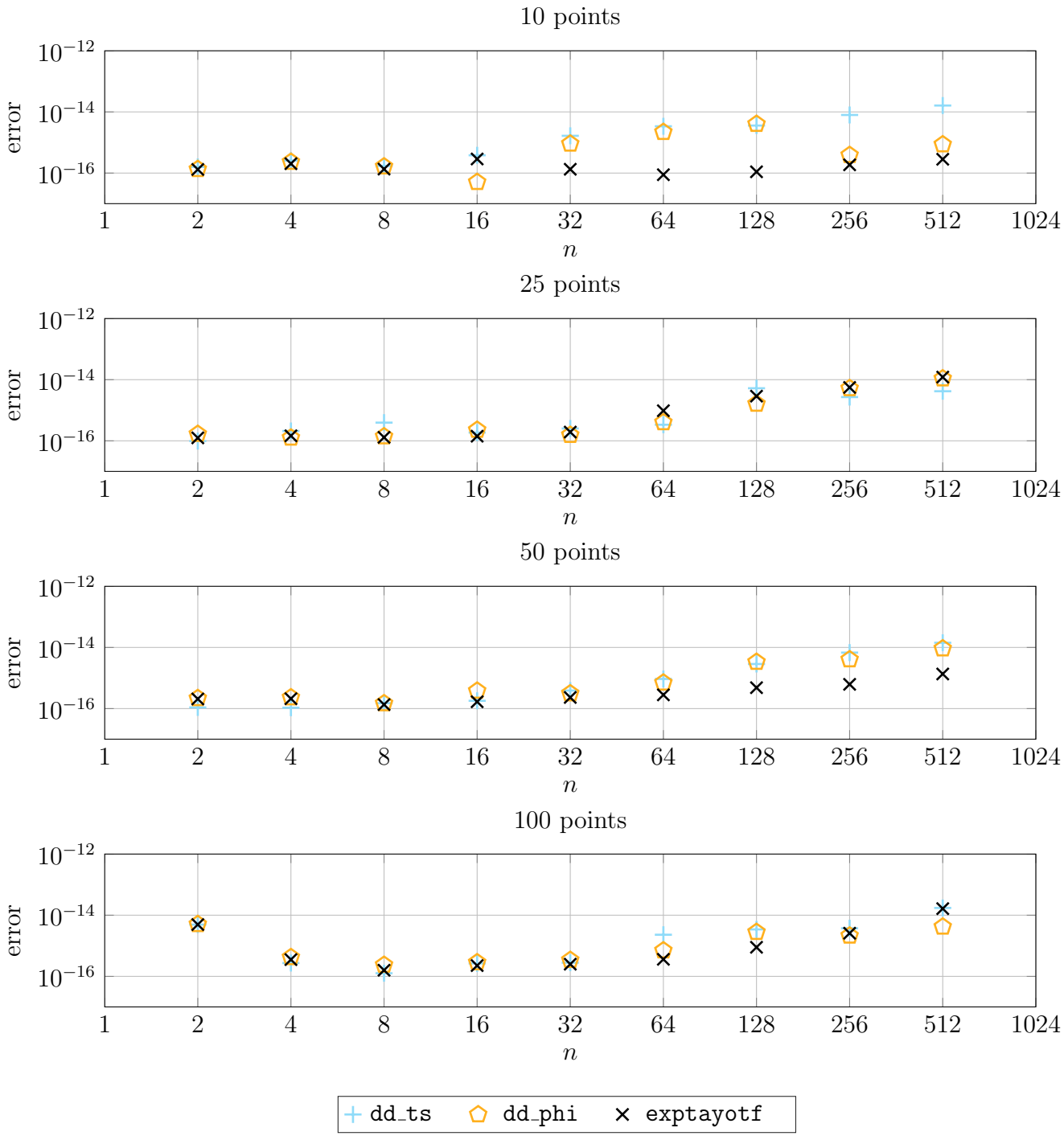


Figure 2.6: Mean average error (y-axis) as the scale parameter γ (x-axis) increases over the sequences described in a6., i.e. coalescing points $z_i = 2^{-i}$.

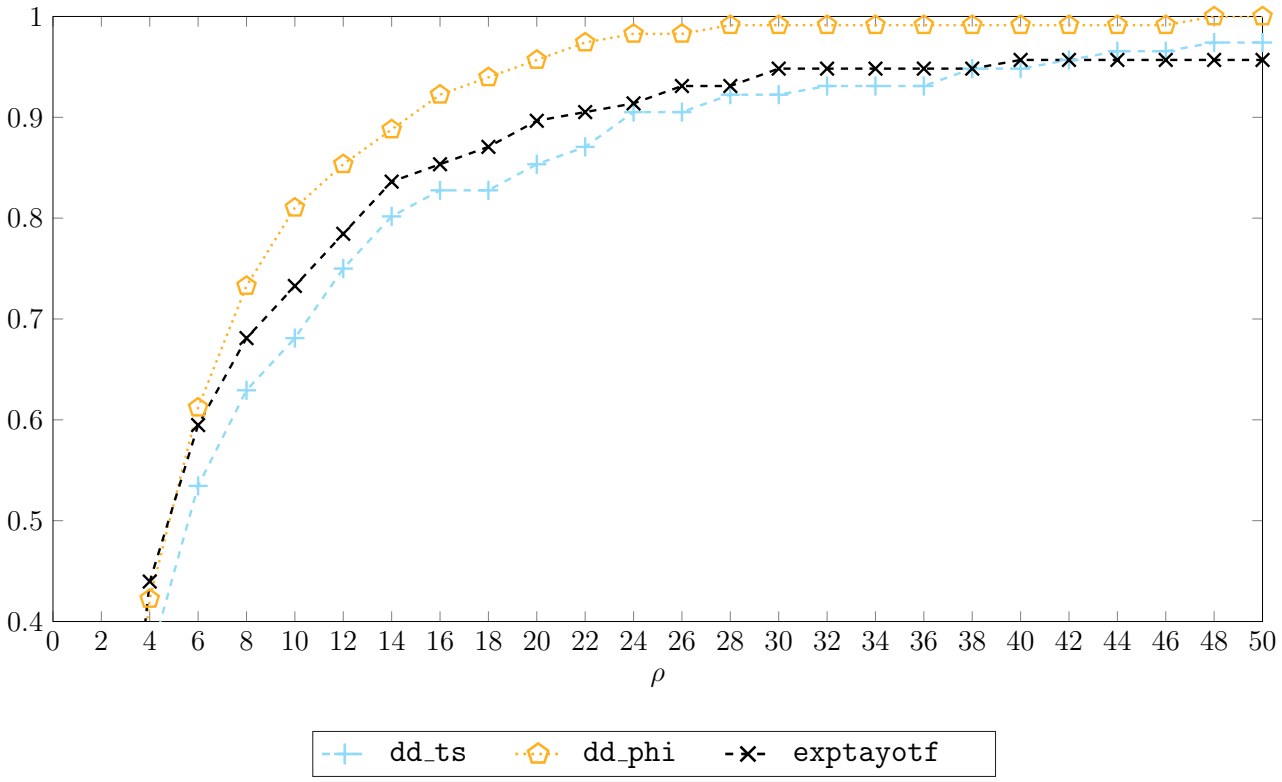


Figure 2.7: Same data as from Figures 2.3–2.6 presented as performance profile.

In conclusion, we can state that the thorough analysis of the matrix representing the operator mapping the coefficients of a polynomial $p(x)$ in the monomial basis $\mathcal{M}$ into those in the Newton basis $\mathcal{N}$ we carried on in the first part of this work gave us a powerful theoretical tool for building innovative algorithms.

The next step is going to be to exploit the structure of the matrices $\mathbf{E}(z_0, z_1, \ldots, z_n)$ and $\mathbf{C}(z_0, z_1, \ldots, z_n)$ in order to tackle difficult problems of interest in the field of numerical polynomial interpolation. As an example, in Chapter 4, we will use the theory developed in the course of this chapter in order to study in-depth the nature of the backward error polynomial using only calculations in low digits arithmetic. Furthermore, as future work, the plan is to develop brand new routines for the computation of the divided differences of other functions of interest such as the trigonometric or logarithmic functions.

Chapter 3

Computing the matrix exponential

New and old challenges brought by the numerical applications and by the always more diffused numerical computations in arbitrary precision arithmetic fuelled lively research in the field of matrix functions. Among the most important is without any doubt the matrix exponential. In this chapter, we report an adaptation of the contents from the manuscripts [9, 60], where the authors developed and then perfected a routine for computing the matrix exponential, in arbitrary precision arithmetic and for any given tolerance, that shows accuracy and performances that are superior to existing alternatives.

3.1 Introduction

Given a complex valued square matrix $\mathbf{A}$ of size N, the matrix exponential $\mathrm{e}^{\mathbf{A}}$, that we recall that can be defined by means of the power series expansion of the exponential function

$$\mathrm{e}^{\mathbf{A}} := \sum_{k=0}^{\infty} \frac{\mathbf{A}^k}{k!},$$

plays a crucial role in many applications of numerical analysis. Among others, we highlight the technique of the exponential integrators, that constitute a competitive tool for the numerical solution of stiff or highly oscillatory problems (see [30]). Since these methods just require the action of $\mathrm{e}^{\mathbf{A}}$ on a vector, and not to form the matrix $\mathrm{e}^{\mathbf{A}}$, that it is generally dense even when $\mathbf{A}$ is sparse, a great deal of methods have been designed for computing the action of the matrix exponential. These methods are particularly efficient for they only perform matrix-vector products, showing a computational complexity decisively smaller than the one needed to handle matrix products. We mention for example the manuscripts [2, 6, 8, 46]. On the other hand there are other cases where forming the matrix exponential is instead convenient. Consider the example of the heat equation

$$u_t = \Delta u, \quad u|_{\partial\Omega} = 0,$$

on $\Omega = (0,1)^2$ with homogeneous Dirichlet boundary conditions. If $\mathbf{A}$ is the discretization of the one dimensional operator ∂_{xx} by finite differences over N points, we have that the matrix

$$\mathbf{L} := \mathbf{A} \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{A}$$

is the two dimensional spatial discretization of the Laplacian operator, where $\mathbf{I}$ is the identity matrix of size N and $\otimes$ is the Kronecker product. Now, if $\mathbf{U}(t)$ is the approximation of $u(x, y, t)$ on the 2D grid and $\mathrm{vec}(\cdot)$ is the operator such that $\mathrm{vec}(\mathbf{A})$ corresponds to the MATLAB command `A(:)`, we can rewrite equation in discretized form as

$$\mathrm{vec}(\dot{\mathbf{U}}(t)) = \mathbf{L}\,\mathrm{vec}(\mathbf{U}(t))$$

and one could thus compute the solution at time t as $\mathrm{e}^{t\mathbf{L}}\,\mathrm{vec}(\mathbf{U}_0)$ and successively reshape it back in matrix form. Unfortunately $\mathbf{L} = \mathbf{A} \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{A}$ is a matrix of size N^2, leading the total complexity of evaluating the solution to drastically grow, as well as to demand a larger amount of storage space.

Alternatively, thanks to the property of the Kronecker product $\mathrm{vec}(\mathbf{X}\,\mathbf{Y}\,\mathbf{Z}) = (\mathbf{Z}^T \otimes \mathbf{X})\,\mathrm{vec}(\mathbf{Y})$, we can rewrite the problem in matrix form as

$$\dot{\mathbf{U}}(t) = \mathbf{A}\,\mathbf{U}(t) + \mathbf{U}(t)\,\mathbf{A}^T$$

whose solution at time t is $\mathrm{e}^{t\mathbf{A}}\,\mathbf{U}_0\,\mathrm{e}^{t\mathbf{A}^T}$, that requires to handle matrix of just size N. We refer to [25, Chapter 10.1] and [37, Section 3.2] for more details. This and many other applications fueled a lively research, we can mention among the others the manuscripts [1] and [18] that are based on a Padé approximation of the exponential function and the manuscripts [9, 18, 51, 57, 58, 59] and [61] that are based instead on a Taylor approximation.

In addition to that, special attention is dedicated to the approximation of the matrix exponential in arbitrary precision arithmetic (see for example the routines from [9, 18]). This is due to the increased interest toward the rich variety of computer algebra systems such as Maple [34], Arithmetica [35], SageMath [54] that natively support arbitrary precision floating-point arithmetic or of MATLAB special toolboxes enabling the multiprecision calculus such as Symbolic Math Toolbox [64] and the Multiprecision Computing Toolbox [39].

Another case, often overlooked, arises when the needed precision is higher than the working precision. Even though this may sound strange, there are cases where this is a crucial point. For example, suppose one approximates the vector $w^T = [P,\, 0]$ by $\tilde{w}^T = [P,\, \epsilon]$, $P \gg 1$, the relative error

$$\frac{\|w - \tilde{w}\|}{\|w\|}$$

can be several orders of magnitude smaller than the working precision while the 0 component is so wrongly approximated that, if the relative error was computed component-wise, then it would be tending to infinite. This phenomenon is not as uncommon as one may think: as an example among others we can mention the case of the matrix

$$\mathbf{Z}(z_0, z_1, \ldots, z_m) = \begin{pmatrix} z_0 & & & \\ 1 & z_1 & & \\ & \ddots & \ddots & \\ & & 1 & z_m \end{pmatrix}$$

that, thanks to the Opitz theorem, we know is such that the exponential of $\mathbf{Z}(z_0, z_1, \ldots, z_m)$

is the divided differences table of the exponential function at the interpolation sequence $(z_0, z_1, \ldots, z_m)$. The matrix

$$\mathrm{e}^{\mathbf{Z}(z_0,z_1,\ldots,z_m)} = \sum_{k=0}^{\infty} \frac{\mathbf{Z}(z_0, z_1, \ldots, z_m)^k}{k!}$$

shows entries that largely vary in norm: in fact the pth subdiagonal of $\mathrm{e}^{\mathbf{Z}(z_0,z_1,\ldots,z_m)}$ only gets filled when, in the summation, j grows larger than p. Hence, it follows, due to the factorial decay of the coefficients, the large variation in norm of the entries. It is also for this reason that in the literature exist dedicated algorithms for the computation of the divided differences of the exponential function, such as the state-of-the-arts algorithm from [68] that we developed in Section 2. Another important case where such a phenomenon may appear, is in the approximation of the action of the matrix exponential using Krylov methods. These methods approximate $\mathrm{e}^{\mathbf{A}}v$, where $\mathbf{A}$ is a N sized squared matrix and v is a vector of according dimension, by forming a linear combination of m orthonormal vectors, $m \ll N$. The coefficients of such linear combination are determined as

$$\mathrm{e}^{\mathbf{H}_m} e_1$$

where $\mathbf{H}_m$ is an m sized upper Hessenberg matrix while e_1 is the first column of $\mathbf{I}_m$. Due to the Hessenberg structure of $\mathbf{H}_m$ also the subdiagonals of $\mathrm{e}^{\mathbf{H}_m}$ fill up with higher powers of $\mathbf{H}_m$ as it happens with $\mathrm{e}^{\mathbf{Z}(z_0,z_1,\ldots,z_m)}$. Therefore it is necessary to use a routine that computes $\mathrm{e}^{\mathbf{H}_m}$ in high precision arithmetic. Or that works in double precision arithmetic but can compute the matrix exponential of $\mathbf{H}_m$ with tolerance set arbitrarily low. As today, the only routine able to run the calculations in double precision arithmetic while setting an arbitrarily low tolerance is `exptayotf` from the manuscript [9] of Caliari and Zivcovich.

The goal of this work is to develop a routine for computing the matrix exponential in arbitrary precision arithmetic that allows the user to prescribe any tolerance regardless of the number of working digits. Moreover, this routine will have to comply with the highest accuracy and speed standards dictated by the state-of-the-arts competitors, that we will introduce in the section on numerical experiments. In order to achieve this result, we decided to enhance the routine `exptayotf`, whose on–the–fly backward error estimate is one of the most promising option for building the core of the routine we wish to create. In particular, we applied four main modifications to `exptayotf`:

- we implemented a new strategy for choosing the approximation degree m and the scaling parameter s;

- we improved the parameters selection process based on the on–the–fly backward error estimate tool. To do so we designed a backward error projection over a small Krylov subspace in order to discard more quickly unfeasible combinations of parameters;

- we implemented for tolerances larger or equal to 2^{-53} a new polynomial evaluation scheme that was developed in [56] and successfully applied to the computation of the matrix exponential in [57];

- following the path traced in the work [18], we got rid of the only conjecture made in [9] that, although weak, poorly fits an algorithm claiming to be highly accurate.

This chapter is structured as follows: in Section 3.2 we introduce the basic polynomial that we use to approximate the matrix exponential, we describe two ways to evaluate such polynomial: the well known Paterson–Stockmeyer evaluation scheme and the recently developed evaluation scheme from [57]. In Section 3.3 we formulate an accuracy criterion based on the matrix of the backward error, we then describe the on–the–fly backward error estimate and show how to project it on a smaller Krylov subspace to boost the process of selecting the approximation parameters. In Section 3.4 we show some numerical experiments and then in Section 3.5 we draw the conclusions of this manuscript.

3.2 Approximation technique and polynomial evaluation scheme

The truncation

$$T_m(\mathbf{A}) = \sum_{k=0}^{m} \frac{\mathbf{A}^k}{k!}$$

at degree m of the matrix exponential coupled with a *shifting* technique constitutes the basic approximation method that we employ to build our algorithm. The idea behind the shifting technique is simple: Taylor approximation corresponds to interpolating the exponential function and its derivatives at 0, it then would be desirable if the eigenvalues of the input matrix were somehow distributed around the origin. Therefore we pick the shifting parameter to be equal to the mean eigenvalue of $\mathbf{A}$, that is

$$\mu := \sum_{i=0}^{N} \frac{a_{ii}}{N}.$$

We set $\mathbf{B} := \mathbf{A} - \mu\mathbf{I}$ where $\mathbf{I}$ is the identity matrix of appropriate dimension and we exploit the properties of the exponential function to approximate $\mathrm{e}^{\mathbf{A}}$ by $\mathrm{e}^{\mu} T_m(\mathbf{B})$.

Increasing m generally corresponds to an increased accuracy, at least in exact arithmetic. On the other hand a larger m leads to an higher computational effort, thus we couple this approximation method with an important technique aimed to keep under control the computational cost: the so called *scaling and squaring* technique. The scaling and squaring algorithm consists in picking s to be a positive power of 2 such that the matrix

$$T_m(s^{-1}\mathbf{B}) = \sum_{k=0}^{m} \frac{(s^{-1}\mathbf{B})^k}{k!} \tag{3.1}$$

accurately approximates $\mathrm{e}^{s^{-1}\mathbf{B}}$ up to the prescribed accuracy. Then the approximation of $\mathrm{e}^{\mathbf{A}}$ can be recovered by $T_m^{(s)}(s^{-1}\mathbf{A})$, that is the matrix coming from the recurrence

$$T_m^{(j+1)}(s^{-1}\mathbf{A}) = T_m^{(j)}(s^{-1}\mathbf{A}) T_m^{(j)}(s^{-1}\mathbf{A}), \quad j = 0, 1, \ldots, \log_2(s) - 1$$

where $T_m^{(0)}(s^{-1}\mathbf{A}) = \mathrm{e}^{s^{-1}\mu} T_m(s^{-1}\mathbf{B})$. Alternatively the approximation of $\mathrm{e}^{\mathbf{A}}$ can be recov-

ered by the matrix $e^{\mu} T_m^{(s)}(s^{-1}\mathbf{B})$ where

$$T_m^{(j+1)}(s^{-1}\mathbf{B}) = T_m^{(j)}(s^{-1}\mathbf{B}) T_m^{(j)}(s^{-1}\mathbf{B}), \quad j = 0, 1, \ldots, \log_2(s) - 1$$

and $T_m^{(0)}(s^{-1}\mathbf{B}) = T_m(s^{-1}\mathbf{B})$. Although the two procedures are mathematically equivalent, the former is less prone to overflow when μ has negative real part (see [9, Section 3]). Therefore we will use the two forms accordingly with the sign of the real part of μ.

3.2.1 Polynomial Evaluation Scheme

The Taylor series for the exponential function is a convergent series, hence there always exists a degree m for which $T_m(s^{-1}\mathbf{B})$ approximates $e^{s^{-1}\mathbf{B}}$ as accurately as desired. On the other hand, m could be quite large, thus requiring many executions of costly matrix products MP. For this reason, in computing polynomial approximations of matrix functions, an appropriate choice of the evaluation scheme plays a crucial role. One of the most famous evaluation schemes is certainly the Horner scheme. It consists in evaluating a polynomial performing the multiplications in a nested fashion. We briefly illustrate this scheme giving a quick example, we show in the following how to evaluate $T_4(\mathbf{X})$ using the Horner scheme:

$$T_4(\mathbf{X}) = \mathbf{I} + \mathbf{X}(\mathbf{I} + \frac{\mathbf{X}}{2}(\mathbf{I} + \frac{\mathbf{X}}{3}(\mathbf{I} + \frac{\mathbf{X}}{4}))).$$

Unfortunately in the matrix case, the Horner scheme does not make us save any matrix-vector product. In fact, we used 3 matrix-vector products to reach degree 4. We, therefore, have to consider more involved polynomial evaluation schemes.

In particular, we consider two evaluation schemes for the computation of $T_m(\mathbf{X})$: the first is the so-called Paterson–Stockmeyer evaluation scheme, which was proven in several works ([9, 18, 51, 57, 58, 59, 61]) to be a robust and successful choice for the computation of the matrix exponential; the second is a new scheme recently developed by J. Sastre in the work [56], this scheme has been optimized in [57] for the approximation of the matrix exponential at most in double precision arithmetic and its great efficiency pushed us to adopt it whenever it's possible.

The Paterson–Stockmeyer evaluation scheme for evaluating $T_m(\mathbf{X})$ consists in computing and storing the first few powers of $\mathbf{X}$:

$$\mathbf{X}^2, \mathbf{X}^3, \ldots, \mathbf{X}^z$$

for some positive integer z. Then, the following regrouping of the terms of $T_m(\mathbf{X})$ is considered

$$T_m(\mathbf{X}) = \mathbf{I} + \sum_{k=0}^{r} (\mathbf{X}^z)^k \mathbf{P}_k, \quad r = \left\lfloor \frac{m}{z} \right\rfloor$$

where the matrices $\mathbf{P}_k$ are defined as

$$\mathbf{P}_k = \begin{cases} \displaystyle\sum_{i=1}^{z} \frac{\mathbf{X}^i}{(zk+i)!}, & k = 0, 1, \ldots, r-1, \\ \displaystyle\sum_{i=1}^{m-zk} \frac{\mathbf{X}^i}{(zk+i)!}, & k = r. \end{cases}$$

Clearly, for a given number MP of matrix products we wish to find a rule for choosing z so that the approximation degree m is maximized. A commonly used rule (see e.g. [9, 18, 59]) consists in choosing

$$z = \left\lceil \frac{\mathrm{MP}}{2} \right\rceil + 1$$

leading to the approximation degree

$$m = (\mathrm{MP} - z + 2)z,$$

the optimality of this rule has been demonstrated in the recent work [16]. Henceforth, we can define the function $\deg_{\mathrm{PS}}$ that maps the number of matrix products into the maximum approximation degree

$$\deg_{\mathrm{PS}}(\mathrm{MP}) = (\mathrm{MP} - \left\lceil \frac{\mathrm{MP}}{2} \right\rceil + 3)(\left\lceil \frac{\mathrm{MP}}{2} \right\rceil + 1),$$

that is the formula for the optimal m where the formula for the optimal z in function of MP was plugged in. The strength of the Paterson–Stockmeyer evaluation scheme lies in the fact that it consists in a mere regrouping of the polynomial to evaluate. Hence it can be directly applied to polynomials of any degree without the need for accessory computations. We briefly illustrate this scheme with a quick example, we show in the following how evaluate $T_6(\mathbf{X})$ with the Paterson–Stockmeyer evaluation scheme

$$T_6(\mathbf{X}) = \mathbf{I} + \frac{\mathbf{X}^2}{2!} + \frac{\mathbf{X}^3}{3!} + \mathbf{X}^3(\frac{\mathbf{I}}{4!} + \frac{\mathbf{X}^2}{5!} + \frac{\mathbf{X}^3}{6!}).$$

To compute $T_6(\mathbf{X})$ we performed 3 matrix products, the same amount required by the Horner scheme to reach degree 4.

On the other hand, there exists a new evaluation scheme, developed by J. Sastre in [56], that consists in a factorization of the polynomial of interest in smaller degree polynomials. The coefficients of these smaller polynomials are not restricted to be equal to the coefficients from the polynomial of interest. Instead, the coefficients of such smaller polynomials are computed in advance, once and for all, by using a software of symbolic calculus.

Differently from the Paterson–Stockmeyer scheme, there is not a rule in common for every polynomial degree m, on the contrary one has to proceed on a case to case basis. In [57] the case of Taylor series truncation at degrees $m = 2, 4, 8, 15, 21, 24, 30$ were treated. In such cases, the numbers z of powers of $\mathbf{X}$ to compute and store are respectively $z = 2, 2, 2, 2, 3, 4, 5$. The reason for treating cases just until degree 30 lies in the fact that the computational efficiency of polynomial approximations in double precision arithmetic of the exponential function is maximum when m is not larger than 30. This is due to the great efficiency of the scaling and squaring algorithm, that allows to roughly double the approximation degree by raising the number of needed matrix products of just one unit.

We briefly illustrate the Sastre evaluation scheme by giving a quick example, we show in the following how evaluate $T_8(\mathbf{X})$. First we compute

$$y_{02}(\mathbf{X}) = \mathbf{X}^2(c_1\mathbf{X}^2 + c_2\mathbf{X}),$$

then we form

$$T_8(\mathbf{X}) = (y_{02}(\mathbf{X}) + c_3\mathbf{X}^2 + c_4\mathbf{X})(y_{02}(\mathbf{X}) + c_5\mathbf{X}^2) + c_6 y_{02}(\mathbf{X}) + \mathbf{X}^2/2 + \mathbf{X} + \mathbf{I}.$$

To compute $T_8(\mathbf{X})$ we performed 3 matrix products, the same amount required by the Paterson–Stockmeyer scheme to reach degree 6 and by the Horner scheme to reach degree 4.

The coefficients $c_1, c_2, \ldots, c_6$ are obtained by comparison with those of $T_8(\mathbf{X})$, this operation requires to solve a nonlinear system of equations in high precision arithmetic. In the following we report a table listing the highest degree reachable from the two scheme for a growing number of matrix products. As we can observe in Table 3.1, the new

MP	0	1	2	3	4	5	6	7
$\deg_{\mathrm{PS}}(\mathrm{MP})$	1	2	4	6	9	12	16	20
$\deg_{\mathrm{Sastre}}(\mathrm{MP})$	1	2	4	8	15	21	24	30

Table 3.1: Maximum degree m reachable by the Paterson–Stockmeyer and by the Sastre evaluation scheme in function of the number MP of executed matrix products.

evaluation scheme is considerably more efficient than the Paterson–Stockmeyer one, i.e. with the same amount of matrix products we can reach higher approximation degrees. For the Sastre evaluation scheme we have to define the function mapping the number of matrix products into the maximum degree reachable $\deg_{\mathrm{Sastre}}$ by cases as shown in Table 3.1.

As a final note, we remark that we are going to call *candidate degrees* those degrees that are of the form $\deg_{\mathrm{PS}}(\mathrm{MP})$, if the Paterson–Stockmeyer evaluation scheme is employed, or of the form $\deg_{\mathrm{Sastre}}(\mathrm{MP})$, if instead we employ the Sastre evaluation scheme. For sake of simplicity, from now on we refer to the candidate degrees as the degrees of the form $\deg(\mathrm{MP})$, omitting to specify the evaluation scheme in the subscript. Besides, we call *candidate scalings* all the scaling parameters that are positive powers of 2.

If a couple of positive integers (m, s) is such that m is a candidate degree and s is a candidate scaling we call (m, s) a *candidate couple*. If a candidate couple is such that the resulting approximation is accurate, then we have that (m, s) is a *feasible couple*. Notice that, for any input matrix $\mathbf{A}$, the feasible couples are infinitely many. In fact, if (m, s) is a feasible couple then, for example, also the couples $(m, 2^p s)$ are feasible for any integer p larger than 1.

3.3 Analysis of the backward error

The approximation method and the evaluation techniques shown in the previous section are broadly used by many other routines for the computation of the matrix exponential. What really makes our routine faster, more flexible and accurate than our competitors is the superior and quicker choice of the approximation parameters m and s. This section is dedicated to describe how to choose the best feasible couple (m, s) in a competitive time.

