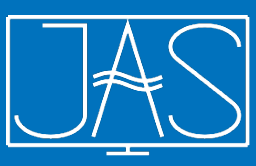
Volume 1, Issue 1

Journal of Approximation Software



23 July 2024

SIRIO@UniTO

Journal of Approximation Software

Volume 1, Issue 1
23 July 2024

Preface

We have the pleasure to present the first issue of the Journal of Approximation Software. It publishes in open-access mode well-structured accompanying articles to open-source software, on all aspects of approximation theory and applications in its broadest sense.

The publisher is University of Torino (Italy) by its open-access platform SIRIO@UniTO.

Implementations can use the most common languages and environments like MATLAB, Python, C, C++, among others. The codes are posted by the authors on stable public platforms with a clear software versioning policy, such as GitHub.

There will be one general issue per year (with articles published continuously by acceptance date) and possible thematic special issues including conference proceedings.

This first issue collects a few regular articles of colleagues and researchers, who work in different fields of approximation implementing numerical algorithms by means of the MATLAB software. The issue includes a variety of approximation topics involving exponential integrators, ordinary differential equations, quadrature of oscillating functions, computation of derivatives and Quasi-MonteCarlo integration.

We sincerely thank the authors of the papers and the anonymous referees helping us to carefully review the papers included in this first issue of the first volume of JAS.

The Editors-in-Chief of JAS

J. Approx. Softw., Vol. 1, No. 1 (2024) 1

Journal of Approximation Software

Direction splitting of φ-functions in exponential integrators for d-dimensional problems in Kronecker form

M. Caliari[1,⋆] and F. Cassini[1]

[1] *Department of Computer Science, University of Verona, Italy*

Received: 05/04/2023 – Published: 23/07/2024
Communicated by: R. Cavoretto and A. De Rossi

Abstract

In this manuscript, we propose an efficient, practical and easy-to-implement way to approximate actions of φ-functions for matrices with d-dimensional Kronecker sum structure in the context of exponential integrators up to second order. The method is based on a direction splitting of the involved matrix functions, which lets us exploit the highly efficient level 3 BLAS for the actual computation of the required actions in a μ-mode fashion. The approach has been successfully tested on two- and three-dimensional problems with various exponential integrators, resulting in a consistent speedup with respect to a technique designed to approximate actions of φ-functions for Kronecker sums.

Keywords: exponential integrators, μ-mode, φ-functions, direction splitting, Kronecker form (MSC2020: 65F60, 65L04, 65M20)

1 Introduction

The problem of computing actions of exponential and exponential-like functions with Kronecker sum structure received a lot of attention in the last years [6, 8, 10, 14, 21, 22, 25]. Indeed, the efficient approximation of such quantities allows to effectively employ exponential integrators for the time integration of large stiff systems of Ordinary Differential Equations (ODEs). More in detail, we suppose to work with the following system of ODEs

$$\begin{cases} \boldsymbol{u}'(t) = K\boldsymbol{u}(t) + \boldsymbol{g}(t,\boldsymbol{u}(t)), & t > 0, \\ \boldsymbol{u}(0) = \boldsymbol{u}_0. \end{cases} \tag{1a}$$

⋆ Corresponding author: marco.caliari@univr.it

The stiff part is represented by the matrix $K \in \mathbb{C}^{N\times N}$ which has d-dimensional Kronecker sum structure, i.e.,

$$K = A_d \oplus A_{d-1} \oplus \cdots \oplus A_1 = \sum_{\mu=1}^{d} A_{\otimes\mu}, \quad A_{\otimes\mu} = I_d \otimes \cdots \otimes I_{\mu+1} \otimes A_\mu \otimes I_{\mu-1} \otimes \cdots \otimes I_1. \quad (1b)$$

Here, $A_\mu \in \mathbb{C}^{n_\mu \times n_\mu}$ and I_μ is the identity matrix of size n_μ. Moreover, $\boldsymbol{g}(t,\boldsymbol{u}(t))$ is a generic nonlinear function of t and of the unknown $\boldsymbol{u}(t) \in \mathbb{C}^N$, with $N = n_1 \cdots n_d$. Throughout the paper the symbol $\otimes$ denotes the standard Kronecker product of matrices, while $\oplus$ is employed for the Kronecker sum of matrices. Finally, we refer to system (1) as a system in *Kronecker form* or with *Kronecker sum structure*.

This kind of systems naturally arises in many contexts. For example, if $d = 2$, such a structure appears in constant coefficient matrix Riccati differential equations (see, for instance, Reference [1, Ch. 3])

$$\begin{cases} \boldsymbol{U}'(t) = A_1\boldsymbol{U}(t) + \boldsymbol{U}(t)A_2^{\mathsf{T}} + C + \boldsymbol{U}(t)B\boldsymbol{U}(t), \\ \boldsymbol{U}(0) = \boldsymbol{U}_0, \end{cases} \quad (2)$$

where $\boldsymbol{U}(t) \in \mathbb{C}^{n_1\times n_2}$, $B \in \mathbb{C}^{n_2\times n_1}$, and $C \in \mathbb{C}^{n_1\times n_2}$. Indeed, using the properties of the Kronecker product [31], we can rewrite equivalently such a matrix equation as a system of ODEs in Kronecker form (1), i.e.,

$$\begin{cases} \boldsymbol{u}'(t) = ((I_2 \otimes A_1) + (A_2 \otimes I_1))\boldsymbol{u}(t) + \operatorname{vec}(C + \boldsymbol{U}(t)B\boldsymbol{U}(t)), \\ \boldsymbol{u}(0) = \operatorname{vec}(\boldsymbol{U}_0), \end{cases} \quad (3)$$

where vec is the operator which stacks the columns of the input matrix in a single vector.

Systems with Kronecker sum structure often arise also when applying the method of lines to approximate numerically the solution of a Partial Differential Equation (PDE) defined on a tensor product domain and appropriate boundary conditions. Indeed, after semidiscretization in space of well-known parabolic equations such as Allen–Cahn, Brusselator, Gray–Scott, advection–diffusion–reaction [8, 10] or Schrödinger equations [6], we obtain a large stiff system of ODEs in form (1).

Once system (1) is given, many techniques can be employed to numerically integrate it in time, and in particular we are interested in the application of exponential integrators [19]. In fact, they are a prominent way to perform the required task since they enjoy favorable stability properties that make them suitable to work in the stiff regime. These kinds of schemes require the computation of the action of the matrix exponential and of exponential-like matrix functions (the so-called φ-functions) on vectors. They are defined, for a generic matrix $X \in \mathbb{C}^{N\times N}$, as

$$\varphi_0(X) = \mathrm{e}^X, \quad \varphi_\ell(X) = \int_0^1 \frac{\theta^{\ell-1}}{(\ell-1)!}\mathrm{e}^{(1-\theta)X}d\theta, \quad \ell > 0, \quad (4a)$$

and their Taylor series expansion is given by

$$\varphi_\ell(X) = \sum_{i=0}^{\infty} \frac{X^i}{(i+\ell)!}, \quad \ell \geq 0. \quad (4b)$$

When the size of X allows, it is common in practice to approximate such matrix functions by means of diagonal Padé approximations [2, 5, 30] or via polynomial approximations [12, 20,

29]. On the other hand, when X is large sized, this approach is computationally unfeasible, and many algorithms have been developed to perform directly the action of φ-functions on vectors. We mention, among the others, Krylov-based techniques [16, 23, 26], direct polynomial methods [3, 7, 11, 20], and hybrid techniques [9]. When X is in fact a matrix K with Kronecker sum structure (1b), it is possible to exploit this information to compute more efficiently the action of the φ-functions on a vector. Indeed, let us consider $\ell = 0$, so that $\varphi_0(K) = \mathrm{e}^K$. Then, it is easy to see [10] that computing

$$\boldsymbol{e} = \mathrm{e}^K \boldsymbol{v} = \mathrm{e}^{A_d \oplus A_{d-1} \oplus \cdots \oplus A_1} \boldsymbol{v} = \left(\mathrm{e}^{A_d} \otimes \mathrm{e}^{A_{d-1}} \otimes \cdots \otimes \mathrm{e}^{A_1} \right) \boldsymbol{v} \tag{5}$$

is mathematically equivalent to the *tensor formulation*

$$\boldsymbol{E} = \boldsymbol{V} \times_1 \mathrm{e}^{A_1} \times_2 \cdots \times_d \mathrm{e}^{A_d}. \tag{6}$$

Here, $\boldsymbol{E}$ and $\boldsymbol{V}$ are order-d tensors of size $n_1 \times \cdots \times n_d$ that satisfy $\mathrm{vec}(\boldsymbol{E}) = \boldsymbol{e}$ and $\mathrm{vec}(\boldsymbol{V}) = \boldsymbol{v}$, respectively, while vec is the operator which stacks the columns of the input tensor into a suitable single column vector. The symbol $\times_\mu$ denotes the tensor–matrix product along the mode μ, which is also known as μ-mode product, and the computation of consecutive μ-mode products (as it happens in formula (6)) is usually referred to as *Tucker operator*. Notice that the element $e_{i_1 \ldots i_d}$ of the tensor $\boldsymbol{E}$ turns out to be

$$e_{i_1 \ldots i_d} = \sum_{j_d=1}^{n_d} \cdots \sum_{j_1=1}^{n_1} v_{j_1 \ldots j_d} \prod_{\mu=1}^{d} e^{\mu}_{i_\mu j_\mu}, \quad 1 \le i_\mu \le n_\mu, \tag{7}$$

being $e^{\mu}_{i_\mu j_\mu}$ the generic element of e^{A_μ}. Although formulas (5), (6), and (7) are mathematically equivalent, the direct usage of both formulas (5) and (7) is much less efficient than formula (6), which is implemented by exploiting the highly performance level 3 BLAS after computing the *small* sized matrix exponentials e^{A_μ}. Indeed, for instance, formula (6) in two dimensions requires two matrix-matrix products, as it reduces to $\mathrm{e}^{A_1} \boldsymbol{V} \left(\mathrm{e}^{A_2}\right)^{\mathsf{T}}$, while in the d-dimensional case it requires d level 3 BLAS calls. This technique led to the so-called μ-mode integrator [6], and has been successfully used to integrate in time semidiscretizations of advection–diffusion–reaction and Schrödinger equations, eventually in combination with a splitting scheme. In particular, it is reported a consistent speedup with respect to state-of-the-art techniques to compute the action of the matrix exponential on a vector, as well as a very good scaling when performing GPUs simulations. We invite a reader interested in more details and applications of the Tucker operator to check References [6, 10].

When computing actions of φ-function of higher order, i.e., $\varphi_\ell(K)\boldsymbol{v}$ with $\ell > 0$, the last equality in formula (5) does not hold anymore. In Reference [8] the authors propose an approach to overcome this difficulty, by developing a method based on the application of a quadrature formula to the integral definition of the φ-functions (4a). In fact, it requires an action of the matrix exponential for each quadrature point, which is performed by a Tucker operator. In this way, it is possible to compute the required action of φ-functions at a given tolerance. The technique, which has been named PHIKS, has been developed for arbitrary dimension d and is designed to compute not only φ-functions applied to a vector but also linear combinations of actions of φ-functions. In addition the desired quantities can be made available simultaneously at suitable different time scales. These features allow to implement high stiff order exponential integrators, such as exponential Runge–Kutta schemes, in a more efficient way compared to the usage of state-of-the-art techniques to compute combinations of actions of φ-functions. Another very recent method based on quadrature rules applied to formula (4a) is presented

in Reference [14], where the technique is described only in dimension $d \leq 3$ for actions of single φ-functions at a given time scale. Other approaches for the action of φ-functions for matrices with Kronecker sum structure are available in the literature. We mention for instance Reference [25], whose algorithm is based on the solution of Sylvester equations and is currently limited to dimension $d = 2$. Another way to approximate the action of φ-functions of the Sylvester operator $A_1\boldsymbol{V} + \boldsymbol{V}A_2^{\mathsf{T}}$ or the Lyapunov operator $A\boldsymbol{V} + \boldsymbol{V}A^{\mathsf{T}}$ for the solution of Riccati differential equations, possibly in the context of low-rank approximation, is presented in References [21, 22].

In this manuscript we propose an alternative way to approximate $\varphi_\ell(K)\boldsymbol{v}$, with $\ell > 0$ and K a matrix with d-dimensional Kronecker sum structure, in the context of exponential integrators up to second order. The approach, that we call PHISPLIT, is based on a direction splitting of the matrix φ-functions of K, which generates an approximation error compatible with the one of the time marching numerical scheme. The evaluation of the required actions is performed in a μ-mode fashion by means of a *single* Tucker operator for each φ-function, exploiting the highly efficient level 3 BLAS. After recalling some popular exponential integrators in Section 2, we describe in Section 3 the proposed technique, as well as how to employ it to implement the just mentioned exponential schemes. Then, in Section 4 we present some numerical experiments that show the effectiveness of PHISPLIT, and we finally draw some conclusions in Section 5.

2 Recall of some exponential integrators up to order two

When numerically integrating stiff semilinear ODEs in form (1), a prominent approach is to use exponential integrators [19]. For convenience of the reader, we report here (for simplicity in a constant time step size scenario) a possible derivation of the exponential schemes that will be employed later in the numerical experiments of Section 4.

The starting point is the variation-of-constants formula

$$\begin{aligned} \boldsymbol{u}(t_{n+1}) &= \mathrm{e}^{\tau K}\boldsymbol{u}(t_n) + \int_{t_n}^{t_{n+1}} \mathrm{e}^{(t_{n+1}-s)K}\boldsymbol{g}(s,\boldsymbol{u}(s))ds \\ &= \mathrm{e}^{\tau K}\boldsymbol{u}(t_n) + \tau\int_0^1 \mathrm{e}^{(1-\theta)\tau K}\boldsymbol{g}(t_n+\tau\theta,\boldsymbol{u}(t_n+\tau\theta))d\theta \end{aligned} \tag{8}$$

which expresses the analytical solution of system (1a) at time $t_{n+1} = t_n + \tau$, where τ is the time step size. If we approximate the integral with the rectangle left rule, we get the scheme

$$\boldsymbol{u}_{n+1} = \mathrm{e}^{\tau K}\boldsymbol{u}_n + \tau\mathrm{e}^{\tau K}\boldsymbol{g}(t_n,\boldsymbol{u}_n) = \mathrm{e}^{\tau K}(\boldsymbol{u}_n + \tau\boldsymbol{g}(t_n,\boldsymbol{u}_n)), \tag{9}$$

which is known as Lawson–Euler scheme (see Reference [4, Sec. A.1.1]). It is of order one, exact if $\boldsymbol{g}(t,\boldsymbol{u}(t))$ is null. The linear part of system (1a) is solved exactly and thus no restriction on the time step size due to the stiffness is necessary. Instead, if the trapezoidal quadrature rule is applied to the integral in equation (8), we get the approximation

$$\boldsymbol{u}(t_{n+1}) \approx \mathrm{e}^{\tau K}\boldsymbol{u}(t_n) + \frac{\tau}{2}\left(\mathrm{e}^{\tau K}\boldsymbol{g}(t_n,\boldsymbol{u}(t_n)) + \boldsymbol{g}(t_{n+1},\boldsymbol{u}(t_{n+1}))\right).$$

An explicit time marching scheme is then obtained by creating an intermediate stage $\boldsymbol{u}_{n2}$ which approximates $\boldsymbol{u}(t_{n+1})$ in the right hand side by the Lawson–Euler scheme (9). Overall, we get

$$\begin{aligned} \boldsymbol{u}_{n2} &= \mathrm{e}^{\tau K}(\boldsymbol{u}_n + \tau\boldsymbol{g}(t_n,\boldsymbol{u}_n)), \\ \boldsymbol{u}_{n+1} &= \mathrm{e}^{\tau K}\left(\boldsymbol{u}_n + \frac{\tau}{2}\boldsymbol{g}(t_n,\boldsymbol{u}_n)\right) + \frac{\tau}{2}\boldsymbol{g}(t_{n+1},\boldsymbol{u}_{n2}), \end{aligned} \tag{10}$$

which is a Lawson method of order two, also known in literature as Lawson2b (see Reference [4, Sec. A.1.6]).

A different approach to the approximation of the integral in formula (8) leads to the so-called exponential Runge–Kutta methods. Indeed, if we approximate only the nonlinear function $\boldsymbol{g}(t_n+\tau\theta,\boldsymbol{u}(t_n+\tau\theta))$ by $\boldsymbol{g}(t_n,\boldsymbol{u}(t_n))$, by using the definition of φ_1 function in equation (4a) we get the scheme

$$\boldsymbol{u}_{n+1} = \mathrm{e}^{\tau K}\boldsymbol{u}_n + \tau\varphi_1(\tau K)\boldsymbol{g}(t_n,\boldsymbol{u}_n),$$

which can be equivalently rewritten as

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \tau\varphi_1(\tau K)(K\boldsymbol{u}_n + \boldsymbol{g}(t_n,\boldsymbol{u}_n)) \tag{11}$$

and is known as exponential Euler (or Nørsett–Euler, see Reference [4, Sec. A.2.1]). It is a first order scheme, exact if $\boldsymbol{g}(t,\boldsymbol{u}(t))$ is constant. Another possibility is to interpolate $\boldsymbol{g}(t_n+\tau\theta,\boldsymbol{u}(t_n+\tau\theta))$ with a polynomial of degree one in θ at 0 and 1, thus obtaining the approximation

$$\boldsymbol{u}(t_{n+1}) \approx \mathrm{e}^{\tau K}\boldsymbol{u}(t_n) + \tau\int_0^1 \mathrm{e}^{(1-\theta)\tau K}(\theta\boldsymbol{g}(t_{n+1},\boldsymbol{u}(t_{n+1})) + (1-\theta)\boldsymbol{g}(t_n,\boldsymbol{u}(t_n)))d\theta.$$

By taking a stage $\boldsymbol{u}_{n2}$ which approximates $\boldsymbol{u}(t_{n+1})$ in the right hand side by the exponential Euler scheme, and using the definitions of φ_1 and φ_2 functions in formula (4a), we obtain the second order exponential Runge–Kutta scheme (also known in literature as ETD2RK, see Reference [4, Sec. A.2.5])

$$\begin{aligned}\boldsymbol{u}_{n2} &= \boldsymbol{u}_n + \tau\varphi_1(\tau K)(K\boldsymbol{u}_n + \boldsymbol{g}(t_n,\boldsymbol{u}_n)),\\ \boldsymbol{u}_{n+1} &= \boldsymbol{u}_{n2} + \tau\varphi_2(\tau K)(\boldsymbol{g}(t_{n+1},\boldsymbol{u}_{n2}) - \boldsymbol{g}(t_n,\boldsymbol{u}_n)).\end{aligned} \tag{12}$$

Finally, we consider the Rosenbrock–Euler method (see Reference [19, Ex. 2.20]) which, in the autonomous case, can be obtained from the application of the exponential Euler scheme to the linearized differential equation

$$\boldsymbol{u}'(t) = \left(\underbrace{K + \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}}(\boldsymbol{u}_n)}_{K_n}\right)\boldsymbol{u}(t) + \left(\boldsymbol{g}(\boldsymbol{u}(t)) - \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}}(\boldsymbol{u}_n)\boldsymbol{u}(t)\right),$$

where K_n is the Jacobian evaluated at $\boldsymbol{u}_n$. The resulting scheme is

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \tau\varphi_1(\tau K_n)(K\boldsymbol{u}_n + \boldsymbol{g}(\boldsymbol{u}_n)). \tag{13}$$

It is a second order method and, in contrast to all the methods presented above, it requires the evaluation of a different matrix function $\varphi_1(\tau K_n)$ at *each time step*. The extension to non-autonomous systems is straightforward, see Reference [19, Ex. 2.21].

Remark 2.1. We considered here only a selected number of exponential integrators which require the action of φ-functions. Other exponential-type schemes of first or second order could benefit from the μ-mode splitting technique for computing φ-functions of Kronecker sums that we present in this work. We mention, among the others, exponential multistep schemes [13], corrected splitting schemes [15], low-regularity schemes [28], and Magnus integrators for linear time dependent coefficient non-homogeneous equations [17].

3 Direction splitting of φ-functions

As mentioned in the introduction, we suppose that we are dealing with a matrix K with Kronecker sum structure (1b), and we are interested in approximating efficiently $\varphi_\ell(\tau K)\boldsymbol{v}$, $\boldsymbol{v} \in \mathbb{C}^N$, $\tau \in \mathbb{R}$, in the context of exponential integrators. In particular, we know that by employing a scheme of order p, we make a local error $\mathscr{O}(\tau^{p+1})$, being τ the (constant) time step size. Hence, if the integrator requires to compute a quantity of the form $\tau^q \varphi_\ell(\tau K)$, with $q > 0$, it is sufficient to approximate $\varphi_\ell(\tau K)$ with an error $\mathscr{O}(\tau^{p+1-q})$ to preserve the order of convergence. For our schemes of interest, i.e., the ones presented in the previous section, we make use of the following result.

Theorem 3.1. *Let K be a matrix with d-dimensional Kronecker sum structure (1b). Then, for $\ell > 0$, we have*

$$\varphi_\ell(\tau K) = (\ell!)^{d-1} \left(\varphi_\ell(\tau A_d) \otimes \varphi_\ell(\tau A_{d-1}) \otimes \cdots \otimes \varphi_\ell(\tau A_1)\right) + \mathscr{O}(\tau^2). \tag{14}$$

Proof. For compactness of presentation, we employ the following notation

$$X_d \otimes X_{d-1} \otimes \cdots \otimes X_1 = \bigotimes_{\mu=d}^{1} X_\mu, \quad X_\mu \in \mathbb{C}^{n_\mu \times n_\mu}.$$

Then, by using the Taylor expansion of the φ_ℓ function (4b) and the properties of the Kronecker product (see Reference [31] for a comprehensive review) we obtain

$$\begin{aligned}
(\ell!)^{d-1} \bigotimes_{\mu=d}^{1} \varphi_\ell(\tau A_\mu) &= (\ell!)^{d-1} \bigotimes_{\mu=d}^{1} \left(\frac{I_\mu}{\ell!} + \frac{\tau A_\mu}{(\ell+1)!} + \mathscr{O}(\tau^2)\right) \\
&= (\ell!)^{d-1} \left(\frac{1}{(\ell!)^d} \bigotimes_{\mu=d}^{1} I_\mu + \frac{\tau}{(\ell!)^{d-1}(\ell+1)!} \sum_{\mu=1}^{d} A_{\otimes\mu} + \mathscr{O}(\tau^2)\right) \\
&= \frac{I}{\ell!} + \frac{\tau K}{(\ell+1)!} + \mathscr{O}(\tau^2) \\
&= \varphi_\ell(\tau K) + \mathscr{O}(\tau^2),
\end{aligned}$$

where I is the identity matrix of size $N \times N$. □

Formula (14) allows for an efficient μ-mode based implementation, similarly to the matrix exponential case (6). Indeed, given an order-d tensor $\boldsymbol{V}$, if we define the tensor formulation

$$\boldsymbol{P}_\ell^{(2)}(\boldsymbol{V}) = \left((\ell!)^{d-1}\boldsymbol{V}\right) \times_1 \varphi_\ell(\tau A_1) \times_2 \varphi_\ell(\tau A_2) \times_3 \cdots \times_d \varphi_\ell(\tau A_d), \tag{15}$$

we have

$$\varphi_\ell(\tau K)\boldsymbol{v} = \boldsymbol{p}_\ell^{(2)}(\boldsymbol{v}) + \mathscr{O}(\tau^2),$$

where $\boldsymbol{v} = \mathrm{vec}(\boldsymbol{V})$ and $\boldsymbol{p}_\ell^{(2)}(\boldsymbol{v}) = \mathrm{vec}(\boldsymbol{P}_\ell^{(2)}(\boldsymbol{V}))$.

This is precisely the formulation that we propose to employ in the above exponential integrators when actions of φ-functions of a matrix with Kronecker sum structure are required. From now on, we refer to this technique as the PHISPLIT approach. Notice that, after the computation of the *small* sized matrix functions $\varphi_\ell(\tau A_\mu)$, with $\mu = 1, \ldots, d$, a single Tucker operator is required to evaluate the approximation.

3.1 Evaluation of small sized matrix φ-functions

The matrices A_μ have a much smaller size compared to K, and the corresponding matrix φ-functions can be directly computed without much effort. In particular, for φ_0 (i.e., the exponential function), we employ the most popular technique for generic matrices, which is based on a diagonal rational Padé approximation coupled with a scaling and squaring algorithm (see Reference [2]). This procedure is encoded in the internal MATLAB function `expm`. Different algorithms that could be used as well, based on Taylor approximations of the matrix exponential, can be found in References [12, 29].

For the computation of higher order matrix φ-functions we rely on a quadrature formula applied to the integral definition (4a). For a generic matrix $X \in \mathbb{C}^{N\times N}$, we have then

$$\varphi_\ell(X) \approx \sum_{i=1}^{q} w_i \mathrm{e}^{(1-\theta_i)X} \frac{\theta_i^{\ell-1}}{(\ell-1)!}, \quad \ell > 0. \tag{16}$$

In order to avoid an impractically large number of quadrature points, we couple the procedure with a modified scaling and squaring algorithm (see Reference [30]). In fact, we scale the original matrix X by 2^s, where s is a natural number defined so that $\|X/2^s\|_1 < 1$, we approximate $\varphi_\ell(X/2^s)$ by means of formula (16) and we recover $\varphi_\ell(X)$ by the recurrence

$$\varphi_\ell(2z) = \frac{1}{2^\ell}\left[\mathrm{e}^z \varphi_\ell(z) + \sum_{k=1}^{\ell} \frac{\varphi_k(z)}{(\ell-k)!}\right]. \tag{17}$$

In order to compute the needed matrix exponentials, we again employ the internal MATLAB function `expm`. Notice that the squaring of φ_ℓ also requires the evaluation of all the φ_j functions, for $0 < j < \ell$. In particular, for the first squaring step, we compute themselves by formula (16) with $\ell = j$ and using the *same* set of matrix exponentials already available for the quadrature procedure of $\varphi_\ell(X/2^s)$. For all the subsequent squaring steps, we use formula (17) itself with $\ell = j$. As a consequence of this procedure, the computation of a single matrix function φ_ℓ makes available all the matrix functions φ_j, with $0 \le j \le \ell$. In practice, for the quadrature we employ the Gauss–Legendre–Lobatto formula, which allows for high precision with a moderate number of quadrature nodes. Moreover, since it uses the endpoints of the quadrature interval, we make use of the matrix exponential $\mathrm{e}^{X/2^s}$ ($\theta_1 = 0$), which is also required for the subsequent squaring procedure, and we avoid generating the last matrix exponential since $\theta_q = 1$. The overall procedure is implemented in MATLAB language in our function `phiquad`.

Alternatively, for the computation of the matrix φ-functions, it is possible to employ the MATLAB routine `phipade` (see Reference [5]), whose algorithm is based on a rational Padé approximation coupled with the squaring formula (17). Another recent technique that employs a polynomial Taylor approximation instead of rational Padé one is presented in Reference [20].

3.2 Practical implementation of the exponential integrators

The implementation of the Lawson methods introduced in Section 2, which require just actions of matrix exponentials, does not suffer from any direction splitting error, thanks to the equivalence between formulas (5) and (6). In particular, the tensor formulation of Lawson–Euler is

$$\boldsymbol{U}_{n+1} = (\boldsymbol{U}_n + \tau \boldsymbol{G}(t_n, \boldsymbol{U}_n)) \times_1 \mathrm{e}^{\tau A_1} \times_2 \cdots \times_d \mathrm{e}^{\tau A_d}, \tag{18}$$

while the Lawson2b scheme is given by

$$\begin{aligned} \boldsymbol{U}_{n2} &= (\boldsymbol{U}_n + \tau \boldsymbol{G}(t_n, \boldsymbol{U}_n)) \times_1 \mathrm{e}^{\tau A_1} \times_2 \cdots \times_d \mathrm{e}^{\tau A_d}, \\ \boldsymbol{U}_{n+1} &= \left(\boldsymbol{U}_n + \frac{\tau}{2}\boldsymbol{G}(t_n, \boldsymbol{U}_n)\right) \times_1 \mathrm{e}^{\tau A_1} \times_2 \cdots \times_d \mathrm{e}^{\tau A_d} + \frac{\tau}{2}\boldsymbol{G}(t_{n+1}, \boldsymbol{U}_{n2}). \end{aligned} \tag{19}$$

The remaining exponential integrators transform as follows. First of all, the action of the matrix K on $\boldsymbol{u}_n$ is computed in tensor form as

$$\sum_{\mu=1}^{d} \left(\boldsymbol{U}_n \times_\mu A_\mu\right) \tag{20}$$

without explicitly assembling the matrix K (see Reference [9]). The exponential Euler PHISPLIT method is

$$\boldsymbol{U}_{n+1} = \boldsymbol{U}_n + \tau \left(\sum_{\mu=1}^{d} (\boldsymbol{U}_n \times_\mu A_\mu) + \boldsymbol{G}(t_n, \boldsymbol{U}_n) \right) \times_1 \varphi_1(\tau A_1) \times_2 \cdots \times_d \varphi_1(\tau A_d). \tag{21}$$

Notice that an alternative version, which we will not consider in this work but could be more appropriate for the cases in which the contribute of $\boldsymbol{G}(t_n, \boldsymbol{U}_n)$ is particularly small, is

$$\boldsymbol{U}_{n+1} = \boldsymbol{U}_n \times_1 \mathrm{e}^{\tau A_1} \times_2 \cdots \times_d \mathrm{e}^{\tau A_d} + \tau \boldsymbol{G}(t_n, \boldsymbol{U}_n) \times_1 \varphi_1(\tau A_1) \times_2 \cdots \times_d \varphi_1(\tau A_d), \tag{22}$$

that is in fact an exact formula if $\boldsymbol{G}(t, \boldsymbol{U}(t))$ is null. The ETD2RK PHISPLIT scheme becomes

$$\begin{aligned} \boldsymbol{U}_{n2} &= \boldsymbol{U}_n + \tau \left(\sum_{\mu=1}^{d} (\boldsymbol{U}_n \times_\mu A_\mu) + \boldsymbol{G}(t_n, \boldsymbol{U}_n) \right) \times_1 \varphi_1(\tau A_1) \times_2 \cdots \times_d \varphi_1(\tau A_d), \\ \boldsymbol{U}_{n+1} &= \boldsymbol{U}_{n2} + \tau \left(2^{d-1} \left(\boldsymbol{G}(t_{n+1}, \boldsymbol{U}_{n2}) - \boldsymbol{G}(t_n, \boldsymbol{U}_n) \right) \right) \times_1 \varphi_2(\tau A_1) \times_2 \cdots \times_d \varphi_2(\tau A_d). \end{aligned} \tag{23}$$

Finally, concerning the exponential Rosenbrock–Euler method for autonomous systems, we assume that the Jacobian K_n can be written as a Kronecker sum, i.e.,

$$K_n = K + \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}}(\boldsymbol{u}_n) = J_d(\boldsymbol{U}_n) \oplus J_{d-1}(\boldsymbol{U}_n) \oplus \cdots \oplus J_1(\boldsymbol{U}_n).$$

Therefore the exponential Rosenbrock–Euler PHISPLIT method is

$$\boldsymbol{U}_{n+1} = \boldsymbol{U}_n + \tau \left(\sum_{\mu=1}^{d} (\boldsymbol{U}_n \times_\mu A_\mu) + \boldsymbol{G}(\boldsymbol{U}_n) \right) \times_1 \varphi_1(\tau J_1(\boldsymbol{U}_n)) \times_2 \cdots \times_d \varphi_1(\tau J_d(\boldsymbol{U}_n)). \tag{24}$$

4 Numerical experiments

In this section we present numerical experiments to validate the proposed approach PHISPLIT. In particular, we will consider a two-dimensional example from linear quadratic control and a three-dimensional example which models an advection–diffusion–reaction equation. To perform the time marching, we will employ the exponential integrators of Section 2 as described in Section 3.2 for the PHISPLIT version.

As term of comparison, we will consider the approximation of actions of φ-functions for matrices with Kronecker sum structure using PHIKS[1] [8]. This algorithm operates in tensor

[1] https://github.com/caliarim/phiks

formulation using μ-mode products, too, but it requires an input tolerance, which we take proportional to the local temporal order of the method and to the norm of the current solution. The proportionality constant is chosen so that the accuracy of the routine does not affect the integration error.

To compute all the relevant tensor operations, i.e., Tucker operators and μ-mode products, we use the functions contained in the package KronPACK[2]. Moreover, to compute the needed matrix φ-functions, we employ the internal MATLAB function `expm` (for φ_0) and the function `phiquad` (for φ_ℓ, $\ell > 0$), as presented in Section 3.1. In terms of hardware, we run all the experiments employing an Intel® Core™ i7-10750H CPU with six physical cores and 16GB of RAM. As a software, we use MathWorks MATLAB® R2022a.

4.1 Example of code usage

All the codes to reproduce the following numerical experiments, together with our implementation of PHISPLIT and the function `phiquad`, can be found in a maintained GitHub repository[3]. They are written in MATLAB language and they are fully compatible with GNU Octave. As an example, we show in Code 1 how to perform a full integration of problem (1) by the ETD2RK PHISPLIT method (23) with constant time step size.

Code 1: Implementation of ETD2RK PHISPLIT

```
% compute once and for all phi1 and phi2 functions and store them
for mu = 1:d
    [phi_store.phi{1:2,mu}] = phiquad(tau * A{mu},2);
end

% time integration
U = U0;
t = 0;
for i = 1:m
    Gn = G(t,U);
    P1 = phisplit(tau,A,kronsumv(U,A) + Gn,1,phi_store); % phi1 approximation
    Un2 = U + tau * P1; % intermediate stage
    P2 = phisplit(tau,A,G(t + tau,U2) - Gn,2,phi_store); % phi2 approximation
    U = Un2 + tau * P2; % updated solution
    t = t + tau;
end
```

Here, we simply notice that the required φ-functions are computed once and for all before time integration by the function `phiquad`, the function `phisplit` computes approximation (15), while the function `kronsumv` of the KronPACK package realizes formula (20).

4.2 Linear quadratic control

We present in this section a classical example from linear quadratic control (see, for instance, References [24, 27]). We are interested in the minimization over the scalar control $v(t) \in \mathbb{R}$ of the functional

$$\mathscr{J}(v) = \frac{1}{2}\int_0^T \left(\alpha s(t)^2 + v(t)^2\right) dt$$

subject to the constraints

$$\begin{aligned} &w'(t) = Aw(t) + bv(t), \quad w(0) = w_0, \\ &s(t) = cw(t). \end{aligned}$$

[2] https://github.com/caliarim/KronPACK
[3] https://github.com/caliarim/phisplit

Here $w(t) \in \mathbb{R}^{n\times 1}$ is a column vector containing the state variables, $s(t) \in \mathbb{R}$ represents the scalar output, $A \in \mathbb{R}^{n\times n}$ is the system matrix, $b \in \mathbb{R}^{n\times 1}$ is the system column vector, $c \in \mathbb{R}^{1\times n}$ is a row vector, and $\alpha \in \mathbb{R}^{+}$ is a positive scalar.

Then, the solution of the constrained optimization problem is determined by the optimal control

$$v^*(t) = -b^{\mathsf{T}}\boldsymbol{U}(t)w(t),$$

where $\boldsymbol{U}(t) \in \mathbb{R}^{n\times n}$ satisfies the symmetric Riccati differential equation

$$\begin{cases} \boldsymbol{U}'(t) = A^{\mathsf{T}}\boldsymbol{U}(t) + \boldsymbol{U}(t)A + C + \boldsymbol{U}(t)B\boldsymbol{U}(t), \\ \boldsymbol{U}(0) = \boldsymbol{Z}, \end{cases} \tag{25}$$

where $C = \alpha c^{\mathsf{T}}c$ and $B = -bb^{\mathsf{T}}$ (see Reference [1, Ch. 4] for a comprehensive introduction to the subject). Here $\boldsymbol{Z} \in \mathbb{R}^{n\times n}$ is a matrix containing all zeros entries. Clearly, equation (25) is in form (2), which in turn can be seen as a problem with two-dimensional Kronecker sum structure (3) and integrated efficiently by means of the techniques described in Section 3. Notice also that the solution of equation (25) converges to a steady state determined by the algebraic Riccati equation

$$A^{\mathsf{T}}\boldsymbol{U}(t) + \boldsymbol{U}(t)A + C + \boldsymbol{U}(t)B\boldsymbol{U}(t) = 0. \tag{26}$$

For our numerical experiment, similarly to what previously done in the literature [21, 22, 24, 27], we take $A \in \mathbb{R}^{\hat{n}^2\times\hat{n}^2}$ as the matrix obtained by the discretization with second order centered finite differences of the operator

$$\partial_{xx} + \partial_{yy} - 10x\partial_x - 100y\partial_y \tag{27}$$

on the domain $[0,1]^2$ with homogeneous Dirichlet boundary conditions. Moreover, the components b_k of the vector b are defined as

$$b_k = \begin{cases} 1 \text{ if } 0.1 < x_i \leq 0.3, \\ 0 \text{ otherwise,} \end{cases} \qquad k = i + (j-1)\hat{n}, \quad i = 1,\ldots,\hat{n}, \quad j = 1,\ldots,\hat{n},$$

while for the components c_k of the vector c we take

$$c_k = \begin{cases} 1 \text{ if } 0.7 < x_i \leq 0.9, \\ 0 \text{ otherwise,} \end{cases} \qquad k = i + (j-1)\hat{n}, \quad i = 1,\ldots,\hat{n}, \quad j = 1,\ldots,\hat{n}.$$

Here x_i represents the ith (inner) grid point along the x direction. Finally, we set $\alpha = 100$.

For the temporal integration of equation (25) we use the exponential Rosenbrock–Euler method, already employed in References [21, 22], and reported in formula (13) (see formula (24) for the PHISPLIT version). In fact, the Jacobian matrix of system (25) has the following Kronecker sum structure

$$K_n = I \otimes (A^{\mathsf{T}} + \boldsymbol{U}_n B) + (A + B\boldsymbol{U}_n)^{\mathsf{T}} \otimes I,$$

where I is the identity matrix of size $n \times n$, with $n = \hat{n}^2$. We remark that the exponential Rosenbrock–Euler PHISPLIT method requires at *each time step* to evaluate the matrix function $\varphi_1(\tau(A^{\mathsf{T}} + \boldsymbol{U}_n B))$, to compute the action $K\boldsymbol{u}_n$ and to perform one Tucker operator. We will employ also the second order exponential Runge–Kutta method ETD2RK, reported in formula (12) and presented in PHISPLIT sense in formula (23). Although each time step of this

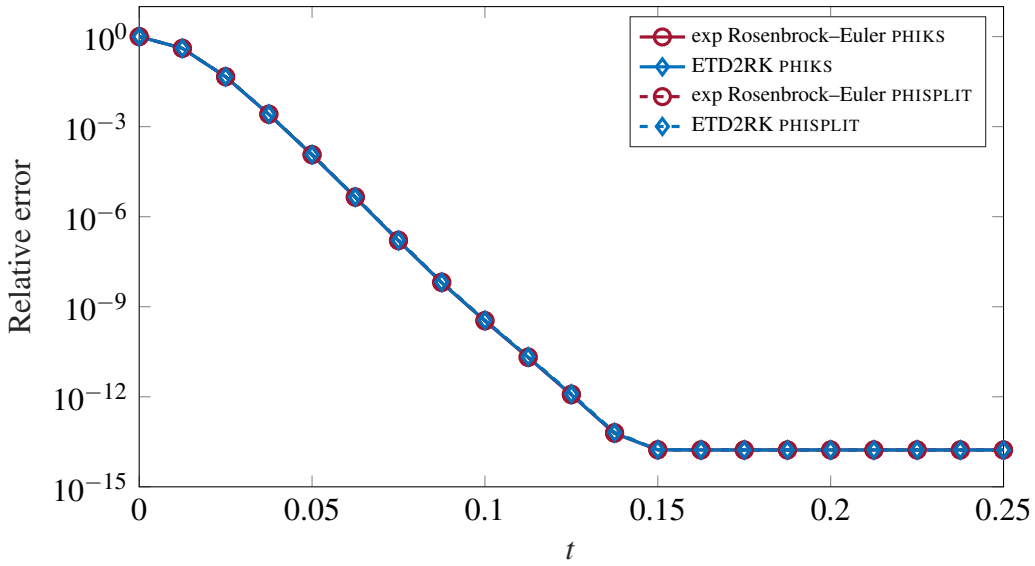


Figure 1: Convergence of the exponential Rosenbrock–Euler and of the ETD2RK methods, both in PHIKS and in PHISPLIT variants, to the steady state of Riccati differential equation (25). Here $\hat{n} = 20$, the integrators have been employed with 200 time steps, and the relative errors, measured in the Frobenius norm with respect to the solution of algebraic Riccati equation (26), are displayed each 10th time step.

integrator requires two Tucker operators plus the action $K\boldsymbol{u}_n$ for the PHISPLIT version, in a constant time step size implementation the needed matrix functions $\varphi_1(\tau A^{\mathsf{T}})$ and $\varphi_2(\tau A^{\mathsf{T}})$ can be computed once and for all at the beginning.

First of all, we verify the implementation of the involved exponential integrators for a long term simulation, i.e., until reaching the steady state. For this experiment, we employ $\hat{n} = 20$ inner discretization points for the x and the y variables. As confirmed by the plot in Figure 1, around time 0.15 the methods, both in their PHIKS and PHISPLIT implementation, approach the solution of equation (26), which is obtained with the MATLAB function `icare` from the Control System Toolbox.

exp Rosenbrock–Euler PHIKS	steps	10	20	30	40	50
	order	–	2.11	2.06	2.05	2.03
ETD2RK PHIKS	steps	7	14	21	28	35
	order	–	2.08	2.05	2.03	2.03
exp Rosenbrock–Euler PHISPLIT	steps	30	65	100	135	170
	order	–	2.05	2.03	2.02	2.02
ETD2RK PHISPLIT	steps	30	65	100	135	170
	order	–	2.05	2.03	2.02	2.02

Table 1: Number of time steps and observed convergence rates for the time integration of Riccati differential equation (25) up to final time $T = 0.025$, with different exponential integrators and $\hat{n} = 30$. The achieved errors and the wall-clock times are displayed in Figure 2.

Then, we compare the performances of the integrators for the solution of equation (25) with $\hat{n} = 30$ and final time $T = 0.025$. All methods are run with different time step sizes

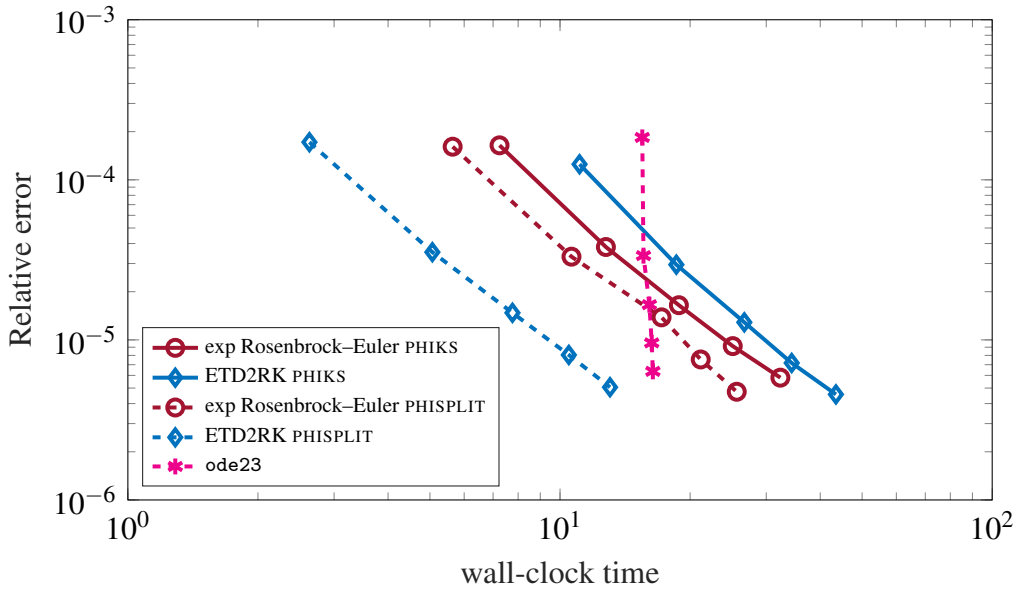


Figure 2: Achieved errors in the Frobenius norm and wall-clock times in seconds for the solution of Riccati differential equation (25) up to final time $T = 0.025$, with different integrators and $\hat{n} = 30$. The number of time steps for each exponential method is reported in Table 1. The input tolerances (both absolute and relative) for `ode23` are 5e−3, 1e−3, 1e−4, 5.5e−5, and 5e−5.

in such a way to reach comparable relative errors with respect to a reference solution. The number of time steps for each method and simulation, together with the numerically observed convergence rate, is reported in Table 1. All the methods appear to be of second order, as expected. In Figure 2 we report the relative errors and the corresponding wall-clock times of the simulations. Here, we also include the performance of the built-in MATLAB function `ode23`. This is an explicit Runge–Kutta method of order three with variable step size, suggested for not stringent tolerances and for moderately stiff problems. In fact, it turned out to be the fastest routine in the ODE suite to reach accuracies in the same range of the other methods. We notice first of all that the exponential Rosenbrock–Euler method is always faster than ETD2RK in the PHIKS implementation, that is with the action of matrix functions computed at a precision that does not affect the temporal error (see the discussion at the beginning of the section). On the other hand, the two implementations with PHISPLIT are always faster compared with their PHIKS counterparts, although they require a larger number of time steps to reach a comparable accuracy. Moreover, the ETD2RK method turns out to be faster with respect to the exponential Rosenbrock–Euler method. This is mainly due to the fact that the matrix functions in the Runge–Kutta case are computed only once before the time marching. This method is in fact at least twice as fast as the other exponential methods and faster than `ode23`, which anyway shows a good performance for the most stringent tolerances.

Finally, we repeat the same experiment with $\hat{n} = 40$. The results are presented in Table 2 and in Figure 3. The global behavior is similar with respect to the previous case, although the speed-ups of the PHISPLIT implementations with respect to their PHIKS counterparts is noticeably larger. In fact, ETD2RK PHISPLIT is still the most efficient method.

Remark 4.1. The discretization of the operator (27) has itself a Kronecker sum structure. Hence,

exp Rosenbrock–Euler PHIKS	steps	15	30	45	60	75
	order	–	2.10	2.06	2.04	2.03
ETD2RK PHIKS	steps	10	20	30	40	50
	order	–	2.12	2.07	2.05	2.04
exp Rosenbrock–Euler PHISPLIT	steps	30	65	100	135	170
	order	–	2.05	2.03	2.02	2.02
ETD2RK PHISPLIT	steps	30	65	100	135	170
	order	–	2.05	2.03	2.02	2.02

Table 2: Number of time steps and observed convergence rates for the time integration of Riccati differential equation (25) up to final time $T = 0.025$, with different exponential integrators and $\hat{n} = 40$. The achieved errors and the wall-clock times are displayed in Figure 3.