As a first step we must introduce a criterion that tells us if a couple (m, s) is feasible or not, i.e. if (m, s) caters to an accurate approximation. To do so, we introduce the backward

error matrix $\Delta\mathbf{B}$ by stating that the approximation $T_m(s^{-1}\mathbf{B})^s$ is the exponential of a slightly perturbed matrix:

$$T_m(s^{-1}\mathbf{B})^s = \mathrm{e}^{\mathbf{B}+\Delta\mathbf{B}}.$$

The approximation is accurate if the backward error matrix $\Delta\mathbf{B}$ has a small norm in comparison to the norm of $\mathbf{A}$, that is:

$$\|\Delta\mathbf{B}\| \leq \mathrm{tol} \cdot \|\mathbf{A}\| \tag{3.2}$$

where tol is a tolerance specified by the user. We rewrite $\Delta\mathbf{B}$ as a function of $\mathbf{B}$ by exploiting the fact that

$$\mathrm{e}^{\Delta\mathbf{B}} = \mathrm{e}^{-\mathbf{B}} T_m(s^{-1}\mathbf{B})^s$$

and therefore

$$\Delta\mathbf{B} = s\log(\mathrm{e}^{-s^{-1}\mathbf{B}} T_m(s^{-1}\mathbf{B}))$$

is the functional form yielding $\Delta\mathbf{B}$. We now introduce some notation, in particular considering the residual function, that for a square matrix $\mathbf{X}$ is defined as

$$r_m(\mathbf{X}) := \mathrm{e}^{\mathbf{X}} - T_m(\mathbf{X}) = \sum_{j=m+1}^{\infty} \frac{\mathbf{X}^j}{j!}.$$

We can consider the identity $\mathbf{I} - \mathrm{e}^{-\mathbf{X}} T_m(\mathbf{X}) = \mathrm{e}^{-\mathbf{X}} r_m(\mathbf{X})$ and use it to define two important functions that will play a key role in this paper:

$$g_{m+1}(\mathbf{X}) := \mathrm{e}^{-\mathbf{X}} r_m(\mathbf{X})$$

that over $\mathbb{C}^{N\times N}$ can be represented by its power series expansion

$$g_{m+1}(\mathbf{X}) = \sum_{k=0}^{\infty} b_{k,m}\mathbf{X}^k$$

and

$$h_{m+1}(\mathbf{X}) := \log(\mathbf{I} - g_{m+1}(\mathbf{X})).$$

that over the set

$$\Omega = \{\mathbf{X} \in \mathbb{C}^{N\times N} \colon \rho(g_{m+1}(\mathbf{X})) < 1\} \tag{3.3}$$

can be represented by its power series expansion

$$h_{m+1}(\mathbf{X}) = \sum_{k=0}^{\infty} f_{k,m}\mathbf{X}^k. \tag{3.4}$$

The analytic expression of the coefficients $b_{k,m}$ and $f_{k,m}$ is available. It was in fact already independently derived in previous works, for more details see [9, 15, 58]. We therefore have everything we need to represent explicitly the backward error matrix $\Delta\mathbf{B}$ as:

$$s^{-1}\Delta\mathbf{B} = h_{m+1}(s^{-1}\mathbf{B}) = \sum_{k=0}^{\infty} f_{k,m} s^{-k}\mathbf{B}^k$$

provided that $s^{-1}\mathbf{B}$ belongs to Ω. Thanks to the power series representation of $\Delta\mathbf{B}$ we now know that, in order to satisfy the inequality (3.2), two requirements have to hold true at the same time

$$s^{-1}\mathbf{B} \in \Omega \tag{3.5a}$$

and

$$\left\|h_{m+1}(s^{-1}\mathbf{B})\right\| \leq \mathrm{tol} \cdot s^{-1} \cdot \|\mathbf{A}\| . \tag{3.5b}$$

Both requirements (3.5a) and (3.5b) are particularly expensive to check, for they involve evaluations of matrix functions. Hence, we apply some inequalities in order to produce more stringent requirements that, in exchange, are far easier to check. To start with, remember that (3.5a) holds true if $\rho(g_{m+1}(s^{-1}\mathbf{B}))$ is smaller than 1. By applying the renowned Gelfand formula, we can avail of the following inequality

$$\rho(g_{m+1}(s^{-1}\mathbf{B})) \leq \left\|g_{m+1}(s^{-1}\mathbf{B})\right\| .$$

Therefore we know that, if

$$\left\|g_{m+1}(s^{-1}\mathbf{B})\right\| \leq 1,$$

then (3.5a) is verified. The requirement (3.5b) can be manipulated too. Bearing in mind that $h_{m+1}(s^{-1}\mathbf{B}) = \log(\mathbf{I} - g_{m+1}(s^{-1}\mathbf{B}))$, we can write

$$\left\|\log(\mathbf{I} - g_{m+1}(s^{-1}\mathbf{B}))\right\| = \left\|\sum_{j=1}^{\infty} \frac{(-1)^j g_{m+1}(s^{-1}\mathbf{B})^j}{j}\right\| \leq \sum_{j=1}^{\infty} \frac{\|g_{m+1}(s^{-1}\mathbf{B})\|^j}{j},$$

where clearly we have

$$\sum_{j=1}^{\infty} \frac{\|g_{m+1}(s^{-1}\mathbf{B})\|^j}{j} = -\log(1 - \left\|g_{m+1}(s^{-1}\mathbf{B})\right\|).$$

We are now able to check (3.5a) and (3.5b) at the same time by verifying that

$$-\log(1 - \left\|g_{m+1}(s^{-1}\mathbf{B})\right\|) \leq \min\{1, \mathrm{tol} \cdot s^{-1}\,\|\mathbf{A}\|\}, \tag{3.6}$$

shifting the problem to the evaluation of $\|g_{m+1}(s^{-1}\mathbf{B})\|$. In fact

$$\left\|g_{m+1}(s^{-1}\mathbf{B})\right\| \leq -\log(1 - \left\|g_{m+1}(s^{-1}\mathbf{B})\right\|)$$

therefore if 3.6 holds, then both 3.5a and 3.5b hold true at the same time.

To verify 3.6 we estimate the quantity $\|g_{m+1}(s^{-1}\mathbf{B})\|$ using a routine for the 1-norm estimation such as `normest1` from [29]. This routine only requires to perform products between $s^{-1}\mathbf{B}$ (or its transpose) and a vector, allowing us to estimate $\|g_{m+1}(s^{-1}\mathbf{B})\|$ without actually forming $g_{m+1}(s^{-1}\mathbf{B})$. Clearly, we cannot actually estimate the quantity $\|g_{m+1}(s^{-1}\mathbf{B})\|$ for any candidate pair (m, s) for it would be too computationally demanding. Therefore we have to carefully plan a strategy, but first we need to determine how large we want to allow the approximation degree m to grow.

3.3.1 Maximum approximation degree

In this section, we explain how to determine a convenient maximum degree of approximation M for the approximation of $e^{\mathbf{A}}$ in the case the user would not specify a preferred maximum approximation degree.

The maximum approximation degree M must be determined in function of the evaluation scheme in use and of the tolerance prescribed by the user. To do so consider for the moment the classical backward error analysis that aims to satisfy a slightly different criterion from (3.2), for it requires the norm of the backward matrix to be smaller than the shifted matrix's one

$$\|\Delta\mathbf{B}\| \leq \text{tol} \cdot \|\mathbf{B}\| .$$

The successive step is to consider the triangular inequality applied to the power series expansion of the backward error matrix as follows:

$$\|\Delta\mathbf{B}\| = \|h_{m+1}(\Delta\mathbf{B})\| = \left\| \sum_{k=0}^{\infty} f_{k,m}\mathbf{B}^k \right\| \leq \sum_{k=0}^{\infty} |f_{k,m}| \, \|\mathbf{B}\|^k =: \tilde{h}_{m+1}(\|\mathbf{B}\|),$$

therefore we know that if the right hand side is smaller than $\text{tol} \cdot \|\mathbf{B}\|$ then so it is $\|\Delta\mathbf{B}\|$. A common strategy is to find the smallest positive scalar $\theta_{m,\text{tol}}$ for which

$$\tilde{h}_{m+1}(\theta_{m,\text{tol}}) = \text{tol} \cdot \theta_{m,\text{tol}}$$

so that if $\|\mathbf{B}\| \leq \theta_{m,\text{tol}}$ then is implied that $\|\Delta\mathbf{B}\|$ is smaller than $\text{tol} \cdot \|\mathbf{B}\|$. If instead $\|\mathbf{B}\| > \theta_{m,\text{tol}}$ one can consider to use a larger candidate degree or to pick s as 2 to the power of

$$\left\lceil \log_2\left(\frac{\|\Delta\mathbf{B}\|}{\theta_{m,\text{tol}}}\right) \right\rceil,$$

that is the smallest power of 2 such that makes $\|s^{-1}\mathbf{B}\| \leq \theta_{m,\text{tol}}$.

m	2	4	6	9	12	16	20	25	30
θ	2.6e-08	3.4e-04	9.1e-03	8.9e-02	3.0e-01	7.8e-01	1.4e-00	2.4e-00	3.5e-00

Table 3.2: Values $\theta_{m,\text{tol}}$ for $\text{tol} = 2^{-53}$ and $m = \deg_{\text{PS}}(\text{MP})$ with $\text{MP} = 1, 2, \ldots, 9$.

By giving a quick glance to the values $\theta_{m,\text{tol}}$ for $\text{tol} = 2^{-53}$ and $m = \deg_{\text{PS}}(\text{MP})$ with $\text{MP} = 1, 2, \ldots, 9$ that we report in Table 3.2, it appears clear that for small values of m we have a rapid increment of $\theta_{m,\text{tol}}$, while for larger candidate degrees this increment is more modest. Hence, it is evident that the feasible couples with small m can be characterized by an unnecessarily large scaling parameter, leading to an higher number of matrix products and to an higher risk of incurring into the *overscaling* phenomenon that is when there is loss of accuracy due to a choice of a scaling parameter that is too large.

The rule of thumb is to consider as maximum scaling degree the last degree M for which $\theta_{M,\text{tol}}$ is at least twice the value of the previous candidate degree. Doing so will surely lead to a reduced scaling parameter and overall cost. This criterion is bound to the evaluation scheme, in fact, an evaluation scheme with a different efficiency (intended as the degree reached over the number of matrix products performed) may lead to different maximum

approximation degrees M. For approximations with tolerance set to 2^{-53} and evaluated with the Paterson–Stockmeyer scheme, the maximum approximation degree should be $M = 16$ (and MP $= 7$), on the contrary in the literature the maximum approximation degree considered is $M = 30$ (and MP $= 9$). Similarly, with the Sastre evaluation scheme, M should be equal to 21 (MP $= 5$) while, in [57], degree $M = 30$ (and MP $= 7$) was instead preferred.

This choice is due to the fear of incurring into the above-mentioned overscaling phenomenon, and a larger M, in fact, may help in mitigating such a problem. Although the parameters' selection algorithm from [9] (that we further perfectioned in this work) greatly reduces the risk of incurring into the overscaling phenomenon, we too adopt for any computation with tolerance larger than 2^{-53} the Sastre evaluation scheme with $M = 30$, while otherwise we employ the Paterson–Stockmeyer evaluation scheme with $M = 30$.

In any other case, we compute M in run time by searching for the first MP such that $\theta_{\deg_{\mathrm{PS}}(\mathrm{MP}+1),\mathrm{tol}} > 1.45 \cdot \theta_{\deg_{\mathrm{PS}}(\mathrm{MP}),\mathrm{tol}}$ where 1.45 is the ratio between $\theta_{30,\mathrm{tol}}$ and $\theta_{25,\mathrm{tol}}$, following in this way the rule of the thumb adopted in the literature for mitigating the incidence of the overscaling phenomenon. Also, we require that $\theta_{\deg_{M,\mathrm{tol}}} \geq 1$ or our numerical experience suggests that we probably incur in overscaling phenomena. To quickly compute M, we take into account an approximation of the values $\theta_{m,\mathrm{tol}}$. Consider, for small values of $\|g_{m+1}(\mathbf{B})\|$, the approximation

$$\|\Delta\mathbf{B}\| = \|h_{m+1}(\Delta\mathbf{B})\| = \|\log(\mathbf{I} - g_{m+1}(\mathbf{B}))\| \approx \|g_{m+1}(\mathbf{B})\|$$

which can be bound by

$$\|g_{m+1}(\mathbf{B})\| \leq \left\|\mathrm{e}^{-\mathbf{B}}\right\| \|r_m(\mathbf{B})\| \leq \mathrm{e}^{\|\mathbf{B}\|} r_m(\|\mathbf{B}\|)$$

that in turn can be approximated by a function of $\|\mathbf{B}\|$:

$$\dot{\tilde{h}}_{m+1}(\|\mathbf{B}\|) := \mathrm{e}^{\|\mathbf{B}\|} \frac{\|\mathbf{B}\|^{m+1}}{(m+1)!}.$$

We can therefore compute an approximation of $\theta_{m,\mathrm{tol}}$ by finding the scalar realizing

$$\dot{\tilde{h}}_{m+1}(x) = \mathrm{tol} \cdot x.$$

We remind that even though our numerical experience confirmed that the approximation of $\theta_{m,\mathrm{tol}}$ is really close to its real value, a gross mistake in its estimate would not cause any major problem to the accuracy of our approximation. In fact, we are just trying to estimate which values of the parameter M would potentially lead to the smallest cost.

Anyway, some precision is welcomed, therefore we carefully engineered the following MATLAB code for finding our approximation of $\theta_{m,\mathrm{tol}}$ avoiding overflow or underflow phenomena.

```
01. function c = bea_tay_approx( m, log_tol )
02. % Compute the estimate c of  theta_{m,exp(log_tol)} relative to
03. % the Taylor approximation of degree m with tolerance exp(log_tol).
04. tol = log_tol + gammaln( m + 2 );
05. a = 0; b = 100; c = 0; c0 = 1;
```

```
06. it = 0; maxit = max( 100, abs( log_tol ) );
07. while ( ( abs( c0 - c ) > ( c * 1e-2 ) ) && ( it < maxit ) )
08.   it = it + 1;
09.   c0 = c;
10.   c = ( a + b ) / 2;
11.   if  ( ( c + m * log( c ) ) < tol )
12.     a = c;
13.   else
14.     b = c;
15.   end
16. end
```

3.3.2 Krylov subspace projection of the backward error

Before the digression on the choice of the maximum approximation parameter M, we agreed over the necessity to plan a strategy to find the best feasible pair (m, s) reducing as much as possible the number candidate couples for which we estimate $\|g_{m+1}(s^{-1}\mathbf{B})\|$. To do so, we have to take one step back: our numerical experience tells us that in the vast majority of the cases the quantity $\|h_{m+1}(s^{-1}\mathbf{B})\|$ is either several orders of magnitude larger than the threshold $\min\{1, \mathrm{tol} \cdot s^{-1} \|\mathbf{A}\|\}$ or several orders of magnitude smaller. In other words, it is very rare that $\|h_{m+1}(s^{-1}\mathbf{B})\|$ and $\min\{1, \mathrm{tol} \cdot s^{-1} \|\mathbf{A}\|\}$ are of comparable magnitude. We believe that this phenomenon was never fully exploited before thus we designed a way to do that.

The process starts by selecting a vector v outside the kernel of $\mathbf{B}$. In order to be sure of that, we pick a randomly generated vector and we multiply the matrix $\mathbf{B}$ into it. The resulting vector is in the image $\mathbf{B}$ and therefore it cannot belong to the kernel. Then, without loss of generality, we normalize v in its 2 norm obtaining v_1. We proceed by running few Arnoldi (or Lanczos) iterations (say $\kappa < m$) with $\mathbf{B}$ and v_1 in order to obtain

$$\mathbf{B}\mathbf{V}_\kappa = \mathbf{V}_\kappa \mathbf{H}_\kappa + h_{\kappa+1,\kappa} v_{\kappa+1} e_\kappa^T$$

with $\mathbf{H}_\kappa$ Hessenberg matrix, $\mathbf{V}_\kappa = [v_1, v_2, \ldots, v_\kappa]$ rectangular matrix such that $\mathbf{V}_\kappa^H \mathbf{V}_\kappa$ is the size κ identity matrix and e_κ is the κth column of $\mathbf{I}_\kappa$. In case $\mathbf{B}$ is an Hermitian or skew-Hermitian matrix, we recall that a simpler and faster process, called Lanczos process, can be used to compute the same decomposition above. The meaning of such decomposition is that the matrix $\mathbf{H}_\kappa$ represents the projection over the smaller Krylov subspace

$$\mathrm{span}\{v_1, \mathbf{B}v_1, \ldots, \mathbf{B}^\kappa v_1\} = \mathrm{span}\{v_1, v_2, \ldots, v_{\kappa+1}\}$$

of the matrix $\mathbf{A}$. This projection is numerically attractive because it allows to approximate the action of function of matrices on vectors, such as $f(\mathbf{B})v_1$, by means of $\mathbf{V}_\kappa f(\mathbf{H}_\kappa) e_1$, where $\mathbf{H}_\kappa$ is of smaller dimension of $\mathbf{B}$ and therefore easier to handle.

Let us come back to our problem: provided that we have a candidate parameter scaling s, if we consider the chain of inequalities

$$\sqrt{n} \left\| h_{m+1}(s^{-1}\mathbf{B}) \right\|_1 \geq \left\| h_{m+1}(s^{-1}\mathbf{B}) \right\|_2 = \sup\{ \left\| h_{m+1}(s^{-1}\mathbf{B}) x \right\|_2 : x \in \mathbb{C}^N \wedge \|x\|_2 = 1 \}$$

whose right hand side is, for the definition of superior, greater or equal than the choice of

$x = v_1$, giving us

$$\sqrt{n}\left\|h_{m+1}(s^{-1}\mathbf{B})\right\|_1 \geq \left\|h_{m+1}(s^{-1}\mathbf{B})v_1\right\|_2.$$

Therefore, we can exploit the Krylov subspace approximation

$$\left\|h_{m+1}(s^{-1}\mathbf{B})v_1\right\|_2 \approx \left\|\mathbf{V}_\kappa h_{m+1}(s^{-1}\mathbf{H}_\kappa)e_1\right\|_2$$

to obtain a cheap underestimate of the norm of the backward matrix for any given couple (m, s). Clearly if $\|\mathbf{V}_\kappa h_{m+1}(s^{-1}\mathbf{H}_\kappa)e_1\|_2$ exceeds $\min\{1, \mathrm{tol} \cdot s^{-1}\|\mathbf{A}\|\}$, even more will $\|h_{m+1}(s^{-1}\mathbf{B})\|_1$ and therefore the couple (m, s) is clearly not feasible.

As the reader may have understood, we need to compute $\|\mathbf{V}_\kappa h_{m+1}(s^{-1}\mathbf{H}_\kappa)e_1\|_2$ repeatedly and it can be quite expensive. We can tackle this issue easily by computing once and for all the diagonal factorization of $\mathbf{H}_\kappa$, i.e.

$$\mathbf{H}_\kappa = \mathbf{Q}_\kappa \mathbf{D}_\kappa \mathbf{Q}_\kappa^{-1}$$

with $\mathbf{D}_\kappa$ diagonal matrix. If the entries $h_{j+1,j}$ of $\mathbf{H}_\kappa$ are different from zero for $j = 1, 2, \ldots, \kappa - 1$, then the matrix $\mathbf{H}_\kappa$ has κ different roots and therefore it is diagonalizable. In case we encounter a $j \leq \kappa$ for which $h_{j+1,j} = 0$, then $\mathbf{H}_{j-1}$ is diagonalizable and we set $\kappa = j - 1$. Then we know that

$$\begin{aligned}
\left\|\mathbf{V}_\kappa h_{m+1}(s^{-1}\mathbf{H}_\kappa)e_1\right\|_2 &= \sqrt{e_1^T h_{m+1}(s^{-1}\mathbf{H}_\kappa)^H \mathbf{V}_\kappa^H \mathbf{V}_\kappa h_{m+1}(s^{-1}\mathbf{H}_\kappa)e_1} \\
&= \sqrt{e_1^T h_{m+1}(s^{-1}\mathbf{H}_\kappa)^H h_{m+1}(s^{-1}\mathbf{H}_\kappa)e_1} \\
&= \left\|h_{m+1}(s^{-1}\mathbf{H}_\kappa)e_1\right\|_2 \\
&= \left\|h_{m+1}(s^{-1}\mathbf{Q}_\kappa \mathbf{D}_\kappa \mathbf{Q}_\kappa^{-1})e_1\right\|_2 \\
&= \left\|\mathbf{Q}_\kappa h_{m+1}(s^{-1}\mathbf{D}_\kappa)\mathbf{Q}_\kappa^{-1}e_1\right\|_2
\end{aligned}$$

where $\mathbf{Q}_\kappa^{-1}e_1$ can be computed once and for all at the beginning. This is really cheap to compute, in fact κ is chosen very small and for any function $f(\cdot)$ we have that $f(s^{-1}\mathbf{D}_\kappa)$ is the diagonal matrix having on its diagonal scalar functions of the diagonal entries of $\mathbf{D}_\kappa$.

Now that we have a quick procedure for ruling out unfeasible couples (m, s) we can follow a simple strategy that aims to narrow down the list of the candidate couples. Briefly, this strategy consists in setting $s = 1$, MP $= 1$, $m = \deg(\mathrm{MP})$ and to observe the following procedure

1. - if the inequality

   $$\left\|\mathbf{Q}_\kappa h_{m+1}(s^{-1}\mathbf{D}_\kappa)\mathbf{Q}_\kappa^{-1}e_1\right\|_2 / \sqrt{n} \leq \min\{1, \mathrm{tol} \cdot s^{-1}\|\mathbf{A}\|\}$$

   holds true, exit the procedure

2. - if $m < M$, increment MP by 1, otherwise double s and set MP to 1

3. - set $m = \deg(\mathrm{MP})$

4. - return to 1.

The couple (m, s) exiting this procedure constitutes the initial educated guess for the rigorous procedure of selection of a convenient feasible couple that we introduce in the next subsection.

Before proceeding, we explain how we can speed up even more the procedure we just introduced by guessing an initial scaling parameter $s = s_0$ instead of simply fixing $s = 1$. To do so, we adapt an idea usually exploited by the classical backward error analysis for Taylor method that we introduced in the past subsection: instead of comparing $\|s^{-1}\mathbf{B}\|$ with the values $\theta_{m,\text{tol}}$ one can use instead the values

$$s^{-1}\alpha_p(\mathbf{B}) = \max(s^{-1}\left\|\mathbf{B}^p\right\|^{\frac{1}{p}}, s^{-1}\left\|\mathbf{B}^{p+1}\right\|^{\frac{1}{p+1}})$$

provided $p(p-1) \leq m-1$ (for more details see [1]). The reason behind this is that $\alpha_p(\mathbf{B})$ may be much smaller than $\|\mathbf{B}\|$. The problem of this approach, that is used by the vast majority of our competitors (for example [1, 9, 18, 51, 57]), is that the computation of the values $\alpha_p(\mathbf{B})$ requires a lot of time, sometimes compromising the efficiency of the routine. In particular, techniques exploiting the behavior of the sequences of $\alpha_p(\mathbf{B})$ have been exploited and many efforts have been spent in order to improve the constraint $p(p-1) \leq m-1$ and to be allowed to use a higher value p for a fixed approximation degree m. In fact, for p growing we have that $\alpha_p(\mathbf{B})$ approaches $\rho(\mathbf{B})$, that is the technical limit of this approach.

Our idea is exactly to use $\rho(\mathbf{B})$, or at least our best approximation of $\rho(\mathbf{B})$ that is $\rho(\mathbf{H}_\kappa)$ (we recall that the eigenvalues of $\mathbf{H}_\kappa$ approximate those of $\mathbf{B}$) and we have already computed it since at this point we have already the diagonal factorization of $\mathbf{H}_\kappa$.

A concern may be that $\rho(\mathbf{B})$ is in general smaller than the value of $\alpha_p(\mathbf{B})$ that we are technically authorized to employ. Moreover, the approximation $\rho(\mathbf{H}_\kappa)$ is generally smaller than $\rho(\mathbf{B})$. Anyways, this is not a problem because at the moment we are still in the guessing phase of the scaling parameter. Once we fix $s = s_0$ with s_0 determined using $\rho(\mathbf{H}_\kappa)$, if s turns out to be an inappropriate choice, then the procedure we just designed will automatically increase it.

After we run the aforementioned procedure, provided that we never had to double the scaling parameter at step 2., we try to split s in half and to rerun the procedure with $s/2$ hoping that the exiting candidate couple still has $s/2$ as scaling parameter. If this is the case, we try over and over the same trick so that we possibly obtain an even smaller scaling parameter.

As a final note, we remark that the candidate couple (m, s) exiting the procedure is in no way granted to be feasible. We let the charge of determining the feasibility of (m, s) to the "unprojected" on-the-fly backward error estimate procedure that we adopted from [9] and that we now introduce.

3.3.3 Accuracy check: on-the-fly backward error estimate

Following the steps described in the past subsection, we obtained three key parameters: the maximum interpolation degree M and m, s forming the couple (m, s), which could potentially be feasible.

It is now the moment to estimate $\|g_{m+1}(\mathbf{B})\|$ and to make sure that equality (3.6) holds true. If it does for (m, s), then this couple can be officially declared feasible and we

proceed to the evaluation part. If it is not, similarly to the procedure described in the past subsection, we increase the computational effort by considering a larger candidate degree m and, in case m was already equal to M, we double the scaling parameter, this time without resetting m to the smallest candidate parameter.

The decision of not resetting m to the smallest candidate parameter is due to the fact we expect (m, s) to be already close to a feasible couple or even to be already a feasible couple. Hence we do not like the idea of lingering on the (costly, this time) decision process. On the contrary, we acknowledge the fact that s was chosen too small and we engage an exit strategy that returns us a feasible couple as quickly as possible.

As a consequence, once we start this new decision process, we never decrease the scaling parameter m, therefore we can start to compute the z powers of $\mathbf{B}$ needed by the evaluation scheme in order to evaluate the polynomial $T_m(s^{-1}\mathbf{B})$. We can exploit this situation by regrouping $g_{m+1}(s^{-1}\mathbf{B})$ à la Paterson-Stockmeyer and then by applying the triangular inequality obtaining

$$\hat{g}_{m+1,z}(s^{-1}\mathbf{B}) := \sum_{j=r}^{\infty} \left\| (s^{-z}\mathbf{B}^z)^j \mathbf{P}_j \right\|, \quad r = \lfloor m/z \rfloor$$

where

$$\mathbf{P}_j := \sum_{i=1}^{z} b_{jz+i,m} s^{-i} \mathbf{B}^i$$

and $\|g_m(s^{-1}\mathbf{B})\| \leq \hat{g}_{m+1,z}(s^{-1}\mathbf{B})$. We point out that the summation in the definition of $\hat{g}_{m+1}(s^{-1}\mathbf{B})$ starts from r because from the analytic formula of the coefficients b_k we know that the first m are all equal to zero.

It is evident that $\hat{g}_{m+1,z}(s^{-1}\mathbf{B})$ is far cheaper than $\|g_m(s^{-1}\mathbf{B})\|$ to evaluate, in fact the matrices $\mathbf{P}_j$ can be composed without needing of any additional matrix product and the quantity $\|(s^{-z}\mathbf{B}^z)^j \mathbf{P}_j\|$ we recall that can be estimated with a routine for the 1-norm estimate.

Let us define $\delta_j(m, s) := \|(s^{-z}\mathbf{B}^z)^j \mathbf{P}_j\|$, in our numerical experience, the vast majority of the times we have that if $\delta_{j+1}(m, s) < \delta_j(m, s)$ then $\delta_j(m, s)$ is several orders of magnitude larger than $\delta_j(m, s)$. Therefore after we overestimate carefully and sharply $\delta_r(m, s)$ we can be rougher and overestimate a bit more $\delta_{r+1}(m, s)$ in exchange of cheaper computations. To do so we consider

$$\bar{\delta}_{r+1}(m, s) = \left\| (s^{-z}\mathbf{B}^z)^r \right\| \cdot \left\| s^{-z}\mathbf{B}^z \mathbf{P}_{r+1} \right\|$$

where both $\|(s^{-z}\mathbf{B}^z)^r\|$ and $\|s^{-z}\mathbf{B}^z \mathbf{P}_{r+1}\|$ are estimated with `normest1` without performing any matrix product. For all the $\delta_j(m, s)$ with $j > r + 1$ instead we use the even rougher overestimates:

$$\hat{\bar{\delta}}_{r+1}(m, s) = \left\| (s^{-z}\mathbf{B}^z)^r \right\| \cdot \left\| s^{-z}\mathbf{B}^z \right\| \cdot \|\mathbf{P}_{r+1}\|$$

that do not even involve the matrix 1-norm estimate process anymore since $\|(s^{-z}\mathbf{B}^z)^r\|$ was already estimated at step $j = r + 1$ and the norm of $s^{-z}\mathbf{B}^z$ is directly available.