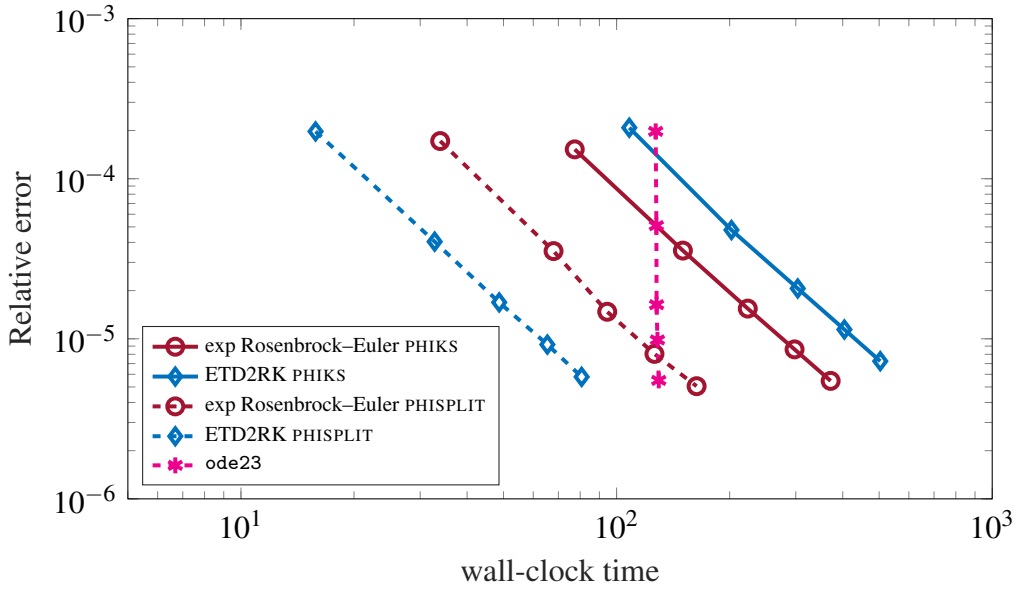


Figure 3: Achieved errors in the Frobenius norm and wall-clock times in seconds for the solution of Riccati differential equation (25) up to final time $T = 0.025$, with different integrators and $\hat{n} = 40$. The number of time steps for each exponential method is reported in Table 2. The input tolerances (both absolute and relative) for `ode23` are 3e−3, 4e−4, 2.3e−4, 9.5e−5, and 8.7e−5.

it is possible to write equation (25) (in vector formulation for simplicity of exposition) as

$$\begin{cases} \boldsymbol{u}'(t) = K\boldsymbol{u}(t) + \boldsymbol{g}(\boldsymbol{u}(t)), \\ \boldsymbol{u}(0) = \boldsymbol{z}, \end{cases}$$

where $\boldsymbol{g}$ and $\boldsymbol{z}$ are the vectorizations of the nonlinearity and of $\boldsymbol{Z}$, respectively, and K has the form

$$K = I \otimes I \otimes I \otimes D_1^{\mathsf{T}} + D_2^{\mathsf{T}} \otimes I \otimes I \otimes I.$$

Here I is an identity matrix of size $\hat{n} \times \hat{n}$ and $D_1 \in \mathbb{R}^{\hat{n}\times\hat{n}}$ and $D_2 \in \mathbb{R}^{\hat{n}\times\hat{n}}$ the discretizations of the operators $\partial_{xx} - 10x\partial_x$ and $\partial_{yy} - 100y\partial_y$, respectively. In the context of temporal integration with exponential Runge–Kutta schemes, we could then use both the PHIKS and the PHISPLIT

approaches with the even smaller sized matrices D_1 and D_2, forming then the approximations at every time steps using Tucker operators with order-4 tensors. However, as this is just possible because of the specific form of the operator (27), we do not pursue this approach here.

4.3 Advection–diffusion–reaction

We now consider the semidiscretization in space of the following three-dimensional evolutionary Advection–Diffusion–Reaction (ADR) equation (see Reference [8])

$$\begin{cases} \partial_t u(t,x_1,x_2,x_3) = \varepsilon\Delta u(t,x_1,x_2,x_3) + \alpha(\partial_{x_1}+\partial_{x_2}+\partial_{x_3})u(t,x_1,x_2,x_3) \\ \qquad\qquad + g(t,x_1,x_2,x_3,u(t,x_1,x_2,x_3)), \\ u_0(x_1,x_2,x_3) = 64x_1(1-x_1)x_2(1-x_2)x_3(1-x_3). \end{cases} \tag{28}$$

The nonlinear function g is given by

$$g(t,x_1,x_2,x_3,u(t,x_1,x_2,x_3)) = \frac{1}{1+u(t,x_1,x_2,x_3)^2} + \Psi(t,x_1,x_2,x_3),$$

where $\Psi(t,x_1,x_2,x_3)$ is such that the analytical solution of the equation is

$$u(t,x_1,x_2,x_3) = \mathrm{e}^t u_0(x_1,x_2,x_3).$$

The problem is solved up to final time $T = 1$ in the domain $[0,1]^3$ and completed with homogeneous Dirichlet boundary conditions. The remaining parameters are set to $\varepsilon = 0.75$ and $\alpha = 0.1$. By semidiscretizing in space with second order centered finite differences, we obtain a system of type (1) with K having three-dimensional Kronecker sum structure, where A_μ approximates $\varepsilon\partial_{x_\mu x_\mu} + \alpha\partial_{x_\mu}$. We first perform simulations with $n_1 = 40$, $n_2 = 41$, and $n_3 = 42$ inner discretization points for the x_1, x_2 and x_3 variables, respectively. The temporal integration is performed with four methods: the Lawson–Euler scheme (9), the exponential Euler method (11), the Lawson2b scheme (10) and the ETD2RK method (12) (see Section 3.2 for their practical implementation and the PHISPLIT versions). In particular, concerning the Lawson schemes, the needed matrix exponentials $\exp(\tau A_\mu)$, with $\mu = 1,2,3$, are computed once and for all at the beginning. Then, one and two Tucker operators per time step, for the first order and second order scheme, respectively, are required to form the approximations during the temporal integration. Concerning the PHISPLIT implementation of exponential Euler and ETD2RK, again we compute once and for all the needed matrix functions $\varphi_1(\tau A_\mu)$ and $\varphi_2(\tau A_\mu)$ before starting the temporal integration, and we then combine them suitably at each time step. This operation requires a single Tucker operator for the first order scheme and two for the second order one, as for the aforementioned Lawson schemes, plus the action $K\boldsymbol{u}_n$. The number of time steps for each method, for both the PHISPLIT and PHIKS implementations, is reported in Table 3, while the reached relative errors and the wall-clock times are summarized in Figure 4. First of all, we notice that all the methods show the expected convergence rate, reported in Table 3 as well. The Lawson–Euler method and the exponential Euler scheme in its PHIKS implementation (see top plot of Figure 4) perform equally well, even if the former requires much more time steps. Overall, the exponential Euler method in its PHISPLIT variant is roughly 10 times faster to reach the highest accuracy in this experiment. If we consider the second order methods (bottom plot of Figure 4), we observe that the Lawson2b scheme needs much more wall-clock time to reach the same level of accuracy of the other methods, and overall the most efficient method is ETD2RK in the PHISPLIT variant. In this plot we report also

Lawson–Euler	steps	800	8800	16800	24800	32800
	order	–	1.00	1.00	1.00	1.00
exp Euler PHIKS	steps	50	450	850	1250	1650
	order	–	1.03	1.00	1.00	1.00
exp Euler PHISPLIT	steps	50	450	850	1250	1650
	order	–	1.03	1.01	1.00	1.00
Lawson2b	steps	1500	5500	9500	13500	17500
	order	–	1.96	1.99	2.00	2.00
ETD2RK PHIKS	steps	20	80	140	200	260
	order	–	1.94	1.97	1.98	1.99
ETD2RK PHISPLIT	steps	40	140	240	340	440
	order	–	2.10	2.04	2.03	2.02

Table 3: Number of time steps and observed convergence rates for the time integration of the semidiscretization of the ADR equation (28) up to final time $T = 1$, with different exponential integrators. Here we considered $n_1 = 40$, $n_2 = 41$ and $n_3 = 42$ inner space discretization points for the x_1, x_2 and x_3 variables, respectively. The achieved errors and the wall-clock times are displayed in Figure 4.

Lawson–Euler	steps	800	8800	16800	24800	32800
	order	–	1.00	1.00	1.00	1.00
exp Euler PHIKS	steps	50	450	850	1250	1650
	order	–	1.02	1.00	1.00	1.00
exp Euler PHISPLIT	steps	50	450	850	1250	1650
	order	–	1.03	1.01	1.00	1.00
Lawson2b	steps	3000	4500	6000	7500	9000
	order	–	1.79	1.87	1.92	1.94
ETD2RK PHIKS	steps	20	80	140	200	260
	order	–	1.94	1.97	1.98	1.99
ETD2RK PHISPLIT	steps	40	140	240	340	440
	order	–	2.10	2.04	2.03	2.02

Table 4: Number of time steps and observed convergence rates for the time integration of the semidiscretization of the ADR equation (28) up to final time $T = 1$, with different exponential integrators. Here we considered $n_1 = 80$, $n_2 = 81$ and $n_3 = 82$ inner space discretization points for the x_1, x_2 and x_3 variables, respectively. The achieved errors and the wall-clock times are displayed in Figure 5.

the results obtained with the internal MATLAB `ode23t` implicit integrator. It is an implementation of the trapezoidal rule with variable step size, which is suggested for stiff problems at low accuracies. Nevertheless, it performs worse than the considered exponential integrators.

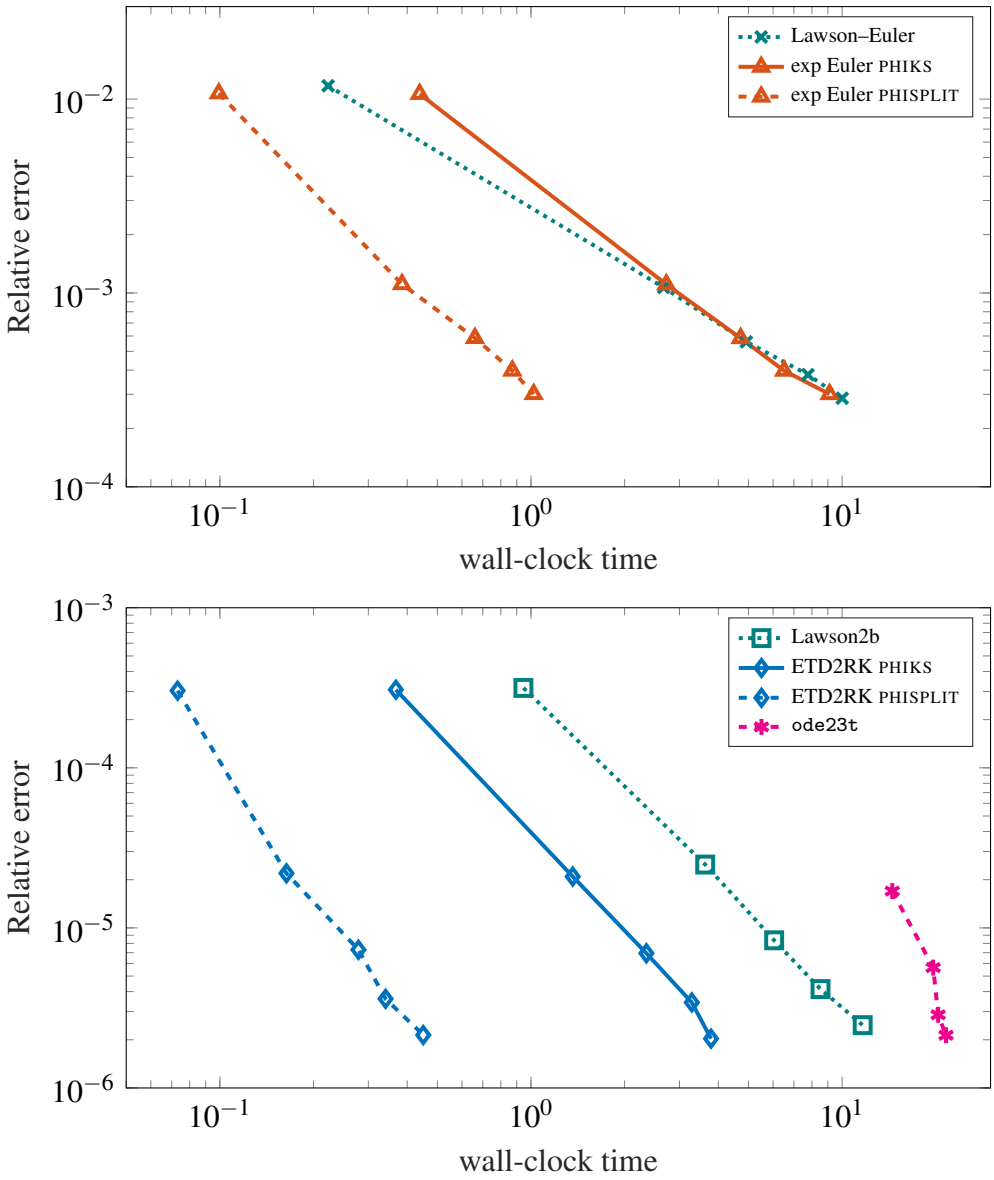


Figure 4: Achieved errors in the infinity norm and wall-clock times in seconds for the solution of the semidiscretization of the ADR equation (28) up to final time $T = 1$, with different exponential integrators of order one (top) and order two (bottom). Here we considered $n_1 = 40$, $n_2 = 41$ and $n_3 = 42$ inner space discretization points for the x_1, x_2 and x_3 variables, respectively. The number of time steps for each exponential method is reported in Table 3. The input tolerances (both absolute and relative) for `ode23t` are 8e−3, 4e−5, 1e−5, and 5e−6.

Finally, we repeat the experiment with $n_1 = 80$, $n_2 = 81$, and $n_3 = 82$ inner discretization points for the x_1, x_2 and x_3 variables, respectively. The number of time steps for each method is reported in Table 4, while the relative errors and the wall-clock times are summarized in Figure 5. Again, we notice that all the methods show the expected convergence rate, reported in Table 4 as well. In particular, for large time step sizes, the Lawson2b method suffers from an order reduction. This is expected, as in these cases the problem is more stiff, and schemes which employ just the exponential function are affected by this phenomenon (see, for instance, Reference [18]). Then, from Figure 5 we observe that the PHISPLIT approach is in any case the most efficient among all the methods and techniques considered, with an increasing speedup for

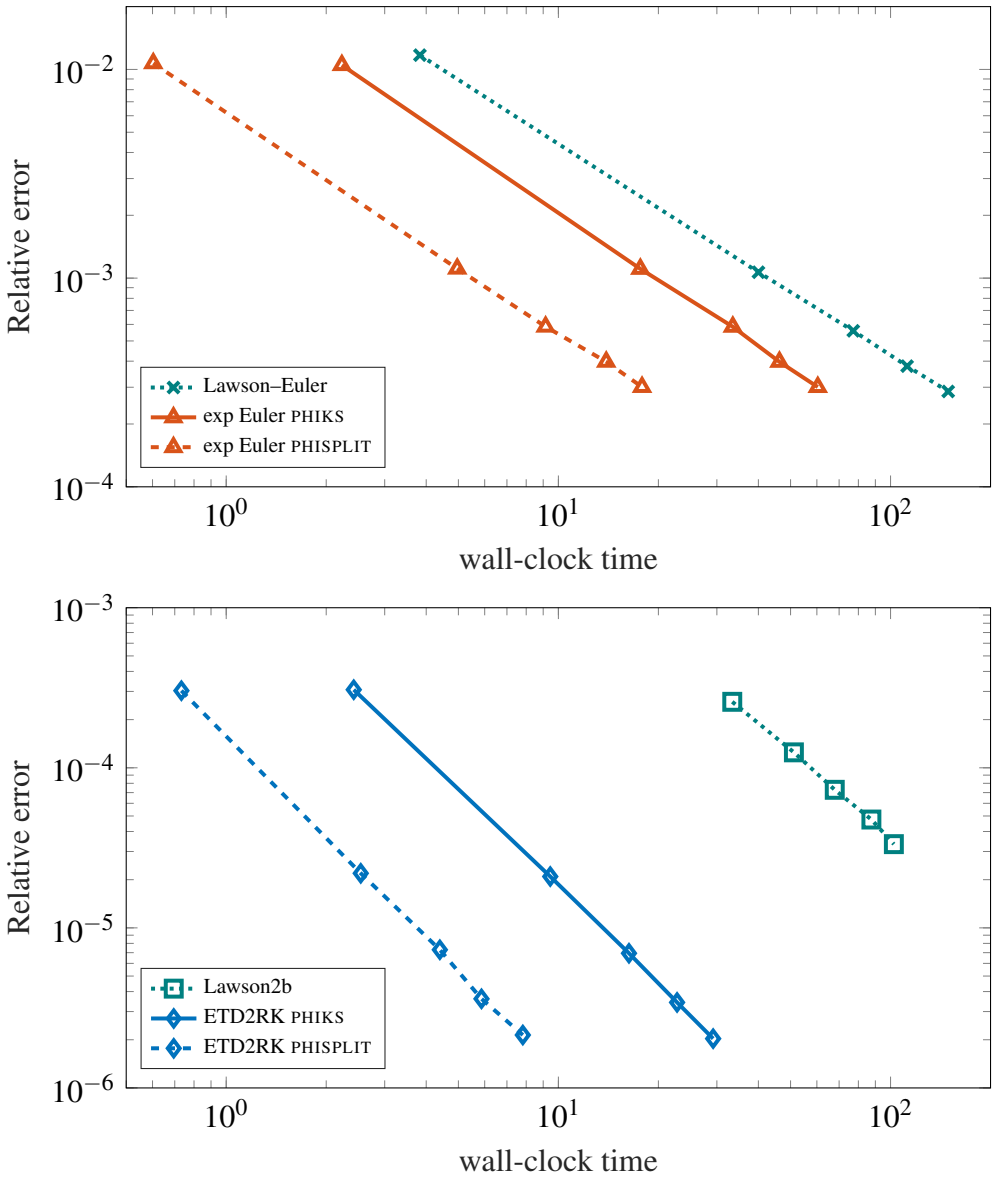


Figure 5: Achieved errors in the infinity norm and wall-clock times in seconds for the solution of the semidiscretization of the ADR equation (28) up to final time $T = 1$, with different exponential integrators of order one (top) and order two (bottom). Here we considered $n_1 = 80$, $n_2 = 81$ and $n_3 = 82$ inner space discretization points for the x_1, x_2 and x_3 variables, respectively. The number of time steps for each method is reported in Table 4.

more stringent accuracies. More in detail, compared with its PHIKS counterparts, the PHISPLIT implementations are roughly 3.5 time faster, even if (in general) they require more time steps to reach a comparable accuracy. On the other hand, the Lawson schemes perform poorly. This is mainly due to the requirement of a large number of time steps to reach the accuracy of the other methods, which is particularly evident for the second order schemes (see bottom of Table 4 and Figure 5). Moreover, while ETD2RK in its PHISPLIT variant reached the most stringent accuracy in less than 10 seconds, Lawson2b was not able to reach an accuracy 10 times larger in 100 seconds. Hence, we decided to stop the simulation with this integrator with a larger number of time steps. Finally, concerning the internal MATLAB ODE suite, none of the methods was able to output a solution within 10 minutes.

5 Conclusions

In this paper, we presented how it is possible to efficiently approximate actions of φ-functions for matrices with d-dimensional Kronecker sum structure using a μ-mode based approach. The technique, that we call PHISPLIT, is suitable when integrating initial valued ordinary differential equations with exponential integrators up to second order. It is based on an inexact direction splitting of the matrix functions involved in the time marching schemes which preserves the order of the method. The effectiveness and superiority of the approach, with respect to another technique to compute actions of φ-functions in Kronecker form, has been shown on a two-dimensional problem from linear quadratic control and on a three-dimensional advection–diffusion–reaction equation, using a variety of exponential integrators. Interesting future developments would be to generalize the approach for higher order integrators and performing GPU simulations with the PHISPLIT technique, possibly in single and/or half precision (which are compatible with the magnitude of the errors of the temporal integration), for different problems from science and engineering.

Acknowledgments

The authors would like to acknowledge partial support from the Program Ricerca di Base 2019 No. RBVR199YFL of the University of Verona entitled “Geometric Evolution of Multi Agent Systems”.

Declaration of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] H. Abou-Kandil, G. Freiling, V. Ionescu, and G. Jank. *Matrix Riccati Equations in Control and Systems Theory*, (Birkhäuser Springer, Basel2003). URL https://doi.org/10.1007/978-3-0348-8081-7

[2] A. H. Al-Mohy and N. J. Higham. *A New Scaling and Squaring Algorithm for the Matrix Exponential*. SIAM J. Matrix Anal. Appl., vol. 31, 3, (2010), 970–989. URL https://doi.org/10.1137/09074721X

[3] 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., vol. 33, 2, (2011), 488–511. URL https://doi.org/10.1137/100788860

[4] H. Berland, B. Skaflestad, and W. M. Wright. *EXPINT — A MATLAB package for exponential integrators*. Tech. Rep. 4, Norwegian University of Science and Technology (2005). URL https://www.math.ntnu.no/preprint/numerics/2005/N4-2005.pdf

[5] H. Berland, B. Skaflestad, and W. M. Wright. *EXPINT—A MATLAB Package for Exponential Integrators*. ACM Trans. Math. Softw., vol. 33, 1, (2007), Article 4. URL https://doi.org/10.1145/1206040.1206044

[6] M. Caliari, F. Cassini, L. Einkemmer, A. Ostermann, and F. Zivcovich. *A μ-mode integrator for solving evolution equations in Kronecker form*. J. Comput. Phys., vol. 455, (2022), 110989. URL https://doi.org/10.1016/j.jcp.2022.110989

[7] M. Caliari, F. Cassini, and F. Zivcovich. *Approximation of the matrix exponential for matrices with a skinny field of values*. BIT Numer. Math., vol. 60, 4, (2020), 1113–1131. URL https://doi.org/10.1007/s10543-020-00809-0

[8] M. Caliari, F. Cassini, and F. Zivcovich. *A μ-mode approach for exponential integrators: actions of φ-functions of Kronecker sums*. arXiv preprint arXiv:2210.07667. URL https://doi.org/10.48550/arXiv.2210.07667

[9] M. Caliari, F. Cassini, and F. Zivcovich. *BAMPHI: Matrix-free and transpose-free action of linear combinations of φ-functions from exponential integrators*. J. Comput. Appl. Math., vol. 423, (2023), 114973. URL https://doi.org/10.1016/j.cam.2022.114973

[10] M. Caliari, F. Cassini, and F. Zivcovich. *A μ-mode BLAS approach for multidimensional tensor-structured problems*. Numer. Algorithms, vol. 92, 4, (2023), 2483–2508. URL https://doi.org/10.1007/s11075-022-01399-4

[11] M. Caliari, P. Kandolf, A. Ostermann, and S. Rainer. *The Leja Method Revisited: Backward Error Analysis for the Matrix Exponential*. SIAM J. Sci. Comput., vol. 38, 3, (2016), A1639–A1661. URL https://doi.org/10.1137/15M1027620

[12] M. Caliari and F. Zivcovich. *On-the-fly backward error estimate for matrix exponential approximation by Taylor algorithm*. J. Comput. Appl. Math., vol. 346, (2019), 532–548. URL https://doi.org/10.1016/j.cam.2018.07.042

[13] S. M. Cox and P. C. Matthews. *Exponential Time Differencing for Stiff Systems*. J. Comput. Phys., vol. 176, 2, (2002), 430–455. URL https://doi.org/10.1006/jcph.2002.6995

[14] M. Croci and J. Muñoz-Matute. *Exploiting Kronecker structure in exponential integrators: Fast approximation of the action of φ-functions of matrices via quadrature*. J. Comput. Sci., vol. 67, (2023), 101966. URL https://doi.org/10.1016/j.jocs.2023.101966

[15] L. Einkemmer and A. Ostermann. *Overcoming Order Reduction in Diffusion-Reaction Splitting. Part 1: Dirichlet Boundary Conditions*. SIAM J. Sci. Comput., vol. 37, 3, (2015), A1577–A1592. URL https://doi.org/10.1137/140994204

[16] S. Gaudreault, G. Rainwater, and M. Tokman. *KIOPS: A fast adaptive Krylov subspace solver for exponential integrators*. J. Comput. Phys., vol. 372, (2018), 236–255. URL https://doi.org/10.1016/j.jcp.2018.06.026

[17] C. González, A. Ostermann, and M. Thalhammer. *A second-order Magnus-type integrator for nonautonomous parabolic problems*. J. Comput. Appl. Math., vol. 189, 1–2, (2006), 142–156. URL https://doi.org/10.1016/j.cam.2005.04.036

[18] M. Hochbruck, J. Leibold, and A. Ostermann. *On the convergence of Lawson methods for semilinear stiff problems*. Numer. Math., vol. 145, (2020), 553–580. URL https://doi.org/10.1007/s00211-020-01120-4

[19] M. Hochbruck and A. Ostermann. *Exponential integrators*. Acta Numer., vol. 19, (2010), 209–286. URL https://doi.org/10.1017/S0962492910000048

[20] D. Li, S. Yang, and J. Lan. *Efficient and accurate computation for the φ-functions arising from exponential integrators*. Calcolo, vol. 59, (2022), Article 11. URL https://doi.org/10.1007/s10092-021-00453-2

[21] D. Li, X. Zhang, and R. Liu. *Exponential integrators for large-scale stiff Riccati differential equations*. J. Comput. Appl. Math., vol. 389, (2021), 113360. URL https://doi.org/10.1016/j.cam.2020.113360

[22] D. Li, Y. Zhang, and X. Zhang. *Computing the Lyapunov operator φ-functions, with an application to matrix-valued exponential integrators*. Appl. Numer. Math., vol. 182, (2022), 330–343. URL https://doi.org/10.1016/j.apnum.2022.08.009

[23] V. T. Luan, J. A. Pudykiewicz, and D. R. Reynolds. *Further development of efficient and accurate time integration schemes for meteorological models*. J. Comput. Phys., vol. 376, (2019), 817–837. URL https://doi.org/10.1016/j.jcp.2018.10.018

[24] H. Mena, A. Ostermann, L.-M. Pfurtscheller, and C. Piazzola. *Numerical low-rank approximation of matrix differential equations*. J. Comput. Appl. Math., vol. 340, (2018), 602–614. URL https://doi.org/10.1016/j.cam.2018.01.035

[25] J. Muñoz-Matute, D. Pardo, and V. M. Calo. *Exploiting the Kronecker product structure of φ–functions in exponential integrators*. Int. J. Numer. Methods Eng., vol. 123, 9, (2022), 2142–2161. URL https://doi.org/10.1002/nme.6929

[26] J. Niesen and W. M. Wright. *Algorithm 919: A Krylov Subspace Algorithm for Evaluating the ϕ-Functions Appearing in Exponential Integrators*. ACM Trans. Math. Softw., vol. 38, 3, (2012), 1–19. URL https://doi.org/10.1145/2168773.2168781

[27] T. Penzl. *LYAPACK: A MATLAB Toolbox for Large Lyapunov and Riccati Equations, Model Reduction Problems, and Linear–Quadratic Optimal Control Problems. Users' Guide (Version 1.0)* (2000). URL https://www.netlib.org/lyapack/guide.pdf

[28] F. Rousset and K. Schratz. *A General Framework of Low Regularity Integrators*. SIAM J. Numer. Anal., vol. 59, 3, (2021), 1735–1768. URL https://doi.org/10.1137/20M1371506

[29] J. Sastre, J. Ibáñez, and E. Defez. *Boosting the computation of the matrix exponential*. Appl. Math. Comput., vol. 340, (2019), 206–220. URL https://doi.org/10.1016/j.amc.2018.08.017

[30] B. Skaflestad and W. M. Wright. *The scaling and modified squaring method for matrix functions related to the exponential*. Appl. Numer. Math., vol. 59, 3–4, (2009), 783–799. URL https://doi.org/10.1016/j.apnum.2008.03.035

[31] C. F. Van Loan. *The ubiquitous Kronecker product*. J. Comput. Appl. Math., vol. 123, 1–2, (2000), 85–100. URL https://doi.org/10.1016/S0377-0427(00)00393-9

J. Approx. Softw., Vol. 1, No. 1 (2024) 2

Journal of Approximation Software

Qsurf: compressed QMC integration on parametric surfaces

G. Elefante [1], A. Sommariva [1], and M. Vianello [1,⋆]

[1] *Department of Mathematics, University of Padova, Italy*

Received: 17/04/2023 – Published: 23/07/2024
Communicated by: R. Cavoretto and A. De Rossi

Abstract

We discuss a "bottom-up" algorithm for Tchakaloff-like compression of QMC (Quasi-MonteCarlo) integration on surfaces that admit an analytic parametrization. The key tools are Davis-Wilhelmsen theorem on the so-called "Tchakaloff sets" for positive linear functionals on polynomial spaces, and Lawson-Hanson algorithm for NNLS. This algorithm shows remarkable speed-ups with respect to Caratheodory-like subsampling, since it is able to work with much smaller matrices. We provide the corresponding Matlab code *Qsurf*, together with integration tests on regions of different surfaces such as sphere, torus, and a smooth Cartesian graph.

Keywords: Quasi-MonteCarlo formulas, surface integrals, analytic parametrization, low-discrepancy sequences, rejection sampling, Tchakaloff sets, quadrature compression, Davis-Wilhelmsen theorem, NonNegative Least Squares. (MSC2020: 65C05, 65D32)

1 Introduction

In the recent study [14], we have considered the compression problem for Quasi-MonteCarlo (QMC) surface integration on multibubbles (the surface of a ball union in $\mathbb{R}^3$), which can have a quite complicated structure. Indeed, numerical modelling with multibubbles is relevant in several applications, but compression of QMC integration seemed an overlooked approach, especially in the case of surface integrals.

In this paper, we extend such an approach to compressed QMC formulas for general integration problems on compact subsets of surfaces in $\mathbb{R}^3$, admitting an analytic parametrization. Such formulas preserve the approximation power of QMC up to the best uniform polynomial approximation error of a given degree to the integrand, but using a much lower number of sampling points.

⋆ Corresponding author: marcov@math.unipd.it

The key tools are Davis-Wilhelmsen theorem on the so-called "Tchakaloff sets" for positive linear functionals and Lawson-Hanson algorithm for NNLS, which allows to extract a set of "equivalent" re-weighted nodes from a huge uniformly distributed sequence with respect to the surface measure, by working in a "bottom-up" mode. Such a sequence can be obtained for example from a bivariate Halton sequence by an area-preserving map, when available, or by the probabilistic method of rejection sampling, which has been extended to the low-discrepancy deterministic setting, cf. e.g. [20, 31]. On the other hand, there are other relevant QMC point sequences on manifolds, see e.g. [2, 3].

The "bottom-up" approach shows remarkable speed-ups with respect to Caratheodory-like subsampling (cf. e.g. [16, 19, 21, 25, 29]), since it is able to work with much smaller matrices. We stress that one of the main difficulties consists in adapting the compression algorithm to work on the appropriate spaces of trivariate polynomials restricted to the surface, since the dimension of trivariate polynomial spaces can collapse in the case of algebraic surfaces.

The paper is organized as follows. In Section 2 we briefly discuss the theoretical background and the main idea of the "bottom-up" compression algorithm. Then, we sketch the algorithm, that has been implemented in Matlab, and comment on the main computational issues. Finally, in Section 3 we present some numerical examples concerning regions of sphere and torus, and the Cartesian graph of an analytic function. All the codes and demos are all freely available at [15].

2 QMC compression on surfaces

The possibility of compressing QMC integration rests on a somehow overlooked but relevant result of quadrature theory, originally proved by Davis [5] and then extended by Wilhelmsen [30]. Only recently this theorem has been rediscovered as a basic tool for positive cubature via adaptive NNLS moment-matching, cf. [13, 18, 26, 27].

Theorem 1. (Davis, 1967 - Wilhelmsen, 1976) Let $\{f_j\}_{1\le j\le N}$ be continuous, real-valued, linearly independent functions defined on a compact set $\Omega \subset \mathbb{R}^d$, and $\mathscr{F} = span(f_1,\dots,f_N)$. Assume that $\mathscr{F}$ satisfies the Krein condition (i.e. there is at least one $f \in \mathscr{F}$ which does not vanish on Ω) and that L is a positive linear functional on $\mathscr{F}$, i.e. $L(f) > 0$ for every $f \in \mathscr{F}$, $f \ge 0$ not vanishing everywhere in Ω.

If $\{P_i\}_{i=1}^{\infty}$ is an everywhere dense subset of Ω, then for sufficiently large m, the set $X_m = \{P_i\}_{i=1,\dots,m}$ is a "Tchakaloff set", i.e. there exist weights $w_k > 0$, $k = 1,\dots,\nu$, and nodes $\{Z_k\}_{k=1,\dots,\nu} \subset X_m \subset \Omega$, with $\nu = \mathrm{card}(\{Z_k\}) \le N$, such that

$$L(f) = \ell(f) = \sum_{k=1}^{\nu} w_k f(Z_k)\ ,\ \ \forall f \in \mathscr{F}\ . \tag{1}$$

Davis-Wilhelmsen theorem is a constructive generalization of the well-known Tchakaloff theorem [28] on the existence of positive quadrature formulas. But, just in view of its generality, it can be directly applied to a discrete functional like a QMC formula on $\Omega = \mathscr{J}$, $\mathscr{J}$ being a compact region of a surface $\mathscr{S} \subset \mathbb{R}^3$

$$L(f) = L_{\mathrm{QMC}}(f) = \frac{\sigma(\mathscr{J})}{M}\sum_{i=1}^{M} f(P_i) \approx \int_{\mathscr{J}} f\, d\sigma\ ,\ \ f \in C(\mathscr{J})\ , \tag{2}$$

where

$$X_M = \{P_i\}_{i=1,\dots,M}\ ,\ \ M > N\ ,$$

is a uniformly distributed sequence on $\mathscr{J}$ and σ is the surface measure. Typically one generates a uniformly distributed sequence of cardinality say M_0 on the bounding surface $\mathscr{S} \supseteq \mathscr{J}$, from which sequence on $\mathscr{J}$ is extracted by a suitable in-domain algorithm. We observe that if $\sigma(\mathscr{J})$ is unknown or difficult to compute, it can be approximated as $\sigma(\mathscr{J}) \approx \sigma(\mathscr{S})M/M_0$.

Positivity of the functional for $f \in \mathscr{F} = \mathbb{P}_n^3(\mathscr{J})$ (the space of trivariate polynomials of total degree not exceeding n restricted to $\mathscr{J}$), is ensured whenever the set X_M is $\mathbb{P}_n^3(\mathscr{J})$-determining, i.e. a polynomial vanishing there vanishes everywhere on $\mathscr{J}$, or equivalently $dim(\mathbb{P}_n^3(X_M)) = N = dim(\mathbb{P}_n^3(\mathscr{J}))$, or even

$$rank(V_M) = N\ ,\ \ V_M = V^{(n)}(X_M) = [f_j(P_i)] \in \mathbb{R}^{M\times N} \tag{3}$$

where V_M is the corresponding rectangular Vandermonde-like matrix. Notice that, X_M being a sequence, for every $k \leq M$ we have that

$$V_k = V^{(n)}(X_k) = [(V_M)_{ij}]\ ,\ \ 1 \leq i \leq k\ ,\ 1 \leq j \leq N\ . \tag{4}$$

We stress that the full rank requirement for V_M is not restrictive, in practice, when $\mathscr{S}$ is a surface that admits an analytic parametrization, the subset $\mathscr{J}$ is $\mathbb{P}_n^3(\mathscr{S})$-determining and the points are uniformly distributed with respect to the surface measure. Indeed, the probability that $det(V_N) = 0$ dealing with uniformly distributed points is null, as is ensured by the following proposition which is a special case of a general result proved in [7] in the case of continuous random point distributions.

Proposition 1. Let $\mathscr{S} \subset \mathbb{R}^3$ be a surface that admits an analytic parametrization $P = \Psi(u,v)$ from a connected open set $D \subset \mathbb{R}^2$, i.e. $\Psi = (\Psi_1, \Psi_2, \Psi_3)$ where $\Psi_i : D \to \mathbb{R}^3$ are analytic and $\Psi(D) = \mathscr{S}$. Moreover, let $\{f_j\}_{1\leq j\leq N}$ be a basis of $\mathbb{P}_n^3(\mathscr{S})$ and $\{(u_i,v_i)\}_{i\geq 1}$ an equidistributed sequence on D with respect to any given probability density $\phi(u,v)$.

Then, the points $\{P_i = \Psi(u_i,v_i)\}_{1\leq i\leq N}$ are almost surely unisolvent for polynomial interpolation in $\mathbb{P}_n^3(\mathscr{S})$.

Remark 1. *We can apply this proposition to the case where the parametrization is regular (so that the surface area element $\|\partial_u\Psi \times \partial_v\Psi\|_2/\sigma(\mathscr{J})$ is well-defined), $dim(\mathbb{P}_n^3(\mathscr{S})) = N = dim(\mathbb{P}_n^3(\mathscr{J}))$, and $d\sigma = \phi(u,v)\,du\,dv$ with density*

$$\phi(u,v) = I_{\mathscr{J}}(\Psi(u,v))\,\|\partial_u\Psi \times \partial_v\Psi\|_2/\sigma(\mathscr{J})\ , \tag{5}$$

$I_{\mathscr{J}}$ denoting the indicator function of $\mathscr{J}$.

Remark 2. *To be rigorous, we should notice that Proposition 1 concerns random sequences, whereas here we deal with quasi-random sequences, where we can expect, and we have indeed verified experimentally, that the full-rank property of V_M in practice holds. In order to construct a sequence $\{P_i\}_{1\leq i\leq N}$ that be uniformly distributed on $\mathscr{J}$ with respect to the surface measure, we can adopt the classical probabilistic method of* rejection sampling *on D applied to the density (5), that has been extended to low-discrepancy sequences; cf. [20, 31] with the references therein. Clearly, a suitable "in-domain" algorithm for $\mathscr{J}$ has to be at hand.*

Remark 3. *We recall that polynomial spaces can collapse on algebraic surfaces, i.e. it happens that $dim(\mathbb{P}_n^3(\mathscr{J})) = dim(\mathbb{P}_n^3(\mathscr{S})) < dim(\mathbb{P}_n^3(\mathbb{R}^3)) = (n+1)(n+2)(n+3)/6$. For example, if $\mathscr{J}$ is a subset with internal points w.r.t. the topology of the sphere S^2 (e.g. a spherical polygon as in the first example below), we have that $dim(\mathbb{P}_n^3(\mathscr{J})) = dim(\mathbb{P}_n^3(S^2)) = (n+1)^2$; we refer the reader, e.g., to [4] concerning the delicate matter of determining polynomial spaces dimension on algebraic varieties.*

In view of the results quoted above, when $M \gg N$ we can then try to find a Tchakaloff set $X_m \subset X_M$, with $N \le m < M$, such that there exists a sparse nonnegative solution vector u to the underdetermined *moment-matching system*

$$V_m^t u = \lambda = V_M^t e \ , \ \ e = \frac{\sigma(\mathscr{J})}{M}\,(1,\ldots,1)^t \ . \tag{6}$$

In practice, we solve (6) via Lawson-Hanson active-set method [17] applied to the NNLS problem

$$\min_{u\ge 0} \|V_m^t u - \lambda\|_2 \ , \tag{7}$$

accepting the solution when the residual size is small, say

$$\|V_m^t u - \lambda\|_2 < \varepsilon \tag{8}$$

where ε is a given tolerance. Then the nonzero components of u provide nodes and weights of a compressed QMC formula extracted from X_m, that is $\{w_k\} = \{u_i : u_i > 0\}$ and $\{Z_k\} = \{P_i : u_i > 0\}$, giving

$$\ell_{\text{QMC}}(f) = \sum_{k=1}^{\nu} w_k f(Z_k) \ , \ \ \nu \le N \ll M \ \ , \tag{9}$$

where $\ell_{\text{QMC}}(f) = L_{\text{QMC}}(f)$ for every $f \in \mathbb{P}_n^3(\mathscr{J})$.

It is worth recalling that, in the case $m = M$, Caratheodory theorem on finite-dimensional conic combinations (applied to the columns of V_M^t) would ensure directly the existence of a Tchakaloff-like representation of the QMC functional (cf. [21] for a discussion on this point in the general framework of discrete measure compression by "Caratheodory-Tchakaloff subsampling"). In such a way, however, working with say an order of $10^5 - 10^6$ nodes, we would have to manage a huge matrix, that is we would have to solve the huge NNLS problem

$$\min_{u\ge 0} \|V_M^t u - \lambda\|_2 \ . \tag{10}$$

On the contrary, we can substantially reduce the computation cost by solving an increasing sequence of much smaller problems like (7) with $m := m_1, m_2, m_3, \ldots$ and $m_1 < m_2 < m_3 < \cdots \le M$,

$$\min_{u\ge 0} \|V_{m_j}^t u - \lambda\|_2 \ , \ \ j = 1,2,3,\ldots \ , \ m_1 \ge N \ , \tag{11}$$

corresponding to increasingly dense subsets $X_{m_1} \subset X_{m_2} \subset \cdots \subseteq X_M$, until the residual becomes sufficiently low. We may call this procedure a "*bottom-up*" approach to QMC compression. Indeed, as shown in [13], with a suitable choice of the sequence $\{m_j\}$ the residual becomes extremely small in few iterations, with a substantial speed-up with respect to (10).

Now, following [13] it is easy to derive the following error estimate

$$\begin{aligned} |\ell_{\text{QMC}}(f) - \int_{\mathscr{J}} f\,d\sigma| &\le \mathscr{E}_{\text{QMC}}(f) + 2\,\mu(\mathscr{J})\,E_n(f;X_M) \\ &\le \mathscr{E}_{\text{QMC}}(f) + 2\,\mu(\mathscr{J})\,E_n(f;\mathscr{J}) \ , \end{aligned} \tag{12}$$

valid for every $f \in C(\mathscr{J})$, where $\mathscr{E}_{\text{QMC}}(f) = |L_{\text{QMC}}(f) - \int_{\mathscr{J}} f\,d\sigma|$ and we define $E_n(f;K) = \inf_{p\in\mathbb{P}_n^3(K)} \|f-p\|_{\infty,K}$ with K discrete or continuous compact set.

The meaning of (12) is that the compressed QMC functional ℓ_{QMC} retains the approximation power of the original QMC formula, up to a quantity proportional to the best polynomial approximation error to f in the uniform norm on X_M (and hence by inclusion in the uniform

norm on $\mathscr{J}$). We recall that the latter can be estimated depending on the regularity of f by multivariate Jackson-like theorems, cf. e.g. [22] for volume integrals where $\mathscr{J}$ is the closure of a bounded open set and [23] for the case of the sphere. On the other hand, we do not deepen here the topic of QMC convergence and error estimates, in particular on manifolds, referring the reader to specific papers and monographs, like e.g. [2, 3, 12] .

2.1 Algorithm description and computational issues

In this section we sketch the method implementation in the form of a pseudo-code and discuss its main computational features.

Algorithm Qsurf: *Bottom-up compression of QMC integration on a compact subset $\mathscr{J}$ of a surface $\mathscr{S} \subset \mathbb{R}^3$ with a regular analytic parametrization on a domain $D \subset \mathbb{R}^2$*

- `input`: the bounding surface measure $\sigma(\mathscr{S})$, possibly the measure $\sigma(\mathscr{J})$, the cardinality M_0 of a uniformly distributed sequence on $\mathscr{S}$, the cardinality increase factor $\theta > 1$, the moment-matching tolerance ε, the residual decrease threshold $\tau > 1$

(i) generate M_0 low-discrepancy points on the bounding surface $\mathscr{S} \supseteq \mathscr{J}$ (for example by rejection sampling on D w.r.t. the surface measure density) and extract the points $X = X_M = \{P_i\}_{i=1,\dots,M}$ that lie on $\mathscr{J}$ (by a suitable "in-domain" algorithm)

(ii) if unknown, approximate $\sigma(\mathscr{J})$ as $\sigma(\mathscr{J}) := \sigma(\mathscr{S})M/M_0$

(iii) *% selecting a basis of* $\mathbb{P}^3_n(X)$

(iii1) take a polynomial basis $\{p_1, \dots, p_{\mathscr{V}}\}$ of $\mathbb{P}^3_n$, $\mathscr{V} = \frac{(n+1)(n+2)(n+3)}{6}$

(iii2) compute the Vandermonde-like matrix $C := [p_j(P_i)] \in \mathbb{R}^{M \times \mathscr{V}}$

(iii3) compute $N := rank(C_{\mathscr{V}})$ where $C_{\mathscr{V}} = [(C)_{ij}]$, $1 \le i, j \le \mathscr{V}$

(iii4) compute the QR factorization with column pivoting $C^{\pi}_{\mathscr{V}} = QR$ where $\pi = (\pi_1, \dots, \pi_{\mathscr{V}})$ is the column permutation vector

(iii5) set $V_M := [(C)_{ij}]$, $1 \le i \le M$, $j = \pi_1, \dots, \pi_N$

(iv) compute the QMC moments $\lambda := V_M^t e$, $e = \sigma(\mathscr{J})/M\,(1, \dots, 1)^t$

(v) *% bottom-up QMC compression*

(v1) initialize m, $N \le m \ll M$ and $momtype := 0$

(v2) set $V_m := [(V_M)_{ij}]$, $1 \le i \le m$, $1 \le j \le N$

(v3) compute the QR factorization $V_m = Q_m R_m$

(v4) if $momtype = 0$ `then`

- compute the modified QMC moments $q_m = (R_m^{-1})^t \lambda$ by solving the system $R_m^t q_m = \lambda$ via Gaussian elimination with row pivoting
- set $A_m = Q_m$

`else`

- compute the modified QMC moments $q_m = (R_m^{-1})^t \lambda = (V_M R_m^{-1})^t e$ as $q_m := A_M^t e$, by solving the matrix equation $R_m^t A_M^t = V_M^t$ via Gaussian elimination with row pivoting
- set $A_m = [(A_M)_{i,j}]$, $1 \le i \le m$, $1 \le j \le N$

($v5$) compute a sparse solution u to the NNLS problem

$$\min_{u \ge 0} \|A_m^t u - q_m\|_2$$

by Lawson-Hanson active-set algorithm

($v6$) compute the relative residual $res := \|V_m^t u - \lambda\|_2 / \|\lambda\|_2$

($v7$) `if` $res_0/res > \tau$ & $m < M$ `then`
($v7a$) `if` $momtype = 0$ `then`
- set $momtype := 1$ and `goto` ($v2$)
`else`
- set $m := M$ and `goto` ($v2$)

($v8$) `if` $res > \varepsilon$ & $\lceil \theta m \rceil \le M$ `then`
- set $m := \lceil \theta m \rceil$, $res_0 := res$ and `goto` ($v2$)

(vi) select the indexes $J = \{i : u_i > 0\}$ and set $w := u(J)$ and $Z := X(J)$

- `output`: the weights $\{w_k\}$ and nodes $\{Z_k\} \subset X$ of a compressed QMC formula on $\mathscr{J}$ with moment-matching residual res

Now, some observations on delicate aspects are in order. Step (iii) is a key point in the case of surface integration. As for the starting polynomial basis, for conditioning problems we adopt the product Chebyshev total-degree basis of the smaller bounding box say $[a_1,b_1] \times [a_2,b_2] \times [a_3,b_3] \supset X$, namely

$$p_j(x,y,z) = T_{\alpha_1(j)}(\sigma_1(x)) \cdot T_{\alpha_2(j)}(\sigma_2(y)) \cdot T_{\alpha_3(j)}(\sigma_3(z)), \;\; j = 1, \ldots, \mathscr{V} \,,$$

$$\sigma_i : [a_i,b_i] \mapsto [-1,1] \,, \;\; \sigma_i(t) = \frac{2t - b_i - a_i}{b_i - a_i} \,, \;\; i = 1,2,3 \,,$$

where $j \mapsto \alpha(j)$ corresponds to the graded lexicographical ordering of the 3-indexes $\alpha = (\alpha_1, \alpha_2, \alpha_3)$, $0 \le \alpha_1 + \alpha_2 + \alpha_3 \le n$.