Therefore, we plan to check if the couple (m, s) satisfies (3.6) by checking if we have that

$$\delta_r(m,s) + \bar{\delta}_{r+1}(m,s) + \sum_{k=r+2}^{K} \hat{\bar{\delta}}_k(m,s) \leq \min(1, \mathrm{tol} \cdot s^{-1} \|\mathbf{A}\|)$$

where K is such that, in machine working precision, we have

$$\delta_r(m,s) + \bar{\delta}_{r+1}(m,s) + \sum_{k=r+2}^{K} \hat{\bar{\delta}}_k(m,s) = \delta_r(m,s) + \bar{\delta}_{r+1}(m,s) + \sum_{k=r+2}^{K-1} \hat{\bar{\delta}}_k(m,s)$$

so that we can do without the conjecture made in [9] that was aimed to arrest the summation in formula

$$\delta_r(m,s) + \bar{\delta}_{r+1}(m,s) + \sum_{k=r+2}^{\infty} \hat{\bar{\delta}}_k(m,s).$$

To resume, as we already set for ourselves at the beginning of this subsection, we observe the following strategy

1. - if the inequality

$$(\delta_r(m,s) + \bar{\delta}_{r+1}(m,s) + \sum_{k=r+2}^{K} \hat{\bar{\delta}}_k(m,s)) \leq \min\{1, \mathrm{tol} \cdot s^{-1} \|\mathbf{A}\|\}$$

   holds true, exit the procedure.

2. - if $m < M$, increment MP by 1, otherwise double s

3. - set $m = \deg(\mathrm{MP})$

4. - return to 1.

Then we proceed to the evaluation of $T_m(s^{-1}\mathbf{B})$ with the couple (m, s) exiting the procedure. We know that $T_m(s^{-1}\mathbf{B})^s$ is an accurate approximation of $\mathrm{e}^{\mathbf{A}}$.

Early terminating the evaluations of $\hat{\bar{\delta}}_k(m,s)$

In high precision arithmetic, the positive integer K defined as before may be very large, therefore we engineered an additional criterion for arresting the evaluation of the quantities $\hat{\bar{\delta}}_k(m,s)$ and declaring the couple (m,s) feasible.

Let us refer with $\Delta_q(m,s)$ to the sharpest overestimate possibly available of the first $q-r$ blocks of $\hat{g}_{m+1,z}(s^{-1}\mathbf{B})$. In our case this is:

$$\Delta_q(m,s) := \begin{cases} \delta_r(m,s), & q = r \\ \delta_r(m,s) + \bar{\delta}_{r+1}(m,s), & q = r+1 \\ \delta_r(m,s) + \bar{\delta}_{r+1}(m,s) + \sum_{k=r+2}^{q} \hat{\bar{\delta}}_k(m,s), & q > r+1 \end{cases}$$

If we apply the following inequality

$$\hat{g}_{m+1,z}(s^{-1}\mathbf{B}) = \sum_{j=r}^{\infty} \left\|(s^{-z}\mathbf{B}^z)^j\mathbf{P}_j\right\| \leq \Delta_q(m,s) + \sum_{j=q+1}^{\infty} \left\|(s^{-z}\mathbf{B}^z)^j\mathbf{P}_j\right\|$$

and consider that

$$\Delta_q(m,s) + \sum_{j=q+1}^{\infty} \left\|(s^{-z}\mathbf{B}^z)^j\mathbf{P}_j\right\| \leq \Delta_q(m,s) + \sum_{k=(q+1)z+1}^{\infty} b_k\alpha_p(s^{-1}\mathbf{B})^k,$$

it follows that, if

$$\sum_{k=(q+1)z+1}^{\infty} b_k\alpha_p(s^{-1}\mathbf{B})^k \leq \min\{1, \mathrm{tol}\cdot s^{-1}\left\|\mathbf{A}\right\|\} - \Delta_q(m,s),$$

then (3.6) is satisfied too. If, on the contrary, the above inequality is not satisfied we do not declare the unfeasability of (m,s), instead we just proceed to evaluate $\Delta_{q+1}(m,s)$. The last detail to take care about is how to evaluate

$$\sum_{k=(q+1)z+1}^{\infty} b_k\alpha_p(s^{-1}\mathbf{B})^k$$

without applying any conjecture for terminating the infinite summation. Analogously as before we can search for a positive integer K' such that

$$\sum_{k=(q+1)z+1}^{K'-1} b_k\alpha_p(s^{-1}\mathbf{B})^k = \sum_{k=(q+1)z+1}^{K'} b_k\alpha_p(s^{-1}\mathbf{B})^k$$

in working precision.

Differently from before, if K' is large we do not worry because now the terms $b_k\alpha_p(s^{-1}\mathbf{B})^k = b_k s^{-k}\alpha_p(\mathbf{B})^k$ are extremely cheap to compute once we spend the effort of computing $\alpha_p(\mathbf{B})$.

3.4 Numerical experiments

The numerical experiments' Section is divided into two main parts: in the first part, we are going to run our routine `expkptotf` (EXPonential Krylov Projection Taylor On-The-Fly) against the state-of-the-arts routines for the approximation of the matrix exponential in double precision arithmetic. These routines are:

- `exptayotf`, the algorithm of Caliari and Zivcovich (see [9]) whose on-the-fly backward error estimate algorithm has been improved in this work. Similarly to `expkptotf`, this routine is based on a scaling and squaring technique coupled with a shifted truncated Taylor series evaluated with the classical Paterson–Stockmeyer evaluation scheme.

- `expm_pol` the algorithm of Sastre, Ibáñez and Defez (see [57]), based on a scaling

and squaring technique coupled with a truncated Taylor series evaluated with the new special evaluation scheme that, due to its high efficiency, was also adopted in our routine.

- `exptaynsv3` the algorithm of Ruiz, Sastre, Ibáñez and Defez (see [51]), based on a scaling and squaring technique coupled with a truncated Taylor series evaluated with the classical Paterson–Stockmeyer evaluation scheme.

- `expm` the built-in function `expm` of MATLAB, which implements the algorithm of Al-Mohy and Higham (see [1]) and it is based on a scaling and squaring technique coupled with a Padé approximation of the exponential function.

In the second part of the numerical experiments instead, we are going to assess the performances of `expkptotf` for computations in arbitrary precision arithmetic. To do so we compare it with the state-of-the-arts routines that are dedicated to the approximation of the matrix exponential in multi precision environments. These routines are:

- once again `exptayotf`, the algorithm of Caliari and Zivcovich (see [9]), that, similarly to `expkptotf`, is meant to run in any given precision arithmetic without needing any component external to MATLAB in order to run.

- `expm_mp`, the routine of Fasi and Higham (see [18]) based on a scaling and squaring method coupled with four techniques for the approximation of the matrix exponential. These techniques characterize the four versions of this code that will constitute four of our competitors:

  - `exp_d`, based on [18, Algorithm 4.1] employing a diagonal Padé approximant
  - `exp_t`, based on [18, Algorithm 4.1] employing a truncated Taylor series
  - `exp_sp_d`, based on the Schur–Padé approach discussed in [18, Section 4.3] where, for the triangular Schur factor, [18, Algorithm 4.1] a diagonal Padé approximant is used.
  - `exp_sp_t`, based on the Schur–Padé approach discussed in [18, Section 4.3] where, for the triangular Schur factor, [18, Algorithm 4.1] a truncated Taylor series is used.

  These routines only run with the aid of a Multiprecision Computing Toolbox named Advanpix and they use the high precision arithmetic for bounding the forward error in an innovative way.

In this section we are going to assess the accuracy of the routines over the set $\mathcal{N} \cup \mathcal{H}$ and the speed of the routines over another set, that we call $\mathcal{M}$. This choice was made in order to make these experiments as comparable as possible with those from the most recent and relevant papers on the topic.

In fact, the set $\mathcal{N} \cup \mathcal{H}$ is exactly the same set used by the authors of [18] in order to assess the accuracy of their four versions of the routine `expm_mp`. This set is composed by the union of $\mathcal{N}$, that is a collection of 97 non-Hermitian matrices, and $\mathcal{H}$, that is a collection of 35 Hermitian matrices. The matrices in the set $\mathcal{N} \cup \mathcal{H}$ are of size ranging from 2 to 1000 and are taken from a collection of benchmark problems for the burnup

equations and from the MATLAB gallery function (see [18] for details), hence this is a formidable set for assessing the accuracy of a method.

On the other hand, the set $\mathcal{M}$ is exactly the same set used by the authors of [9] in order to prove that their routine, `exptayotf`, was accurate and fast for the approximation of the matrix exponential in double precision arithmetic . The matrices in the set $\mathcal{M}$ come from the Matrix Computation Toolbox [23] with the addition of few random complex matrices (see [9] for details). The property of this set (that by the way presents some intersections with $\mathcal{N} \cup \mathcal{H}$) that makes it so attractive for measuring the speed of a method is that the size of the matrices from $\mathcal{M}$ can be decided by the user and therefore it is possible to plot the evolution of the CPU time in function of the size of the matrices.

The four versions of the routine `expm_mp` use the Advanpix Multiprecision Computing Toolbox (version 4.4.7.12739) which provides the class `mp` to represent arbitrary precision floating-point numbers and overloads all the MATLAB functions they need in their implementations. In order to compare with them, the experiments from the part of this Section dedicated to the computations in arbitrary precision arithmetic are entirely run with the aid of the Multiprecision Computing Toolbox, although neither `exptayotf` nor `expkptotf` would need it. This toolbox allows the user to specify the number of decimal digits of working precision, but not the number of bits in the fraction of its binary representation, thus, in this section, similarly to [18], whenever we refer to d (decimal) digits of precision, we mean that the working precision is set using the command `mp.Digits(d)`.

The experiments were performed using the 64-bit (glxna64) version of MATLAB® 9.2 (R2017a) on a machine equipped with 16Gb of RAM and four Intel Core i7 processors running at 3.30GHz.

3.4.1 Tests in double precision arithmetic

In Figure 3.1a we compare the forward errors committed by the routines optimized for the double precision arithmetic, that we remind to be `expkptotf`, `exptayotf`, `exptaynsv3`, `expm_pol` and `expm`, on the matrix test sets $\mathcal{N}$ and $\mathcal{H}$. The matrices are sorted by decreasing condition number $\kappa_{\exp}(\mathbf{A})$, number that we estimated using the function `funm_condest1` from the Matrix Function Toolbox [23] on `expm`. The same data are displayed in Figure 3.1b as a performance profile: a point (γ, ρ) on a curve related to a method represents the fraction of matrices in the matrix collection $\mathcal{N} \cup \mathcal{H}$ for which the corresponding error is bounded by ρ times the error of the algorithm that delivers the most accurate result for that matrix.

In Figure 3.1a we observe that the errors of all the five routines are approximately bounded by $\kappa_{\exp}(\mathbf{A}) \cdot \mathrm{tol}$ and we can confirm the conclusion drawn in [18]: the algorithms based on truncated Taylor series are overall more accurate than those based on diagonal Padé approximants. The performance profile curve that appears to be the most favorable, although it does not clearly stand out, is the one of `exptayotf`, confirming the results displayed in [9].

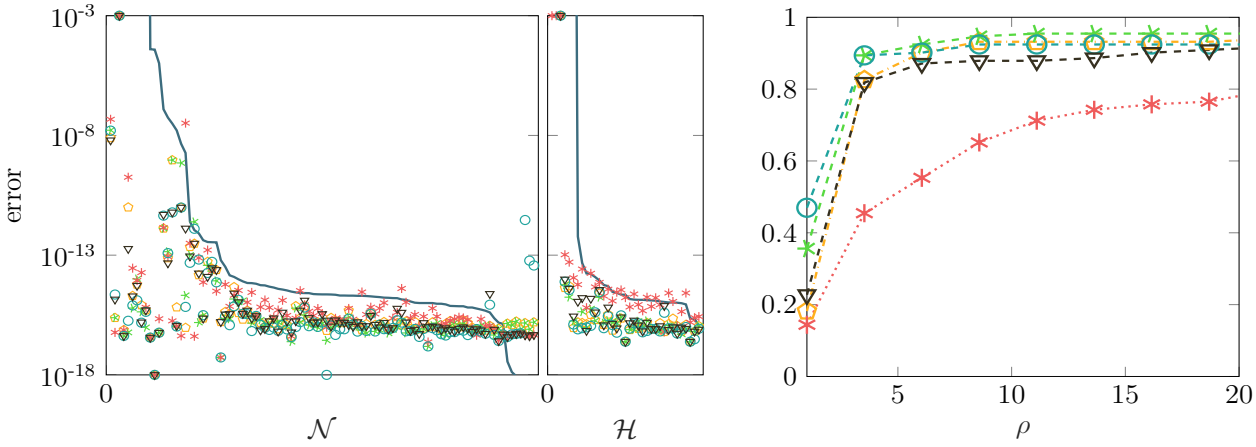
In Table 3.3, we captured how varies, for each routine, the average CPU time taken to compute an approximation of the matrix exponential of each matrix from $\mathcal{M}$ when the size N of the matrices from the test set increases. Also, we reported the maximum CPU time taken by each routine for computing the exponential of the matrices from $\mathcal{M}$ at each considered dimension. It appears clear from Table 3.3 that the routines `expkptotf` and

	$N = 16$		$N = 32$		$N = 64$	
	avg CPU	max CPU	avg CPU	max CPU	avg CPU	max CPU
`expkptotf`	9.72e-04	2.96e-03	1.21e-03	4.36e-03	3.99e-03	1.23e-02
`exptayotf`	4.57e-04	1.94e-03	6.88e-04	2.95e-03	3.43e-03	9.36e-03
`expm`	2.50e-04	6.12e-04	8.20e-04	2.05e-03	2.02e-03	5.51e-03
`exptaynsv3`	4.75e-04	2.04e-03	6.28e-04	3.08e-03	3.17e-03	1.10e-02
`expm_pol`	3.57e-04	1.84e-03	4.55e-04	2.18e-03	1.96e-03	8.72e-03

	$N = 128$		$N = 256$		$N = 512$	
	avg CPU	max CPU	avg CPU	max CPU	avg CPU	max CPU
`expkptotf`	5.09e-03	1.67e-02	1.87e-02	7.00e-02	1.17e-01	4.27e-01
`exptayotf`	4.74e-03	1.31e-02	1.98e-02	6.33e-02	1.36e-01	6.39e-01
`expm`	3.68e-03	1.35e-02	2.01e-02	5.67e-02	1.42e-01	5.73e-01
`exptaynsv3`	4.47e-03	1.32e-02	2.00e-02	8.33e-02	1.35e-01	5.50e-01
`expm_pol`	3.77e-03	1.65e-02	1.67e-02	6.99e-02	1.11e-01	4.31e-01

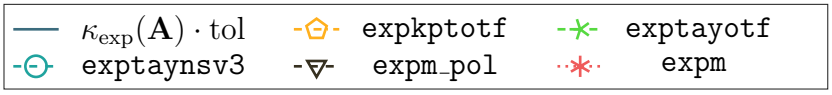
	$N = 1024$		$N = 2048$		$N = 4096$	
	avg CPU	max CPU	avg CPU	max CPU	avg CPU	max CPU
`expkptotf`	7.75e-01	2.46e+00	4.92e+00	1.52e+01	3.48e+01	1.07e+02
`exptayotf`	8.84e-01	2.77e+00	5.43e+00	1.74e+01	3.85e+01	1.16e+02
`expm`	9.28e-01	3.85e+00	5.85e+00	2.09e+01	4.14e+01	1.61e+02
`exptaynsv3`	8.97e-01	3.22e+00	5.69e+00	2.25e+01	4.01e+01	1.50e+02
`expm_pol`	7.44e-01	2.43e+00	4.72e+00	1.86e+01	3.34e+01	1.28e+02

Table 3.3: Average and maximum CPU time for computing the exponential of the matrices from the set $\mathcal{M}$ with size N set to 2^k ($k = 4, 5, \ldots, 12$). Calculations run with 16 digits and tol set to 2^{-53}.



(a) 16 digits (tol $= 2^{-53}$).

(b) Performance profile for $\mathcal{N} \cup \mathcal{H}$.



(c) Legend for the data in (a)-(b).

Figure 3.1: Left: forward error (y-axis) of the methods on the matrices in the test sets. Right: corresponding performance profiles for the matrices in $\mathcal{N} \cup \mathcal{H}$.

`expmpol` stand out as the fastest when N increases. This is due to the high efficiency of the Sastre evaluation scheme developed in [56] and adopted by both routines. On the other hand, we observe that on matrices of small size the routine `expkptotf` suffers particularly. This is because running the Arnoldi iterations and searching for the eigenvalues of $\mathbf{H}_k$ represents a fixed cost that penalizes the routine especially for matrices of small dimension. While this problem could be easily tackled by avoiding to project the backward error on the Krylov subspace for those matrices that are so small that the classical on-the-fly backward error estimates is not excessively slow, this is out of the scope of this manuscript. A careful implementation oriented to be commercialized would instead take this aspect in account, as well as considering the Sastre evaluation scheme for degrees larger than 30 in order to make it available for computations in multiprecision arithmetic, that constitute the next topic treated in this Section.

3.4.2 Tests in multiple precision arithmetic

In Figures 3.2a, 3.2c, 3.2e we compare the forward errors committed by the routines designed for running in multiprecision arithmetic, that we remind to be `expkptotf`, `exptayotf`, `exp_sp_t`, `exp_t`, `exp_sp_d` and `exp_`, on the matrix test sets $\mathcal{N}$ and $\mathcal{H}$. The experiment was repeated performing the calculations with 64, 256 and 1024 digits and setting the tolerance respectively to $2^{-213}, 2^{-851}$ and 2^{-3402}. The matrices are sorted by decreasing condition number $\kappa_{\exp}(\mathbf{A})$, number that we estimated using the function `funm_condest1` from the Matrix Function Toolbox [23] on `expm`. The same data are displayed in Figures 3.2b, 3.2d, 3.2f as a performance profile: a point (γ, ρ) on a curve related to a method represents the fraction of matrices in the matrix collection $\mathcal{N} \cup \mathcal{H}$ for which the corresponding error is bounded by ρ times the error of the algorithm that

delivers the most accurate result for that matrix.

In Figures 3.2a, 3.2c, 3.2e we observe that the errors of all the six routines are approximately bounded by $\kappa_{\exp}(\mathbf{A}) \cdot \text{tol}$. Once again we can confirm the conclusions drawn in [18]: the algorithms based on truncated Taylor series are overall more accurate than those based on diagonal Padé approximants and the algorithms based on the Schur decomposition of $\mathbf{A}$ tend to give larger errors, with sensibly worse performance profile curves. From Figures 3.2b, 3.2d, 3.2f we can deduce that the routines based on the on-the-fly estimate of the backward error, namely `expkptotf` and `exptayotf`, stand out as the routines with the most favorable performance profile curve, in particular, it is `expkptotf` the routine with the most favorable performance profile curve.

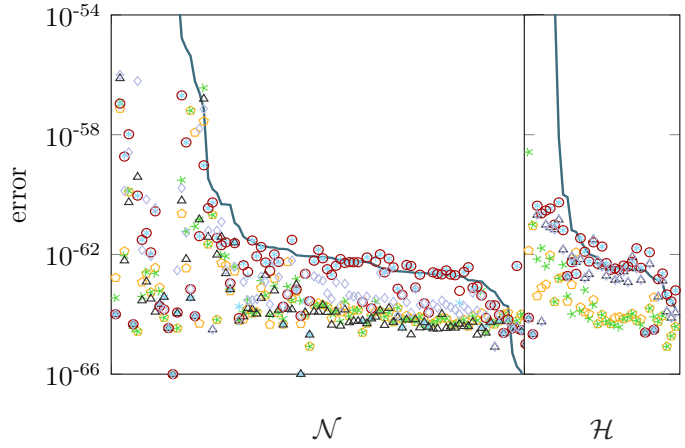
In Table 3.4 we captured how varies, for each routine, the average CPU time taken to compute an approximation of the matrix exponential of each matrix from $\mathcal{M}$ when the size N of the matrices from the test set and when the number of digits with which we run the calculations increase. Also, we reported the maximum CPU time taken by each routine for computing the exponential of the matrices from $\mathcal{M}$ at each considered dimension and number of digits. The reported CPU time is an impartial measurement of the performances of algorithms which are characterized by structurally different ways to choose approximation degrees and scaling parameters.

It appears clear from Table 3.4 that the routine `expkptotf` stands out as the fastest routine in every case but two, namely the case of calculations performed with 64 digits and tolerance set to 2^{-213} and matrix size N equal to 16 and 32. In the double precision arithmetic case in which `expkptotf` was standing out as one of the fastest because of Sastre's evaluation scheme. This explanation is not satisfactory now, in fact, `expkptotf` does not make use of such an evaluation scheme when the required tolerance is lower than 2^{-53}. It is instead thanks to the new Krylov projection of the backward error that the routine `expkptotf` turns out to be the fastest, for it contributed to speed up the process of selecting a convenient feasible couple (m, s).

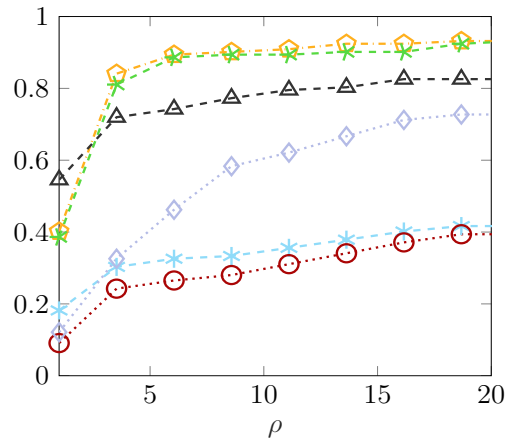
3.5 Conclusions

We have confirmed that the on-the-fly backward error estimate is a quick, robust and reliable way to determine the approximation parameters m and s. In particular, we designed a technique that consists in projecting the backward error over a small Krylov subspace speeding up the process of selection of the parameters m and s. In addition to that, we proved that the strong decay rate of the terms of the backward error polynomial can be exploited to our advantage in order to boost even more the process of estimating the backward error on-the-fly. Finally, we adopted a recently developed polynomial evaluation scheme for computations run with tolerance larger than the double precision arithmetic roundoff. Such a scheme further contributed to improving the performances of our routine `expkptotf`.

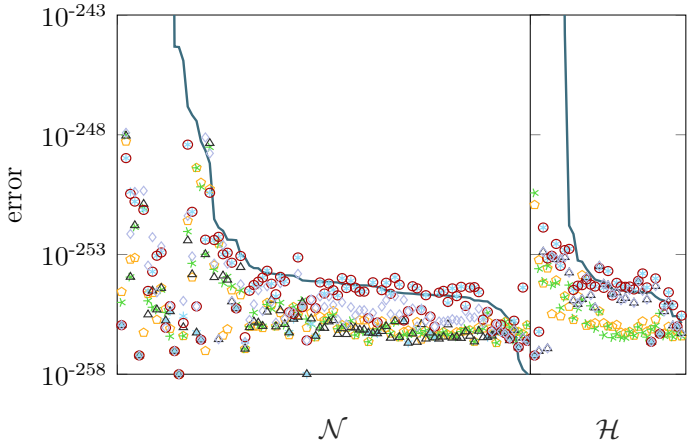
As we mentioned in the introduction of this manuscript, the high precision computation of the matrix exponential handling a low number of digits is a very important task. For motivating this statement we made the example of the computation of the table of the divided differences by Opitz theorem and the computation of the exponential of Hessenberg matrices appearing in Krylov approximation of the action of the matrix exponential. Both tasks must be performed with the highest level of accuracy possible



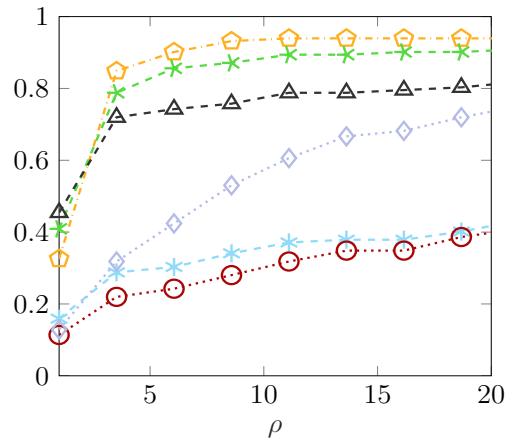
(a) 64 digits ($\text{tol} = 2^{-213}$).



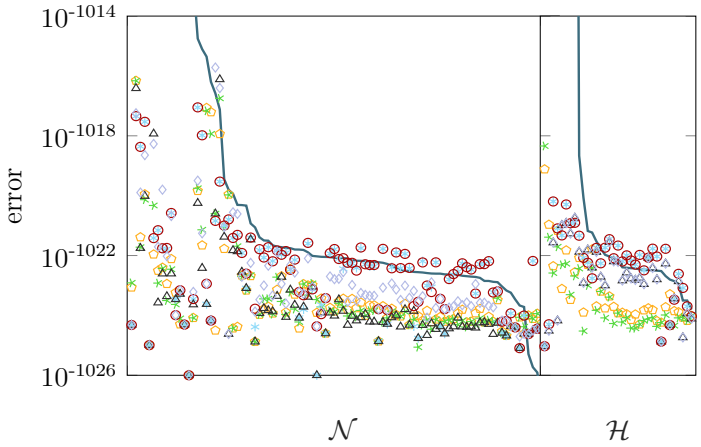
(b) Performance profile for $\mathcal{N} \cup \mathcal{H}$.



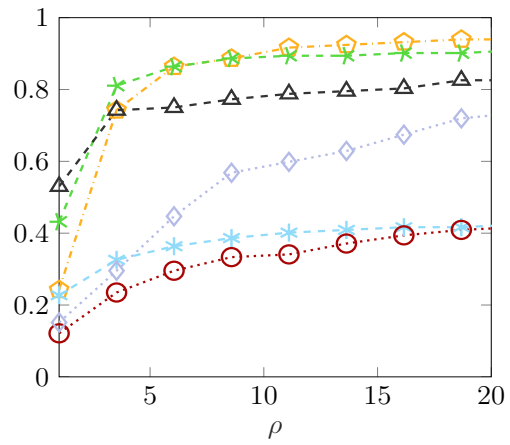
(c) 256 digits ($\text{tol} = 2^{-851}$).



(d) Performance profile for $\mathcal{N} \cup \mathcal{H}$.



(e) 1024 digits ($\text{tol} = 2^{-3402}$).



(f) Performance profile for $\mathcal{N} \cup \mathcal{H}$.

(g) Legend for the data in (a)–(f).

Figure 3.2: Left: forward error (y-axis) of the methods on the matrices in the test sets. Right: corresponding performance profiles for the matrices in $\mathcal{N} \cup \mathcal{H}$.

	$N = 16$		$N = 32$		$N = 64$	
	avg CPU	max CPU	avg CPU	max CPU	avg CPU	max CPU
`expkptotf`	6.85e-02	1.30e-01	1.44e-01	4.10e-01	5.83e-01	2.44e+00
`exptayotf`	5.36e-02	1.33e-01	1.35e-01	4.37e-01	5.83e-01	2.64e+00
`exp_sp_t`	1.58e-01	2.23e-01	4.28e-01	7.43e-01	2.15e+00	3.38e+00
`exp_t`	8.62e-02	1.30e-01	1.57e-01	3.87e-01	6.38e-01	2.47e+00
`exp_sp_d`	2.48e-01	4.13e-01	6.42e-01	1.02e+00	3.35e+00	5.01e+00
`exp_d`	1.49e-01	2.23e-01	3.03e-01	6.63e-01	1.24e+00	3.62e+00

(a) 64 digits (tol = 2^{-213})

	$N = 16$		$N = 32$		$N = 64$	
	avg CPU	max CPU	avg CPU	max CPU	avg CPU	max CPU
`expkptotf`	1.37e-01	2.70e-01	3.45e-01	1.12e+00	1.62e+00	7.01e+00
`exptayotf`	1.43e-01	3.50e-01	4.09e-01	1.45e+00	1.85e+00	7.98e+00
`exp_sp_t`	3.56e-01	7.53e-01	1.11e+00	2.22e+00	6.15e+00	9.61e+00
`exp_t`	1.96e-01	3.97e-01	4.19e-01	1.12e+00	1.84e+00	7.06e+00
`exp_sp_d`	7.33e-01	1.19e+00	2.05e+00	3.26e+00	1.11e+01	1.57e+01
`exp_d`	4.19e-01	7.90e-01	9.61e-01	2.90e+00	4.50e+00	1.34e+01

(b) 256 digits (tol = 2^{-851})

	$N = 16$		$N = 32$		$N = 64$	
	avg CPU	max CPU	avg CPU	max CPU	avg CPU	max CPU
`expkptotf`	6.69e-01	1.94e+00	2.37e+00	9.19e+00	1.26e+01	5.63e+01
`exptayotf`	7.55e-01	2.45e+00	2.94e+00	1.09e+01	1.46e+01	6.40e+01
`exp_sp_t`	2.02e+00	1.11e+01	9.76e+00	4.27e+01	6.26e+01	1.72e+02
`exp_t`	8.32e-01	1.71e+00	2.67e+00	7.58e+00	1.41e+01	5.31e+01
`exp_sp_d`	3.92e+00	1.32e+01	1.67e+01	5.12e+01	9.25e+01	2.04e+02
`exp_d`	1.86e+00	4.31e+00	6.30e+00	1.82e+01	3.18e+01	1.06e+02

(c) 1024 digits (tol = 2^{-3402})

Table 3.4: Average and maximum CPU time for computing the exponential of the matrices from the set $\mathcal{M}$ with size N set to 2^k ($k = 4, 5, 6$). Calculations run with 64, 256, 1024 digits and tol set to $2^{-213}, 2^{-851}, 2^{-3402}$.

in the lowest time possible for they constitute the core of high performance computing routines. For this reason one, generally cannot afford to compute the exponential of such matrices with a large number of digits and successively cast the result in double or single precision arithmetic.