Moreover, we recall that $dim(\mathbb{P}_n^3(X))$ is simply the rank of the corresponding rectangular Vandermonde-like matrix C. In step ($iii4$), instead, we work with the principal square submatrix $C_{\mathscr{V}}$. As already observed in Section 2, with $\mathscr{V} \ge dim(\mathbb{P}_n^3(\mathscr{J}))$ uniformly distributed points on $\mathscr{J}$, the probability that such a rank be lower than $dim(\mathbb{P}_n^3(\mathscr{J}))$ is null, so that "almost-surely" Wilhelmsen theorem applies. In Matlab, one can use directly the built-in function `rank` based on an economy-size version of SVD. Notice that we are using a numerical rank (obtained by discarding the singular values below a tolerance close to machine precision), not the true rank. Nevertheless, dealing with polynomials restricted to X this is numerically equivalent to work, up to very small errors, with the true polynomial space. We stress that when $\mathscr{V} \ll M$, using $C_{\mathscr{V}}$ instead of C gives experimentally a substantial speed-up to the rank computation, by a factor roughly of the order of $M/\mathscr{V}$.

The polynomial basis selection, i.e. the determination of a set of linearly independent polynomials on $\mathscr{J}$ within the starting basis, is performed in ($iii4$) by a QR factorization with column pivoting of the Chebyshev-Vandermonde matrix $C_{\mathscr{V}}$ (again, an economy-size version

can be used in Matlab that produces only the first N columns of Q and a column permutation vector). In such a way we select a polynomial basis of $\mathbb{P}^3_n(X)$ by the first N components $\pi_1,\ldots,\pi_N$ of the column permutation, say $(f_1,\ldots,f_N)=(p_{\pi_1},\ldots,p_{\pi_N})$.

We can now turn to the second key step of the algorithm, that is the extraction of a compressed QMC formula in (v). As already observed, this is based on Wilhelmsen theorem, using just X_M as extraction set, in a "bottom-up" fashion. This procedure avoids working directly on the complete matrix V_M (cf. (10)), as done instead in other previous approaches to QMC compression like [6], cf. also the discussion in [1, 13]. Indeed, the overall number of points, i.e. of rows of V_M, can be huge, up to the order of 10^5-10^6. In practice, we proceed along increasingly dense subsequences of the overall sequence, solving the corresponding NNLS problems and stopping when the relative moment-matching residual becomes sufficiently small.

To this purpose the classical Lawson-Hanson iterative method turns out to be a good choice, since it automatically seeks a sparse solution with a number of nonzeros not exceeding N. The method is implemented in most numerical programming environments, e.g. in Matlab by the built-in function `lsqnonneg`. On the other hand, there are improvements of the algorithm, cf. for example [24] for a survey, and the recent implementation named LHDM based on the concept of "Deviation Maximization" instead of "column pivoting" for the underlying QR factorizations, cf. [8, 9]. Indeed, in the present framework we have adopted LHDM, since it gives experimentally a speed-up of at least 2 with respect to `lsqnonneg`.

In order to cope ill-conditioning of the matrices used in the sequence of NNLS problem, that worsens increasing the degree, we perform an orthogonalization of V_m by QR factorization, that corresponds to work with the discrete orthogonal basis $(f_1,\ldots,f_N)R_m^{-1}$. Such a basis is orthogonal with respect the counting measure supported at X, i.e. with respect to the discrete scalar product $\langle f,g\rangle_{X_m}=\sum_{i=1}^m f(P_i)g(P_i)$. Consequently, the original QMC moments have to be modified as in $(v4)$.

It should be stressed that, due to the inherited ill-conditioning of the triangular factor R_m by V_m, that increases with the degree, explicit inversion of R_m in $(v4)$ is avoided by solving linear systems via Gaussian elimination with row pivoting (that is in Matlab simply by applying the backslash operator).

We also notice that the complete matrix V_M is used only to compute the QMC moments in (iv), unless $(v7a)$ has to be followed due to a residual decrease factor below the required threshold. Such a phenomenon turns out to occur seldom with high degrees and strong ill-conditioning. In such a case, computation of $A_M=V_MR_m^{-1}$ becomes the computational bulk slowing down the whole process.

3 Numerical tests and demos

In order to show the effectiveness of the bottom-up compression procedure of QMC surface integration, we present some numerical tests, where we compare "Caratheodory-Tchakaloff" compression of multivariate discrete measures as implemented in the general-purpose package *dCATCH* [10], with the bottom-up approach described above. The Matlab codes and demos, collected in a package named *Qsurf*, are freely available at [15].

In all the tests we have set the parameters of the algorithm to $\varepsilon=10^{-10}$, $\theta=2$, $\tau=10$, and m has been initialized to $2N$. The tests have been performed with a CPU AMD Ryzen 5 3600 with 48 GB of RAM, running Matlab R2022a.

We point out that in all the numerical experiments we have adopted Halton points but alternatively the software provides the usage of other sequences (e.g. Sobol sets).

3.1 Sphere region

In the first example we consider a large region $\mathscr{J}$ of the sphere, namely a *spherical polygon* (a polygon whose vertices are on the sphere and whose sides are great circle arcs) representing an approximation of continental Africa (see Fig. 1). In this case it is convenient to choose a spherical cap (say $\mathscr{C} \supset \mathscr{J}$) centered at the polygon centroid as bounding surface, $\mathscr{S} = \mathscr{C}$, and we can apply a rotation to the sphere in such a way that the centroid is at the north pole (this does not clearly affect surface integration on the region).

The indicator function of $\mathscr{J}$ can be easily implemented by stereographic projection from the south pole on the tangent plane at the north pole, that generates a planar polygon for which the Matlab `inpolygon` works quite efficiently. Observe that this procedure can be applied to any rotated spherical polygon that does not contain the south pole.

Then, we can parametrize the polar cap by the area-preserving map (i.e., $\|\partial_u \Psi \times \partial_v \Psi\|_2 = 1$)

$$\Psi(u,v) = r(\sqrt{1-u^2}\cos(v), \sqrt{1-u^2}\sin(v), u)\ , \tag{13}$$

$(u,v) \in D = (c,1) \times (0,2\pi)$, where r is the sphere radius and c is the z-quote of the cap boundary (in practice, working with the open rectangle D we loose the Greenwich 0-meridian arc cutting the cap, that has null surface measure and thus surface integration is not affected).

Now, starting from low-discrepancy points in D, e.g. Halton points, we get low-discrepancy points on the cap $\mathscr{S}$ and finally on the spherical polygon $\mathscr{J}$. On the other hand, Proposition 1 substantially applies since the map Ψ is analytic and regular on D (see also Remarks 1-2), and hence we can resort to the bottom-up algorithm *Qsurf* in order to compress QMC integration on a huge number M of mapped low-discrepancy points in $\mathscr{J}$. To the purpose of illustration, in Figure 1 we show the distribution of 64 compressed QMC points extracted from about 2400 Halton points, still matching the QMC moments on $\mathscr{J}$, up to degree 7.

In Table 1 we report the results obtained by applying the QMC compression with more than one million points on the spherical polygon, taking degrees $n = 3,6,9,12,15$, and accepting (8) with a tolerance $\varepsilon = 10^{-10}$. In particular, we display the cardinalities and compression ratios, the cpu-times for the construction of the low-discrepancy sequence (cpu Halton seq.) and those for the computation of the compressed rules.

The advantage of the new approach is two-fold, since in all the tests an inferior cputime with respect to *dCATCH* is required to determine the compressed rule and, differently from *dCATCH*, the solution of (7) always satisfies the moment residual criterion (8). In addition, less memory is necessary due to the inherent structure of the bottom-up approach, which works on much smaller matrices.

Finally, in Table 2, we approximate the integrals $\int_{\mathscr{J}} g_k \, d\sigma$ on three test functions, namely setting $P = (x,y,z)$

$$g_1(P) = \exp(-\|P - P_0\|_2) \tag{14}$$
$$g_2(P) = \cos(x+y+z) \tag{15}$$
$$g_3(P) = \|P - P_0\|_2^5 \tag{16}$$

P_0 being the centroid of the spherical polygon $\mathscr{J}$. The reference values of the integrals have been computed by a QMC rule with very high cardinality (more than 20 million points). We display the relative errors of the QMC rule with more than one million points and of the two proposed compressions. As expected from estimate (12), by increasing the QMC moment-matching degree the errors tend to stabilize around the underlying QMC error.

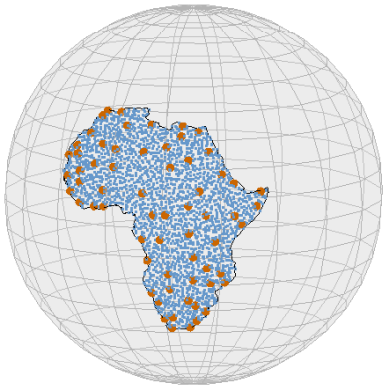


Figure 1: 64 compressed QMC points (red) at exactness degree $n = 7$, extracted from about 2400 mapped Halton points (blue) on the surface of a spherical polygon approximating continental Africa.

3.2 Torus region

The second example concerns surface integration on a region $\mathscr{J}$ of a torus $\mathscr{T}$, corresponding to a section by a plane, excluding the points that are internal to a ball intersecting the torus; see Fig. 2. In particular, we consider the torus with center $(0,0,0)$ and radii $r = 2$, $R = 3$, cut by the ball $B((0,4,0),\sqrt{6})$ and the upper half-space of $\mathbb{R}^3$ w.r.t. the plane of equation $-x/4+y+4z = 0$.

In this case it is not straightforward to apply a standard integrator, since one should track the domain $\Psi^{-1}(\mathscr{J})$ in D and then apply there a suitable cubature rule. On the contrary, QMC integration can be more easily constructed by rejection sampling in standard toroidal coordinates (here the bounding surface $\mathscr{S} = \mathscr{T}$ is the whole torus)

$$\Psi(u,v) = ((R+r\cos(u))\cos(v),(R+r\cos(u))\sin(v),r\sin(u)) \,, \tag{17}$$

$(u,v) \in D = (0,2\pi)\times(0,2\pi)$, where R and r are the big and small torus radii respectively, and $\|\partial_u\Psi\times\partial_v\Psi\|_2 = r(R+r\cos(v))$. Observe that considering the open rectangle Ω we loose the possible intersection of $\mathscr{J}$ with two circles, that have null surface measure and do not affect surface integration. Moreover, the indicator function of $\mathscr{J}$ can be implemented by the simple inequalities that describe an half-space determined by the cutting plane, and the interior of the ball. Again, the map Ψ is analytic and regular so that Proposition 1 with Remarks 1-2 applies and algorithm *Qsurf* can be used. In Figure 2 we show the distribution of 64 compressed QMC points, extracted from about 8000 mapped Halton points after selection by rejection sampling w.r.t. the surface measure density, still matching the QMC moments on $\mathscr{J}$ up to degree 7.

In Table 3 we again report the results obtained by applying QMC compression with more than one million points on the region $\mathscr{J}$. As for the spherical polygon, we consider degrees $n = 3,6,9,12,15$, accepting (8) with a tolerance $\varepsilon = 10^{-10}$. In all the tests an inferior cputime is required by *Qsurf* to determine the compressed rule and, while *dCATCH* fails for degree $n = 15$. Moreover, the solution by the new approach to (7) always satisfies the moment residual criterion (8).

deg	3	6	9	12	15
card. *QMC*	$M = 1{,}184{,}341$				
card. *dCATCH*	16	49	98	165	239
card. *Qsurf*	16	49	100	169	256
compr. ratio	7.4e+04	2.4e+04	1.1e+04	7.0e+03	4.6e+03
cpu Halton seq.	4.53e+01s				
cpu *dCATCH*	4.1e+00s	1.5e+01s	4.6e+01s	1.3e+02s	3.1e+02s
cpu *Qsurf*	3.8e-01	1.2e+00s	3.1e+00s	6.4e+00s	1.3e+01
speed-up	10.8	12.5	14.8	20.3	23.8
mom. resid. *dCATCH*	4.3e-12	4.3e-12	$\star$ 2.9e-04	$\star$ 6.2e-04	$\star$ 2.0e-03
mom. resid. *Qsurf*					
iter. 1	3.7e-16	8.5e-01	2.9e+01	7.9e+01	1.4e+01
iter. 2		6.5e-02	1.7e-04	3.8e-02	7.6e-01
iter. 3		6.3e-16	1.1e-15	1.3e-15	1.7e-02
iter. 4					2.8e-15

Table 1: QMC compression by with more than one million points on a spherical polygon approximating continental Africa.

deg	3	6	9	12	15
$E^{QMC}(g_1)$	3.0e-05				
$E^{dCATCH}(g_1)$	1.0e-03	3.1e-05	2.7e-05	3.2e-05	1.7e-05
$E^{Qsurf}(g_1)$	1.2e-04	3.0e-05	3.0e-05	3.0e-05	3.0e-05
$E^{QMC}(g_2)$	1.5e-05				
$E^{dCATCH}(g_2)$	8.9e-05	1.5e-05	2.5e-05	4.9e-07	3.7e-06
$E^{Qsurf}(g_2)$	1.4e-05	1.5e-05	1.5e-05	1.5e-05	1.5e-05
$E^{QMC}(g_3)$	8.6e-04				
$E^{dCATCH}(g_3)$	2.4e-02	1.3e-03	7.8e-04	8.2e-04	5.8e-04
$E^{Qsurf}(g_3)$	2.3e-02	7.7e-04	8.3e-04	8.6e-04	8.6e-04

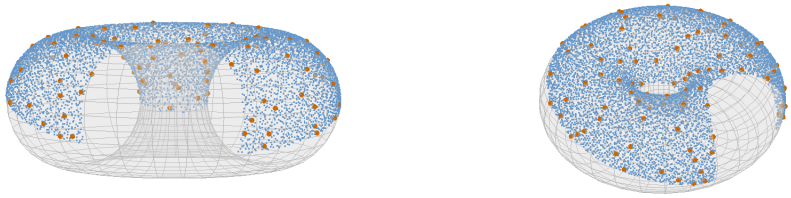
Table 2: Relative integration errors for the three test functions (14)-(16) on a spherical polygon approximating continental Africa, by means of QMC, *dCATCH* and *Qsurf*.

Lastly, in Table 4 we approximate the value of $\int_{\mathscr{S}} g_k \, d\sigma$, $k = 1,2,3$, with the same functions defined in (14)-(16) and $P_0 = (0,-3,2)$. The reference values of the integrals have been computed by means of a QMC rule with very high cardinality (more than 20 million points). We display the relative errors of the QMC rule with about one million points and of the two proposed compressions. Notice again that, as expected from estimate (12), by increasing the QMC moment-matching degree the errors tend to stabilize around the underlying QMC error.

3.3 Cartesian graph

In the third example we consider as a regular surface $\mathscr{S}$ the Cartesian graph of an analytic function, namely the popular Franke's surface, which is the graph of a linear combination of Gaussians

$$F(u,v) = \frac{3}{4} e^{-\frac{1}{4}((9u-2)^2+(9v-2)^2)} + \frac{3}{4} e^{-\frac{1}{49}((9u+1)^2+(9v+1)^2)} + \frac{1}{2} e^{-\frac{1}{4}((9u-7)^2+(9v-3)^2)} - \frac{1}{5} e^{-((9u-4)^2+(9v-7)^2)}, \tag{18}$$



(a) (b)

Figure 2: 100 compressed QMC points (red) at exactness degree $n = 7$, extracted from about 8000 mapped Halton points (blue) by rejection sampling on a torus region, determined by a cutting ball and plane (view from different perspectives).

deg	3	6	9	12	15
card. *QMC*	$M = 1{,}006{,}200$				
card. *dCATCH*	20	74	164	290	450
card. *Qsurf*	20	74	164	290	452
compr. ratio	5.0e+04	1.3e+04	6.1e+03	3.5e+03	2.2e+03
cpu Halton seq.	1.0e+01s				
cpu *dCATCH*	2.8e+00s	1.6e+01s	4.4e+01s	1.2e+02s	3.0e+02s
cpu *Qsurf*	2.7e-01s	9.9e-01s	2.9e+00s	6.3e+00s	2.2e+01s
speed-up	10.4	16.2	15.2	19.0	13.6
mom. resid. *dCATCH*	1.2e-11	1.2e-11	1.2e-11	1.2e-11	$\star$ 9.1e-07
mom. resid. *Qsurf*					
iter. 1	3.0e-16	8.9e-01	1.3e+00	6.4e+00	2.5e+01
iter. 2		1.1e-15	1.9e-15	2.6e-01	1.3e-01
iter. 3				3.3e-15	4.5e-15

Table 3: QMC compression with more than one million points on the torus region in Fig. 2.

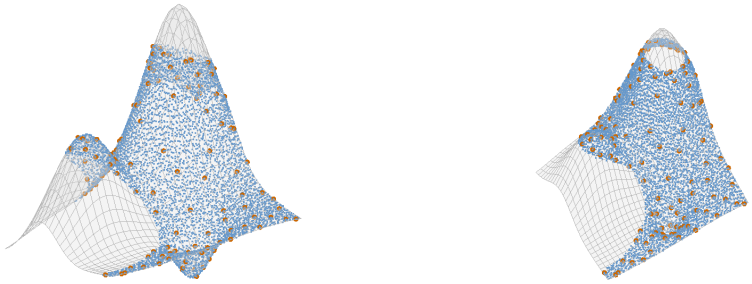
$(u,v) \in D = (0,1) \times (0,1)$.

We take two regions of such a surface, the first determined by a cutting ball and plane, whereas the second is a disconnected one determined by three cutting balls; see Figs. 3 and 4. Again, the map Ψ is analytic and regular, since $\|\partial_u \Psi \times \partial_v \Psi\|_2 = \sqrt{1 + (\partial_u F)^2 + (\partial_v F)^2}$, so that Proposition 1 with Remarks 1-2 applies and algorithm *Qsurf* can be used.

The numerical tests are collected in Tables 5-8, and show results that are in line with those of the previous examples, apart from the fact that the numerically determined dimension of the trivariate polynomial spaces does not collapse on the surface (at least up to degree 9). This is expected since Franke's surface is a transcendental, i.e. not algebraic, surface. Notice in particular that at degrees 9, 12, 15, *dCATCH* fails to reach the required residual tolerance, whereas *Qsurf* always succeeds in at most 4-5 iterations.

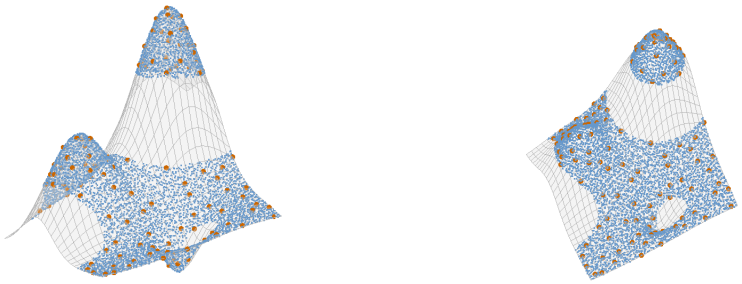
Remark 4. *In the numerical code there are some parameters ε, θ and τ that must be specified.*

The cardinality increase factor θ, becomes important when compression is not achieved, using the current subset of the QMC nodes. A small θ would propose a new pointset that may fail at the next stage because not dense enough, while a large value would instead define a too dense set, possibly increasing the algorithm cputime. Thus we decided as reasonable choice to



(a) (b)

Figure 3: 120 compressed QMC points (red) at exactness degree $n = 7$, extracted from about 6500 mapped Halton points (blue) by rejection sampling on a Franke's surface region, determined by a cutting ball and plane (view from different perspectives).



(a) (b)

Figure 4: 120 compressed QMC points (red) at exactness degree $n = 7$, extracted from about 6500 mapped Halton points (blue) by rejection sampling on a Franke's surface disconnected region, determined by three cutting balls (view from different perspectives).

deg	3	6	9	12	15
$E^{QMC}(g_1)$	1.7e-04				
$E^{dCATCH}(g_1)$	3.5e-01	1.2e-02	2.5e-03	2.2e-04	2.2e-04
$E^{Qsurf}(f_1)$	5.5e-01	6.5e-02	2.4e-03	5.4e-04	1.5e-04
$E^{QMC}(g_2)$	2.4e-04				
$E^{dCATCH}(g_2)$	3.5e-01	2.5e-01	7.2e-03	1.3e-04	2.4e-04
$E^{Qsurf}(f_2)$	1.7e+00	1.3e-01	1.5e-03	1.8e-04	2.4e-04
$E^{QMC}(g_3)$	5.2e-06				
$E^{dCATCH}(f_3)$	4.3e-03	2.3e-06	5.2e-06	5.2e-06	5.2e-06
$E^{Qsurf}(g_3)$	8.0e-03	3.1e-06	5.2e-06	5.2e-06	5.2e-06

Table 4: Relative errors for the three test functions (14)-(16) on the torus region of Fig. 2, by means of QMC, *dCATCH* and *Qsurf*.

deg	3	6	9	12	15
card. *QMC*	$M = 1{,}293{,}600$				
card. *dCATCH*	20	84	212	407	586
card. *Qsurf*	20	84	220	442	701
compr. ratio	6.5e+04	1.5e+04	5.9e+03	2.9e+03	1.8e+03
cpu Halton seq.	1.8e+00s				
cpu *dCATCH*	3.6e+00s	2.1e+01s	5.7e+01s	1.8e+02s	4.9e+02s
cpu *Qsurf*	3.5e-01s	1.4e+00s	3.8e+00s	2.0e+01s	2.7e+01s
speed-up	10.3	15.0	15.0	9.0	18.1
mom. resid. *dCATCH*	7.6e-12	7.6e-12	$\star$ 8.9e-04	$\star$ 2.9e-03	$\star$ 5.9e-03
mom. resid. *Qsurf*					
iter. 1	1.9e-16	5.5e-01	1.5e+00	1.4e+01	3.4e+01
iter. 2		1.0e-15	1.8e-15	3.9e-01	1.9e+00
iter. 3				1.2e-02	4.2e-15
iter. 4				2.5e-15	

Table 5: QMC compression with more than one million points on on the Franke's surface region in Fig. 3.

set $\theta = 2$, so doubling the cardinality of the pointset in case of failure.

To illustrate the effect of this choice, we consider the torus region in Figure 2, for degree of exactness 6 and different values of θ, that is $\theta = 1.2, 1.4, \ldots, 5$. In Figure 5 we take as ordinate the median of the cputime of the bottom-up QMC compression over 10 tests. In such experiment, values of θ in the interval $[2,3]$ show a good behavior, justifying our choice.

Concerning the relative moment matching tolerance ε, as default we used $\varepsilon = 10^{-10}$, in view of the typical approximation quality of the QMC approximation. However if the initial cardinality of the QMC rule is sufficiently large, there is numerical evidence that for mild degrees of precision the value of ε can be set smaller, e.g. $\varepsilon = 10^{-14}$.