Therefore routines able to compute the matrix exponential with high accuracy only handling computations in a low number of digits, such as `expkptotf`, are of primary importance. As of today, no existing routine except for `expkptotf` and `exptayotf` allow to accomplish this.

To illustrate this feature we briefly show the behavior of the multiprecision routines in the case of computing the divided differences of the exponential function at the Leja interpolation sets $\mathcal{L}_{32}$ and $\mathcal{L}_{128}$ via Opitz theorem requiring the resulting output to be respectively in single and double precision arithmetic. The Leja points are interpolation points heavily used in the applications for they asymptotically distribute as the Chebyshev points and enjoy nice numerical properties, one among the others the fact that $\mathcal{L}_k \subset \mathcal{L}_{k+1}$. Hence this set of points is reasonable but our numerical experience tells us that any other set of point would lead to the same conclusion. We recall that the Opitz theorem states that the divided differences $d_0[z_0], d_1[z_0, z_1], \ldots, d_m[z_0, z_1, \ldots, z_m]$ of the exponential function over the interpolation set $\{z_0, z_1, \ldots, z_m\}$ can be computed as $\mathrm{e}^{\mathbf{Z}(z_0,z_1,\ldots,z_m)}e_1$ where

$$\mathbf{Z}(z_0, z_1, \ldots, z_m) = \begin{pmatrix} z_0 & & & \\ 1 & z_1 & & \\ & \ddots & \ddots & \\ & & 1 & z_m \end{pmatrix}$$

and e_1 is the first vector of the canonical basis of $\mathbb{C}^m$.

To be sure that the divided differences are accurately approximated we demand the tolerance to be set to the unit roundoff of the single and double precision arithmetic, i.e. $2^{-126} =$ `realmin('single')` for the single and respectively $2^{-1022} =$ `realmin('double')` for the double. In order to measure the accuracy of the routines, if `d` is the approximate data cast to respectively single or double data type, we define the maximum forward component-wise relative error $\mathrm{e}_{\mathrm{fcr}}(\mathtt{d})$ as

$$\mathrm{e}_{\mathrm{fcr}}(\mathtt{d}) = \max_{k=0,1,\ldots,m} \left\{ \left| \frac{d_k[z_0, z_1, \ldots, z_k] - \mathtt{d}_k}{d_k[z_0, z_1, \ldots, z_k]} \right| \right\}$$

where $\mathtt{d}_k$ is the $(k+1)$th component of the vector `d`. Clearly each exact component $d_k[z_0, z_1, \ldots, z_k]$ is rounded in order to match the data type of `d`. For the computations of the divided differences over $\mathcal{L}_{32}$ the routines `expkptotf` and `exptayotf` could handle the calculations using just 8 digits both committing a maximum forward component-wise error of 3.56e-07 and 3.56e-07 in respectively 1.22e-03 and 9.39e-04 seconds. For the same task, the routines `exp_sp_t`, `exp_t`, `exp_sp_d` and `exp_d` were forced to compute the divided differences with 38 operative digits (these routines are not designed for tolerances different from the unit roundoff of the working precision and therefore a comparison would be unfair) committing a maximum forward pointwise error of 9.01e-13, 9.01e-13, 5.54e-24 and 5.54e-24 in respectively 1.35e-01, 9.88e-02, 1.89e-01 and 1.66e-01 seconds.

Clearly, the error committed by `expkptotf` and `exptayotf` is larger than their competitors because in handling just 8 digits the accuracy is capped to the roundoff error.

Nevertheless, the accuracy of all six routines turns out to be the same for all the routines when the output is converted to single precision arithmetic, that was the goal, but with the slowest of the on-the-fly routines being 82 times faster than the fastest of the not on-the-fly ones.

For the double precision arithmetic case with interpolation set $\mathcal{L}_{64}$ the disproportion is even larger: the routines `expkptotf` and `exptayotf` could handle the calculations using just 16 digits both committing a maximum forward pointwise error of 9.35e-16 and 1.40e-15 in respectively 9.06e-03 and 9.49e-03 seconds. For the same task the routines `exp_sp_t`, `exp_t`, `exp_sp_d` and `exp_d` were forced to compute the divided differences with 308 operative digits committing a maximum forward componentwise error of 4.15e-143, 4.15e-143, 4.70e-143 and 4.70e-143 in respectively 3.72e+00, 3.26e+00, 1.35e+01 and 1.32e+01 seconds. Once again the error committed by `expkptotf` and `exptayotf` is larger than their competitors because in handling just 16 digits the accuracy is capped to the roundoff error. Nevertheless, the accuracy of all six routines turns out to be the same for all the routines when the output is converted to double precision arithmetic with the slowest of the on-the-fly routines being 344 times faster than the fastest of the not on-the-fly ones.

The case of computation of the table of the divided differences is not the only instance of a computation that requires high accuracy in a standard working precision. Another example is represented by the exponentiation of the Hessenberg matrices arising from the widely used Krylov approximations of the action of the matrix exponential. Consider in fact the matrix $\mathbf{A}$, arising from the discretization of the 1D advection–diffusion operator $\partial_{xx}+\partial_x$ with homogeneous Dirichlet boundary conditions, that can be created in MATLAB using the commands

```
N = 256;
h = 1 / (N + 1);
A = toeplitz( sparse([1, 1], [1, 2], [-2, 1] / ( h*h ), 1, N)) + ...
    toeplitz( sparse(1, 2, -1 / (2 * h), 1, N), ...
    sparse (1, 2, 1 / (2 * h), 1, N));
```

and the vector v

```
v = transp( sin( pi * linspace(0, 1, N + 2) ) );
v = v( 2:N+1);
```

and suppose we run the Arnoldi process until degree $m = 100$. We compare in Table 3.5 how accurate are the routines `expm` and `expkptotf` when we require to compute the first column of $\mathrm{e}^{\tau\mathbf{H}_{100}}$, with $\tau = 10^{-5}$ so that $\|\tau\mathbf{H}_{100}\|_2 \approx 2.64$, when both are run in double precision arithmetic, but to the latter we prescribe a tolerance equal to `realmin`. For sake of completeness, we mention that the CPU time needed by `expm` for approximating $\mathrm{e}^{\tau\mathbf{H}_{100}}e_1$ is around 1.10e-03 seconds. The value for `expkptotf` amounts to 4.60e-03 seconds. For computing instead $\mathrm{e}^{\tau\mathbf{H}_{100}}e_1$ using the Multiprecision Computing Toolbox required by the routines `exp_sp_t`, `exp_t`, `exp_sp_d`, and `exp_d` the time required would be in the order of tens of seconds. Therefore, it is not a viable solution, in high performance computing, to compute $\mathrm{e}^{\tau\mathbf{H}_{100}}e_1$ using one of these routines in high precision arithmetic and then to convert them in double precision data type.

i	`expm`	`exact`	`expkptotf`
1	9.999013095670584e-01	9.999013095670584e-01	9.999013095670581e-01
2	3.115436398118518e-05	3.115436398118519e-05	3.115436398118518e-05
3	8.846542020076177e-07	8.846542020076174e-07	8.846542020076174e-07
4	1.574632605641432e-07	1.574632605641431e-07	1.574632605641431e-07
5	2.267163210996956e-08	2.267163210996955e-08	2.267163210996955e-08
...	...	...	...
20	5.686441429828880e-27	5.686441429782687e-27	5.686441429782688e-27
...	...	...	...
30	1.174361293640533e-42	1.174361274486545e-42	1.174361274486545e-42
...	...	...	...
50	4.388439314516036e-71	4.095918401733461e-78	4.095918401733464e-78
...	...	...	...
75	5.792148930723989e-109	2.321501202667806e-127	2.321501202667806e-127
...	...	...	...
100	7.973862992039284e-145	2.548175675177905e-180	2.548175675177906e-180

Table 3.5: Approximation of the ith component of the vector $\mathrm{e}^{\tau\mathbf{H}_{100}}e_1$ using `expm` and the routine `expkptotf` with tolerance set to `realmin`.

The encouraging results that the routine `expkptotf` obtained in double precision arithmetic and arbitrary precision arithmetic, as well as the capacity of successfully handling data in arbitrary precision arithmetic for any prescribed tolerance, make this routine a formidable tool for the numerical applications. We, therefore, plan to apply the ideas elaborated in this section to the computation in multiprecision of other matrix functions.

Chapter 4

Computing the action of the matrix exponential

The main goal of this chapter is to produce algorithms for the efficient approximation of the action of the matrix exponential on a vector. To do so, we consider the simple polynomial methods. First, we study how to conveniently order an interpolation set to minimize the chances of the insurgence of the hump phenomenon.

Then we propose a family of interpolation sequences coming from the reordering of the Leja–Hermite sets of points. These sequences are tailored to those matrices whose spectrum is skinny and distributed along the real or imaginary axis. A dedicated contour integral approximation of the backward error matrix is then illustrated. The result is an algorithm, `explhe`, that shows a clear improvement with respect to other polynomial methods on a fairly large share of matrices: those with a very skinny spectrum.

After that, we show a more general way to exploit certain information readily available about the spectrum of $\mathbf{A}$. Acknowledged the great effectiveness of the Krylov method, we tried to replicate its characteristics while avoiding its vulnerabilities using a polynomial interpolation at the extended Ritz's values, that are known to lie close to the largest eigenvalues of $\mathbf{A}$. Such interpolation set varies with $\mathbf{A}$, thus we developed a novel technique to bound the backward error matrix in run-time in double precision arithmetic. The result is an algorithm, `pkryexp`, that appears to be among the fastest routines for approximating $\mathrm{e}^{\mathbf{A}}v$ while showing exceptional accuracy properties.

4.1 Introduction

Exponential integrators play a key role in the field of the applications due to their effectiveness when applied to stiff or highly oscillatory problems. The efficiency of this class of methods strongly depends on the fast and accurate approximation of the action of the matrix exponential on a vector, denoted by

$$\mathrm{e}^{\mathbf{A}}v,$$

where $\mathbf{A}$ is a complex-valued square matrix of size N and where v is a vector of compatible dimension. As we anticipated already in the introduction of this thesis, since the matrix $\mathbf{A}$ may be sparse and large, it is crucial to avoid to form the matrix $\mathrm{e}^{\mathbf{A}}$ in approximating

the vector $\mathrm{e}^{\mathbf{A}}v$. Therefore it is necessary to form the approximation of $\mathrm{e}^{\mathbf{A}}v$ by linearly combining (few) vectors from

$$\{v, \mathbf{A}v, \mathbf{A}^2v, \ldots, \mathbf{A}^{\nu-1}v\},$$

where ν is the grade of v with respect to $\mathbf{A}$. In this endeavor, we also set for ourselves to avoid to use rational approximations and to focus on the contrary on methods that are based on polynomial approximations of the exponential function.

Let us recall that a degree m polynomial approximation of $\mathrm{e}^{\mathbf{A}}v$ in Newton form is

$$p_m(\mathbf{A})v = \sum_{k=0}^{m} d[z_0, z_1, \ldots, z_m] \prod_{j=0}^{k-1} (\mathbf{A} - z_j\mathbf{I})v,$$

where $d[z_0], d[z_0, z_1], \ldots, d[z_0, z_1, \ldots, z_m]$ are the divided differences of the exponential function at the interpolation sequence $\sigma_m = (z_0, z_1, \ldots, z_m)$.

In order to determine if such approximation is accurate, we represent the backward error matrix, that is the matrix $\Delta\mathbf{A}$ such that

$$p_m(\mathbf{A})v = \mathrm{e}^{\mathbf{A}+\Delta\mathbf{A}}v,$$

as a function of $\mathbf{A}$. Namely, similarly to what we have done in Chapter 3, we apply the properties of the exponential function and we write

$$h_{m+1}(\mathbf{A}) := \log(\mathrm{e}^{-\mathbf{A}}p_m(\mathbf{A})), \tag{4.1}$$

that is the matrix $\Delta\mathbf{A}$. We declare $p_m(\mathbf{A})v$ to be a satisfying approximation of $\mathrm{e}^{\mathbf{A}}$ if we somehow manage to grant that the inequality

$$\|h_{m+1}(\mathbf{A})\| \leq \mathrm{tol} \cdot \|\mathbf{A}\|,$$

where tol is the tolerance prescribed by the user, holds true. Since it is not practical to compute the matrix function h_{m+1}, the common practice is to control $\|h_{m+1}(\mathbf{A})\|$ by some larger quantity that is in turn easier to compute. For practical examples we refer to Section 3.3, 3.3.1 and 3.3.3 of this thesis.

If the analysis of the backward error matrix reveals that the approximation may be inaccurate the options are two: either we increase m or we apply a sub-stepping strategy. That is we can set $v^{(0)} := v$, then march as

$$v^{(l+1)} := p_m(\tau_{l+1}\mathbf{A})v^{(l)}, \quad l = 0, 1, \ldots, s-1 \tag{4.2}$$

and recover the desired approximation $v^{(s)}$. Since such polynomial interpolations do not vary through the sub-steps, we select a positive integer s and we set $\tau_l = s^{-1}$ for each $l = 0, 1, \ldots, s-1$.

In order to shift the eigenvalues of $\mathbf{A}$ to a more favorable location for the interpolant p_m, we consider to work with the shifted version of the input matrix

$$\mathbf{B} := \mathbf{A} - \mu\mathbf{I},$$

for some scalar μ. Popular choices of μ all aim to drag the spectrum in a neighborhood of the origin. Due to numerical stability reasons, if the real part of μ is positive we recover the desired approximation as $\mathrm{e}^{\mu}v^{(s)}$, otherwise we multiply $\mathrm{e}^{\tau_{l+1}\mu}$ into $v^{(l+1)}$ at each sub-step l.

We also equip with an early termination criterion. Suppose that at sub-step l we encounter a positive integer i smaller than m such that

$$\left\| d[z_0, z_1, \ldots, z_{i-1}] \prod_{j=0}^{i-2} (\mathbf{B} - z_j \mathbf{I}) v \right\|_\infty + \left\| d[z_0, z_1, \ldots, z_i] \prod_{j=0}^{i-1} (\mathbf{B} - z_j \mathbf{I}) v \right\|_\infty$$

is not larger than

$$\mathrm{tol} \cdot \left\| \sum_{k=0}^{i} d[z_0, z_1, \ldots, z_k] \prod_{j=0}^{k-1} (\mathbf{B} - z_j \mathbf{I}) v \right\|_\infty .$$

Then we stop the computations, i.e. we set

$$v^{(l+1)} = \sum_{k=0}^{i} d[z_0, z_1, \ldots, z_k] \prod_{j=0}^{k-1} (\mathbf{B} - z_j \mathbf{I}) v^{(l)}$$

and we proceed processing the successive sub-step.

The main problem affecting this class of methods lies in the choice of the interpolation sequence. If the interpolation points lie far away from the eigenvalues of $\mathbf{B}$, the hump phenomenon may destructively kick in causing a sensible loss of precision in finite precision arithmetic.

In addition to that, we also showed an example where even though the interpolation points lied sufficiently close to the spectrum of $\mathbf{B}$, the hump phenomenon was bound to strike anyway (see end of 1.1.2). This is because it is important to strategically reorder the interpolation set so that the largest eigenvalues of $\mathbf{B}$ get "marked" by an interpolation point in the earliest stages of the approximation.

In the following we are going to briefly analyze the problem of conveniently reordering a set of interpolation points, illustrating our contribution to the problem.

After that we enter the core topic of this chapter by starting, in Section 4.2, to design a polynomial method that is tailored to those matrices which are characterized by a skinny spectrum. For such matrices, it is in fact easy to place the interpolation points close to the eigenvalues, for they must be distributed along the real or imaginary axis. In addition to that, we exploit the peculiar distribution of the spectrum of $\mathbf{B}$ to control the norm of the backward error matrix with a sequence of sharp inequalities based on a contour integral approximation of $h_{m+1}(\mathbf{B})$.

Despite in the field of the applications the matrices with a skinny spectrum can be encountered quite commonly, this is not always the case. In Section 4.3, we design a routine that is based on a polynomial approximation exploiting the characteristics of the Krylov subspace linked with $\mathbf{A}$ and v.

4.1.1 On the reordering of an interpolation set

To this aim we showed in the Introduction a reordering algorithm, the Leja ordering, that helps avoiding the hump phenomenon. For it takes just few lines to be described and it helps clarify the improvements we brought to it, we are going to illustrate the Leja ordering algorithm once again.

Consider the set $P = \{x_0, x_1, \ldots, x_k\}$ of k distinct interpolation nodes such that m_j+1 is the multiplicity of the node x_j and suppose $m_0 + m_1 + \ldots + m_k + k + 1 = m$. In [50] Reichel suggests an ordering of P, called Leja ordering, that returns an interpolation sequence $(z_0, z_1, \ldots, z_m)$. For a fixed initial point $y_0 \in P$, one recursively chooses

$$y_{i+1} \in \arg\max_{x \in P} \prod_{j=0}^{i} |x - y_j|^{m_j+1}, \quad i = 0, 1, \ldots, k-1,$$

the ordered sequence of interpolation points is given by the selected nodes y_i repeated according to their multiplicity in P:

$$(z_0, z_1, \ldots, z_m) = (\underbrace{y_0, \ldots, y_0}_{m_0+1}, \underbrace{y_1, \ldots, y_1}_{m_1+1}, \ldots, \underbrace{y_k, \ldots, y_k}_{m_k+1}).$$

When $z_0, z_1, \ldots, z_i$ are all distinct, since at step i the interpolation error is proportional to $\pi_{i,\sigma_i}(x)$, the procedure above selects at each step the point $z_{i+1} \in P$ where the interpolation error is bound to be the largest, forcing the interpolation error to be now 0 at z_{i+1}. This should greedily reduce the interpolation error.

For repeated points the above procedure diverges from this idea, that we would like instead to pursue. In [8] we came out with a modification of the Leja ordering that properly handles the case of interpolation nodes with multiplicity greater than one. Fixing the first point $z_0 \in P = \{x_0, x_1, \ldots, x_k\}$, we recursively pick from the set

$$z_{i+1} \in \arg\max_{x \in P} \prod_{j=0}^{i} |x - z_j| = \arg\max_{x \in P} |\pi_{i,\sigma_i}(x)|, \quad i = 0, 1, \ldots, k-1.$$

Of course, when i equals k, since all the points in P have been already selected once, we have

$$\max_{x \in P} |\pi_{k,\sigma_k}(x)| = 0,$$

making it impossible to proceed in a standard way. Differently from the standard Leja ordering we have now selected merely $k+1$ points and not the desired m. In order to complete the sequence of interpolation points up to m, let us suppose that we can look for z_{k+1} in a slightly translated set $P + \nu$, $\nu \in \mathbb{C}$, that is

$$z_{k+1}^{\nu} \in \arg\max_{x \in P+\nu} |\pi_{k,\sigma_k}(x)| = \arg\max_{x \in P} |\pi_{k,\sigma_k}(x+\nu)| + \nu.$$

Now, we can express $\pi_{k,\sigma_k}(x+\nu)$ by means of Taylor's formula

$$\pi_k(x+\nu) = \pi_{k,\sigma_k}(x) + \pi'_{k,\sigma_k}(x)\nu + o(\nu),$$

and therefore

$$\arg\max_{x\in P}|\pi_{k,\sigma_k}(x)+\pi'_{k,\sigma_k}(x)\nu+o(\nu)|+\nu=\arg\max_{x\in P}\frac{|\pi'_{k,\sigma_k}(x)\nu+o(\nu)|}{\nu}+\nu.$$

Finally, letting ν tend to 0, our suggested choice for z_{k+1} is therefore

$$z_{k+1}\in\arg\max_{x\in P_1}|\pi'_{k,\sigma_k}(x)|.$$

where $P_1\subseteq P$ is the set of points with multiplicity at least two. Then, we can set ourselves to pick the successive points as

$$z_{k+j}\in\arg\max_{x\in P_1}|\pi'_{k+j-1,\sigma_{k+j-1}}(x)|,\quad j=1,2,\ldots,k_1.$$

where k_1 is the cardinality of P_1. If $k+k_1<m$ then we have once again that

$$\max_{x\in P_1}|\pi'_{k+k_1,\sigma_{k+k_1}}(x)|=0$$

by following an analogous reasoning as above, we select

$$z_{k+k_1+j}\in\arg\max_{x\in P_2}|\pi''_{k+k_1+j-1,\sigma_{k+k_1+j-1}}(x)|,\quad j=1,2,\ldots,k_2,$$

where $P_2\subseteq P_1$ is the set of points with multiplicity at least three and k_2 is its cardinality. We iterate this reasoning until we complete the sequence $\sigma_m=(z_0,z_1,\ldots,z_m)$.

Although this ordering loyally preserves the idea of greedily maximize the chances of triggering an early termination condition for a given set of interpolation nodes P, it is sometimes not the most convenient way to go. The matrix $\mathbf{A}$ and the vector v may be real-valued while the interpolation points are chosen to be pairwise complex conjugated. In this case, it is possible to rewrite the Newton interpolation in real arithmetic thanks to the so called *Tal-Ezer algorithm* (see [66] for details and [8, Alg. 4] for a simple pseudocode), provided that the pair of complex conjugated interpolation points are taken in a succession. This is crucial for keeping the calculations in real arithmetic and saving computational effort. To contemplate this possibility, it is enough to perform a slight modification of the reordering algorithm above: once a complex point is chosen we immediately pick its conjugate and then we proceed normally.

It is worth pointing out that this reordering algorithm and its modification aimed to use real arithmetic over real-valued inputs has been successfully applied in [8] and [10]. The beneficial effects of such a reordering algorithm were not only limited to reduce the computational effort but as well to improve the overall accuracy of the methods.

4.2 Matrices with skinny field of values

When it comes to the real applications, very often the spectrum of the matrices of interest is not just a scattered bunch of points on the complex plane. On the contrary, after a proper shifting, the spectrum of $\mathbf{A}$ often shows a shape that is well contained in a rectangle centered at the origin of the complex plane. We just mention the spatial discretization of diffusion, advection–diffusion, advection, and Schrödinger operators, among others.

These very practical applications led us into refining the existing polynomial techniques to achieve better accuracy and performances over a fairly specific class of matrices: those having a skinny field of values.

In order to do so, we consider families of interpolation points that distribute accordingly with the shape of said rectangles. Since, ideally, such rectangles are very skinny and can be easily centered at the origin, we can consider interpolation points distributed on an interval $[-c, c]$, where $c \in \mathbb{R}^+$ or $c \in \mathrm{i}\mathbb{R}^+$. The most famous family of interpolation points with such characteristics is the set of Chebyshev interpolation points. Despite this family of points is an optimal choice when it comes to polynomial approximation, it may not be the most convenient choice in this particular scenario. In fact, the Chebyshev interpolation points of order m are only optimal if considered as a whole, while we actually hope that the early termination criterion triggers well before the mth interpolation step. Furthermore, later on, we will agree on the necessity of having a couple of interpolation points at 0, while for construction the Chebyshev interpolation points only show one or no interpolation points at all at the origin.

The Leja–Hermite interpolation points are instead a satisfying solution. We can decide in fact how many interpolation points at 0 there will be and, even more importantly, the Leja–Hermite points are expressly designed to aid the early termination to trigger. In fact the Leja–Hermite interpolation points are defined as

$$\begin{aligned} z_0 &= z_1 = \ldots = z_\ell = 0, \\ z_{i+1} &\in \arg\max_{x \in [-c,c]} \prod_{j=0}^{i} |x - z_j|, \quad i = \ell, \ell+1, \ldots, m-1, \end{aligned} \tag{4.3}$$

where $\ell \geq 0$. Points $z_{\ell+1}$, $z_{\ell+2}$, and $z_{\ell+3}$ are not uniquely determined and we select them as $z_{\ell+1} = c$, $z_{\ell+2} = -c$, and $z_{\ell+3} = c\sqrt{(\ell+1)/(\ell+3)}$. In [50] it was shown that when $\ell = 0$, the Leja–Hermite interpolation points asymptotically distributed as the Chebyshev interpolation points, constituting a strong interpolation set. Furthermore, when the rectangle is skinny and horizontally oriented we have that the Leja–Hermite points lie close to the estimated location of the eigenvalues of $\mathbf{B}$. When on the other hand the rectangle is skinny and vertically oriented we can consider the complex conjugate version of the Leja–Hermite interpolation points, defined as

$$\begin{aligned} z_0 &= \ldots = z_\ell = 0, \quad \ell + m \text{ even}, \\ z_{i+1} &\in \arg\max_{x \in [-\mathrm{i}|c|, \mathrm{i}|c|]} \prod_{j=0}^{i} |x - z_i|, \quad z_{i+2} = \overline{z_{i+1}}, \quad i = \ell, \ell+2, \ldots, m-2 \end{aligned} \tag{4.4}$$

where the points $z_{\ell+1}$, $z_{\ell+2}$, and $z_{\ell+3}$ are selected in a similar way as above. The advantage in using this complex conjugate version of the Leja–Hermite points is sure to keep the interpolation points close to the eigenvalues, that for vertically oriented rectangles we estimate to be distributed along the imaginary axis. On the other hand, by picking ℓ and m so that $\ell + m$ is even, we force the existence of complex conjugated pairs in order to be able to exploit the Tal-Ezer algorithm we mentioned before.

4.2.1 Contour integral approximation of the backward error matrix

In the previous section, we agreed over a family of interpolation sequences to use that helps us exploiting the potentially skinny shape of the spectrum of $\mathbf{A}$. The successive step consists in estimating the norm of the backward error matrix so that we can determine if the approximation is accurate. In doing so we are determined to exploit the peculiar skinny shape of the spectrum of $\mathbf{A}$.

As a first step, we need to draw a rectangle around the spectrum of the input matrix. Given a matrix $\mathbf{A}$, we split it into its Hermitian and skew-Hermitian parts, that is

$$\mathbf{A} = \mathbf{A}_{\mathrm{H}} + \mathbf{A}_{\mathrm{SH}}$$

where

$$\mathbf{A}_{\mathrm{H}} := (\mathbf{A} + \mathbf{A}^*)/2$$

and

$$\mathbf{A}_{\mathrm{SH}} := (\mathbf{A} - \mathbf{A}^*)/2,$$

then we estimate the extreme eigenvalues of the two parts by using, for example, the Gershgorin's disks. Since we know the eigenvalues of $\mathbf{A}_{\mathrm{H}}$ are real and those of $\mathbf{A}_{\mathrm{SH}}$ are purely imaginary, we obtain

$$\mathrm{conv}(\sigma(\mathbf{A}_{\mathrm{H}})) \subseteq [\alpha_1, \alpha_2], \quad \mathrm{conv}(\sigma(\mathbf{A}_{\mathrm{SH}})) \subseteq \mathrm{i}[\eta_1, \eta_2],$$

where conv denotes the convex hull of a set. Therefore, we have

$$\begin{aligned}\mathcal{W}(\mathbf{A}) = \mathcal{W}(\mathbf{A}_{\mathrm{H}} + \mathbf{A}_{\mathrm{SH}}) &\subseteq \mathcal{W}(\mathbf{A}_{\mathrm{H}}) + \mathcal{W}(\mathbf{A}_{\mathrm{SH}}) = \\ &= \mathrm{conv}(\sigma(\mathbf{A}_{\mathrm{H}})) + \mathrm{conv}(\sigma(\mathbf{A}_{\mathrm{SH}})) \subseteq [\alpha_1, \alpha_2] + \mathrm{i}[\eta_1, \eta_2].\end{aligned}$$

We denoted by $\mathcal{W}(\mathbf{A})$ the field of values of a matrix, that is

$$\mathcal{W}(\mathbf{A}) = \{z \in \mathbb{C} \colon z = x^*\mathbf{A}x, \quad \text{for } x \in \mathbb{C}^N \text{ with } x^*x = 1\ \},$$

and used its sub-additivity property and the equivalence between field of values and convex hull of the spectrum for normal matrices. Given the rectangle containing the field of values of the matrix $\mathbf{A}$, a choice of the shifting parameter μ can be given by its center, that is

$$\mu := \frac{\alpha_1 + \alpha_2}{2} + \mathrm{i}\frac{\eta_1 + \eta_2}{2}. \tag{4.5}$$

This shifting strategy is sensibly different from the one applied in Chapter 3, that is because we want to make sure that the rectangle is symmetrically centered at the origin. Therefore, we set

$$\nu := |\frac{\alpha_1 - \alpha_2}{2}|, \quad \beta := |\frac{\eta_1 - \eta_2}{2}|,$$

and we work in practice with $\mathbf{B} := \mathbf{A} - \mu\mathbf{I}$, whose rectangle

$$R(\mathbf{B}) = [-\nu, \nu] + \mathrm{i}[-\beta, \beta]$$

lays symmetrically about the origin of the complex plane. Now, if we consider the 2-norm ε-pseudo-spectrum of $\mathbf{B}$, that is

$$\Lambda_\varepsilon(\mathbf{B}) = \{z \in \mathbb{C}\colon \left\|(z\mathbf{I} - \mathbf{B})^{-1}\right\|_2 \geq \varepsilon^{-1}\},$$

we can write the chain of inclusions

$$\Lambda_\varepsilon(\mathbf{B}) \subseteq \mathcal{W}(\mathbf{B}) + \Delta_\varepsilon \subseteq R(\mathbf{B}) + \Delta_\varepsilon$$

where Δ_ε is the closed disk of radius ε. Both the quantities $\mathcal{W}(\mathbf{B})$ and $R(\mathbf{B})$ scale with $\mathbf{B}$, that is $\mathcal{W}(z\mathbf{B}) = z\mathcal{W}(\mathbf{B})$ and $R(z\mathbf{B}) = zR(\mathbf{B})$ for $z \in \mathbb{C}$, but not the ε-pseudo-spectrum. If we considered instead, for a given δ, the $\delta \left\|\mathbf{B}\right\|_2$-pseudo-spectrum, we have the chain of inclusions

$$\begin{aligned}\Lambda_{\delta\|z\mathbf{B}\|_2}(z\mathbf{B}) \subseteq \mathcal{W}(z\mathbf{B}) + \Delta_{\delta\|z\mathbf{B}\|_2} \subseteq R(z\mathbf{B}) + \Delta_{\delta\|z\mathbf{B}\|_2} = \\ = z(R(\mathbf{B}) + \Delta_{\delta\|\mathbf{B}\|_2}) \subseteq zR_\delta(\mathbf{B})\end{aligned} \tag{4.6}$$

where

$$R_\delta(\mathbf{B}) = [-\nu - \delta\left\|\mathbf{B}\right\|_2, \nu + \delta\left\|\mathbf{B}\right\|_2] + \mathrm{i}[-\beta - \delta\left\|\mathbf{B}\right\|_2, \beta + \delta\left\|\mathbf{B}\right\|_2],$$

is the *extended rectangle* with the strip of width $\delta\left\|\mathbf{B}\right\|_2$ around $R(\mathbf{B})$. If we select 0 as interpolation point at least *twice* (that is if we pick $\ell > 0$), provided that $\mathbf{X}$ belongs to

$$\Omega = \{\mathbf{X} \in \mathbb{C}^{N\times N}\colon \rho(\mathbf{I} - \mathrm{e}^{-\mathbf{X}}p_m(\mathbf{X})) < 1\},$$

we can develop the function $h_{m+1}(\mathbf{X})$ from (4.1) in power series and write

$$h_{m+1}(\mathbf{X}) = \mathbf{X}^2 \sum_{k=\ell+1}^{\infty} f_k \mathbf{X}^{k-2} =: \mathbf{X}^2 \bar{h}_{m+1}(\mathbf{X}). \tag{4.7}$$

Provided that $\mathbf{X}$ is different from the null matrix, by using the Cauchy integral representation for the matrix function $\bar{h}_{m+1}(\mathbf{X})$ we get

$$\left\|\bar{h}_{m+1}(\mathbf{X})\right\|_2 = \left\|\frac{1}{2\pi\mathrm{i}}\int_\Gamma \bar{h}_{m+1}(z)(z\mathbf{I} - \mathbf{X})^{-1}\mathrm{d}z\right\|_2 \leq \frac{\mathcal{L}(\Gamma)}{2\pi\delta\left\|\mathbf{X}\right\|_2}\left\|\bar{h}_{m+1}\right\|_\Gamma$$

where $\Gamma = \partial K$ denotes the boundary of a domain $K \subset \mathbb{C}$ that contains the $\delta\left\|\mathbf{X}\right\|_2$-pseudo-spectrum of $\mathbf{X}$ and

$$\left\|\bar{h}_{m+1}\right\|_\Gamma = \max_{z\in\Gamma}|\bar{h}_{m+1}(z)| = \max_{z\in K}|\bar{h}_{m+1}(z)|.$$

Therefore

$$\begin{aligned}\frac{\left\|h_{m+1}(\mathbf{X})\right\|_2}{\left\|\mathbf{X}\right\|_2} \leq \left\|\mathbf{X}\right\|_2\left\|\bar{h}_{m+1}(\mathbf{X})\right\|_2 = \\ = \left\|\mathbf{X}\right\|_2\left\|\frac{1}{2\pi\mathrm{i}}\int_\Gamma \bar{h}_{m+1}(z)(z\mathbf{I} - \mathbf{X})^{-1}\mathrm{d}z\right\|_2 \leq \frac{\mathcal{L}(\Gamma)}{2\pi\delta}\left\|\bar{h}_{m+1}\right\|_\Gamma\end{aligned} \tag{4.8}$$

if $\Gamma = \partial K$ with K now containing the $\delta\left\|\mathbf{X}\right\|_2$-pseudo-spectrum of $\mathbf{X}$. Thanks to (4.6),

this is certainly true if

$$R_\delta(\mathbf{X}) \subseteq K. \tag{4.9}$$

Now we have to restrict the choice of possible domains K of interest. We consider the domain K circumscribed by an ellipses Γ_γ

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1, \quad z = x + \mathrm{i}y$$

whose focal interval is the interpolation interval $[-c, c]$ and the capacity is

$$\gamma = \frac{a+b}{2}.$$

Since $c^2 = a^2 - b^2$, where c can be real or purely imaginary, it turns out that the ellipse Γ_γ has semi-axes

$$a = \gamma + \frac{c^2}{4\gamma}, \quad b = \gamma - \frac{c^2}{4\gamma}.$$

Such a choice for the domains K makes it possible to select the ellipse Γ_{γ_δ}, for given c and δ, which realizes

$$\frac{\mathcal{L}(\Gamma_{\gamma_\delta})}{2\pi\delta} \left\| \bar{h}_{m+1} \right\|_{\Gamma_{\gamma_\delta}} = \mathrm{tol} \tag{4.10}$$

by finding the root (using the secant method, e.g.) of the uni-variate function

$$\gamma \mapsto \frac{\mathcal{L}(\Gamma_\gamma)}{2\pi\delta} \left\| \bar{h}_{m+1} \right\|_{\Gamma_\gamma} - \mathrm{tol}.$$

Such a function has at most one positive root. In fact, $\gamma \geq |c|/2$ and the function is monotonically increasing with γ, by the maximum modulus principle. Moreover, $\mathcal{L}(\Gamma_\gamma) \to +\infty$ for $\gamma \to +\infty$. Therefore, either the error estimate exceeds tol already for $\Gamma_{|c|/2} = [-c, c]$, meaning that the interpolation degree m is not large enough for the given interval, or there exists one positive root.

Therefore, for each polynomial of interest $p_m \colon [-c, c] \to \mathbb{R}$ which interpolates e^x at $m+1$ points containing $\ell + 1$ zeros and a given δ, it is possible to pre-compute once and for all, with a software for multiple precision arithmetic, the semi-axes a and b of the ellipse satisfying (4.10).

At the moment of the computation, we compare the rectangle $R(\mathbf{B})$ with the ellipses of the polynomials p_m. If none of the pre-computed ellipses contain $R(\mathbf{B})$, or if we want to have more freedom of choice, we can consider a sub-stepping strategy. That is for each p_m we select the smallest positive integer s such that $s^{-1}R(\mathbf{B})$ fits into the ellipse of p_m and we recover the wanted approximation by marching as follows:

$$v^{(l+1)} = \mathrm{e}^{s^{-1}\mathbf{B}} v^{(l)}, \quad l = 0, 1, \ldots, s-1 \tag{4.11}$$

where $v^{(0)} = v$. In order to find the smallest integer value s for which inclusion (4.9) is satisfied, it is now possible to solve the inequality

$$\frac{(\nu + \delta \left\| \mathbf{B} \right\|_2)^2}{s^2 a^2} + \frac{(\beta + \delta \left\| \mathbf{B} \right\|_2)^2}{s^2 b^2} \leq 1,$$

where $s^{-1}(\nu+\delta\,\|\mathbf{B}\|_2\,,\beta+\delta\,\|\mathbf{B}\|_2)$ is the top-right vertex of the rectangle $s^{-1}R_\delta(\mathbf{B})$, which gives

$$s=\left\lceil\sqrt{\frac{(\nu+\delta\,\|\mathbf{B}\|_2)^2}{a^2}+\frac{(\beta+\delta\,\|\mathbf{B}\|_2)^2}{b^2}}\right\rceil. \tag{4.12}$$

By doing so the computational cost needed to build this approximation amounts to $s\cdot m$ matrix-vector products.

We briefly sketch again the procedure:

1. For a given matrix $\mathbf{A}$, compute the rectangle $R(\mathbf{A})$ which contains its field of values $\mathcal{W}(\mathbf{A})$, shift it by μ (see (4.5)) and compute the final centered symmetric rectangle

$$R(\mathbf{B})=[-\nu,\nu]+\mathrm{i}[-\beta,\beta].$$

2. Compute s as ruled in (4.12).

3. Recover the approximation of $\mathrm{e}^{\mathbf{A}}v$ marching as in (4.11).

The backward error analysis assures that $v^{(s)}$ accurately approximates $\mathrm{e}^{\mathbf{A}}v$. If approximations at different matrix scales $\exp(t_i\mathbf{B})v_i$, $t_i\geq 0$, are required (this is quite common in the exponential integrators), it is possible to compute the matrix-dependent quantity

$$r_\delta(\mathbf{B})=\sqrt{\frac{(\nu+\delta\,\|\mathbf{B}\|_2)^2}{a^2}+\frac{(\beta+\delta\,\|\mathbf{B}\|_2)^2}{b^2}}$$

once and for all and later to select the scaling parameter as

$$s_i=\lceil t_i r_\delta(\mathbf{B})\rceil.$$

This result is possible thanks to the choice of a pseudo-spectrum level which scales with $\|\mathbf{B}\|_2$, improving the work done in [6, Section 3.2], in which the estimate of the backward error analysis based on the contour integral expansion was introduced for the Leja points and the level of the pseudo-spectrum was chosen independent of the matrix.

Refinement of the rectangle $R(\mathbf{B})$

When it comes to the real applications the input matrices are usually extremely large and sparse. Therefore it is not convenient to store the matrix $\mathbf{A}_\mathrm{H}$ and $\mathbf{A}_\mathrm{SH}$ in order to compute the Gershgorin's disks and draw the rectangle $[\alpha_1,\alpha_2]+\mathrm{i}[\eta_1,\eta_2]$. In addition to that, the Gershgorin's disks technique estimate the approximate location of every eigenvalue of $\mathbf{A}_\mathrm{H}$ (and respectively $\mathbf{A}_\mathrm{SH}$) that is an information much more expensive than necessary when the goal is just to find the scalars α_1, α_2, η_1 and η_2 such that

$$\mathrm{conv}(\sigma(\mathbf{A}_\mathrm{H}))\subseteq[\alpha_1,\alpha_2],\quad \mathrm{conv}(\sigma(\mathbf{A}_\mathrm{SH}))\subseteq\mathrm{i}[\eta_1,\eta_2].$$

Alternatively, we can effectively compute the rectangle $R(\mathbf{A})$ as

$$R(\mathbf{A})=[-\,\|\mathbf{A}_\mathrm{H}\|_2\,,\|\mathbf{A}_\mathrm{H}\|_2]+\mathrm{i}[-\,\|\mathbf{A}_\mathrm{SH}\|_2\,,\|\mathbf{A}_\mathrm{SH}\|_2],$$

thanks to the inclusions

$$\sigma(\mathbf{A}_{\mathrm{H}}) \subseteq [-\rho(\mathbf{A}_{\mathrm{H}}), \rho(\mathbf{A}_{\mathrm{H}})] = [-\left\|\mathbf{A}_{\mathrm{H}}\right\|_2, \left\|\mathbf{A}_{\mathrm{H}}\right\|_2],$$
$$\sigma(\mathbf{A}_{\mathrm{SH}}) \subseteq \mathrm{i}[-\rho(\mathbf{A}_{\mathrm{SH}}), \rho(\mathbf{A}_{\mathrm{SH}})] = \mathrm{i}[-\left\|\mathbf{A}_{\mathrm{SH}}\right\|_2, \left\|\mathbf{A}_{\mathrm{SH}}\right\|_2].$$

Clearly even if the eigenvalues of $\mathbf{A}$ are not so far from each other but they happen to be unfortunately far from the origin, the rectangle traced with this method, differently from the Gershgorin's disks technique, is unreasonably large. Therefore it is good practice to shift $\mathbf{A}$ so that its eigenvalues are distributed around the origin before drawing the rectangle. But we cannot apply the shift μ introduced in the previous section, because for computing it we would need to compute the Gershgorin's disks, that we want to avoid. Therefore the shift μ is determined as in Chapter 3, i.e. as the mean eigenvalue of $\mathbf{A}$, that is

$$\mu := \sum_{i=0}^{N} \frac{a_{i,i}}{N}.$$

Such a rectangle is generally smaller than the one obtained using the (shifted) Gershgorin's disks technique. As a consequence, we expect smaller values of s leading to an increased efficiency of the method.

On the other hand, it requires two 2-norm computations (or estimates), that can be computationally demanding. In addition to that it seems that we did not tackle the problem of not forming the matrices $\mathbf{A}_{\mathrm{H}}$ and $\mathbf{A}_{\mathrm{SH}}$ since the state-of-the-arts routine for estimate the 2-norm of a matrix would explicitly require as an input the matrices $\mathbf{A}_{\mathrm{H}}$ and $\mathbf{A}_{\mathrm{SH}}$.

There is a way around in order to tackle this issue, in fact we know that for Hermitian matrices their 2-norm is equivalent to their spectral radius. A similar result holds also for skew-Hermitian matrices. Therefore we can modify the method of powers in order to compute the spectral radius of

$$\mathbf{B}_{\mathrm{H}} := (\mathbf{B} + \mathbf{B}^*)/2$$

and

$$\mathbf{B}_{\mathrm{SH}} := (\mathbf{B} - \mathbf{B}^*)/2,$$

without explicitly forming the matrices $\mathbf{B}_{\mathrm{H}}$ and $\mathbf{B}_{\mathrm{SH}}$ if we carefully implement the method of powers algorithm. We report verbatim this routine in the following.

```
 1. function e = sym_power_method( B, skewness, tol )
 2. % Compute the spectral radius of B + skewness * B' using the method
 3. % of powers without actually forming the matrix B + skewness * B'.
 4.   maxit = 100;
 5.   x = randn( length( B ), 1 );
 6.   x = full( B * x + skewness * ctranspose( ctranspose( x ) * B ) );
 7.   e = norm( x );
 8.   if ( e == 0 )
 9.     return;
10.   end
11.   e0 = 0;
```

```
12.   x = x / e;
13.   it = 0;
14.   while ( abs( e - e0 ) >= tol * e ) && ( it < maxit )
15.      e0 = e;
16.      if ( skewness == 1 )
17.        x = B * x + ctranspose( ctranspose( x ) * B );
18.      else
19.        x = ctranspose( ctranspose( x ) * B ) - B * x;
20.      end
21.      e = norm( x );
22.      if ( e == 0 )
23.        break;
24.      end
25.      x = x / e;
26.      it = it + 1;
27.   end
28. end
```

Since we know that the numerical radius of $\mathbf{B}$, defined by

$$w(\mathbf{B}) = \max_{z \in \mathcal{W}(\mathbf{B})} |z|,$$

is such that

$$w(\mathbf{B}) = \sup_{\|x\|_2=1} \langle x, \mathbf{B}x \rangle \leq \sup_{\|x\|_2=1} \|x\|_2 \, \|\mathbf{B}x\|_2 = \|\mathbf{B}\|_2$$

and

$$\|\mathbf{B}\|_2 \leq \|\mathbf{B}_{\mathrm{H}}\|_2 + \|\mathbf{B}_{\mathrm{SH}}\|_2 = w(\mathbf{B}_{\mathrm{H}}) + w(\mathbf{B}_{\mathrm{SH}}) \leq 2w(\mathbf{B})$$

we know that the

$$w(\mathbf{B}) \leq \|\mathbf{B}\|_2 \leq 2w(\mathbf{B})$$

and therefore $\|\mathbf{B}\|_2$ could be over-estimated at no extra cost by the diagonal of the rectangle $R(\mathbf{B})$, that is:

$$\sqrt{\|\mathbf{B}_{\mathrm{H}}\|_2^2 + \|\mathbf{B}_{\mathrm{SH}}\|_2^2}.$$

A possible choice for δ

In this section we suggest a way to choose the parameter δ. We start by considering that, for given $c \neq 0$ and δ, the minimum of

$$\gamma \mapsto \frac{\mathcal{L}(\Gamma_\gamma)}{2\pi\delta} \left\| \bar{h}_{m+1} \right\|_{\Gamma_\gamma}$$

is attained for the degenerate ellipse $\Gamma_{|c|/2}$ with perimeter $\mathcal{L}(\Gamma_{|c|/2}) = 4|c|$. Therefore, the minimum value for δ satisfying (4.10) is

$$\delta_1 = \frac{4|c|}{2\pi \cdot \mathrm{tol}} \left\| \bar{h}_{m+1} \right\|_{\Gamma_\gamma}.$$

Of course, this value is of no interest, since no extended rectangle can be contained into the degenerate ellipse. If $\delta > \delta_1$, then there exists a non-degenerate ellipse Γ_{γ_δ} with semi-axes $a = a_\delta$ and $b = b_\delta$ and capacity γ_δ which satisfies

$$\frac{\mathcal{L}(\Gamma_{\gamma_\delta})}{2\pi\delta} \left\|\bar{h}_{m+1}\right\|_{\Gamma_{\gamma_\delta}} = \text{tol}.$$

From the formula above, a larger δ allows a larger ellipse Γ_δ, but it also requires a larger extended rectangle which has to be included into the ellipse.

We need a strategy to "maximize" the rectangles which can be contained into the ellipse Γ_{γ_δ}. For a given ellipse with semi-axes a_δ and b_δ, it is easy to compute the inscribed rectangles with the largest area or the longest perimeter. By elementary calculations, they have semi-edges $a_\delta/\sqrt{2}$, $b_\delta/\sqrt{2}$ and $a_\delta^2/\sqrt{a_\delta^2+b_\delta^2}$, $b_\delta^2/\sqrt{a_\delta^2+b_\delta^2}$, respectively. The rectangle with longest perimeter appears to be skinnier than the one with largest area. Since we are precisely interested in this feature, we suppose that the rectangle of interest is a re-scaling of the rectangle of longest perimeter, that is it has semi-edges ta_δ^2 and tb_δ^2, for a scalar $t > 0$. In order to compute the dimensions of the related extended rectangle we assume that the 2-norm of the matrix whose field of values is contained in the rectangle is half of its diagonal, that is $t\sqrt{a_\delta^4+b_\delta^4}$.

Therefore, the extended rectangle is contained into the ellipse Γ_{γ_δ} if

$$t\sqrt{\left(\frac{a_\delta^2+\delta\sqrt{a_\delta^4+b_\delta^4}}{a_\delta}\right)^2+\left(\frac{b_\delta^2+\delta\sqrt{a_\delta^4+b_\delta^4}}{b_\delta}\right)^2} \leq 1.$$

Now, we can try to maximize the quantity $t(a_\delta^2+b_\delta^2)$ (that is the semi-perimeter of the rectangle) under the constraint above. We have therefore to find the maximum of the uni-variate function

$$\delta \mapsto \frac{a_\delta^2+b_\delta^2}{\sqrt{\left(\frac{a_\delta^2+\delta\sqrt{a_\delta^4+b_\delta^4}}{a_\delta}\right)^2+\left(\frac{b_\delta^2+\delta\sqrt{a_\delta^4+b_\delta^4}}{b_\delta}\right)^2}}. \tag{4.13}$$

The above maximum can be approximated, for instance, by using the golden section method. We will denote by δ_m, a_m, b_m, and γ_m the parameters of the ellipse found after this optimization procedure. The optimal choice of δ is a second neat advantage over the procedure described in [6], where ε-pseudo-spectra where considered, with ε independent on the norm of the matrix and fixed to 1/50.

If instead $c = 0$, the ellipse Γ_γ is in fact the circumference of radius γ. Since $\mathcal{L}(\Gamma_\gamma) \to 0$ for $\gamma \to 0$, for each δ is it possible to find γ_δ in order to satisfy (4.10). The function to maximize (4.13) simplifies to

$$\delta \mapsto \frac{\sqrt{2}\gamma_\delta}{1+\sqrt{2}\delta}.$$

Example. *As an example, we compute the ellipse which corresponds to degree* $m = 30$, $c = 3$, *and* $\ell = 1$ *for the double precision tolerance* $\text{tol} = 2^{-53}$. *The best value for* δ *we found with the strategy described above is* $\delta_m = 0.0179961382516439$. *The corresponding ellipse* Γ_{γ_m} *of semi-axes* a_m *and* b_m *and rectangle* R *with edges proportional to the squares*

of the semi-axes are drawn in magenta dashed-dotted line in Figure 4.1. If we consider the ellipse and rectangle associated to a value of δ ten times larger (a value similar to the value $1/50$ used in [6]), we get the figures drawn with a blue dashed line in Figure 4.1. As already noticed, the ellipse we get is quite large, but the strip of width proportional to δ is large and the rectangle which should contain the field of values turns out to be smaller than the previous one. Finally, we consider the result coming from taking the interpolation set with four zeros ($\ell = 3$). The optimization technique gives $\delta_m = 0.0177640774067962$ and the corresponding figures are drawn with a solid black line in Figure 4.1. For these choices of m and c, $\ell = 3$ turns out in fact to be the value giving the largest rectangle.

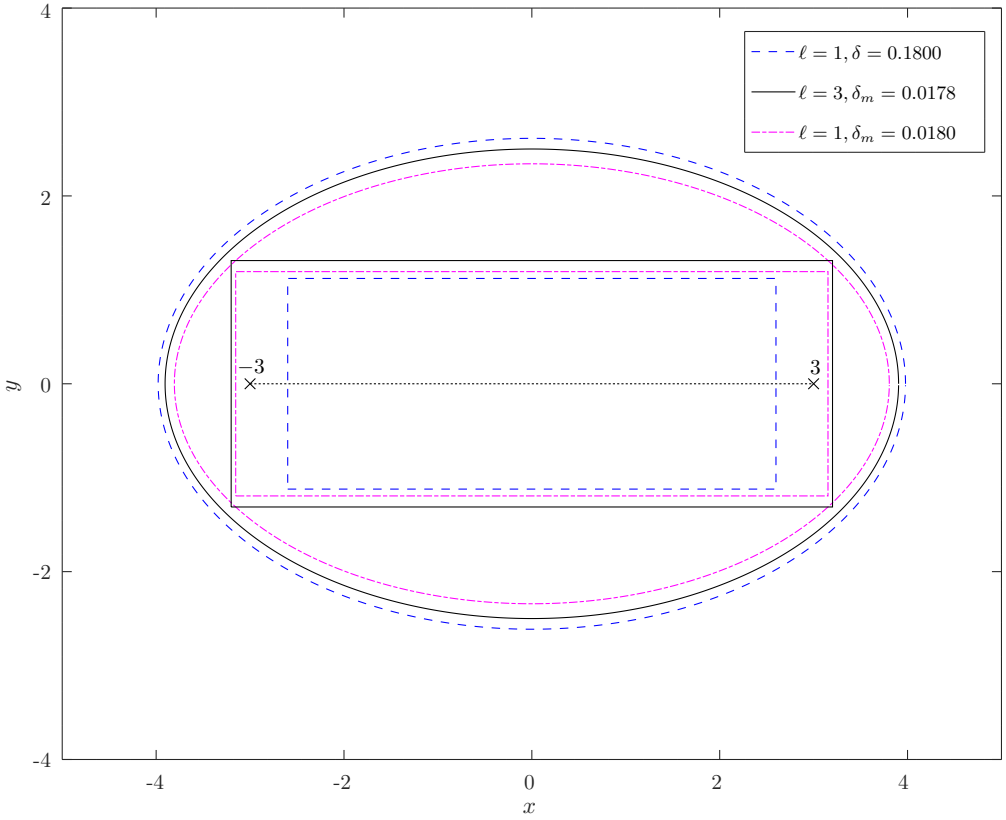


Figure 4.1: Ellipses and rectangles of longest perimeter corresponding to $m = 30$, $c = 3$, $\ell = 1$ and $\delta = 0.179961382516439$ (blue dashed line), $\delta_m = 0.0179961382516439$ (magenta dashed-dotted line), $\ell = 3$ and $\delta_m = 0.0177640774067962$ (solid black line).

4.2.2 Final selection of the polynomial interpolation

The final choice of the polynomial interpolant to use for approximating $\mathrm{e}^{\mathbf{A}}v$ with the technique introduced in this section is divided into two parts. The first part is to be completed in advance before the user is ready to use the routine, for it requires to use a software of symbolic calculus to predetermine certain key parameters. The second is to

be performed during the computations, for it consists in finding the rectangle $R(\mathbf{B})$ and in comparing it with the parameters precomputed in the first part.

The family of polynomial interpolants

In the first part, we have to form a family of polynomials that we are willing to use for our interpolation and for each polynomial we compute the connected ellipse. Clearly, the larger is the family the better will be the performance of the routine, a commercial implementation of this algorithm would surely take into account large families of polynomials. Nevertheless, this is out of the scope of this manuscript, therefore we kept the number of components of said family relatively low.

The variables are three: the interpolation degree m, the number of interpolation points taken at the origin $\ell + 1$ and the interpolation interval of parameter c. In line with the literature we consider interpolation degrees m of at most 55, see for example [2, 6, 8]. Then we decided to set

$$\ell = q(q-1) - 1, \quad q = 2, 3, \ldots, 8. \tag{4.14}$$

The reason for such a peculiar choice of the parameter ℓ is to build interpolation sequences with certain fractions of the interpolation points in zero that are somehow well-distinguished from each other. If the interval $[-c, c]$ is characterized by a purely imaginary c then the interpolation points different from zero must be taken in complex conjugate pairs. To do so, they need to be in an even number and therefore we adapt the choice of ℓ accordingly.

The last parameter left to fix is c. Clearly, we cannot vary c continuously or we would have to compute infinitely many ellipses. Therefore we consider the sequence $|c| = 0, 0.5, 1, \ldots$ and try to fulfill (4.10), with the relative optimal value of δ. We stress that by "compute an ellipse" we mean that given m, ℓ and c we solve for γ equation (4.10) and therefore we uniquely determined the ellipse.

As we already mentioned, for any given m, there exists a c for which the ellipse $\Gamma_{|c|/2}$ is degenerate and it makes impossible to satisfy (4.10). This is the upper bound for the sequence of the absolute values of c above.

To complete the first part we have to compute the ellipse relative to each member of the family, that is the ellipses satisfying (4.10), and store the results for a later use.

Choosing the interpolation parameters

In this second part, we have to associate a polynomial and a scaling parameter s to the input matrix $\mathbf{B}$. To do so, we narrow down the choice of the candidate polynomials by discarding those not fitting certain criteria. Before describing such criteria we stress once again that each polynomial is characterized by the three parameters m, ℓ and c.

The first criterion concerns the interpolation interval. Once we compute the rectangle $R(\mathbf{B})$ we immediately half the possible candidates: if $\nu \geq \beta$ then c will belong to $\mathbb{R}^+$, otherwise c will be taken purely imaginary.

The second criterion concerns once again the parameter c. Since we want to make sure that the interpolation points fall close to the eigenvalues of $\mathbf{B}$, we want c to be broadly equal to $s^{-1}\max(\nu, \beta)$. In particular we discarded all those polynomials whose c is such

that

$$|s^{-1}\max(\nu,\beta)-c|>c\cdot 4\text{e-}2$$

where 4e-2 is a parameter we engineered a posteriori in order to discard a fair amount of polynomials.

The third criterion concern the shape of the ellipses. Among the polynomials that "survived" the second criterion we only keep the twenty whose ellipse has axis ratio

$$\frac{\min(a_m,b_m)}{\max(a_m,b_m)}$$

most similar to the ratio of the edges

$$\frac{\min(\nu,\beta)}{\max(\nu,\beta)}$$

of the rectangle $R(\mathbf{B})$. Once again, twenty is a parameter we engineered a posteriori to keep a fair amount of polynomials.

The final criterion is about the total approximation cost $s\cdot m$, among the 20 polynomials "surviving" the previous criterion we select the one will the smallest expected total cost.

At this point, we proceed to form the desired approximation of $\mathrm{e}^{\mathbf{A}}v$ using the polynomial we are left with.