Finally, recalling that the relative residual decrease threshold $\tau > 1$ is important to determine if the residual error made by successive stages of the algorithm, say res_{old} and res_{new}, is stagnating, we set as default $\tau = 10$, so detecting such an event when $res_{old} < 10\, res_{new}$. We decided for this option, since a much higher value would indicate stagnation even when it does not happen, while a much smaller value would rarely detect such a situation.

deg	3	6	9	12	15
card. *QMC*	$M = 1{,}305{,}444$				
card. *dCATCH*	20	84	212	405	612
card. *Qsurf*	20	84	220	448	735
compr. ratio	6.5e+04	1.6e+04	5.9e+03	2.9e+03	1.8e+03
cpu Halton seq.	1.74e+00s				
cpu *dCATCH*	3.8e+00 s	2.2e+01	5.2e+01	1.9e+02s	5.0e+02s
cpu *Qsurf*	3.6e-01	1.4e+00	9.0e+00	3.9e+01	6.3e+01
speed-up	10.6	15.7	5.8	4.9	7.9
mom. resid. *dCATCH*	1.8e-12	1.8e-12	$\star$ 8.2e-04	$\star$ 2.4e-03	$\star$ 3.4e-03
mom. resid. *Qsurf*					
iter. 1	6.0e-16	7.4e-02	7.7e-01	3.2e+01	3.4e+01
iter. 2		1.2e-15	2.4e-01	4.0e-01	6.1e+00
iter. 3			2.4e-01	8.5e-02	6.1e+00
iter. 4			1.6e-11	8.5e-02	9.2e-12
iter. 5				7.9e-12	

Table 6: QMC compression with more than one million points on the on the Franke's surface disconnected region in Fig. 4.

deg	3	6	9	12	15
$E^{QMC}(g_1)$	1.2e-05				
$E^{dCATCH}(g_1)$	4.4e-03	1.1e-05	1.5e-06	8.1e-06	4.6e-05
$E^{Qsurf}(g_1)$	6.7e-04	9.6e-06	1.2e-05	1.2e-05	1.2e-05
$E^{QMC}(g_2)$	3.0e-07				
$E^{dCATCH}(g_2)$	7.8e-05	2.9e-07	3.6e-05	1.2e-04	6.3e-05
$E^{Qsurf}(g_2)$	4.0e-05	3.0e-07	3.0e-07	3.0e-07	3.0e-07
$E^{QMC}(g_3)$	6.0e-05				
$E^{dCATCH}(g_3)$	1.4e-01	3.8e-05	1.1e-04	2.8e-05	1.2e-04
$E^{Qsurf}(g_3)$	2.2e-02	1.1e-04	6.1e-05	6.0e-05	6.0e-05

Table 7: Relative integration errors for the three test functions (14)-(16) on the Franke's surface region of Fig. 3, by means of QMC, *dCATCH*, *Qsurf*.

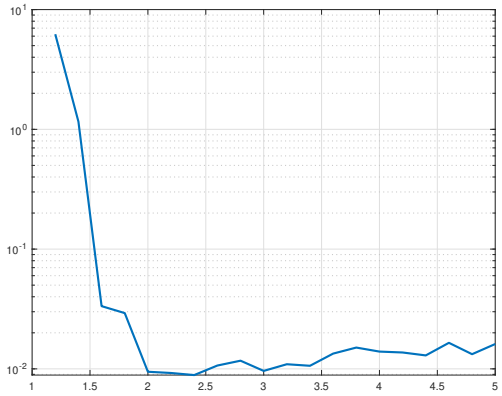


Figure 5: Median of the cputime of the bottom-up QMC compression with degree of exactness $n = 6$, over 10 tests, on a subregion of the torus, for several values of θ.

deg	3	6	9	12	15
$E^{QMC}(g_1)$	1.3e-06				
$E^{dCATCH}(g_1)$	4.6e-03	2.3e-06	1.7e-05	8.1e-06	2.1e-05
$E^{Qsurf}(g_1)$	2.1e-03	9.7e-07	1.3e-06	1.3e-06	1.3e-06
$E^{QMC}(g_2)$	6.6e-05				
$E^{dCATCH}(g_2)$	5.3e-05	6.6e-05	6.7e-05	9.1e-05	4.8e-05
$E^{Qsurf}(g_2)$	1.1e-04	6.6e-05	6.6e-05	6.6e-05	6.6e-05
$E^{QMC}(g_3)$	1.7e-06				
$E^{dCATCH}(g_3)$	2.3e-01	4.1e-05	1.6e-04	6.3e-05	8.1e-05
$E^{Qsurf}(g_3)$	1.1e-01	4.0e-05	1.3e-06	1.7e-06	1.7e-06

Table 8: Relative integration errors for the three test functions (14)-(16) on the Franke's surface disconnected region of Fig. 4, by means of QMC, *dCATCH*, *Qsurf*.

4 Software

We have implemented and tested in Matlab all the described routines.

The demos `demo_CQMC_sphpoly`, `demo_CQMC_torus`, `demo_CQMC_franke` illustrate the numerical experiments performed in the previous section. Their structure is essentially similar and can be modified to treat other subsets and/or parametric surfaces, adapting the function `pts_domain` to the new instance. This corresponds to items (*i*) and (*ii*) of *Algorithm Qsurf*.

The routine `cqmc_v2` implements its remaining items from (*iii*) to (*vi*). To this purpose, the basis selection in (*iii*) is obtained by means of the function `dCHEBVAND_v2`, while the computation of a sparse solution in (*v*5) is achieved by an user's choice implementation of the Lawson-Hanson algorithm (namely, the Matlab built-in function `lsqnonneg` or the alternative open-source codes `lawsonhanson` and `LHDM` proposed respectively in [24] and [9]). Moreover, having in mind to compare algorithm Qsurf with previous approaches, we also provide the routine `dCATCH` from [11], which implements Caratheodory-like compression via NNLS.

The open source software is available at [15].

Acknowledgements

The authors thank the referees for their careful reading of the manuscript and their numerous suggestions, which have improved this work. Work partially supported by the DOR funds and the biennial project BIRD 192932 of the University of Padova, and by the INdAM-GNCS 2022 Project "Methods and software for multivariate integral models". This research has been accomplished within the RITA "Research ITalian network on Approximation", the UMI Group TAA "Approximation Theory and Applications" (G. Elefante, A. Sommariva) and the SIMAI Activity Group ANA&A (A. Sommariva, M. Vianello).

Declaration of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] C. Bittante, S. De Marchi, and G. Elefante. *A new quasi-Monte Carlo technique based on nonnegative least squares and approximate Fekete points*. Numer. Math. Theory Methods Appl., vol. 9, 4, (2016), 640–663. ISSN 1004-8979,2079-7338. URL http://dx.doi.org/10.4208/nmtma.2016.m1516

[2] L. Brandolini, C. Choirat, L. Colzani, G. Gigante, R. Seri, and G. Travaglini. *Quadrature rules and distribution of points on manifolds*. Ann. Sc. Norm. Super. Pisa Cl. Sci. (5), vol. 13, 4, (2014), 889–923. ISSN 0391-173X,2036-2145. URL http://dx.doi.org/10.2422/2036-2145.201103_007

[3] J. S. Brauchart, E. B. Saff, I. H. Sloan, and R. S. Womersley. *QMC designs: optimal order quasi Monte Carlo integration schemes on the sphere*. Math. Comp., vol. 83, 290, (2014), 2821–2851. ISSN 0025-5718,1088-6842. URL http://dx.doi.org/10.1090/S0025-5718-2014-02839-1

[4] D. A. Cox, J. Little, and D. O'Shea. *Ideals, varieties, and algorithms*. Undergraduate Texts in Mathematics, (Springer, Cham2015), fourth edn. ISBN 978-3-319-16720-6; 978-3-319-16721-3. An introduction to computational algebraic geometry and commutative algebra, URL http://dx.doi.org/10.1007/978-3-319-16721-3

[5] P. J. Davis. *A construction of nonnegative approximate quadratures*. Math. Comp., vol. 21, (1967), 578–582. ISSN 0025-5718,1088-6842. URL http://dx.doi.org/10.2307/2005001

[6] S. De Marchi and G. Elefante. *Quasi-Monte Carlo integration on manifolds with mapped low-discrepancy points and greedy minimal Riesz s-energy points*. Appl. Numer. Math., vol. 127, (2018), 110–124. ISSN 0168-9274,1873-5460. URL http://dx.doi.org/10.1016/j.apnum.2017.12.017

[7] F. Dell'Accio, A. Sommariva, and M. Vianello. *Random sampling and unisolvent interpolation by almost everywhere analytic functions*. Appl. Math. Lett., vol. 145, (2023), Paper No. 108734, 6. ISSN 0893-9659,1873-5452. URL http://dx.doi.org/10.1016/j.aml.2023.108734

[8] M. Dessole, M. Dell'Orto, and F. Marcuzzi. *The Lawson-Hanson algorithm with deviation maximization: finite convergence and sparse recovery*. Numer. Linear Algebra Appl., vol. 30, 5, (2023), Paper No. e2490, 19. ISSN 1070-5325,1099-1506. URL http://dx.doi.org/10.1002/nla.2490

[9] M. Dessole, F. Marcuzzi, and M. Vianello. *Accelerating the Lawson-Hanson NNLS solver for large-scale Tchakaloff regression designs*. Dolomites Res. Notes Approx., vol. 13, 1, (2020), 20–29. ISSN 2035-6803. URL http://dx.doi.org/10.14658/PUPJ-DRNA-2020-1-3

[10] M. Dessole, F. Marcuzzi, and M. Vianello. *dCATCH: a numerical package for d-variate near G-optimal Tchakaloff regression via fast NNLS*. Mathematics, vol. 8, 7. ISSN 2227-7390. URL http://dx.doi.org/10.3390/math8071122

[11] M. Dessole, F. Marcuzzi, and M. Vianello. *dCATCH: numerical package for d-variate discrete measure compression, near-optimal design and polynomial fitting - v1.1*. https://www.math.unipd.it/~marcov/dCATCH.html (2020)

[12] J. Dick and F. Pillichshammer. *Digital nets and sequences*, (Cambridge University Press, Cambridge2010). ISBN 978-0-521-19159-3. Discrepancy theory and quasi-Monte Carlo integration, URL http://dx.doi.org/10.1017/CBO9780511761188

[13] G. Elefante, A. Sommariva, and M. Vianello. *CQMC: an improved code for low-dimensional compressed quasi-MonteCarlo cubature*. Dolomites Res. Notes Approx., vol. 15, 2, (2022), 92–100. ISSN 2035-6803. URL http://dx.doi.org/10.14658/PUPJ-DRNA-2022-2-8

[14] G. Elefante, A. Sommariva, and M. Vianello. *Compressed QMC volume and surface integration on union of balls* (2023). URL https://arxiv.org/abs/2303.01460

[15] G. Elefante, A. Sommariva, and M. Vianello. *Qsurf: a software package for compressed QMC integration on parametric surfaces (in Matlab)*. https://github.com/alvisesommariva/Qsurf (2023)

[16] S. Hayakawa. *Monte Carlo cubature construction*. Jpn. J. Ind. Appl. Math., vol. 38, 2, (2021), 561–577. ISSN 0916-7005,1868-937X. URL http://dx.doi.org/10.1007/s13160-020-00451-x

[17] C. L. Lawson and R. J. Hanson. *Solving least squares problems*, *Classics in Applied Mathematics*, vol. 15, (Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA1995). ISBN 0-89871-356-0. Revised reprint of the 1974 original, URL http://dx.doi.org/10.1137/1.9781611971217

[18] G. Legrain. *Non-negative moment fitting quadrature rules for fictitious domain methods*. Comput. Math. Appl., vol. 99, (2021), 270–291. ISSN 0898-1221,1873-7668. URL http://dx.doi.org/10.1016/j.camwa.2021.07.019

[19] C. Litterer and T. Lyons. *High order recombination and an application to cubature on Wiener space*. Ann. Appl. Probab., vol. 22, 4, (2012), 1301–1327. ISSN 1050-5164,2168-8737. URL http://dx.doi.org/10.1214/11-AAP786

[20] N. Nguyen and G. Ökten. *The acceptance-rejection method for low-discrepancy sequences*. Monte Carlo Methods Appl., vol. 22, 2, (2016), 133–148. ISSN 0929-9629,1569-3961. URL http://dx.doi.org/10.1515/mcma-2016-0104

[21] F. Piazzon, A. Sommariva, and M. Vianello. *Caratheodory-Tchakaloff subsampling*. Dolomites Res. Notes Approx., vol. 10, 1, (2017), 5–14. ISSN 2035-6803. URL http://dx.doi.org/10.14658/PUPJ-DRNA-2017-1-2

[22] W. Pleśniak. *Multivariate Jackson inequality*. J. Comput. Appl. Math., vol. 233, 3, (2009), 815–820. ISSN 0377-0427,1879-1778. URL http://dx.doi.org/10.1016/j.cam.2009.02.095

[23] D. L. Ragozin. *Constructive polynomial approximation on spheres and projective spaces*. Trans. Amer. Math. Soc., vol. 162, (1971), 157–170. ISSN 0002-9947,1088-6850. URL http://dx.doi.org/10.2307/1995746

[24] M. Slawski. *Non-negative least squares: comparison of algorithms*. https://sites.google.com/site/slawskimartin/code

[25] A. Sommariva and M. Vianello. *Compression of multivariate discrete measures and applications*. Numer. Funct. Anal. Optim., vol. 36, 9, (2015), 1198–1223. ISSN 0163-0563,1532-2467. URL http://dx.doi.org/10.1080/01630563.2015.1062394

[26] A. Sommariva and M. Vianello. *Computing Tchakaloff-like cubature rules on spline curvilinear polygons*. Dolomites Res. Notes Approx., vol. 14, 1, (2021), 1–11. ISSN 2035-6803. URL http://dx.doi.org/10.14658/PUPJ-DRNA-2021-1-1

[27] A. Sommariva and M. Vianello. *Low cardinality positive interior cubature on NURBS-shaped domains*. BIT, vol. 63, 2, (2023), [Paper No. 22], 20. ISSN 0006-3835,1572-9125. URL http://dx.doi.org/10.1007/s10543-023-00958-y

[28] V. Tchakaloff. *Formules de cubatures mécaniques à coefficients non négatifs*. Bull. Sci. Math. (2), vol. 81, (1957), 123–134. ISSN 0007-4497

[29] M. Tchernychova. *Caratheodory cubature measures*. Ph.D. thesis, University of Oxford (2015). URL https://ora.ox.ac.uk/objects/uuid:a3a10980-d35d-467b-b3c0-d10d2e491f2d

[30] D. R. Wilhelmsen. *A nearest point algorithm for convex polyhedral cones and applications to positive linear approximation*. Math. Comp., vol. 30, 133, (1976), 48–57. ISSN 0025-5718,1088-6842. URL http://dx.doi.org/10.2307/2005429

[31] H. Zhu and J. Dick. *Discrepancy bounds for deterministic acceptance-rejection samplers*. Electron. J. Stat., vol. 8, 1, (2014), 678–707. ISSN 1935-7524. URL http://dx.doi.org/10.1214/14-EJS898

J. Approx. Softw., Vol. 1, No. 1 (2024) 3

Journal of Approximation Software

NDED - Numerical derivatives from equispaced data

N. Egidi [1,⋆], J. Giacomini [1], P. Maponi [1], and M. Youssef[1]

[1] *Scuola di Scienze e Tecnologie, Università di Camerino*

Received: 28/04/2023 – Published: 23/07/2024
Communicated by: R. Cavoretto and A. De Rossi

Abstract

Procedure NDED computes the numerical derivatives of order ν from equispaced data. This is based on the iterated application of a spectral algorithm for the computation of the first order derivative. A preliminary test of the procedure gives satisfactory results.

Keywords: numerical derivative, integral equation, singular value expansion, FFT
(MSC2020: 45C05, 65-04, 65R20)

1 Introduction

A numerical differentiation problem consists in the computation of the derivative of order ν of an unknown function from the knowledge of the values of the function at prescribed points. Numerical differentiation is an interesting topic in many fields of applied sciences, such as biology, chemistry and physics, and it has a fundamental role in numerical analysis [3], [5], [21], [23]. For instance, operators approximating derivatives can be used to numerically solve differential equations [12], [13]. Due to its central role in scientific computing, several numerical differentiation methods are present in the scientific literature [4], [6], [7], [14], [15], [22], [24], [25]. All the methods for numerical differentiation are generally classified into these categories: finite difference methods, interpolation methods, regularization methods and integral methods. Finite difference methods and interpolation methods are well known and have the advantage of simplicity, moreover, they are considered to give satisfactory results when the function to be differentiated is given very precisely. Most of the regularization methods make use of the variational approach. The derivative is written as the solution of a Volterra integral equation and the resulting integral equation is reduced to a well-posed problem that depends on a regularization parameter. The main issue with these methods is the determination of the optimal parameter value that is generally a nontrivial task. Other interesting problems in the

⋆ Corresponding author: nadaniela.egidi@unicam.it

field of numerical differentiation are the differentiation of multivariate functions [2], [20] and the numerical differentiation from scattered data [8].

The main difficulty in the numerical differentiation is that small errors in the function data may cause large errors in the computed derivative, due to the unboundedness of the derivative operator. However, in practice, data are almost always corrupted by noise and in many applications it is necessary to estimate the derivative from known noisy data. Thus, proper regularization schemes are usually considered by methods for numerical differentiation [1], [16], [17], [18], [19], [26].

We present the procedure NDED for the numerical approximation of the derivatives of order $\nu \geq 1$. This procedure is based on a recursive application of an algorithm to compute first order derivatives from the Singular Value Expansion (SVE) of the derivative operator. In the present version, the procedure is intended for equispaced univariate data but the structure of the algorithm is easily generalizable to the cases of irregular grid spacing and multivariate data. The procedure NDED has been implemented in MATLAB and the "code metadata" are

Current code version	v. 1.0
Permanent link to repository	https://github.com/josgiac/NumDer.git
Code versioning system used	git
Software code languages	MATLAB

In Section 2, we summarize the theoretical basis of the proposed algorithm. In Section 3, we give the algorithm and its implementation in MATLAB. In Section 4, we show some numerical results. In Section 5, we provide conclusions and future developments.

2 Theoretical Background

The proposed algorithm is based on papers [9], [10] and [11], which we summarize in this section, for the reader's convenience.

We consider differentiable functions $f : I \to \mathbb{R}$ defined on a closed interval I; without losing generality, we can assume that $I = [0,1]$. The first derivative $f^{(1)}$ of f is the unique solution $w : I \to \mathbb{R}$ of

$$\int_0^1 K(t,s)w(s)ds = f(t) - f(0), \quad t \in [0,1], \tag{1}$$

which is a Volterra integral equation of first kind having kernel, $K : I \times I \to \mathbb{R}$,

$$K(t,s) = \begin{cases} 1, & 0 \leq s < t \leq 1, \\ 0, & 0 \leq t \leq s \leq 1. \end{cases} \tag{2}$$

We note that, with K defined as in (2), integral equation (1) is a direct consequence of the Fundamental Theorem of Calculus, that is, for each $t \in [0,1]$,

$$\int_0^t 1 \cdot f'(s)ds = f(t) - f(0).$$

In compact notation, equation (1) becomes

$$\mathscr{K} w = f - f_0, \tag{3}$$

where $f_0 = f(0)$ and $\mathscr{K}$ is the integral operator with kernel defined by (2). This integral operator $\mathscr{K}$ associated with the first order derivatives has a known SVE given by the following theorem.

Theorem 1. The SVE of kernel (2) is

$$K(t,s) = \sum_{k=0}^{\infty} \mu_k u_k(t) v_k(s), \quad t,s \in I, \tag{4}$$

where $\mu_k = \frac{2}{(2k+1)\pi}$, $k = 0,1,\ldots,$ are the singular values of $\mathscr{K}$ and the singular functions corresponding to μ_k are

$$v_k(s) = \sqrt{2}\cos\left(\frac{s}{\mu_k}\right), \qquad s \in I, \tag{5}$$

$$u_k(t) = \sqrt{2}\sin\left(\frac{t}{\mu_k}\right), \qquad t \in I. \tag{6}$$

Proof. See [11] for a detailed proof.□

The SVE of $\mathscr{K}$ allows the definition of an FFT method to compute the numerical derivatives of a given function starting from its values at prescribed points. Let $n > 0$ and $h = 1/n$, supposing that we know the values of f at $n+1$ equispaced points

$$\xi_j = jh, \ j = 0,1,\ldots,n, \tag{7}$$

that is $f_j = f(\xi_j)$ are known, then the following theorem gives such a method and the corresponding accuracy properties. This theorem considers the approximation of $f'(x_j)$, $j = 0,1,\ldots,n-1$ where

$$x_j = \left(j + \frac{1}{2}\right)h. \tag{8}$$

Moreover, the following notations are used: for $j,k = 0,1,\ldots,n-1$:

$$\gamma_k = \frac{1}{\mu_k}, \qquad \tilde{s}_{j,k} = \sin\left(\gamma_j \xi_k\right) \tag{9}$$

$$c_{j,k} = \cos\left(\gamma_j x_k\right), \qquad s_{j,k} = \sin\left(\gamma_j x_k\right). \tag{10}$$

Theorem 2. For $k = 0,1,\ldots,n-1$,

$$f'(x_k) = f_k^p + O(h^4), \tag{11}$$

where

$$f_k^p = \sqrt{2}\sum_{j=0}^{n-1} f_{v,j}^p c_{j,k}, \tag{12}$$

and for $j = 0,1,\ldots,n-1$,

$$f_{v,j}^p = \alpha_0 c_{j,0} + \beta_j(27 s_{j,0} - s_{j,1}) + \alpha_n c_{j,n}, \tag{13}$$

$$\alpha_0 = \frac{\sqrt{2}}{24}(2f_0 - 5f_1 + 4f_2 - f_3), \tag{14}$$

$$\alpha_n = \frac{\sqrt{2}}{24}(-f_{n-3} + 4f_{n-2} - 7f_{n-1} + 4f_n), \tag{15}$$

$$\beta_j = \frac{\sqrt{2}}{24}\left(2\sum_{l=1}^{n-1}(f_l - f_0)\tilde{s}_{j,l} + (-1)^j(f_n - f_0)\right). \tag{16}$$

Proof. See [10] for details.□

We note that (12) can be computed by the Discrete Fourier Transformation (DFT) of a vector that depends on $f^p_{v,j}$, $j = 0,1,\ldots,n-1$. Moreover, $f^p_{v,j}$, $j = 0,1,\ldots,n-1$, depend only on the data f_k, $k = 0,1,\ldots,n$, and (16) can be computed by the DFT. In particular, by using Theorem 2 and the FFT algorithm, in [10] we give two algorithms *FOD* and *NOD*. Algorithm *FOD* allows calculating the numerical derivative of order 1 by knowing the values of the function in equally spaced points of a closed interval $[a,b]$, while *NOD* computes the numerical derivative of order $\nu \geq 1$ by using *FOD* iteratively.

In the next section, we propose a revised version of FOD algorithm, that in some cases provides more accurate results than the original one.

3 The algorithm

We propose a new algorithm for numerical differentiation based on the following formulas. The algorithm has been coded in MATLAB, the current code version (v. 1.0) is available at https://github.com/josgiac/NumDer.git where the git code versioning system is used.