4.3 Extended Ritz's values interpolation

Although it is true that, when it comes to the real applications, it is frequent that the spectrum of the matrices of interest is usually contained in a quite skinny rectangle, this is not always the case. It is not uncommon, in fact, that such a rectangle resembles more a fatty rectangle or even a square. In those cases, it is not clear how it is best to orient the interpolation points, vertically or horizontally, that is real or complex conjugated case. Also, for Hermitian and skew-Hermitian matrices, it is not clear if the Leja–Hermite interpolation is automatically the best choice. In fact, for such matrices, the slow and involved Arnoldi process simplifies into the Lanczos process, which is on the opposite very simple and fast.

Therefore, our idea is to exploit the equivalence between Krylov methods and the interpolation at Ritz's values in order to build an original approximation that interpolates the exponential function at what we call *extended Ritz's values*. Trivially, the extended Ritz's values are

$$\{\rho_1,\rho_2,\dots\rho_\kappa\}\cup\{\underbrace{0,0,\dots,0}_{\ell+1\text{ zeros}}\}$$

where the zeros are taken with multiplicity $\ell+1$, a value that will be determined, broadly speaking, according with the non normality of $\mathbf{A}$. We set $m=\kappa+\ell$. Clearly, before being ready to use such an interpolation set, we must reorder it into the interpolation sequence $\sigma_m=(z_0,z_1,\dots,z_m)$, following the algorithm described in Section 4.1.1.

The main advantage of such interpolation sequence is that we can avail of an interpolation set that lies close to the eigenvalues of $\mathbf{A}$ while saving the effort of running the

Arnoldi (or Lanczos) process at each sub-step. In fact, once we build the first time the matrix $\mathbf{H}_\kappa$ and compute its eigenvalues $\rho_1, \rho_2, \dots \rho_\kappa$, we have a strong interpolation set. This set needs not to be updated at each sub-step when a sub stepping strategy is considered. In fact, the Ritz's values are supposed to approximate the eigenvalues of $\mathbf{A}$ that, at worst, get scaled when a sub stepping strategy is considered.

Furthermore, after we run the Arnoldi (or Lanczos) process up to step κ for the first (and only) time, we can recycle the matrix $\mathbf{V}_\kappa$ in order to form the approximation $p_m(\mathbf{A})v$ with only $\ell + 1$ additional matrix-vector products. To see how is that possible, consider the equivalence

$$\|v\|_2 \mathbf{V}_\kappa \mathrm{e}^{\mathbf{H}_\kappa} e_1 = \sum_{k=1}^{\kappa} d[\rho_1, \rho_2, \dots, \rho_\kappa] \prod_{j=1}^{k-1} (\mathbf{A} - \rho_j \mathbf{I}) v,$$

that comes directly from Theorem 1. In order to form the approximation $p_m(\mathbf{A})v$, we need explicitly the vector

$$\pi_{k,\sigma_\kappa}(\mathbf{A})v = \prod_{j=1}^{k-1} (\mathbf{A} - \rho_j \mathbf{I}) v$$

with $\sigma_{\kappa-1} := (\rho_1, \rho_2, \dots \rho_\kappa)$, so that we can add, at step k, the vector

$$d[\rho_1, \rho_2, \dots, \rho_\kappa, \underbrace{0, \dots, 0}_{k-\kappa}] \mathbf{A}^{k-m-1} \prod_{j=1}^{\kappa} (\mathbf{A} - \rho_j \mathbf{I}) v$$

for $k = \kappa + 1, m + 2, \dots, m + 1$ and form the desired approximation. Since $\pi_{k,\sigma_{\kappa-1}}(x)$ is a polynomial of degree lesser than κ, we know, from the previous lemmas, that the vector $\pi_{k,\sigma_{\kappa-1}}(\mathbf{A})v$ is a linear combination of vectors from the basis of K_κ. In other words, we can form $\pi_{k,\sigma_{\kappa-1}}(\mathbf{A})v$ exactly as

$$\|v\|_2 \mathbf{V}_\kappa \pi_{k,\sigma_{\kappa-1}}(\mathbf{H}_\kappa) e_1.$$

From the second sub step (if it is necessary) onward, we do not need to compute the Arnoldi (or Lanczos) process anymore. Therefore we do not need to recycle any matrix-vector product. Hence we can proceed to reorder the interpolation set, made out of the extended Ritz's values, into the interpolation sequence σ_m and continue as a normal interpolation method.

4.3.1 Backward error analysis

On the other hand, the main disadvantage of interpolating at the extended Ritz's values is that the interpolation set varies at each instance of $\mathbf{A}$ and v. Therefore, we are forced to perform the backward error analysis in run time and not in advance and once and for all, as we were accustomed to with other routines for computing the action of the matrix exponential. In fact, in the literature, standard fixed sets of interpolation points were considered because the analysis of the backward error needed the aid of slow software of symbolic calculus and therefore to be run in advance.

Let us now briefly retread the steps of the backward error analysis. With the polyno-

mial $p_m(x)$ we approximate exactly the matrix exponential of a slightly perturbed matrix, i.e.

$$p_m(\mathbf{A}) = \mathrm{e}^{\mathbf{A}+\Delta\mathbf{A}}.$$

We declare the approximation $p_m(\mathbf{A})v$ satisfying if, for the prescribed tolerance tol, we have that the norm of the backward error matrix is small enough:

$$\|\Delta\mathbf{A}\| \leq \mathrm{tol} \cdot \|\mathbf{A}\|. \tag{4.15}$$

Thanks to elementary calculations, we have that said perturbation can be expressed as a function of $\mathbf{A}$ as follows

$$\Delta\mathbf{A} = \log(\mathrm{e}^{-\mathbf{A}} p_m(\mathbf{A})) =: h_{m+1}(\mathbf{A}),$$

see also [8] for further details. Now, let us define the matrix set

$$\Omega = \{\mathbf{X} \in \mathbb{C}^{N\times N} : \rho(\mathbf{I} - \mathrm{e}^{-\mathbf{X}} p_m(\mathbf{X})) < 1\},$$

where ρ is the spectral radius. For every matrix $\mathbf{X}$ belonging to Ω we can write $h_{m+1}(\mathbf{X})$ as the power series expansion

$$h_{m+1}(\mathbf{X}) = \sum_{k=\ell+1}^{\infty} f_k \mathbf{X}^k.$$

Now, we can employ the scalar function $\tilde{h}_{m+1}(x) = \sum_{k=0}^{\infty} |f_k| x^k$ in order to control the backward error:

$$\|h_{m+1}(\mathbf{A})\| \leq \tilde{h}_{m+1}(\|\mathbf{A}\|). \tag{4.16}$$

In this way, if we compute the unique positive scalar root $\theta^\star$ of the equation

$$\tilde{h}_{m+1}(x) - \mathrm{tol} \cdot x = 0,$$

that exists provided $\mathrm{tol} > c_1$, thanks to (4.16) and the monotonicity of $\tilde{h}_{m+1}(x) - \mathrm{tol} \cdot x$ we know that (4.15) holds true provided that

$$\|\mathbf{A}\| \leq \theta^\star.$$

In case instead that $\|\mathbf{A}\|$ is not smaller than $\theta^\star$, we have two options. The first is to increase the cardinality $m+1$ of the interpolation set, in fact larger sets generally lead to larger values of $\theta^\star$. The second option is to select the smallest positive integer s such that $s^{-1}\|\mathbf{A}\|$ is smaller than $\theta^\star$, and to recover the wanted approximation by marching as follows:

$$v^{(l+1)} = \mathrm{e}^{s^{-1}\mathbf{A}} v^{(l)}, \quad l = 0, 1, \ldots, s-1,$$

where $v^{(0)} = v$. By doing so, the predicted computational cost necessary to form this approximation amounts to $s \cdot m$ matrix-vector products.

The inequality (4.16) it is not sharp, hence it generally leads to choices of s and m unnecessarily large, making the overall computational cost to rise. In order to tackle this

issue, we can apply two techniques. We are already familiar with the first one: we set

$$\mu := \sum_{i=1}^{N} \frac{a_{i,i}}{N},$$

that is the mean eigenvalue of $\mathbf{A}$, and we work with $\mathbf{B} := \mathbf{A} - \mu\mathbf{I}$. The second technique is considerably more involved: it was proven in [1] that

$$\|h_{m+1}(\mathbf{B})\| \leq \tilde{h}_{m+1}(\alpha_q(\mathbf{B})) \leq \tilde{h}_{m+1}(\|\mathbf{B}\|) \tag{4.17}$$

with

$$\alpha_q(\mathbf{B}) := \max(\|\mathbf{B}^q\|^{1/q}, \left\|\mathbf{B}^{q+1}\right\|^{1/q+1})$$

with q arbitrary but subject to $p(p-1) \leq \ell+1$. We recall $\ell+1$ to be the total number of interpolation points at 0. Hence the norm of the backward error matrix can be controlled by a sharper inequality. As a consequence we can select the parameters m and s by comparing to $\theta^\star$ the value $\alpha_q(\mathbf{B})$ for a suitable q instead of $\|\mathbf{B}\|$.

Runtime backward error analysis

Provided that we are able to compute the coefficients of the backward error function $h_{m+1}(x)$, it is straightforward to compute the root of $\tilde{h}_{m+1}(x) - \mathrm{tol} \cdot x = 0$. In fact, by running few iterations of the Newton root-finder method, we reach a relative precision of a couple of digits that is enough for our purposes (provided that we truncate the remaining digits so that we underestimate $\theta^\star$ rather than overestimate it).

The delicate task is to compute in double precision arithmetic the coefficients of the power series expansion of

$$h_{m+1}(x) = \log(\mathrm{e}^{-x}p_m(x)),$$

without resorting to any slow software of symbolic calculus. The difficulty is hidden in the calculus of the coefficients of $\mathrm{e}^{-x}p_m(x)$. Let us denote the $m+1$ coefficients of the explicit form of $p_m(x)$ by $\{a_i\}_{i=0}^{\infty}$, where $a_k = 0$ if k is larger than m. Then, by the Cauchy product formula we have that

$$\mathrm{e}^{-x}p_m(x) = \sum_{k=0}^{\infty}(\sum_{j=0}^{k} \frac{(-1)^j}{j!}a_{k-j})x^k$$

that, in its expanded form looks like

$$a_0 + (a_1 - a_0)x + (a_2 - a_1 + \frac{a_0}{2})x^2 + (a_3 - a_2 + \frac{a_1}{2} - \frac{a_0}{6})x^3 + \cdots$$

which it's prone to numerical instabilities because, even for low degrees, we encounter huge loss of significance due to catastrophic cancellation. In fact, for legit polynomial approximants, we have that when $i \leq m$ the coefficient a_i is really close to the corresponding coefficient of the exponential function. Therefore, the coefficients of x^k with $k > 0$ are extremely close to zero and their value is computed by means of algebraic sums of values extremely close to each other. Clearly, for $i \leq \ell+1$ we have that a_i is exactly equivalent to the corresponding coefficient of the exponential function and hence the problem is just

delayed to the coefficients of higher order.

In order to tackle this issue, we consider the residue function $r_m(x) = \mathrm{e}^x - p_m(x)$ and the identity

$$h_{m+1}(x) = \log(1 - \mathrm{e}^{-x} r_m(x)).$$

Being the coefficients of $r_m(x)$ and those of the inverse exponential function extremely different in magnitude the loss of significance is averted provided that the coefficients of $r_m(x)$ are computed accurately, which is the problem we face next.

The coefficients of the function $r_m(x)$ can be easily represented operating according the following steps. First we extend $\sigma_m = (z_0, z_1, \ldots, z_m)$ with infinitely many points, without loss of generality in 0. Then, we develop exactly in Newton series the exponential function and we set the first m divided differences to 0. We have that

$$r_m(x) = \sum_{i=m+1}^{\infty} d[z_0, z_1, \ldots, z_i] \prod_{j=0}^{i-1} (x - z_j)$$

where, we say it again, $d[z_0, z_1, \ldots, z_i]$ is the ith divided difference of the exponential function over the interpolation sequence $\sigma_m = (z_0, z_1, \ldots, z_m)$. Due to practical constraints, we will just pick L additional points in zero and not infinite, with $M := L + m$ sufficiently larger than m.

Thanks to the routine `dd_phi`, that we developed in Chapter 2, we can compute the divided differences $d[z_0], d[z_0, z_1], \ldots, d[z_0, z_1, \ldots, z_m]$ in just a fraction of the time taken by the previously existing routines. We can therefore compute relatively quickly the coefficients of $r_m(x)$ following the procedure that we just explained. At this point, it is straightforward to derive the coefficients of $h_{m+1}(x)$ through repeated applications of the Cauchy formula for the series product, that is equivalent to convolving the vectors containing their coefficients. With the coefficients of $h_{m+1}(x)$ calculated accurately, we can now compute the desired $\theta^\star$.

However, for our specific purposes, we need to perform the backward error analysis several times at each call of our routine. In fact, suppose we decide to interpolate the exponential function at σ_m and we hence compute the relative value $\theta^\star$. Suppose also that it appears necessary to scale the matrix $\mathbf{A}$ for the approximation to be accurate. Now, we have that the eigenvalues of $s^{-1}\mathbf{A}$ are a scale of those of $\mathbf{A}$ and the Ritz's values too. Therefore, the right thing to do is to repeat the backward error analysis over the scaled sequence $s^{-1}\sigma_m$. Then, it is possible that a different scaling s is to be considered and therefore another launch of the backward error analysis is necessary over another scaled interpolation set. Since it is likely that we have to iterate this procedure many times, this operation could result in being costly.

To tackle this issue, we exploit the results obtained in Chapter 2. It was shown that, if $a(M)$ is the vector having on its ith entry the ith coefficient of any degree M polynomial, and $d(M)$ is the vector having on its ith entry the ith divided difference over $\sigma_M = (z_0, z_1, \ldots, z_M)$ of said polynomial, then we have that

$$d(M) = \mathbf{C}(z_0, z_1, \ldots, z_M) \cdot a(M)$$

and

$$a(M) = \mathbf{E}(z_0, z_1, \ldots, z_M) \cdot d(M)$$

where $\mathbf{C}(z_0, z_1, \ldots, z_M)$ and $\mathbf{E}(z_0, z_1, \ldots, z_M)$ are square upper triangular matrices one the inverse of the other.

Their analytically definition is given in Chapter 2 and it involves two class of symmetrical polynomials. The firsts are the complete homogeneous symmetric polynomials of degree $k-j$ over the variables $z_0, z_1, \ldots, z_j$, that are:

$$c_{k-j}(z_0, z_1, \ldots, z_j) = \sum_{0 \leq i_1 \leq \cdots \leq i_{k-j} \leq j} z_{i_1} \cdots z_{i_{k-j}},$$

with $c_{k-j}(z_0, z_1, \ldots, z_j)$ equal to 0 if $k-j < 0$ and to 1 if $k-j = 0$. In particular $\mathbf{C}(z_0, z_1, \ldots, z_M)$ is the matrix having $c_{k-j}(z_0, z_1, \ldots, z_j)$ on its $(j+1, k+1)$ position. As an example, we show the matrix $\mathbf{C}(z_0, z_1, z_2, z_3)$ mapping $a(3)$ into $d(3)$:

$$\mathbf{C}(z_0, z_1, z_2, z_3) = \begin{pmatrix} 1 & z_0 & z_0^2 & z_0^3 \\ 0 & 1 & z_0 + z_1 & z_0^2 + z_0 z_1 + z_1^2 \\ 0 & 0 & 1 & z_0 + z_1 + z_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

The seconds are the elementary symmetric polynomials of degree $k-j$ over the variables $z_0, z_1, \ldots, z_{k-1}$, that are:

$$e_{k-j}(z_0, z_1, \ldots, z_{k-1}) = \sum_{0 \leq i_1 < \cdots < i_{k-j} \leq k-1} z_{i_1} \cdots z_{i_{k-j}},$$

with $e_{k-j}(z_0, z_1, \ldots, z_{k-1})$ equal to 0 if $k-j < 0$ and to 1 if $k-j = 0$. In particular $\mathbf{E}(z_0, z_1, \ldots, z_M)$ is the matrix having $(-1)^{k-j} e_{k-j}(z_0, z_1, \ldots, z_{k-1})$ on its $(j+1, k+1)$ position. As an example, we show the matrix $\mathbf{E}(z_0, z_1, z_2, z_3)$ mapping $d(3)$ into $a(3)$:

$$\mathbf{E}(z_0, z_1, z_2, z_3) = \begin{pmatrix} 1 & -z_0 & z_0 z_1 & -z_0 z_1 z_2 \\ 0 & 1 & -z_0 - z_1 & z_0 z_1 + z_0 z_2 + z_1 z_2 \\ 0 & 0 & 1 & -z_0 - z_1 - z_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

By means of these matrices, we can represent the vector having on its entries the first $M+1$ coefficients of $r_m(x)$ as

$$\mathbf{E}(z_0, z_1, \ldots, z_M) \mathbf{Y}_{M,m} \mathbf{C}(z_0, z_1, \ldots, z_M) a(M)$$

where $a(M)$ is the vector having on its ith entry the ith coefficient of the exponential function truncated at M and $\mathbf{Y}_{M,m}$ is the null matrix having the L sized identity matrix $\mathbf{I}_L$ on its bottom right corner.

If the matrix-vector products are considered from right to left, it is easy to see that we are following the passages described above to obtain the wanted coefficients of $r_m(x)$. In fact, $\mathbf{C}(z_0, z_1, \ldots, z_M) a(M)$ is the vector containing the divided differences of the exponential function. By left multiplying such vector by $\mathbf{Y}_{M,m}$, we obtain

$$\mathbf{Y}_{M,m} \mathbf{C}(z_0, z_1, \ldots, z_M) a(M),$$

that is the vector containing the divided differences of the residual function $r_m(x)$. Finally, by left multiplying this vector by $\mathbf{E}(z_0, z_1, \ldots, z_M)$, we obtain the coefficients of $r_m(x)$. If we rewrite the previous formula block wise

$$\begin{pmatrix} \mathbf{E}_{1,1} & \mathbf{E}_{1,2} \\ & \mathbf{E}_{2,2} \end{pmatrix} \begin{pmatrix} 0 & \\ & \mathbf{I}_{M-m} \end{pmatrix} \begin{pmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ & \mathbf{C}_{2,2} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$$

with blocks $\mathbf{C}_{j,k}$, $\mathbf{E}_{j,k}$ and a_j of suitable dimensions, we can easily see that it all boils down to the computation of the vector

$$\begin{pmatrix} \mathbf{E}_{1,2}\mathbf{C}_{2,2}a_2 \\ a_2 \end{pmatrix}$$

whose bottom part is trivial, thus we are left with the task of computing the vector $\mathbf{E}_{1,2}\mathbf{C}_{2,2}a_2$.

From Claim 2.1 with $p = j$ and $z_j = 0$ we have that

$$c_{k-j}(z_0, z_1, \ldots, z_j) = c_{k-j}(z_0, z_1, \ldots, z_{j-1}),$$

hence we know that the $(j+1, k+1)$th entry of $\mathbf{C}(z_0, z_1, \ldots, z_M)$ equals the (j, k)th one for $j > m$ and provided that $z_i = 0$ for $i = m+1, m+2, \ldots, M$.

From Claim 2.2 with $p = k - 1$ and $z_{k-1} = 0$ we have that

$$e_{k-j}(z_0, z_1, \ldots, z_{k-1}) = e_{k-j}(z_0, z_1, \ldots, z_{k-2}),$$

hence we know that the $(j+1, k+1)$th entry of $\mathbf{E}(z_0, z_1, \ldots, z_M)$ equals the (j, k)th one for $k > m$ and provided that $z_i = 0$ for $i = m+1, m+2, \ldots, M$.

By showing this, we have proven that the choice of taking the additional $L = M - m$ interpolations points equal to zero leads $\mathbf{C}_{2,2}$ to be an upper triangular band matrix and $\mathbf{E}_{1,2}$ to be a lower triangular band matrix. Therefore, knowing just the L entries of $\mathbf{C}_{2,2}$ corresponding to the bottom row of $\mathbf{C}_{1,1}$:

$$c_{k-m}(z_0, z_1, \ldots, z_m), \quad k = 0, 1, \ldots, N - m - 1,$$

and the m entries of $\mathbf{E}_{1,2}$ corresponding to the rightmost column of $\mathbf{E}_{1,1}$:

$$e_{m-j}(z_0, z_1, \ldots, z_{m-1}), \quad j = 0, 1, \ldots, m - 1,$$

it is possible to form the $m \times L$ matrix $\mathbf{E}_{1,2}\mathbf{C}_{2,2}$. Furthermore we know that each entry of the matrix $\mathbf{E}_{1,2}\mathbf{C}_{2,2}$ is homogeneous in $z_0, z_1, \ldots, z_m$. To see that, consider for example the product of the matrices

$$\begin{pmatrix} e_0 & 0 & 0 & 0 \\ e_{-1} & e_0 & 0 & 0 \\ e_{-2} & e_{-1} & e_0 & 0 \\ e_{-3} & e_{-2} & e_{-1} & e_0 \end{pmatrix} \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ 0 & c_0 & c_1 & c_2 \\ 0 & 0 & c_0 & c_1 \\ 0 & 0 & 0 & c_0 \end{pmatrix}$$

whose structures mimic the band structure of the matrices $\mathbf{E}_{1,2}$ and $\mathbf{C}_{2,2}$ (even though their dimensions differ). The indexing has been assigned in order to give the idea of how

the homogeneous degrees vary moving away from the diagonals. Their product is

$$\begin{pmatrix} e_0h_0 & e_0c_1 & e_0c_2 & e_0c_3 \\ e_{-1}c_0 & e_0h_0+e_{-1}c_1 & e_0c_1+e_{-1}c_2 & e_0c_2+e_{-1}c_3 \\ e_{-2}c_0 & e_{-1}c_0+e_{-2}c_1 & e_0c_0+e_{-1}c_1+e_{-2}c_2 & e_0c_1+e_{-1}c_2+e_{-2}c_3 \\ e_{-3}c_0 & e_{-2}c_0+e_{-3}c_1 & e_{-1}c_0+e_{-2}c_1+e_{-3}c_2 & e_0c_0+e_{-1}c_1+e_{-2}c_2+e_{-3}c_3 \end{pmatrix}$$

and it is immediate to notice that the homogeneity of the entries its preserved. This is especially important because, once we build the matrix $\mathbf{E}_{1,2}\mathbf{C}_{2,2}$ for a given set of points $\{z_0, z_1, \ldots, z_m\}$, when we need to perform the backward error analysis at different scales of the set, for instance $t \cdot \{z_0, z_1, \ldots, z_m\}$, we simply have to perform the products

$$t^{m+1}\mathbf{T}_{m,m}^{-1}\mathbf{E}_{1,2}\mathbf{C}_{2,2}\mathbf{T}_{L,L}a_2 \tag{4.18}$$

where $\mathbf{T}_{p,p}$ is the p sized diagonal matrix having t^j on its $(j+1, j+1)$ entry. The scalar multiplication with t^{m+1} appearing in the formula is justified by the fact that e_0 has degree $m+1$ while c_0 has degree 0 and the other entries follow accordingly with their indexes.

We now list a simple MATLAB algorithm for computing the matrix $\mathbf{E}_{1,2}\mathbf{C}_{2,2}$ given the interpolation sequence $\sigma_m = (z_0, z_1, \ldots, z_m)$. We suppose, without loss of generality, that σ_m is normalized so that

$$\max_{i=0,1,\ldots,m} |z_i| = 1.$$

This algorithm exploits the recursive structure of both the matrix $\mathbf{C}(z_0, z_1, \ldots, z_m)$ and $\mathbf{E}(z_0, z_1, \ldots, z_m)$ that was highlighted in the Chapter 2 of this thesis. In forming the matrix $\mathbf{E}_{1,2}\mathbf{C}_{2,2}$ it was exploited the pattern for which its $(j+1, k+1)$th entry always contains its (j, k)th one.

Furthermore, it was taken in account the number ℓ of interpolation points equal to zero that are in the tail of the interpolation sequence $\sigma_m = (z_0, z_1, \ldots, z_m)$ in order to save calculations. This is going to be particularly important since, for the class of interpolating polynomials we are going to elect, the number of tailed zeros greatly outnumbers the points lying outside the origin.

```
 1. function EC = pts2EC( z, m, M, ell )
 2. % Compute the matrix E_12C_22 given the
 3. % interpolation sequence z.
 4.   e = [ 0, 1, zeros( 1,m+1 ) ];
 5.   for k = 1:m-ell+1
 6.     for j = k+1:-1:1
 7.       e( j+1 ) = e( j ) - z( k ) * e( j+1 );
 8.     end
 9.   end
10.   e = [ zeros( 1,ell ), e( 2:m-ell+2 ) ];
11.   c = [ 1, zeros( 1,M-m-1 ) ];
12.   for k = 0:m
13.     for j = 1:M-m-1
14.       c( j+1 ) = c( j+1 ) + z( k+1 ) * c( j );
15.     end
16.   end
```

```
17.   EC = zeros( m+1,M-m );
18.   EC( 1,: ) = e( 1 ) * c;
19.   for i = 2+ell:m+1
20.     EC( i,: ) = [ 0, EC( i-1,1:M-m-1 ) ] + e( i ) * c;
21.   end
22. end
```

4.3.2 Final selection of the polynomial interpolation

In this section, we resume the choices that will lead to the determination of the parameters κ, ℓ and s characterizing the polynomial interpolant $p_{\kappa+\ell+1}(x)$ of the exponential function.

The first step is to shift the matrix $\mathbf{A}$ as we set for ourselves and we work with the shifted matrix

$$\mathbf{B} := \mathbf{A} - \mu\mathbf{I}.$$

The second step is to compute some values $\alpha_q(\mathbf{B})$ in order to assess the non-normality of $\mathbf{B}$ and to perform a much sharper overestimate of the norm of the backward error matrix. In order to save computational effort, we employ the criterion [2, Formula (3.13)], that indicates if it is computationally worth to employ $\alpha_q(\mathbf{B})$ in place of using just $\|\mathbf{B}\|$. Namely if

$$\|\mathbf{B}\| \leq 2\frac{\theta_{\max}}{M} q_{\max}(q_{\max}+3)$$

then it is not worth to compute any $\alpha_q(\mathbf{B})$. The integer $q_{\max}$ represents the largest integer q for which we are willing to compute $\alpha_q(\mathbf{B})$. In our case, we set $q_{\max} = 7$, broadly in line with other routines from the literature (see for example [2]). The integer M represents, instead, the maximum order of polynomial approximation that we are willing to take into account. In our case, we set $M = 56$, once again a value in line with other routines from the literature. For simplicity, and only for evaluating the previous formula, instead of computing θ_M for the actual set of extended Ritz's values we are going to use, we approximate now this value by the θ_M corresponding to the Taylor interpolation of degree M. This approximation is usually quite close to the actual value. In case it was not, that would not harm the accuracy of the routine, since at worst it would only lead to a slightly increased computational effort. In order to quickly compute the approximation of θ_M, we use the routine `bea_tay_approx` that we introduced at the end of Section 3.3.1 of this manuscript. In case the criterion tells us that is worth to compute the values $\alpha_q(\mathbf{B})$, as a rule of the thumb, we compute $\alpha_q(\mathbf{B})$ until $q = q_{\max}$ or until we find a $q^\star \leq q_{\max}$ such that

$$\alpha_{q^\star-1}(\mathbf{B}) \geq 0.8\,\alpha_{q^\star}(\mathbf{B}),$$

that is when the values $\alpha_q(\mathbf{B})$ stop decreasing sharply. At this point we set

$$\ell = q^\star(q^\star-1) - 1,$$

so that we are allowed to employ the value $\alpha_{q^\star}(\mathbf{B})$ in place of $\|\mathbf{B}\|$, and

$$\kappa = 2q^\star.$$

We stress that the value we assigned to κ is expressly meant to be quite small in com-

parison with the degree of approximation $m = \kappa + \ell$. This choice is due to the fact that we desire to run the Arnoldi (or Lanczos) process up to low degrees in order to avoid the numerical inefficiencies we called out at the beginning of this chapter.

The third step consists in running the Arnoldi (or Lanczos) process in order to obtain the matrices $\mathbf{V}_\kappa$ and $\mathbf{H}_\kappa$. From $\mathbf{H}_\kappa$ we compute the Ritz's values $\{\rho_1, \rho_2, \ldots, \rho_\kappa\}$, then we extend them with $\ell + 1$ zeroes and we form the matrix

$$\mathbf{E}_{1,2}\mathbf{C}_{2,2}$$

as we explained in detail in the previous section. Now, we have to be careful: suppose the backward error analysis at $\{\rho_1, \rho_2, \ldots, \rho_\kappa\} \cup \{0, 0, \ldots, 0\}$ returns a certain value $\theta^\star$ such that we need to apply a scaling s strictly larger than 1. As we mentioned already, if we scale the matrix $\mathbf{B}$ of a factor s^{-1}, also the Ritz's values will be scaled of the same factor. Therefore, we need to rerun the backward error analysis over the set $s^{-1}\{\rho_1, \rho_2, \ldots, \rho_\kappa\} \cup \{0, 0, \ldots, 0\}$, which will return a different $\theta^\star$ and therefore to a different scaling factor s^{-1}. If we iterate this process over and over we will eventually encounter two values $\theta^\star$ yielding the same scaling factor s^{-1}. This is the scaling factor we will adopt for our approximation.

The existence of such scaling factor is granted by the fact that when s tends to infinity then $s^{-1}\alpha_{q^\star}(\mathbf{B})$ tends to zero and the extended Rit'z values tend to become a set of m interpolation points in zero. Under very weak assumptions (it is enough that $\kappa+\ell+1 > 1$), we know that for such an interpolation set $\theta^\star > 0$. In fact the interpolation of the exponential function at zero is equivalent to a truncated Taylor technique. Clearly, for running the backward error analysis over different scales of the same interpolation set, we exploit formula (4.18).

Now that we have the desired interpolation sequence of length $\kappa+\ell+1$ and the scaling parameter s, the final step is to form

$$v^{(1)} = p_m(s^{-1}\mathbf{A})v$$

paying attention to recycle the columns of $\mathbf{V}_\kappa$ as explained in the course of this section. Then we march, sub-step after sub-step, without changing the interpolation set, until we obtain the desired approximation $v^{(s)}$.

4.4 Numerical Experiments

In this Section about numerical experiments, we test our two routines: `explhe` (EXPonential Leja-Hermite, see [9]) and `pkryexp` (Polynomial KRYlov EXPonential, see [11]) against the state-of-the-arts MATLAB routines for the approximation of the matrix exponential in double precision arithmetic. These routines are:

- `expmv`, the algorithm of Al-Mohy and Higham (see [2]), that is based on a truncated Taylor approximation of degree m of the exponential function. As today, this routine represents one of the most reliable and fast implementation of a polynomial method for computing the action of the matrix exponential on a vector;

- `phipm`, the algorithm of Niesen and Wright (see [46]), that is based on a Krylov subspaces approximation of degree m. As today, this routine is considered the

state-of-the-arts MATLAB implementation of a Krylov method with full reorthogonalization for computing the action of the matrix exponential, [21]. Differently from `expmv`, `phipm` can also compute linear combinations of the functions $\varphi_\ell(x)$ (see 1.3). However, we do not test this feature of `phipm` but just its performance when it comes to the computation of the action of the matrix exponential.

This section is divided in two parts: in the first part we will examine the behavior of the four routines over the test set $\mathcal{A}$. The set $\mathcal{A}$ is a very modest collection of matrices that are recurrent in the field of numerical integration and that have very specific spectral distributions. In particular, in the set $\mathcal{A}$ will appear the matrices stemming from the discretization by second-order finite differences of the diffusion and advection–diffusion partial differential equation, characterized by skinny spectra distributed along the real axis. In addition to that, we include the matrices coming from the discretization by second-order finite differences of the advection operator and those of the free Schrödinger operator, both characterized by purely imaginary spectra.

In the second part we will assess the performances of the four routines over a large set $\mathcal{S}$ built out of 1980 matrices from the *SuiteSparse Matrix Collection* (formerly the *University of Florida Sparse Matrix Collection*), that is a *"widely used set of sparse matrix benchmarks collected from a wide range of applications"* [1]. For more detailed information see [65].

The experiments were performed using the 64-bit (glxna64) version of MATLAB® 9.2 (R2017a) with the `-singleCompThread` option over a a machine equipped with 16Gb of RAM and 4 Intel Core i7 processors running at 3.30GHz.

4.4.1 Tests over the test set $\mathcal{A}$

In this experiment we want to prove a point: a convenient choice of the interpolation set leads to an improved accuracy and a reduced computational effort. For each matrix from the set $\mathcal{A}$, we compare and measure the forward error committed by the four routines and the computational effort measured in terms of the number of matrix-vector products.

In addition to that, we report the average elapsed CPU time (on 100 launches) that each routine takes to compute the action of the matrix exponential on a vector. The reason for this is that the mere number of matrix-vector products can sometimes be misleading. For example, the routine `phipm`, based on Krylov method, performs a particularly low number of matrix-vector products while the average CPU time does not reflect this efficiency. This is because when $\mathbf{A}$ is very sparse, the cost of a matrix-vector product has a complexity similar to a vector-vector product. In this case, the complexity of the Arnoldi process roughly shifts from $O(mN^2)+O(m^2N)$ to $O(mN)+O(m^2N)$, revealing that an important share of the final cost is not represented by the m matrix-vector products. On the other hand, the routine `expmv`, based on Taylor interpolation, often performs the highest number of matrix-vector products while its extreme simplicity and absence of side computations make up for it.

[1] The Collection is widely used by the numerical linear algebra community for the development and performance evaluation of sparse matrix algorithms. It allows for robust experiments because performance results with artificially-generated matrices can be misleading. Its matrices cover a wide spectrum of domains, include those arising from problems with underlying 2D or 3D geometry and those that typically do not have such geometry.

For this experiment, we asked the tolerance to be equal to 2^{-53}, corresponding to the double precision arithmetic. The trusted solution $e^{\mathbf{A}}v$ is obtained using `expkpotf` to compute $e^{\mathbf{A}}$ in high precision arithmetic with tol set to $\text{eps}^2 \approx 4.93\text{e-}32$.

Two-dimensional advection-diffusion matrices

We consider the discretization by second order finite differences of the advection-diffusion partial differential equation

$$\frac{\partial u}{\partial t} + \vec{b} \cdot \nabla u = d\nabla^2 u$$

defined in the two-dimensional spatial domain $[0,1]^2$, subject to homogeneous Dirichlet boundary conditions and initial solution $u_0(x,y) = 16x(1-x)y(1-y)$. The discretization is done with 49 inner points and thus $h = 1/50$. The diffusion coefficient is fixed to $d = 1/100$ and the advection term is $\vec{b} = (b,b)$. The grid Péclet number turns out to be

$$\text{Pe} = \frac{hb}{2d} = b.$$

Suppose $t_0 = 0$ and that we are interested in computing the solution at time $t = 3$. The matrices considered in this example have size $N = 2401$.

In the first example we consider the truly diffusive case, where $b = 0$, leading to a symmetric matrix $\mathbf{A}$. After we shift $\mathbf{A}$, the (estimated) "rectangle" $R(t\mathbf{B})$ collapses to the real interval $[-299, 299]$. Table 4.1 collects the results. From this test we can learn that, as

Method	Substeps	Act. prods	Rel. fwd. err.	Avg. Elaps. CPU Time
`expmv`	31	1501	6.33e-15	2.92e-02
`explhe`	44	1146	2.91e-15	2.99e-02
`pkryexp`	31	903	7.58e-15	2.16e-02
`phipm`	5	256	2.55e-15	1.53e-02

Table 4.1: Results for the diffusion case ($b = 0$), tolerance set to 1.11e-16.

we expected, the methods take more and more matrix-vector products the less information on the spectrum of $\mathbf{A}$ they exploit. In fact `expmv`, based on a truncated Taylor series that interpolates the exponential function and its derivatives at the origin, needs almost 6 times more matrix-vector products than `phipm`, based instead on Krylov method. On the other hand both `explhe` and `pkryexp`, with their wisely placed interpolation points, need sensibly less matrix-vector products than `expmv`, while `phipm` stands out as the method requiring by far the least amount of matrix-vector products. The reason for this is that `phipm` employs more Ritz's values than `pkryexp` (and clearly of `expmv` and `explhe`) and, in addition to that, it rearranges the interpolation points at each sub-step, tailoring them to the application vector.

In the second example we consider the advection-diffusion case with $b = 0.25$. The (estimated) rectangle $R(t\mathbf{B})$ is $[-299, 299] + \mathrm{i}[-73, 73]$. Table 4.2 collects the results. The first element to be noticed is that `phipm` takes drastically more time than in the previous example. This can be only partially justified by the additional 75 matrix-vector products it computes in this second example. The truth is that being $\mathbf{A}$ not Hermitian anymore

Method	Substeps	Act. prods	Rel. fwd. err.	Avg. Elaps. CPU Time
`expmv`	31	1488	1.09e-14	2.39e-02
`explhe`	31	970	1.26e-14	1.83e-02
`pkryexp`	30	1046	4.99e-15	2.58e-02
`phipm`	6	331	2.58e-14	7.64e-02

Table 4.2: Results for the advection-diffusion case ($b = 0.25$), tolerance set to 1.11e-16.

the routine is forced to employ the Arnoldi process, which is considerably more expensive than the Lanczos variant. Similarly, we can guess that `pkryexp` suffers the passage to Arnoldi process as well but somehow in a reduced measure. This is because `pkryexp` had to perform the Arnoldi process only once against the 6 times of `phipm`. Finally, we notice that both `expmv` and `explhe` selected exactly the same sub-stepping strategy but the latter performed two thirds of the matrix-vector products computed by `expmv`. This is, therefore, purely due to the superior choice of the interpolation points performed by `explhe`.

In the third and final example of this series, we consider the case $b = 0.5$. The (estimated) rectangle $R(t\mathbf{B})$ is $[-299, 299] + \mathrm{i}[-146, 146]$. Table 4.3 collects the results. What is to be noticed in this final example is that as the spectrum of $\mathbf{B}$ gets fatter the

Method	Substeps	Act. prods	Rel. fwd. err.	Avg. Elaps. CPU Time
`expmv`	31	1458	1.29e-14	2.37e-02
`explhe`	36	1156	2.03e-15	2.09e-02
`pkryexp`	30	1104	1.70e-14	2.63e-02
`phipm`	4	331	3.05e-11	1.02e-01

Table 4.3: Results for the advection-diffusion case ($b = 0.5$), tolerance set to 1.11e-16.

performance of routine `explhe` gets slightly worse. Nevertheless `explhe` proved to be the fastest and more accurate routine in this case. On the other hand, we see that `phipm` ran into problems for it took a while to return a not satisfying approximation. We evince from this example that the underlying Krylov methods must be having troubles in capturing the nature of $\mathbf{B}$. In fact, `pkryexp` too took quite long in returning an approximation, but, differently from `phipm`, its strictly polynomial nature together with a rigorous analysis of the backward error, kept it from committing a large error.

Schrödinger matrix

In this example we consider the discretization by second order central finite differences of the free Schrödinger operator $\mathrm{i}\partial_{xx}$ in the one-dimensional spatial domain $[-1, 1]$ with homogeneous Dirichlet boundary conditions. The space step size h is $1/35$, the matrix $\mathbf{A}$ has dimension 69×69 and is skew-Hermitian. The application vector is the discretization of $u_0(x) = 1/(2+\cos(2\pi x)) - 1/3$. Suppose $t_0 = 0$ and that we are interested in computing the solution at time $t = 2$. The (estimated) rectangle $R(t\mathbf{B})$ is $\mathrm{i}[-4857, 4857]$. Table 4.4 collects the results.

Method	Substeps	Act. prods	Rel. fwd. err.	Avg. Elaps. CPU Time
`expmv`	249	13373	5.61e-11	5.62e-02
`explhe`	350	11848	2.04e-13	9.42e-02
`pkryexp`	246	10553	5.10e-13	6.61e-02
`phipm`	×	×	×	×

Table 4.4: Results for the Schrödinger case, tolerance set to 1.11e-16.

Before commenting on the data collected in this case, we have to stress that the CPU time reported here just gives a broad idea of the efficiency of a method. In fact, the matrix $\mathbf{A}$ just have size 69 while the problems from the real applications can produce matrices with far more rows and columns. The first thing to be noticed is the absence of `phipm` in Table 4.4, this is due to the fact that this routine fails to find a sequence of sub-steps delivering a satisfying approximation. We notice that both `explhe` and `pkryexp` need considerably less matrix-vector products than `expmv` to deliver the desired approximation

In addition to that, Table 4.4 suggests that `pkryexp` performs slightly better than `explhe` in the sense that it requires to compute less matrix-vector products. While this is true on the one hand, on the other hand, we point out that `explhe` performs averagely 34 matrix-vector products per sub-step against the 43 of `pkryexp`, suggesting that the Leja–Hermite interpolation sequence is still a superior choice in this scenario.

Lastly, we notice that the routine `expmv` returns an inaccurate approximation. This is due to the hump phenomenon which is made even more evident in the Schrödinger case because $\left\|\mathrm{e}^{\mathbf{A}}\right\|_2$ equals 1 while $\|\mathbf{A}\|_2$ is very large. The choice of good interpolation points here is crucial, it can be seen by the fact that even though `pkryexp` selects a similar sub-stepping strategy to `expmv`, it commits a much smaller error.

Conclusion

From these tables we can infer that in those cases where Lanczos process can be used in place of Arnoldi, the routines `phipm` and `pkryexp` are by far the most effective. In the other cases, that is when the spectra of the input matrices are skinny, we notice that `explhe` is the most effective routine. We remark that, despite the reported CPU time is to be considered with a grain of salt (the matrices in $\mathcal{A}$ are moderately small), there is an inconsistency between the number of matrix-vector products performed by `phipm` and the CPU time shown. This is due to the inefficiencies affecting the Arnoldi process we discussed at the end of section 1.1.3. On the other hand, the importance of the number of matrix-vector products should not be underestimated. For example, a matrix-vector product may require several all-to-all communications in a parallel application and a large number of communications could deteriorate the strong scalability of the algorithm.

4.4.2 Tests over the test set $\mathcal{S}$

In the section we assess the performances of the four routines over a large set $\mathcal{S}$ built out of the 1980 square matrices from the *SuiteSparse Matrix Collection* (formerly the *University of Florida Sparse Matrix Collection*), that is a *"widely used set of sparse matrix*

benchmarks collected from a wide range of applications".

To give to the reader a quick overview of the characteristics of $\mathcal{S}$, in Table 4.5 we report the number of matrices that have or have not a (skew-)Hermitian structure for subsets of $\mathcal{S}$ characterized by similar size. It is worth notice that the larger the matrices from

	not Hermitian	(skew-)Hermitian	total
$N \leq 10^2$	40	37	77
$10^2 < N \leq 10^3$	216	151	367
$10^3 < N \leq 10^4$	350	302	652
$10^4 < N \leq 10^5$	235	327	562
$10^5 < N \leq 10^6$	87	156	243
$10^6 < N \leq 10^7$	24	49	73
$10^7 < N \leq 10^8$	2	10	12
total	953	1027	1980

Table 4.5: Number of matrices that have or have not (skew-)Hermitian structure for subsets of $\mathcal{S}$ characterized by similar size.

$\mathcal{S}$ are, the more likely they show a (skew-)Hermitian structure. Moreover, even though this is not displayed, a similar trend can be noticed in the sparsity of the matrices from $\mathcal{S}$: the larger they get, the sparser they tend to be. The reason for this lies in the fact that researchers and engineers are willing to handle large matrices only provided that such matrices are really sparse or show some symmetry. These trends we just highlighted clearly suggest that the performance of a routine may largely vary in function of the size of $\mathbf{A}$. Therefore, running tests over a set large and heterogeneous as $\mathcal{S}$ is fundamental to properly assess the quality of a routine for the computation of the action of the matrix exponential.

The problem with the set $\mathcal{S}$ is that it needs to be manipulated a bit before we are ready to use it for our tests. In fact, many elements from $\mathcal{S}$ have norm in the order of the tens of millions and their exponential may not even be representable in machine precision. Clearly, it would not be correct to normalize every matrix from $\mathcal{S}$ to a specific predetermined norm. First of all, because we would introduce a bias in our tests. Second of all, we would like the matrices with norm larger (or smaller) than the average to keep having a larger (or smaller) norm than the average, so that they can keep their "nature". A solution we came up with is to normalize every matrix from $\mathcal{S}$ to

$$\omega \cdot |\log_2(\|\mathbf{A}\|)|$$

for some positive scalar parameter ω. This transformation fits our requirements for it associates to those matrices $\mathbf{A}$ with a large norm a larger than average norm and vice-versa. The only exception would arise when $\mathbf{A}$ originally has norm close to 0. Luckily for us, there are no such matrices in $\mathcal{S}$. We plot in Figure 4.2 the norm of the elements of $\mathcal{S}$ sorted by their new norm when we set, for example, $\omega = 16$. We invite the reader to notice that the vast majority of the matrices now have a norm between 10 and 1000, while a very small subset has norm smaller than 10. We believe this distribution is heterogeneous

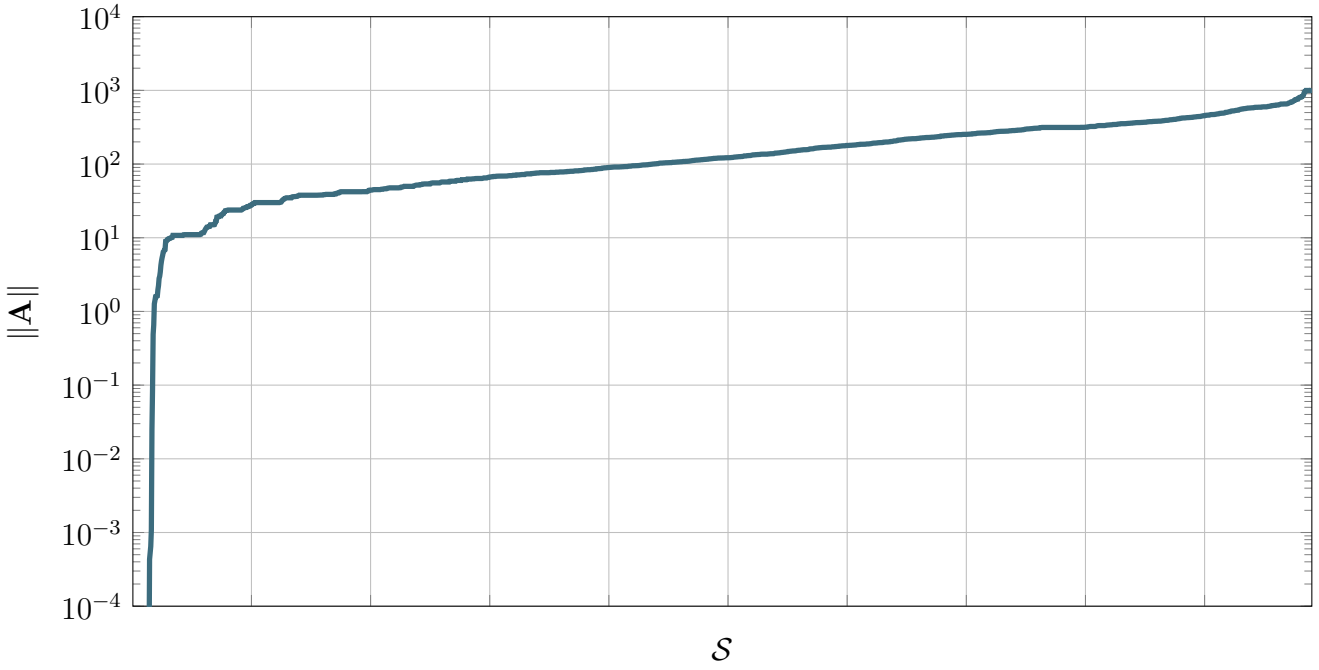


Figure 4.2: New norms ($\omega = 16$) of the matrices from $\mathcal{S}$ (y-axis) displayed in ascending order.

enough while it allows us to focus on hard problems, i.e. those matrices showing a large norm. Therefore, in the next section, when we treat the choice of ω we will pick an ω of this order of magnitude.

As important as the choice of $\mathbf{A}$, is the choice of the vector v, which is not given in the considered test set. We discarded a priori the possibility of using randomly generated vectors for they do not allow for a perfect replicability of the experiments. We found that a satisfying compromise is to generate v as

$$\frac{\mathrm{abs}(\mathbf{A})u}{\|\mathrm{abs}(\mathbf{A})u\|_2},$$

where u is a vector of ones of size compatible with $\mathbf{A}$ and where with $\mathrm{abs}(\mathbf{A})$ we mean the matrix whose (i,j)th is the absolute value of the (i,j)th entry of $\mathbf{A}$. Unfortunately, for some matrices from $\mathcal{S}$ we had that such a v was in the kernel of $\mathbf{A}$. For such matrices we built v using $\mathbf{A}^\star$ in the formula above. Of these matrices, 26 where (skew-)Hermitian, therefore the new v is in the kernel of $\mathbf{A}$ as well. We decide to discard these 26 matrices and to move on.

We noticed that the routine `phipm` behaves very unpredictably. On 21 matrices from $\mathcal{S}$ we noticed that `phipm` is not able to determine a sub-stepping strategy to accomplish the task in practical time. In one case, `phipm` exceeded the 16Gb of memory of the machine running the test. This issue was provoked by the attempt of allocating the dense matrix $\mathbf{V}_m$, necessary to the Arnoldi process, for a large m, which required 18Gb of storage space. Therefore, we exclude also these 22 matrices from $\mathcal{S}$.

The set $\mathcal{S}$ now counts 1932 matrices left.

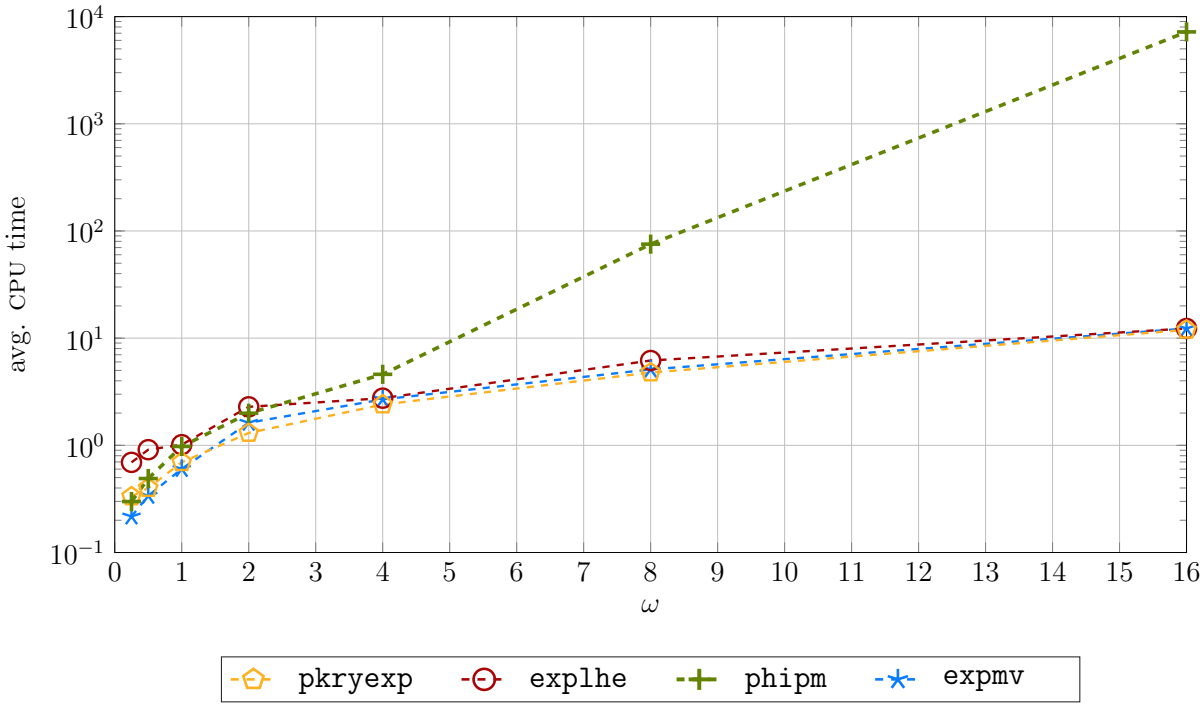


Figure 4.3: Average CPU time (y-axis) against $\omega = 2^j$ (x-axis) with $j = -2, -1, \ldots, 4$, matrices from the test set $\mathcal{S}_1$, tolerance set to 2^{-53}.

CPU time versus norm

Before agreeing on a value of ω, we want to assess the sensitivity of the routines with respect to the norm of the input matrix. In particular, we noticed that `phipm` is very sensitive to the norm of $\mathbf{A}$, i.e., `phipm` works far more efficiently when $\|\mathbf{A}\|$ is not large. There is a small subset $\mathcal{S}_1$ of $\mathcal{S}$, whose four matrices are all titled

```
'ABAQUS benchmark: pt.loaded fluid-filled spherical shell'
```

and we find that it exacerbates the sensitivity of `phipm` with respect to the norm of $\mathbf{A}$. In fact, this four matrices seem to be a formidable source of troubles for `phipm` when their norm grows. In Figure 4.3 we display the data obtained plotting how the CPU time (average over 5 launches) needed by each routine to compute $\mathrm{e}^{\mathbf{A}}v$ with $\mathbf{A} \in \mathcal{S}_1$, varies in function of the parameter ω.

From Figure 4.3 we can observe that while the CPU time needed by the polynomial methods increase linearly with the norm of the input matrix, the CPU time taken by `phipm` increases exponentially. Clearly, the matrices from $\mathcal{S}_1$ are pathological and `phipm` may not be in general so sensitive to the variations of the norm of $\mathbf{A}$. On the other hand, we cannot think to scale down every matrix in $\mathcal{S}$ using a too small value of ω. First of all, because it would be unfair toward those routines whose CPU time scales linearly with the norm of the input matrix. Second of all, exponential integrators are expressly meant for integration over time steps that are as large as possible. Running tests on a biased set made out of matrices with small norm would not be correct. In conclusion we decided to set $\omega = 9$.

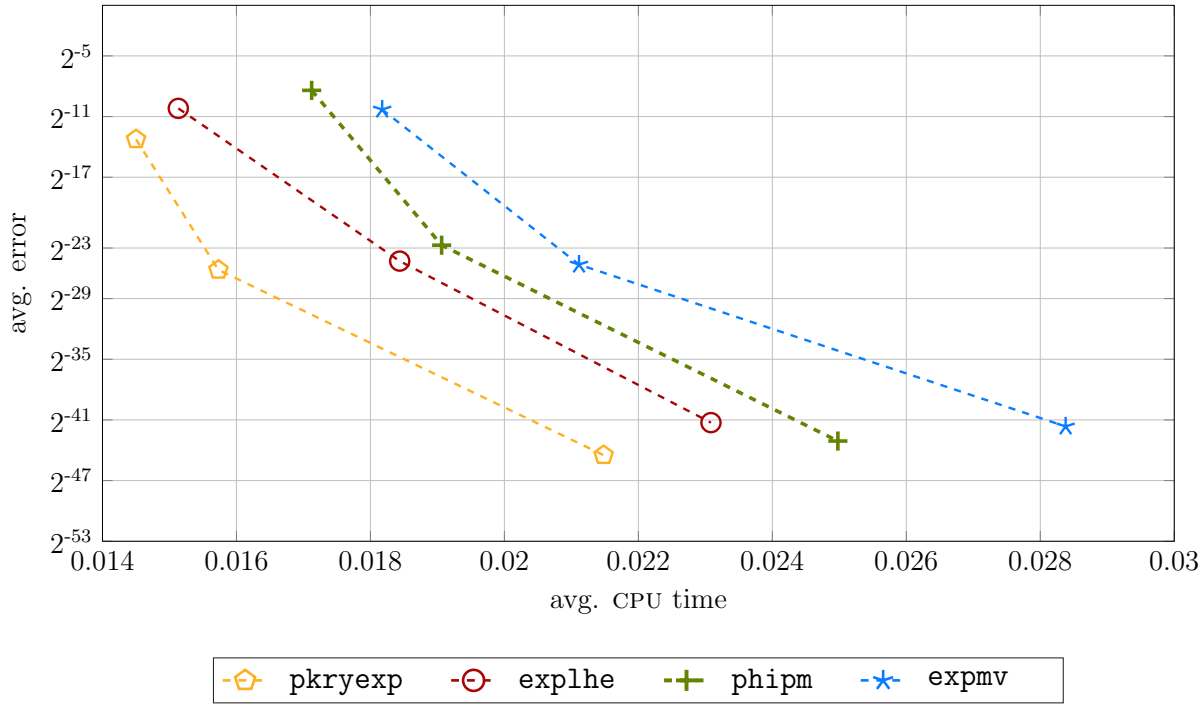


Figure 4.4: Average error (y-axis) against average CPU time (x-axis), matrices from the test set $\mathcal{S}_2$, tolerance set to 2^{-11}, 2^{-24} and 2^{-53}.

Precision diagram

In this experiment we run the four routines setting different prescribed tolerances. We are interested in observing how the CPU time (average over 5 launches), needed by each routine to compute $\mathrm{e}^{\mathbf{A}}v$, varies when the demanded precision increases.

We run the experiment over the set $\mathcal{S}_2$ of the matrices belonging to $\mathcal{S}$ that have size between 1000 and 5000. Then, we display the data we obtained by plotting the average CPU time taken by each routine to compute $\mathrm{e}^{\mathbf{A}}v$ for every $\mathbf{A}$ from $\mathcal{S}_2$ against the average relative forward error committed.