Let $f : I \to \mathbb{R}$ be a sufficiently regular function, by following the proof of Theorem 3.2 in [10], we can prove that, for $k = 0,1,\ldots,n-1$,

$$f'(x_k) = \tilde{f}^p_k + O(h^4), \tag{17}$$

where

$$\tilde{f}^p_k = \sqrt{2}\sum_{l=0}^{n-1} \tilde{f}^p_{v,l} c_{l,k}, \tag{18}$$

and for $j = 0,1,\ldots,n-1$,

$$\tilde{f}^p_{v,j} = \tilde{\alpha}_0 c_{j,0} + \beta_j(27 s_{j,0} - s_{j,1}) + \tilde{\alpha}_n c_{j,n}, \tag{19}$$

$$\tilde{\alpha}_0 = \frac{\sqrt{2}}{1920}(311 f_0 - 1075 f_1 + 1510 f_2 - 1110 f_3 + 435 f_4 - 71 f_5), \tag{20}$$

$$\tilde{\alpha}_n = \frac{\sqrt{2}}{1920}(471 f_n - 1235 f_{n-1} + 1510 f_{n-2} - 1110 f_{n-3} + \\ + 435 f_{n-4} - 71 f_{n-5}), \tag{21}$$

where we recall that $\beta_j, j = 0,1,\ldots,n-1$, are given by (16). We consider the following more general problem where the sampled function F is defined on a closed interval J not necessarily equal to I, that is the domain of f. We suppose that $\nu, n \in \mathbb{N}$, $\nu \geq 1$ and $n \geq \nu + 2$ and that $F : J \to \mathbb{R}$, with $J = [a,b] \subset \mathbb{R}$ and $a < b$, is a sufficiently regular function on the closed interval J of $\mathbb{R}$. Let $L = b - a$, $h = 1/n$ and $m = n - \nu + 1$, then for $i = 1,2,\ldots,\nu$, we consider the following points in J:

$$x^{(i)}_k = a + hL\left(k + \frac{i}{2}\right), \quad k = 0,1,\ldots,n-i, \tag{22}$$

$$\xi^{(i)}_j = a + hL\left(j + \frac{i-1}{2}\right), \quad j = 0,1,\ldots,n-i+1. \tag{23}$$

We note that $x^{(1)}_k = a + x_k L$ for $k = 0,1,\ldots,n-1$ and $\xi^{(1)}_j = a + \xi_j L$ for $j = 0,1,\ldots,n$.

Suppose that we know the values of F at the $n+1$ uniformly distributed points $\xi_j^{(1)}$, $j = 0,1,\ldots,n$, of J, the corresponding function data are $\underline{f} = (f_0, f_1, \ldots, f_n) \in \mathbb{R}^{n+1,}$ where

$$f_j = F(a + \xi_j L), \quad j = 0,1,\ldots,n. \tag{24}$$

We note that the vector of samples $\underline{f}$ may be considered as obtained from the function $f(t) = F((b-a)t+a))$ defined for $t \in I$, moreover $f'(t) = (b-a)F'((b-a)t+a))$. The proposed Algorithm 1, for $k = 0,1,\ldots,m-1$, computes the approximation $D_k^{(\nu)}$ of $F^{(\nu)}\left(x_k^{(\nu)}\right)$.

Algorithm 1 (ν-order derivative) **NDED**$\left(a,b,n,\nu,\underline{f};\underline{D}^{(\nu)}\right)$
Input: $a,b \in \mathbb{R}$; $n,\nu \in \mathbb{N}$; $\underline{f} = (f_0, f_1, \ldots, f_n) \in \mathbb{R}^{n+1}$.
Output: $\underline{D}^{(\nu)} = (D_0, D_1, \ldots, D_{m-1}) \in \mathbb{R}^m$, $m = n - \nu + 1$.

for $l = 0,\ldots,n-1$ **do**
  Compute the quantity $\tilde{f}_{\nu,l}^p$ by using formula (19)
end for
for $k = 0,\ldots,n-1$ **do**
  Compute $\tilde{f}_k^p$ by using formula (18)
  Compute $D_k^{(1)} = \frac{\tilde{f}_k^p}{b-a}$
end for
for $l = 2,3,\ldots,\nu$ **do**
  $m = n - l + 1$;
  Compute $\underline{D}^{(l)} \in \mathbb{R}^m$ by NDED$\left(\xi_0^{(l)}, \xi_m^{(l)}, m, 1, \underline{D}^{(l-1)}; \underline{D}^{(l)}\right)$
end for
return $\underline{D}^{(\nu)}$

We note that the FFT algorithm is used for computing formulae (19) and (18).

3.1 MATLAB implementation

Algorithm 1 has been implemented in MATLAB, here we illustrate the corresponding function.

- **Syntax.** $[d, ifail] = NumDerEquispacedData(a,b,nu,f)$
- **Purpose.** Compute the derivatives of a function F starting from its values at uniformly distributed points.
- **Description.** $[d, ifail] = NumDerEquispacedData(a,b,nu,f)$, given a vector f containing the $n+1$ values of function F at

$$\xi_k^{(1)} = a + k\frac{b-a}{n}, \quad k = 0,1,\ldots,n,$$

computes $d = (d_1, d_2, \ldots, d_m)$ the derivatives of order $\nu = nu$ of F at

$$x_k^{(nu)} = a + \left(k + \frac{nu}{2}\right)\frac{b-a}{n}, \quad k = 0,1,\ldots,m-1, \quad m = n - nu + 1,$$

with Algorithm 1.

- **Parameters.**
  - **input** a, b - double scalar. The closed interval $[a,b]$ is the domain of F. *Constraints:* $a < b$.
  - **input** *nu* - integer scalar. The value of *nu* is the order of the searched derivatives. *Constraints:* $nu \geq 1$.
  - **input** f - double vector with $n+1$ components. $f(j+1)$ must contain the quantity $F(a+j(b-a)/n)$, $j = 0,1,\ldots,n$. *Constraints:* $n \geq nu+2$.
  - **output** d - double vector with $m = n-nu+1$ components. $d(j+1)$ contains the approximation of

$$F^{(nu)}\left(a+\left(j+\frac{nu}{2}\right)\frac{b-a}{n}\right), \quad j = 0,1,\ldots,m-1.$$

  - **output** *ifail* – integer scalar, $ifail = 0$ unless the function detects an error (see Error Indicators and Warnings).
- **Error Indicators and Warnings.** Here is the list of errors or warnings detected by the function:
  - $ifail = 1$ - on entry $a \geq b$ or $nu \leq 0$.
  - $ifail = 2$ - the method cannot be applied because $n < nu+2$.

4 Numerical results

The performance of the proposed algorithm is tested against the following three functions:

- $F_1(x) = \frac{1}{1+x^2}, \quad x \in [0,1]$,
- $F_2(x) = \cos\left((1+x)^2\right), \quad x \in [0,1]$,
- $F_3(x) = e^x, \quad x \in [-0.1, 0.5]$.

We note that the first two functions are the same test functions chosen in [10] and are used for the comparison of the two algorithm versions. Let

- $f_k^{(\nu)}$ be the ν-derivative of F at $x_k^{(\nu)}$, $k = 0,1,\ldots,n-\nu$,
- $\hat{f}_k^{(\nu)}$ be a computed approximation of $f_k^{(\nu)}$, $k = 0,1,\ldots,n-\nu$,

We consider the following performance indices:

$$e_f = \left|f_0^{(1)} - \hat{f}_0^{(1)}\right|, \quad e_l = \left|f_{n-1}^{(1)} - \hat{f}_{n-1}^{(1)}\right|, \tag{25}$$

$$E_\infty = \max_{0\leq k\leq n-\nu}\left|f_k^{(\nu)} - \hat{f}_k^{(\nu)}\right|, \tag{26}$$

$$E_\infty^I = \max_{1\leq k\leq n-\nu-1}\left|f_k^{(\nu)} - \hat{f}_k^{(\nu)}\right|, \tag{27}$$

$$E_r = \sqrt{\frac{\sum_{k=0}^{n-\nu}\left(f_k^{(\nu)} - \hat{f}_k^{(\nu)}\right)^2}{\sum_{k=0}^{n-\nu}\left(f_k^{(\nu)}\right)^2}}. \tag{28}$$

The numerical results related to algorithm NDED have been obtained by using the MATLAB *Script* **Test** that uses **NumDerEquispacedData**.

The results of this test are reported in Tables 1-4 and Fig. 1. In particular, in Tables 1 and 2, we can see that, for both functions F_1 and F_2, the values of e_f and e_l obtained by NDED are significantly lower than those obtained with NOD. This shows that new formulae (19)-(21) actually give an improved approximation at the extremes of the computation interval with respect to the formula implemented in NOD, without changing the performance at the internal points, indeed, the errors E^I_∞ computed with NDED are the same of those computed with NOD.

Table 1: The comparison of the absolute errors e_f and e_l for the first derivative of function F_1 obtained with NOD and NDED, where $x(z)$ denotes the real number $x \cdot 10^z$.

h	*NOD*			*NDED*		
	e_f	e_l	E^I_∞	e_f	e_l	E^I_∞
4.00(−2)	6.18(−5)	9.92(−6)	1.20(−6)	1.90(−6)	1.27(−7)	1.20(−6)
2.00(−2)	7.93(−6)	1.12(−6)	7.53(−8)	7.04(−8)	4.50(−9)	7.53(−8)
1.00(−2)	9.98(−7)	1.32(−7)	4.71(−9)	2.29(−9)	1.45(−10)	4.71(−9)

Table 2: The comparison of the absolute errors e_f and e_l for the first derivative of function F_2 obtained with NOD and NDED, where $x(z)$ denotes the real number $x \cdot 10^z$.

h	*NOD*			*NDED*		
	e_f	e_l	E^I_∞	e_f	e_l	E^I_∞
4.00(−2)	1.33(−4)	7.66(−4)	1.07(−5)	7.38(−7)	1.20(−5)	1.07(−5)
2.00(−2)	1.54(−5)	9.92(−5)	6.69(−7)	7.32(−9)	5.23(−7)	6.69(−7)
1.00(−2)	1.84(−6)	1.26(−5)	4.18(−8)	1.93(−11)	1.87(−8)	4.18(−8)

Table 3 and Table 4 report the errors obtained by NDED, in particular, they show the errors in the numerical derivatives of order $\nu \geq 1$ with $h = 1/100$, but similar behaviors are obtained for a different choice of h. From these tables, we can see that the precision decreases as the order of the derivative increases, which suggests that the algorithm needs a more in-depth study to limit, as far as possible, this natural behavior.

Table 3: The errors obtained by computing with NDED the derivatives of order $\nu = 1,2,3$ of functions F_1 and F_2 with step $h = 1/100$, where $x(z)$ denotes the real number $x \cdot 10^z$.

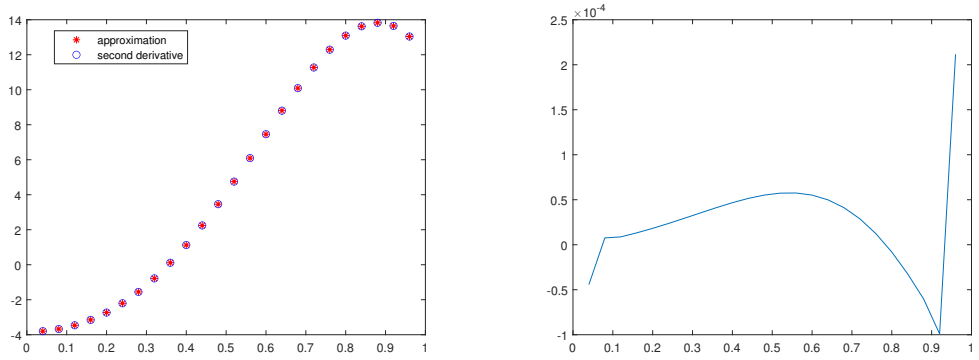
ν	F_1		F_2	
	E_∞	E_r	E_∞	E_r
1	4.71(−9)	4.67(−9)	4.18(−8)	1.20(−8)
2	1.57(−7)	3.16(−8)	6.56(−7)	2.53(−8)
3	2.00(−5)	7.03(−7)	7.81(−5)	4.56(−7)

Table 4: The errors obtained by computing with NDED the derivative of order ν of function F_3 with step $h = 1/100$, where $x(z)$ denotes the real number $x \cdot 10^z$.

ν	1	2	3	4	5
E_∞	8.71(−12)	1.77(−9)	2.69(−7)	4.19(−5)	6.80(−3)
E_r	5.58(−12)	1.56(−10)	2.43(−8)	4.16(−6)	9.05(−4)

Finally, in Figure 1 we have the graph of $F_2^{(2)}\left(x_k^{(2)}\right), k = 0,1,\ldots,n-2$, its approximation $D_k^{(2)}, k = 0,1,\ldots,n-2$, and the corresponding error $E_k = D_k^{(2)} - F_2^{(2)}\left(x_k^{(2)}\right), k = 0,1,\ldots,n-2$, when the approximation is computed with NDED and $h = 1/25$. Figure 1 gives graphical evidence of the accuracy of the derivative approximation obtained by NDED, even with few data points.

Figure 1: On the left, the graph of the second derivative of F_2 and its approximation computed with NDED and $h = 1/25$. On the right, the corresponding error $E_k = D_k^{(2)} - F_2^{(2)}\left(x_k^{(2)}\right)$, $k = 0,1,\ldots,n-2$.



5 Conclusion

In the present paper, we described the procedure NDED for the numerical computation of the derivative of order ν, starting from function data obtained at $n+1$ uniformly distributed points on an interval $[a,b]$. This procedure is based on a recursive application of a numerical method to compute the first order derivative with an error $O(h^4)$. The procedure NDED has been implemented in MATLAB and the current code version is available on github. The current code version (v. 1.0) of the NDED procedure gives satisfactory results for equispaced univariate data. The next versions will be able to consider scattered function data and multivariate function data.

Declaration of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] R. S. Anderssen and P. Bloomfield. *Numerical differentiation procedures for non-exact data*. Numer. Math., vol. 22, (1973/74), 157–182. ISSN 0029-599X,0945-3245. URL http://dx.doi.org/10.1007/BF01436965

[2] R. S. Anderssen and M. Hegland. *For numerical differentiation, dimensionality can be a blessing!* Math. Comp., vol. 68, 227, (1999), 1121–1141. ISSN 0025-5718,1088-6842. URL http://dx.doi.org/10.1090/S0025-5718-99-01033-9

[3] F. S. V. Bazán and L. Bedin. *Filtered spectral differentiation method for numerical differentiation of periodic functions with application to heat flux estimation*. Comput. Appl. Math., vol. 38, 4, (2019), Paper No. 165, 23. ISSN 2238-3603,1807-0302. URL http://dx.doi.org/10.1007/s40314-019-0968-4

[4] B. Chen, Z. Zhao, Z. Li, and Z. Meng. *Numerical differentiation by a Fourier extension method with super-order regularization*. Appl. Math. Comput., vol. 334, (2018), 1–10. ISSN 0096-3003,1873-5649. URL http://dx.doi.org/10.1016/j.amc.2018.04.005

[5] J. Cheng, X. Z. Jia, and Y. B. Wang. *Numerical differentiation and its applications*. Inverse Probl. Sci. Eng., vol. 15, 4, (2007), 339–357. ISSN 1741-5977,1741-5985. URL http://dx.doi.org/10.1080/17415970600839093

[6] O. Davydov and R. Schaback. *Error bounds for kernel-based numerical differentiation*. Numer. Math., vol. 132, 2, (2016), 243–269. ISSN 0029-599X,0945-3245. URL http://dx.doi.org/10.1007/s00211-015-0722-9

[7] O. Davydov and R. Schaback. *Minimal numerical differentiation formulas*. Numer. Math., vol. 140, 3, (2018), 555–592. ISSN 0029-599X,0945-3245. URL http://dx.doi.org/10.1007/s00211-018-0973-3

[8] F. Dell'Accio, F. Di Tommaso, N. Siar, and M. Vianello. *Numerical differentiation on scattered data through multivariate polynomial interpolation*. BIT, vol. 62, 3, (2022), 773–801. ISSN 0006-3835,1572-9125. URL http://dx.doi.org/10.1007/s10543-021-00897-6

[9] N. Egidi, J. Giacomini, and P. Maponi. *A Fredholm integral operator for the differentiation problem*. Comput. Appl. Math., vol. 41, 5, (2022), Paper No. 220, 21. ISSN 2238-3603,1807-0302. URL http://dx.doi.org/10.1007/s40314-022-01923-1

[10] N. Egidi, J. Giacomini, P. Maponi, and M. Youssef. *An FFT method for the numerical differentiation*. Appl. Math. Comput., vol. 445, (2023), Paper No. 127856, 13. ISSN 0096-3003,1873-5649. URL http://dx.doi.org/10.1016/j.amc.2023.127856

[11] N. Egidi and P. Maponi. *The singular value expansion of the Volterra integral equation associated to a numerical differentiation problem*. J. Math. Anal. Appl., vol. 460, 2, (2018), 656–681. ISSN 0022-247X,1096-0813. URL http://dx.doi.org/10.1016/j.jmaa.2017.12.007

[12] N. Egidi and P. Maponi. *An SVE Approach for the Numerical Solution of Ordinary Differential Equations*. Y. D. Sergeyev and D. E. Kvasov (eds.), *Numerical Computations: Theory and Algorithms*, (Springer International Publishing, Cham2020). ISBN 978-3-030-39081-5, 70–85. URL http://dx.doi.org/10.1007/978-3-030-39081-5_8

[13] N. Egidi and P. Maponi. *A spectral method for the solution of boundary value problems*. Appl. Math. Comput., vol. 409, (2021), Paper No. 125812, 12. ISSN 0096-3003,1873-5649. URL http://dx.doi.org/10.1016/j.amc.2020.125812

[14] X. Huang, C. Wu, and J. Zhou. *Numerical differentiation by integration*. Math. Comp., vol. 83, 286, (2014), 789–807. ISSN 0025-5718,1088-6842. URL http://dx.doi.org/10.1090/S0025-5718-2013-02722-6

[15] I. Knowles and R. Wallace. *A variational method for numerical differentiation*. Numer. Math., vol. 70, 1, (1995), 91–110. ISSN 0029-599X,0945-3245. URL http://dx.doi.org/10.1007/s002110050111

[16] S. Lu and S. V. Pereverzev. *Numerical differentiation from a viewpoint of regularization theory*. Math. Comp., vol. 75, 256, (2006), 1853–1870. ISSN 0025-5718,1088-6842. URL http://dx.doi.org/10.1090/S0025-5718-06-01857-6

[17] Y. Luo. *Galerkin method with trigonometric basis on stable numerical differentiation*. Appl. Math. Comput., vol. 370, (2020), 124912, 26. ISSN 0096-3003,1873-5649. URL http://dx.doi.org/10.1016/j.amc.2019.124912

[18] A. G. Ramm and A. B. Smirnova. *On stable numerical differentiation*. Math. Comp., vol. 70, 235, (2001), 1131–1153. ISSN 0025-5718,1088-6842. URL http://dx.doi.org/10.1090/S0025-5718-01-01307-2

[19] T. J. Rivlin. *Optimally stable Lagrangian numerical differentiation*. SIAM J. Numer. Anal., vol. 12, 5, (1975), 712–725. ISSN 0036-1429. URL http://dx.doi.org/10.1137/0712053

[20] T. Tsuda. *Numerical differentiation of functions of very many variables*. Numer. Math., vol. 18, (1971/72), 327–335. ISSN 0029-599X,0945-3245. URL http://dx.doi.org/10.1007/BF01404683

[21] X. Q. Wan, Y. B. Wang, and M. Yamamoto. *Detection of irregular points by regularization in numerical differentiation and application to edge detection*. Inverse Problems, vol. 22, 3, (2006), 1089–1103. ISSN 0266-5611,1361-6420. URL http://dx.doi.org/10.1088/0266-5611/22/3/022

[22] Y. B. Wang, X. Z. Jia, and J. Cheng. *A numerical differentiation method and its application to reconstruction of discontinuity*. Inverse Problems, vol. 18, 6, (2002), 1461–1476. ISSN 0266-5611,1361-6420. URL http://dx.doi.org/10.1088/0266-5611/18/6/301

[23] B. Yin and Y. Ye. *Recovering the local volatility in Black-Scholes model by numerical differentiation*. Appl. Anal., vol. 85, 6-7, (2006), 681–692. ISSN 0003-6811,1563-504X. URL http://dx.doi.org/10.1080/00036810500475025

[24] Z. Zhao. *A Hermite extension method for numerical differentiation*. Appl. Numer. Math., vol. 159, (2021), 46–60. ISSN 0168-9274,1873-5460. URL http://dx.doi.org/10.1016/j.apnum.2020.08.016

[25] Z. Zhao, Z. Meng, and G. He. *A new approach to numerical differentiation*. J. Comput. Appl. Math., vol. 232, 2, (2009), 227–239. ISSN 0377-0427,1879-1778. URL http://dx.doi.org/10.1016/j.cam.2009.06.001

[26] Z. Zhao and L. You. *A numerical differentiation method based on legendre expansion with super order Tikhonov regularization*. Appl. Math. Comput., vol. 393, (2021), Paper No. 125811, 11. ISSN 0096-3003,1873-5649. URL http://dx.doi.org/10.1016/j.amc.2020.125811

J. Approx. Softw., Vol. 1, No. 1 (2024) 4

Journal of Approximation Software

A MATLAB implementation of TASE-RK methods

D. Conte [1], G. Pagano [1,⋆], and B. Paternoster [1]

[1] *Department of Mathematics, University of Salerno*

Received: 09/05/2023 – Published: 23/07/2024
Communicated by: R. Cavoretto and A. De Rossi

Abstract

In this paper, we analyze theoretical and implementation aspects of Time-Accurate and highly-Stable Explicit Runge-Kutta (TASE-RK) methods, which have been recently introduced by Bassenne et al. (2021) [5], for the numerical solution of stiff Initial Value Problems (IVPs). These methods are obtained by combining explicit RK schemes with suitable matrix operators, called TASE operators, involving in their expression a matrix J related to the Jacobian of the differential problem to be solved. By analyzing the formulation and order conditions of TASE-RK methods, we show that they can be interpreted as particular linearly implicit RK schemes, and that their consistency properties are independent of the choice of J. Using this information, we propose a MATLAB implementation of TASE-RK methods, which makes use of matrix factorizations and allows setting J according to user preferences.

Keywords: RK methods, TASE preconditioners, TASE-RK methods, MATLAB code, stiff problems (MSC2020: 65L04, 65L06, 65M06, 65Y99)

1 Introduction

In this manuscript, we focus on the numerical solution of IVPs of the form

$$\begin{cases} y'(t) = f(t, y(t)), \\ y(t_0) = y_0, \end{cases} \qquad t \in [t_0, t_e], \quad f : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}^d, \tag{1}$$

characterized by severe stiffness, usually arising from the spatial semi-discretization of Partial Differential Equations (PDEs) in several applications. Stiffness is a well-known characteristic of differential equations, and several definitions have been given over the years to formalize

⋆ Corresponding author: gpagano@unisa.it

this concept, see, e.g., [10, 11]. Roughly speaking, a system of differential equations is stiff when an explicit numerical method is forced to use very small discretization steps in order to furnish an accurate solution, thus becoming totally inefficient.

Research in the field of efficient methods for solving problems of this type has developed a lot over the years, and still continues. Indeed, most of the models of differential equations that derive from application in several contexts, e.g., corrosion [4, 15], biology [19], chemistry [8, 7], physics [3, 6, 20, 21], are characterized by severe stiffness. The first methods that have been proposed to deal with stiffness are the implicit ones, among which the most famous are the fully implicit RK schemes (e.g. the Gauss-Legendre and RADAU formulas [11, 28, 29]). Fully implicit methods manage to be particularly stable with quite large values of the discretization step. However, fully implicit RK methods are particularly expensive since they require the solution of systems of non-linear equations (of the size of the problem times the number of stages) at each time step, and this constitutes an obstacle especially for problems of large dimensions, such as semi-discretized PDEs. For this reason, several implementation procedures have been proposed in the scientific literature to optimize the efficiency of implicit methods by reducing the number of required operations or the size of the underlying non-linear system, see, e.g., [26]. To reduce the complexity of implicit methods, alternative RK schemes have been formulated, such as DIRK (Diagonally Implicit RK) (see, e.g., [27] and references therein contained), or IMEX (IMplicit EXplicit) (see, e.g. [14, 13] and references therein contained). Furthermore, particularly efficient and stable methods for stiff problems are the so-called linearly implicit RK schemes, which arise for example from a linearization of DIRK. Such methods require the solution of a fixed number of linear systems at each step.