The reason for this choice of $\mathcal{S}_2$ is to sample a subset of $\mathcal{S}$ which is significant ($\mathcal{S}_2$ includes 471 matrices of the 1932 from $\mathcal{S}$) and whose matrices are not so small to not be meaningful when it comes to measure the performances of the routines. We chose not to exceed size 5000 so that the reference solutions could be computed in a reasonable time using the routine `expkpotf` from Chapter 3 with tolerance set to $\texttt{eps}^2 \approx 4.93\text{e-}32$. The results are collected in Figure 4.4.

From Figure 4.4. it appears evident that the fastest and most accurate routine over the considered subset of $\mathcal{S}$ is `pkryexp`. When double precision is required, differently from the half or single precision accuracy cases, no routine reaches on average the full 16 digits of accuracy. This is justified by the fact that the experiments are run in double precision arithmetic, therefore, the approximation error gets inevitably mixed with the rounding errors.

CPU time versus matrix size

In this experiment we set the tolerance to 2^{-24} and we measure how the CPU time (average over 5 launches), needed by each routine to compute $e^{\mathbf{A}}v$ with $\mathbf{A} \in \mathcal{S}$, varies in function of the size N of $\mathbf{A}$.

In Figure 4.5 we display the data we collected in a log-log plot. Clearly, we could not plot the CPU time taken by each routine to compute $e^{\mathbf{A}}v$ for each $\mathbf{A} \in \mathcal{S}$, for this would require to display 7728 (scattered) dots. To help readability, we regrouped the data obtained in 28 groups of 69 matrices with similar size each (1932 is the product of $23, 7, 3, 2$ and 2, so our options were limited). We then plotted the average CPU time that each routine needs to compute the action of the matrix exponential of all the matrices of each group.

From Figure 4.5 it appears evident that routines that are the fastest over the "small" matrices become the slowest on the large matrices. In particular, this is the case of `expmv`, whose performance dramatically drop as the size N grows larger than roughly 2000. This confirms the considerations we made at the beginning of this section on numerical experiments on $\mathcal{S}$.

The best performing routine, without any doubt, appears to be `pkryexp` which, roughly after size 2000, confirms itself as the fastest routine. It is worth notice that, when there is a marked discrepancy between the performance of `phipm` and the strictly polynomial methods `expmv` and `explhe`, the routine `pkryexp` usually performs as the fastest one. We believe that this is due to the dual nature of `pkryexp` which conjugates the best aspect of the two approaches while dodging the drawbacks.

The behavior of the curve representing the performance of `phipm` is quite unpredictable. We believe this is due to the lack of a tool for predicting the approximation parameters at the beginning of the computations. Such a tool, for the polynomial methods, is represented by the backward error analysis.

Performance profile

In running the last test over $\mathcal{S}$, we also collected information about the accuracy of the routines. We plot the data obtained as performance profile in Figure 4.6 that is such that a point (γ, ρ) on a curve related to a method represents the fraction of computed divided differences for which the corresponding error is bounded by ρ times the "unit" error, that we set to $2^{-25} \approx 2.98\text{e-}08$.

In order to measure the error we need a trusted reference. While for the tests over the set $\mathcal{S}_2$ such a reference could be obtained using the routine `expkpotf`, this is now impossible due to the prohibitive size of the matrices from $\mathcal{S}$. We decided to produce the trusted reference using the routine `pkryexp` with tolerance set to $\text{eps}^2 \approx 4.93\text{e-}32$. The reason for this is that, in Figure 4.4 `pkryexp`, proved to be the most accurate routine.

Figure 4.6 shows that the routine `explhe` is evidently the one with the most favorable curve, closely followed by `pkryexp`. This is because it does not matter the behavior of any of the curves as long as it is before $\rho = 4$. In fact we recall that we set the tolerance to 2^{-23} and the "unit" error to 2^{-25}. Therefore $\rho \cdot 2^{-25}$, with $\rho = 4$, equals exactly the tolerance 2^{-23}, therefore we are only interested in the behavior of the curves for $\rho > 4$.

It is remarkable that `explhe` stands out as the most accurate routine even though it is forced to interpolate the exponential function either on a real symmetric interval or a

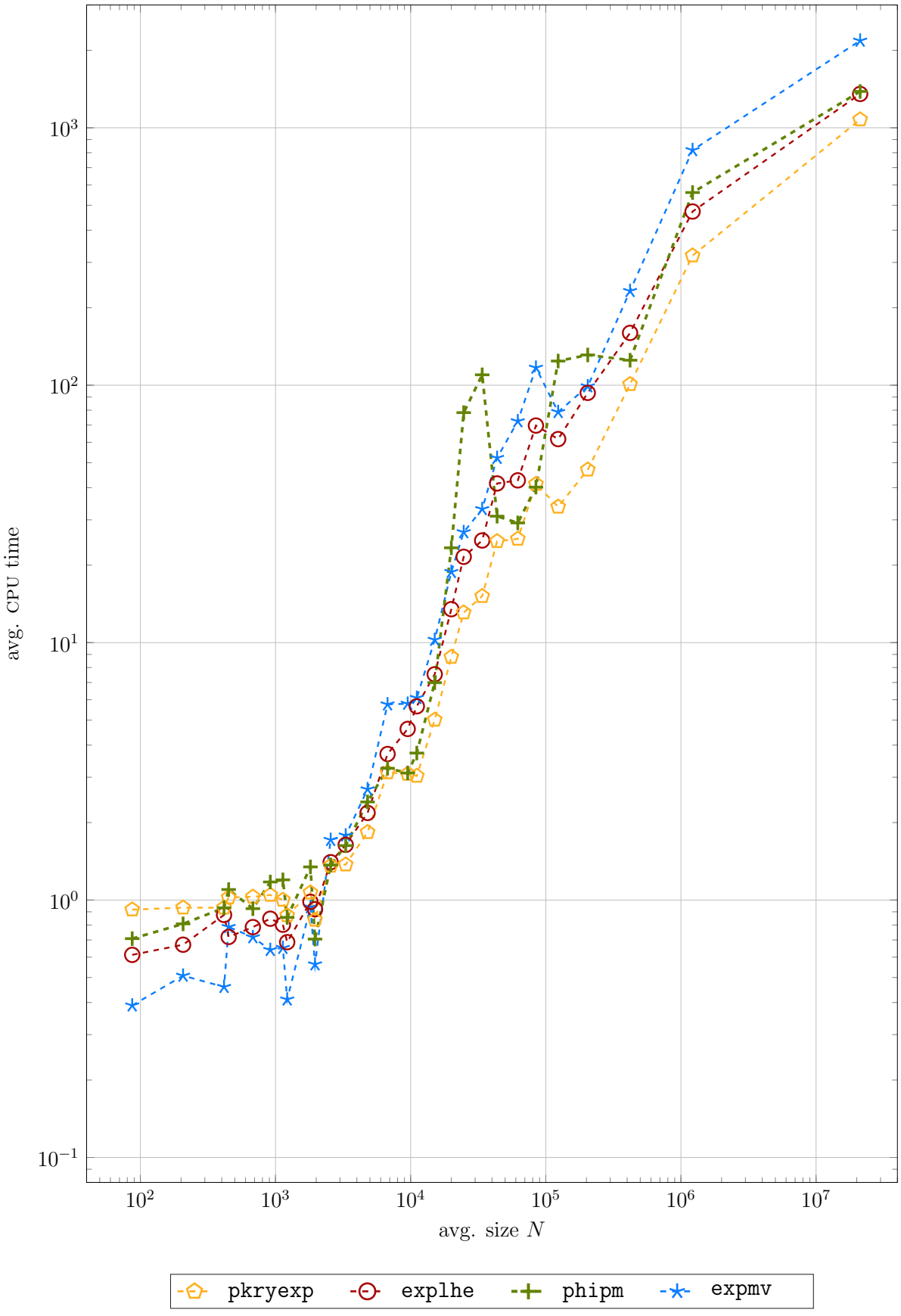


Figure 4.5: Average CPU time (y-axis) versus average matrix size N (x-axis), tolerance set to 2^{-24}.

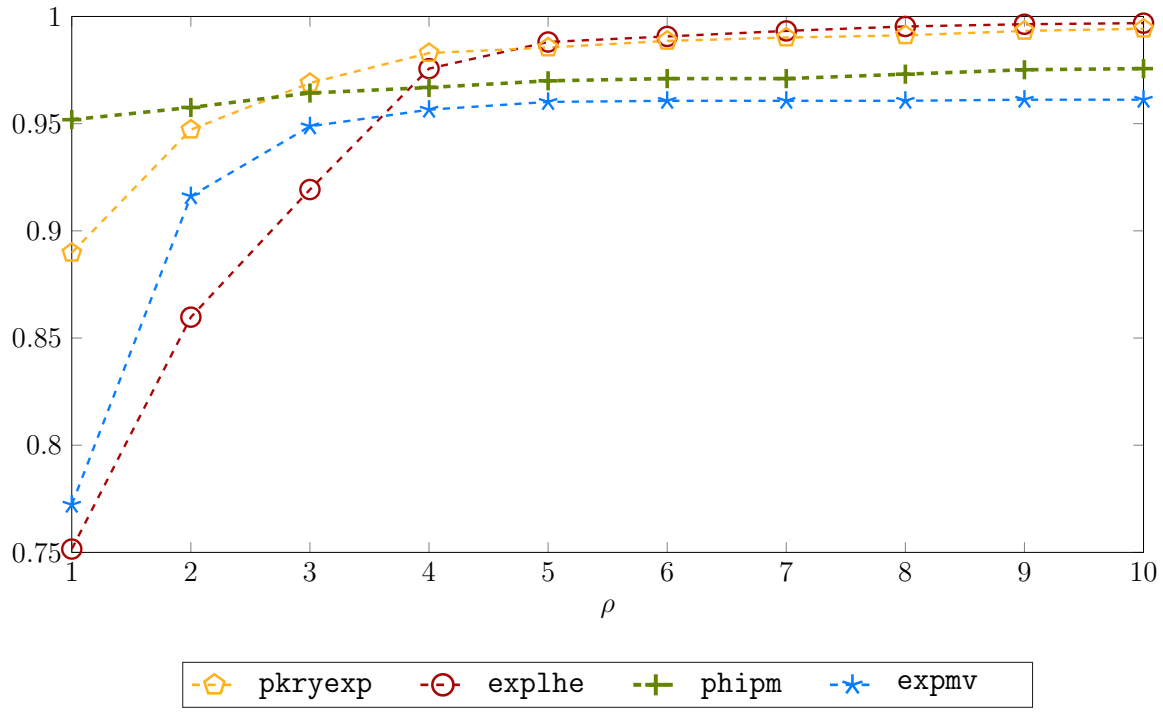


Figure 4.6: Performance profile over the test set $\mathcal{S}$, tolerance set to 2^{-24}.

purely imaginary one. Such a great accuracy must be due to the powerful contour integral approximation of the backward error, which allows for optimal choices of the interpolation parameters.

It is worth notice that, on the contrary, `phipm`, despite it can avail of an interpolation set which is tailored at every sub-step l on $\mathbf{A}$ and v, achieve average performances.

4.5 Conclusions

The first conclusion we can draw from the section on numerical experiments is that the contrast between the Krylov methods and the strictly polynomial methods should be no more the paradigm when it comes to polynomial methods. In fact, the inefficiencies of `expmv` and `phipm` clearly suggest that a rigorous analysis of the approximation error without a good set of interpolation points, and vice-versa, can only go so far.

The contour integral expansion technique proved to be an effective way to bound the backward error. In fact, the routine `explhe`, which was expressly meant for a specific class of matrices, can compete on equal terms with routines having a larger scope.

The interpolation at the extended Ritz's values proved to be a formidable solution to the problems of both the Krylov methods and the strictly polynomial methods. In fact, `pkryexp` mimics the interpolation features of the Krylov methods while it is not subject to its typical instabilities. We recall as an example when we had to discard 22 matrices from $\mathcal{S}$ because `phipm` could not converge to a solution or it was requiring more storage space than the 16Gb available on the machine used to perform the tests. Just few years ago, the typical storage capacity was far less. We can only imagine how many other matrices would have to be discarded if we were to run these tests over an older machine.

We believe these are exactly the problems of the Krylov methods that the authors of `expmv` were addressing when they first developed this routine, that, we recognize, is very reliable. On the other hand, the truncated Taylor series method underlying to the routine `expmv` is a little simplistic and this is reflected both in the accuracy and speed tests.

The next step will be to conjugate the contour integral expansion of the backward error of `explhe` and the interpolation at the extended Ritz's values of `pkryexp` into one routine. Our numerical experience suggests that the resulting routine should be able to easily outperform the four routines we presented in this chapter.

Chapter 5

Conclusions and future work

When I first got started with my research work, the field of numerical analysis dealing with the computation of the action of the matrix exponential was already quite crowded. Among others, I refer to the acclaimed survey paper of C. Moler and C. Van Loan [42], that collects a vast, although a bit dated, literature on this topic. Nevertheless, scientists and engineers tend to polarize on a few well-established methods, mainly polynomial, that are the methods I focused on.

Some scientists prefer the great effectiveness of the Krylov methods, which are usually quick in recovering a fair level of accuracy and that therefore greatly fit PDEs solvers (see for instance [27, 28, 46, 52]). The underlying Arnoldi process, however, is costly, it is prone to loss of orthogonality and it requires a possibly very large Krylov basis to store. Moreover, the exponentiation of the Hessenberg matrix, while it appears to be a simple task, it hides certain pitfalls, see for example Table 3.5 from Chapter 3.

Some others researchers, instead, value more the reliability and stability of polynomial methods which *directly* approximate the scalar exponential function. The performance of these methods are very predictable for they can be equipped with the backward error analysis tool (see [2, 4, 8, 6, 7, 12, 36, 44, 45, 62, 66]).

For this reason, the hump phenomenon, affecting direct methods and whose occurrence is instead unpredictable, is particularly hideous and needed to be addressed. So there I started, studying interpolation at the Leja–Hermite sets, which proved to greatly reduce the incidence of such a phenomenon. This branch of my investigation, through the manuscripts [8] and [10], led to the routine `explhe`, that I described in Chapter 4.

Another long-standing problem, affecting in particular direct methods in Newton form, is the computation of the divided differences. Although this problem was already elegantly and efficiently addressed by A.C. McCurdy in [38], I felt that I could give my input. The result is an innovative representation of the divided differences for analytic functions, which led to the routine `dd_phi` from [68], that I described in Chapter 2. This routine shown a ten-fold improvement in CPU time with respect to its fastest competitor, with a far more favorable performance profile.

In addition to this, the novel representation of the divided differences made possible to study, avoiding cancellation errors, certain polynomials linked to the truncation error. This innovation was crucial for developing, in [11], a tool for performing the backward error analysis in run-time. This tool allowed to consider interpolation sets such as the extended Ritz's values, which remain unknown until the moment of computing the action

of the matrix exponential. The result is the routine `pkryexp`, that I described in Chapter 3 of this thesis. On the one hand, this routine excellently mimics the features of the Krylov methods, being close-to-immune to hump phenomenon. On the other hand, `pkryexp` is not subject to the instabilities and vulnerabilities of the Krylov methods. Evidence for this can be found in the numerical experiments of Section 4.4.2, which shows the remarkable performance and accuracy of `pkryexp`.

In the meantime, a second branch of my investigation consisted in studying the problem of computing the matrix exponential *itself*. Although this problem is strongly interconnected with the problem of computing the action of the matrix exponential, the differences run deep. In particular, one of the main challenges that the numerical linear algebraists are facing nowadays is to convert the existing algorithms for computing matrix functions to arbitrary-precision. For this reason, in [9], I developed the on-the-fly backward error estimate, which turned out to be the trump card to the remarkable performance shown by the routine `expkptotf` from [60], described in Chapter 3. In fact, other than being the best performing and most accurate routine for the arbitrary precision arithmetic computation of the matrix exponential, `expkptotf` is also the only existing method able to successfully produce arbitrarily accurate approximations even when working with standard precision arithmetic (see for example Table 3.5). I refer to Section 3.4 for the numerical experiments supporting my claim.

As for the future, my plan is to produce new routines for computing (the action of) the matrix exponential which are meant to preserve certain quantities or geometric properties, a very important feature in certain fields of the numerical analysis. In addition to that, I noticed that certain routines for the computation of the action of the matrix exponential perform best when embedded into exponential integrator. This phenomenon is due to the optimization of parameters, such as the step size, that takes in account the efficiency of the whole system. This level of efficiency is something I would be thrilled to pursue, therefore, I would like this to be my very next research topic.

Bibliography

[1] A.H. Al-Mohy and N. J. Higham, *A new scaling and squaring algorithm for the matrix exponential*, SIAM J. Matrix Anal. Appl. 31 (3) (2009) 970–989, https://doi.org/10.1137/09074721X.

[2] A.H. Al-Mohy and N.J. Higham, *Computing the action of the matrix exponential with an application to exponential integrators*, SIAM J. Sci. Comput. 33 (2) (2011), 488–511, https://doi.org/10.1137/100788860.

[3] L.P. Bos and M. Caliari, *Application of modified Leja sequences to polynomial interpolation*, Dolomites Res. Notes Approx., 8 (2015), 66–74.

[4] L. Bergamaschi, M. Caliari and M. Vianello, *Efficient approximation of the exponential operator for discrete 2D advection–diffusion problems*, Numer. Linear Algebra Appl. 10 (3) (2003), 271–289.

[5] M. Caliari, *Accurate evaluation of divided differences for polynomial interpolation of exponential propagators*, Computing 80 (2) (2007), 189–201.

[6] M. Caliari, P. Kandolf, A. Ostermann and S. Rainer, *The Leja method revisited: backward error analysis for the matrix exponential*, SIAM J. Sci. Comput. 38 (3) (2016), A1639–A1661, https://doi.org/10.1137/15M1027620.

[7] M. Caliari, M. Vianello and L. Bergamaschi, *Interpolating discrete advection–diffusion propagators at Leja sequences*, J. Comput. Appl. Math. 172 (1) (2004), 79–99.

[8] M. Caliari, P. Kandolf and F. Zivcovich, *Backward error analysis of polynomial approximations for computing the action of the matrix exponential*, Bit Numer. Math. 58 (2018), 907, https://doi.org/10.1007/s10543-018-0718-9.

[9] M. Caliari and F. Zivcovich, *On–the–fly backward error estimate for matrix exponential approximation by Taylor algorithm*, J. Comput. Appl. Math. 346 (2019), 532–548, https://doi.org/10.1016/j.cam.2018.07.042.

[10] M. Caliari and F. Zivcovich, *Approximation of the matrix exponential for matrices with skinny fields of values*, submitted, (2019).

[11] M. Caliari and F. Zivcovich, *Extended Ritz interpolation for computing the action of the matrix exponential*, in preparation, (2020).

[12] V.L. Druskin and L.A. Knizhnerman, *Two polynomial methods of calculating functions of symmetric matrices*, USSR Comput. Math. Math. Phys. 29(6) (1989), 112–121.

[13] M. Eiermann and O. Ernst, *A restarted Krylov subspace method for the evaluation of matrix functions*, SIAM J. Numer. Anal., 44 (2006), 2481–2504.

[14] J. van den Eshof and M. Hochbruck, *Preconditioning Lanczos Approximations to the Matrix Exponential*, SIAM J. Sci. Comput., 27 (4) (2006), 1438–1457, https://doi.org/10.1137/040605461.

[15] T.M. Fischer, *On the algorithm by Al-Mohy and Higham for computing the action of the matrix exponential: A posteriori roundoff error estimation*, Linear Alg. Appl. 531 (2017), 141–168, https://doi.org/10.1016/j.laa.2017.05.042.

[16] M. Fasi, *Optimality of the Paterson–Stockmeyer method for evaluating matrix polynomials and rational matrix functions*, Linear Algebra App., 574 (2019), 182-200, https://doi.org/10.1016/j.laa.2019.04.001.

[17] A. Frommer, S. Güttel and M. Schweitzer, *Efficient and Stable Arnoldi Restarts for Matrix Functions Based on Quadrature*, SIAM J. Matrix Anal. Appl., 35 (2) (2014), 661–683, https://doi.org/10.1137/13093491X.

[18] M. Fasi and N.J. Higham, *An Arbitrary Precision Scaling and Squaring Algorithm for the Matrix Exponential*, SIAM J. Matrix Anal. Appl., 40 (4) (2019), 1233–1256, https://doi.org/10.1137/18M1228876.

[19] W. Gander, *Change of basis in polynomial interpolation*, Linear Algebra Appl., 12 (2005), 769–778.

[20] F.R. Gantmacher, *The Theory of Matrices*, Chelsea, New York, 1959.

[21] S. Gaudreault, G. Rainwater and M. Tokman, *KIOPS: A fast adaptive Krylov subspace solver for exponential integrators*, J. Comput. Phys., 372 (2018), 236-–255,

[22] S. Güttel, *Rational Krylov approximation of matrix functions: Numerical methods and optimal pole selection*, GAMM-Mitteilungen, 36 (2013), 8-31, https://doi.org/10.1002/gamm.201310002.

[23] N.J. Higham, *The Matrix Computation Toolbox*, http://www.ma.man.ac.uk/~higham/mctoolbox.

[24] N.J. Higham, *The scaling and squaring method for the matrix exponential revisited*, SIAM J. Matrix Anal. Appl. 26 (4) (2005) 1179–1193, https://doi.org/10.1137/04061101X.

[25] N.J. Higham, *Function of Matrices, Theory and Computation*, SIAM, Philadelphia, 2008.

[26] N. J. Higham, *Estimating the matrix p-norm*, Numer. Math., 62 (1992), 539–555.

[27] M. Hochbruck and C. Lubich, *On Krylov Subspace Approximations to the Matrix Exponential Operator*, SIAM J. on Numer. Anal. 34 (5) (1997), 1911–1925.

[28] M. Hochbruck, C. Lubich and H. Selhofer, *Exponential Integrators for Large Systems of Differential Equations*, SIAM J. Sci. Comput., 19 (5) (1998), 1552–1574.

[29] N.J. Higham and F. Tisseur, *A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra*, SIAM J. Matrix Anal. Appl. 21 (4) (2000), 1185–1201, https://doi.org/10.1137/S0895479899356080.

[30] M. Hochbruck and A. Ostermann, *Exponential integrators*, Acta Numerica, 19 (2010), 209-286, doi:10.1017/S0962492910000048.

[31] F. Leja, *Sur certaines suites liées aux ensembles plans et leur application à la reprèsentation conforme*, Ann. Polon. Math., 4 (1957), 8–13.

[32] L. Li and E. Celledoni, *Krylov projection method for linear Hamiltonian systems*, Numer Algor 81 (2019), 1361, https://doi.org/10.1007/s11075-018-00649-8.

[33] M. López–Fernández and S.A. Sauter, *Fast and Stable Contour Integration for High Order Divided Differences via Elliptic Functions*, Math. Comput., 84 (2015), 1291–1315.

[34] *Maple*, Waterloo Maple Inc., Waterloo, Ontario, Canada, http://www.maplesoft.com.

[35] *Mathematica*, Wolfram Research, Inc., Champaign, IL, USA, http://www.wolfram.com.

[36] I. Moret and P .Novati, *An interpolatory approximation of the matrix exponential based on Faber polynomials*, J. Comput. Appl. Math. 131 (2001), 361–380.

[37] H. Mena, A. Ostermann, M.L. Pfurtscheller and C. Piazzola, *Numerical low-rank approximation of matrix differential equations*, J. Comput. Appl. Math. 340 (2018), 602-614, https://doi.org/10.1016/j.cam.2018.01.035.

[38] A.C. McCurdy, K.C. Ng and B.N. Parlett, *Accurate computation of divided differences of the exponential function*, University of California — Berkeley, CPAM-160 (1983).

[39] *Multiprecision Computing Toolbox*, Advanpix, Tokyo, https://www.advanpix.com.

[40] R.I. Mclachlan and G.R.W. Quispel, *Splitting Methods*, Acta Numerica 2002, Acta Numerica, (2002), 341-434, https://doi.org/10.1017/CBO9780511550140.005.

[41] I. Moret and P. Novati, *RD-Rational Approximations of the matrix exponential*, Bit Numer Math 44 (2004), 595-615.

[42] C.B. Moler and C.F. Van Loan, *Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later*, SIAM Rev. 45 (1) (2003), 3–49.

[43] I. Moret and M. Popolizio, *The restarted shift-and-invert Krylov method for matrix functions*, Numer. Linear Algebra Appl., 21 (2014), 68-80, https://doi.org/10.1002/nla.1862.

[44] P. Novati, *A polynomial method based on Fejér points for the computation of functions of unsymmetric matrices*, Appl. Numer. Math 44 (2003), 201–224, https://doi.org/10.1002/nla.1862.

[45] I. Novati and P. Moret, *The computation of functions of matrices by truncated Faber series*, Numer. Funct. Anal. Opt. 22 (2001), 5–6, 697-719.

[46] J. Niesen and W.M. Wright, *A Krylov Subspace Algorithm for Evaluating the phi-Functions appearing in Exponential Integrators*, ACM Trans. Math. Software, 38(3) (2012), Article 22, https://doi:10.1145/2168773.2168781.

[47] G. Opitz, *Steigungsmatrizen*, Z. Angew. Math. Mech. 44 (1964),T52–T54.

[48] B.N. Parlett, *A recurrence among the Elements of Functions of Triangular Matrices*, Linear Algebra Appl., 14 (1976), 117–121.

[49] M. Popolizio and V. Simoncini, *Acceleration Techniques for Approximating the Matrix Exponential Operator*, SIAM J. Matrix Anal. Appl., 30(2) (2008), 657–683, https://doi.org/10.1137/060672856.

[50] L. Reichel, *Newton interpolation at Leja points*, L. BIT 30 (2) (1990), 332–346 https://doi.org/10.1007/BF02017352.

[51] P. Ruiz, J. Sastre, J. Ibáñez and E. Defez, *High performance computing of the matrix exponential*, J. Comput. Appl. Math. 291 (1) (2016), 370–379, https://doi.org/10.1016/j.cam.2015.04.001.

[52] Y. Saad, *Analysis of some Krylov subspace approximations to the matrix exponential operator*, SIAM. 29 (1) (1992), 209-228.

[53] Y. Saad, *Iterative methods for sparse linear systems*, SIAM, Philadelphia, 2003.

[54] *SageMath*, Sage Mathematical Software System, https://www.sagemath.org.

[55] R.B. Sidje, *Expokit: A software package for computing matrix exponentials*, ACM Transactions on Mathematical Software, (TOMS) 24 (1) (1998) 130–156.

[56] J. Sastre, *Efficient evaluation of matrix polynomials*, Linear Algebra App., 539 (2018), 229-250, https://doi.org/10.1016/j.laa.2017.11.010.

[57] J. Sastre, J. Ibáñez and E. Defez, *Boosting the computation of the matrix exponential*, Appl. Math. Comp., 34 (2019), 206–220, https://doi:10.1016/j.amc.2018.08.017.

[58] J. Sastre, J. Ibáñez, E. Defez and P. Ruiz, *Accurate matrix exponential computation to solve coupled differential models in engineering*, Math. Comput. Model. 54 (2011), 1835–1840, https://doi.org/10.1016/j.mcm.2010.12.049.

[59] J. Sastre, J. Ibáñez, E. Defez and P. Ruiz, *New scaling-squaring Taylor algorithms for computing the matrix exponential*, SIAM J. Sci. Comput. 37(1) (2015), A439–A455, https://doi.org/10.1137/090763202.

[60] J. Sastre, J. Ibáñez, E. Defez and F. Zivcovich, *On–the–fly backward error Krylov projection for arbitrary precision high performance Taylor approximation of the matrix exponential*, in preparation (2020).

[61] J. Sastre, J. Ibáñez, P. Ruiz and E. Defez, *Accurate and efficient matrix exponential computation*, Int. J. Comp. Math. 91 (1) (2014), 97–112, https://doi.org/10.1080/00207160.2013.791392.

[62] M.J. Schaefer, *A polynomial based iterative method for linear parabolic equations*, J. Comput. Appl. Math. 29 (1990), 35–50.

[63] M. Schreiber, P.S. Peixoto, T. Haut and B. Wingate, *Beyond spatial scalability limitations with a massively parallel method for linear oscillatory problems*, Int. J. High Perform. Comput. Appl., 32(6) (2018), 913–933, https://doi.org/10.1177/1094342016687625.

[64] *Symbolic Math Toolbox*, The MathWorks, Inc., Natick, MA, USA, https://www.mathworks.com/products/symbolic.

[65] *SuiteSparse Matrix Collection,*, https://https://sparse.tamu.edu/.

[66] H. Tal-Ezer, *High degree polynomial interpolation in Newton form*, SIAM J. Sci. Stat. Comput. 12 (3) (1991), 648–667.

[67] L.N. Trefethen, J.A.C. Weideman and T. Schmelzer, *Talbot quadratures and rational approximations*, BIT. Numer. Math. 46 (2006), 653–670.

[68] F. Zivcovich, *Fast and accurate computation of divided differences for analytic functions, with an application to the exponential function*, Dolomites Res. Notes Approx, 12 (2019), 28–42.