The most famous linearly implicit RK schemes are the Rosenbrock and W-methods, see, e.g., [23, 25, 24, 34]. Moreover, recently Bassenne et al. [5] proposed a new class of RK methods, called TASE-RK methods. These numerical schemes have been subsequently improved by Calvo et al. [12]. As pointed out in [5, Introduction], the idea of the former is based on the fact that a user is not a-priori aware of the severity of the stiffness of a differential problem. Thus, a convenient approach may be to always start using an explicit RK method, which is very simple and fast to program. Then, if the numerical results are not good, to avoid using another code and reprogramming a new method, the user can keep the one already applied by pre-conditioning the problem to solve. In this way the stiffness of the problem is moderated and therefore the probability that the explicit RK method works well increases. TASE-RK methods are very interesting and promising, as shown by the large number of scientific articles that have been produced based on them [2, 18, 17, 22, 30, 33, 36]. Some of these manuscripts show that TASE operators are very efficient also when used with other classes of numerical schemes, such as peer methods [1, 16, 18, 31, 32, 35].

In this manuscript, we focus on the implementation aspects of TASE-RK methods. In particular, by analyzing their formulation, we express them in such a way that their implementation and also the study of the related properties of accuracy and stability simplify. Indeed, we revise TASE-RK methods as linearly implicit numerical schemes, and compute an efficient solution of the underlying linear systems. These systems involve matrices that depend on the Jacobian $J_f = f_y(t, y(t))$ of the differential problem. However, since the consistency analysis shows that the TASE-RK methods preserve their order regardless of the choice of these matrices, we propose an implementation that allows the user to fix them in a convenient way. Finally, we show two examples of use of the proposed MATLAB code.

Summarizing, this paper is organized as follows: in Section 2 we recall the original TASE-RK methods and formulate them as linearly implicit RK methods; in Section 3 we discuss the related properties of accuracy and stability; in Section 4 we show and explain the improvement

of TASE-RK methods performed by Calvo et al.; in Section 5 we propose and describe a MATLAB function for implementing TASE-RK methods; in Section 6 we show an example of use through two numerical tests concerning a system of Ordinary Differential Equations (ODEs) and a semi-discretized PDE; finally, some conclusions are drawn in Section 7.

2 Formulation

Let us fix, from now on, the discrete grid $\{t_n = t_0 + nh; n = 0, \dots, N; t_N = t_e\}$, $h > 0$. The idea of derivation of the TASE-RK methods arises from the following observations.

First, consider the implicit Euler method for solving the problem $y'(t) = Jy(t)$, with J representing a generic matrix of order d:

$$y_{n+1} = y_n + hJy_{n+1}. \tag{2}$$

Note that the method (2) can be rewritten as $(I_d - hJ)y_{n+1} = y_n$, where I_d indicates the Identity matrix of order d. Assuming the matrix $I_d - hJ$ to be invertible, we get

$$y_{n+1} = (I_d - hJ)^{-1} y_n \iff y_{n+1} = y_n + ((I_d - hJ)^{-1} - I_d)y_n.$$

Furthermore, using that

$$(I_d - hJ)^{-1} - I_d = (I_d - hJ)^{-1}[I_d - (I_d - hJ)] = (I_d - hJ)^{-1} hJ,$$

we can finally write

$$y_{n+1} = y_n + hT_1(hJ)Jy_n, \qquad \text{with } T_1(hJ) = (I_d - hJ)^{-1}. \tag{3}$$

Note that the numerical scheme (3) corresponds to the explicit Euler method applied to the problem $y'(t) = T_1(hJ)Jy(t)$. Therefore, solving

$$y'(t) = Jy(t), \qquad J \in \mathbb{R}^{d,d}, \tag{4}$$

with implicit Euler method is equivalent to solving

$$y'(t) = T_1(hJ)Jy(t), \qquad T_1 : \mathbb{R}^{d,d} \to \mathbb{R}^{d,d}, \quad T_1(A) := (I_d - A)^{-1} \in \mathbb{R}^{d,d}, \tag{5}$$

using explicit Euler, which in principle has bad stability properties. However, the preconditioning made to the vector field of the problem (4) by means of the matrix T_1 definitely improves the stability of explicit Euler method, since we get implicit Euler.

Using the above observations, Bassenne et al. have proposed a general setting for constructing new RK schemes with s stages and order $p = s$ suitable for stiff problems. The general approach consists of the following two main steps.

First, the differential problem (1) is modified as follows:

$$\begin{cases} u'(t) = T_p(hJ_f) f(t, u(t)), & T_p : \mathbb{R}^{d,d} \to \mathbb{R}^{d,d}, \\ u(t_0) = y_0. \end{cases} \tag{6}$$

Here, T_p is a matrix operator which depends on the product between the step-size h and the Jacobian $J_f = f_y(t, y(t))$ of the problem, where in J_f for simplicity of notation we do not indicate

the time dependence. The only property required for T_p is that it must be an approximation of order p of the Identity. This means that

$$T_p(hJ_f) = I_d + O(h^p). \tag{7}$$

Subsequently, the perturbed problem (6) is solved through an explicit RK method of order p.

Therefore, more conveniently, we can express the TASE-RK methods directly in the following way:

$$\begin{cases} Y_{n,i} = y_n + h\sum_{j=1}^{i-1} a_{ij} T_p(hJ_n) f(t_n + c_j h, Y_{n,j}), & i = 1,\dots,s, \\ y_{n+1} = y_n + h\sum_{j=1}^{s} b_j T_p(hJ_n) f(t_n + c_j h, Y_{n,j}). \end{cases} \tag{8}$$

Here, $Y_{n,j} \approx y(t_n + c_j h)$, $y_n \approx y(t_n)$, and $A = (a_{ij})$, $b = (b_j)$, $c = (c_j)$ represent the coefficients of the underlying explicit RK method. Note that, in the discrete setting, the exact Jacobian J_f is replaced by $J_n = f_y(t_n, y_n)$. Hence, TASE-RK methods require the Jacobian to be updated at each integration time step.

The TASE operator T_p proposed by Bassenne et al. is a natural extension of the function T_1 in Equation (5). In particular, adding in T_1 the dependency on a real positive parameter α, the TASE operator of order one reads

$$T_1(\alpha, hJ_f) = (I_d - \alpha hJ_f)^{-1}, \qquad \alpha > 0.$$

Using Richardson extrapolation, Bassenne et al. have then recursively defined a generic family of TASE operators of order p, as follows:

$$T_p(\alpha, hJ_f) = \begin{cases} (I_d - \alpha hJ_f)^{-1}, & \text{if } p = 1, \\ \dfrac{2^{p-1}}{2^{p-1}-1} T_{p-1}(\alpha/2, hJ_f) - \dfrac{1}{2^{p-1}-1} T_{p-1}(\alpha, hJ_f), & \text{if } p \geq 2. \end{cases} \tag{9}$$

The TASE operator T_p (9) can also be expressed as

$$T_p(\alpha, hJ_f) = \sum_{k=0}^{p-1} \beta_{p,k} (2^k - \alpha hJ_f)^{-1}, \tag{10}$$

where the coefficients $\beta_{p,k}$ must be suitably fixed (see [5, Table 2]).

From the formulation (8), and from the expression of the TASE operator (10), it is clear that TASE-RK methods are linearly implicit numerical schemes. In particular, these methods require the Jacobian to be updated at each step, and the solution of p linear systems depending on J_n for each stage $Y_{n,i}$, $i = 2,\dots,s$ (since $Y_{n,1} = y_n$, we do not take into account the first stage). Furthermore, there are p extra linear systems concerning the computation of the advancing solution. Thus, TASE-RK methods require the solution of sp linear systems at each step.

3 Consistency and stability analysis

In this section, we analyze the consistency and stability of TASE-RK methods.

We start by showing below that a TASE-RK scheme has the same order as the underlying explicit RK method.

Theorem 3.1. *[5, 12] Let us consider an explicit RK method of order p, and assume that the TASE operator T_p satisfies the property (7). Then, the corresponding TASE-RK method (8) is consistent of order p.*

This theorem is quite natural by observing that the exact solutions $y(t)$, $u(t)$, of the original and perturbed problems (1), (6), respectively, satisfy $||y(t)-u(t)|| = O(h^p)$, thanks to property (7). Using an explicit RK method of order p for the perturbed problem, we obtain that $||u(t_n) - u_n|| = O(h^p)$, where $u_n \approx u(t_n)$, for each n. With u_n we here denote the numerical solution of the perturbed problem through the explicit RK method, i.e. the solution of the TASE-RK method. Therefore, it holds that $||y(t_n) - u_n|| = O(h^p)$, i.e. the TASE-RK method has order p.

Several interesting observations can be made starting from Theorem 3.1.

We first write the Taylor series expansion of the TASE operator T_p (9) proposed by Bassenne et al. as follows:

$$T_p(\alpha, hJ_f) = I_d + Q_p(hJ_f)^p + O(h^{p+1}).$$

By making simple calculations, it can be shown that $Q_p = \alpha^p/c$, where c is a known positive constant whose value depends on p.

Remark 3.1. The smaller $|Q_p|$ is, the more T_p approximates the Identity matrix in an accurate way, and therefore the perturbed problem (6) gets closer to the initial one (1). Indeed, in the manuscripts [5, 12] it is observed that the smaller $|Q_p|$ is, the lower the error provided by the TASE-RK methods is.

Remark 3.2. In the manuscripts [5, 12], TASE-RK methods of order $p = s(\leq 4)$ have been derived. This choice allows to attain the maximum possible order using the minimum number of stages and simplifies the study of linear stability, as discussed below.

Now, we analyze the stability properties of TASE-RK methods. It is known that the stability function of explicit s-stage RK methods with order $p = s \leq 4$ is independent of their coefficients. In particular, it can be expressed as follows:

$$R(z) = 1 + z + \cdots + \frac{1}{p!}z^p.$$

Here, $z = h\lambda$, where λ is a complex parameter with $\mathrm{Re}(\lambda) < 0$ associated with the classical test equation $y'(t) = \lambda y(t)$. The stability function of TASE-RK methods can be easily derived from it. Indeed, considering the perturbed problem $y'(t) = T_p(\alpha, h\lambda)\lambda y(t)$, we get the following stability function:

$$RT_p(\alpha, z) = 1 + zT_p(\alpha, z) + \cdots + \frac{1}{p!}(zT_p(\alpha, z))^p. \tag{11}$$

Note that for TASE-RK methods with order $p = s$ the stability function is independent of the coefficients of the underlying explicit RK scheme. The only parameter on which the stability function depends is that of the TASE operator α. Therefore, adequately fixing the free parameter α of T_p, using TASE technique it is possible to improve the stability properties of any explicit RK method with order $p = s \leq 4$.

Remark 3.3. Property (7) holds for the operator T_p (9) for any value of the parameter α and matrix J_f. Thus, according to Remark 3.1, the free parameter α can be set to minimize $|Q_p|$, in order to have a small error. Moreover, since the function RT_p (11) depends on α, this parameter can be determined in order to get A-stability (or at least $A(\theta)$-stability) for the corresponding TASE-RK method. See, e.g., [11, p. 230] for the stability definitions. Furthermore, we can

choose a generic matrix other than J_n in the formulation of TASE-RK methods without altering their order of consistency; however, good stability properties are preserved provided that J_n is a suitable approximation of the Jacobian.

In the paper [5], Bassenne et al. have set, for the cases $p = s = 2,3,4$, the α parameter in order to obtain a good balance between the minimum $|Q_p|$ value, and the best possible stability properties for the corresponding TASE-RK method. The related results are reported in Table 1.

Table 1: Properties of TASE-RK methods in correspondence of the values of α proposed in the manuscript [5].

$p = s$	Stability properties	$\lvert R(\infty)\rvert$	$\lvert Q_p\rvert$	α
2	A-stability	1	1.13	1.5
2	Strong A-stability	0.5	4.50	3
3	$A(\theta)$-stability, $\theta = 89.31°$	1	2.70	2.7858
4	$A(\theta)$-stability, $\theta = 88.36°$	1	13.14	5.3854

4 Improved TASE-RK methods

The TASE-RK methods have been improved by Calvo et al., who in the paper [12] proposed the following generalization of the family (9):

$$\begin{aligned} T_p(\boldsymbol{\alpha}, hJ_f) &= \sum_{j=1}^{p} \gamma_j (I_d - \alpha_j hJ_f)^{-1}, \\ \gamma_j &= \Big(\frac{1}{\alpha_j}\Big)^{p-1} \Big/ \prod_{k\neq j} \Big(\frac{1}{\alpha_j} - \frac{1}{\alpha_k}\Big), \quad \boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_p) \in \mathbb{R}^p. \end{aligned} \tag{12}$$

Here, $\alpha_j > 0$ and $\alpha_j \neq \alpha_k$ for all $j \neq k$. It can be easily shown that the TASE family (12) generalizes the one given by Bassenne et al., which can be derived using $\alpha_j = \alpha/2^{j-1}$, $j = 1, \ldots, p$. By construction it holds that (see [12, Eqs. (6)-(7)])

$$T_p(\boldsymbol{\alpha}, hJ_f) = I_d + Q_p (hJ_f)^p + O(h^{p+1}) \qquad \text{with } Q_p = \prod_{j=1}^{p} \alpha_j.$$

The motivation that led Calvo et al. to this generalization is based on the fact that the first family of TASE operators depends on a single free parameter α. As seen in the previous section, α should be set to obtain the optimal balance between the minimum value of the error constant $|Q_p|$ and the best stability of the corresponding TASE-RK method. However, the results in Table 1 are not fully satisfactory, mainly because it is not possible to achieve strongly $A(\theta)$-stable or even $L(\theta)$-stable methods for $s = p = 3,4$. For problems with high stiffness, these properties are very important.

Thanks to this generalization, there are p ($p > 1$) free parameters α_j to fix for stability and accuracy reasons, and therefore more possibilities to improve TASE-RK methods. We note that obviously for this generalization the stability function becomes

$$RT_p(\boldsymbol{\alpha}, z) = 1 + zT_p(\boldsymbol{\alpha}, z) + \cdots + \frac{1}{p!}(zT_p(\boldsymbol{\alpha}, z))^p.$$

Furthermore, the computational cost of the TASE-RK methods (8) with the new T_p (12) remains the same. Indeed, it is still required the solution of sp linear systems at each step depending on J_n. In Table 2, we report the properties of the TASE-RK methods in correspondence of the α_j parameters of the operator T_p (12) proposed by Calvo et al. in the paper [12].

Table 2: Properties of TASE-RK methods in correspondence of the values of $\boldsymbol{\alpha}$ proposed in the manuscript [12].

$p=s$	Stability properties	$\lvert R(\infty)\rvert$	$\lvert Q_p\rvert$	$\boldsymbol{\alpha}$
2	Strong A-stability	0.5	4.50	$(3, 1.50)$
3	$L(\theta)$-stability, $\theta = 89.02°$	0	6.88	$(2.315, 1.880, 1.582)$
4	Strong $A(\theta)$-stability, $\theta = 87.34°$	0.270	44.32	$(3.940, 2.451, 2.227, 2.061)$

By comparing Tables 1, 2, it is clear that the error constant $|Q_p|$ is of the same order of magnitude for both families of operators (9), (12), for $p = 2,3,4$. Also the angle of $A(\theta)$-stability is more or less similar for the versions proposed in the two tables. However, TASE-RK methods with operator (12) have much better strong stability properties. Therefore, from now on we refer directly to the operators proposed by Calvo et al. (12).

5 MATLAB code and computational effort

From the analysis made in the previous sections, it is clear that the consistency of the TASE-RK methods is independent of the choice of J_n. Indeed, for TASE-RK methods, to get order p, it suffices that the underlying explicit RK method has order p and property (7) holds (see Theorem 3.1).

As observed in Remark 3.3, stability reasons require J_n to be a suitable approximation of the Jacobian, obtained, for example, by fixing J_n as a constant matrix (thus avoiding updating J_n at all points of the time grid). For this reason, we report below an implementation of TASE-RK methods allowing the user to choose whether to update the Jacobian at each step, or fix it as a constant matrix. Furthermore, with this implementation, methods can be applied with any number of stages and with a TASE operator having a generic number of terms (therefore also for $p \neq s$). Finally, referring to formulation (8) of TASE-RK methods, we exploit the MATLAB `lu` function for factorizing the coefficient matrices and then the command `backslash` for solving the required linear systems by means of the forward/backward substitution. At the end of this section, we also provide a brief analysis of the computational effort of TASE-RK methods with and without exact Jacobian, showing that the choice of constant J_n allows to remarkably reduce the number of required operations.

5.1 Function `TASERK.m`

Let us describe below the input and output arguments, together with the employed auxiliary MATLAB functions, of the code `TASERK.m`, which allows to apply the TASE-RK methods proposed by Calvo et al. [12] to a differential problem of the type (1) chosen by the user.

Input arguments

- `N` - integer scalar

Number of (equally spaced) discrete time intervals into which the user decides to subdivide the continuous grid $[t_0, t_e]$.

- `tspan` - double array
  Row vector of length two containing the first and last grid points t_0, t_e, respectively.

- `y0` - double array
  Column vector with the initial condition $y_0 \in \mathbb{R}^d$.

- `Fun` - function handle
  Function which returns the vector field f, evaluated at `(t,y)`, of the problem (1) that the user wants to solve; `t` $\in \mathbb{R}$, `y` $\in \mathbb{R}^d$, constitute the input arguments of `Fun`, and the column vector f(`t`,`y`) $\in \mathbb{R}^d$ is the output.

- `Jac` - function handle
  Function which returns the Jacobian matrix of the problem (1) that the user wants to solve, evaluated at a point `(t,y)`, or a suitable fixed approximation of J_f; in the first case, `t` $\in \mathbb{R}$, `y` $\in \mathbb{R}^d$, constitute the input arguments of `Jac`, and the matrix f_y(`t`,`y`) $\in \mathbb{R}^{d,d}$ is the output.

- `Method` - integer array
  Array with the TASE-RK methods to apply; in particular:

  - `20` corresponds to the TASE-RK with $s = p = 2$, using the midpoint rule as underlying explicit RK;
  - `30` corresponds to the TASE-RK with $s = p = 3$, using the Ralston's method as underlying explicit RK;
  - `40` corresponds to the TASE-RK with $s = p = 4$, using the Kutta's method as underlying explicit RK.

  For example, if we want to use methods `20` and `30`, then `Method=[20,30]`. We will later show an example where we apply all the TASE-RK methods using the same main program.

- `jacup` - integer scalar
  Parameter which is equal to 0 if the user wants to use constant J_n, 1 otherwise.

Output arguments

- `yT` - double array
  Column vector of length d with the numerical solution computed by the chosen TASE-RK method at last grid point t_e.

- `y` - double array
  Matrix of size $d \times (N+1)$ having in column $n+1$ the numerical solution y_n computed by the chosen TASE-RK method.

- `t` - double array
  Row vector of length $N+1$ with all the discrete grid points $\{t_n = t_0 + nh; n = 0, \ldots, N; t_N = t_e\}$.

- `CPUtime` - double scalar
  Total CPU time in seconds taken by the chosen TASE-RK method.

Auxiliary MATLAB functions

- `backslash`
  Computes $A^{-1}B$, where A and B are input matrices.
- `cputime`
  Returns the current CPU time in seconds.
- `eye`
  Returns the Identity matrix of required dimension.
- `kron`
  Computes, in general, the Kronecker product of two matrices.
- `length`
  Returns the length of a vector.
- `linspace`
  Generates a row vector of linearly equally spaced points.
- `lu`
  Computes the LU factorization of a matrix.
- `ones`
  Generates a matrix of the required size with all elements equal to one.
- `sum`
  Computes the sum by rows or columns of the elements of a matrix.
- `zeros`
  Generates a matrix of the required size with all elements equal to zero.

Below, we report the MATLAB function `TASERK.m`.

Code 1: function `TASERK.m`.

```
function [yT,y,t,CPUtime] = TASERK(N,tspan,y0,Fun,Jac,Method,jacup)

% Fixing explicit RK tableau and TASE operator coefficients
switch Method
    case 20 % 20-midpoint method of order s=p=2
        s = 2; p = 2; alpha = [3 3/2];
        A = [0 0;1/2 0]; c = [0 1/2]; b = [0 1];

    case 30 % 30-Ralston method of order s=p=3
        s = 3; p = 3; alpha = [2.31469 1.87961 1.58222];
        A = [0 0 0;1/2 0 0;0 3/4 0];
        c = [0 1/2 3/4]; b = [2/9 1/3 4/9];

    case 40 % 40-Kutta method of order s=p=4
        s = 4; p = 4; alpha = [3.939556 2.450558 2.227083 2.061235];
        A = [0 0 0 0;1/2 0 0 0;0 1/2 0 0;0 0 1 0];
        c = [0 1/2 1/2 1]; b = [1/6 1/3 1/3 1/6];
end

% Computation of gamma
alpham1 = 1./alpha; gamma = [];
for i = 1:p
```

```
    prod = 1;
    for j = 1:p
        if i ~= j
            prod = prod*(alpham1(i)-alpham1(j));
        end
    end
    gamma(i) = alpham1(i)^(p-1)/prod;
end

% Initialization
t = linspace(tspan(1),tspan(2),N+1); % t: discrete time grid
h = (tspan(2)-tspan(1))/N; % h: constant time step-size
d = length(y0); % d: dimension of the problem
Id = eye(d); % Id: Identity matrix of order d
n = 1; % n: index representing the current step
y = y0;

if (jacup == 0) % If we want a 'constant Jacobian'
    C = cputime;
    Jn = Jac();
    for l = 1:p % Compute, outside the loop, the LU factorizations of Id-alpha(l)*(h*Jn)
        [Ll(1:d,(l-1)*d+1:l*d),Ul(1:d,(l-1)*d+1:l*d)] = lu(Id-alpha(l)*(h*Jn));
    end
    for n = 2:N+1
        Y = zeros(d,s); % Block matrix with stages in columns
        Y(:,1) = y(:,n-1);
        f = zeros(d,s); % Block matrix with h*fi in columns
        F = zeros(d,s); % Block matrix with Tp*h*fi in columns
        for i = 1:s-1 % Compute all the stages
            f(:,i) = h*Fun(t(n-1)+h*c(i),Y(:,i));
            for l = 1:p
                F(:,i) = F(:,i) + gamma(l)*(Ul(1:d,(l-1)*d+1:l*d) \ (Ll(1:d,(l-1)*d+1:l*d)\f
    (:,i)));
            end
            Y(:,i+1) = y(:,n-1) + sum(kron(A(i+1,:),ones(d,1)).*F,2);
        end
        f(:,s) = h*Fun(t(n-1)+h*c(s),Y(:,s));
        for l = 1:p
            F(:,s) = F(:,s) + gamma(l)*(Ul(1:d,(l-1)*d+1:l*d) \ (Ll(1:d,(l-1)*d+1:l*d)\f(:,s))
    );
        end
        y(:,n) = y(:,n-1) + sum(kron(b,ones(d,1)).*F,2);
    end
    Cf = cputime;
elseif (jacup == 1) % If we want exact Jacobian
    C = cputime;
    for n = 2:N+1
        Jn = Jac(t(n-1),y(:,n-1)); % Update Jn at each step
        for l = 1:p % Compute, at each step, the LU factorizations of Id-alpha(l)*(h*Jn)
            [Ll(1:d,(l-1)*d+1:l*d),Ul(1:d,(l-1)*d+1:l*d)] = lu(Id-alpha(l)*(h*Jn));
        end
        Y = zeros(d,s); % Block matrix with the stages in column
        Y(:,1) = y(:,n-1);
        f = zeros(d,s); % Block matrix with h*fi in column
        F = zeros(d,s); % Block matrix with Tp*h*fi in column
        for i = 1:s-1 % Compute all the stages
            f(:,i) = h*Fun(t(n-1)+h*c(i),Y(:,i));
            for l = 1:p
                F(:,i) = F(:,i) + gamma(l)*(Ul(1:d,(l-1)*d+1:l*d) \ (Ll(1:d,(l-1)*d+1:l*d)\f
    (:,i)));
            end
            Y(:,i+1) = y(:,n-1) + sum(kron(A(i+1,:),ones(d,1)).*F,2);
        end
        f(:,s) = h*Fun(t(n-1)+h*c(s),Y(:,s));
        for l = 1:p
            F(:,s) = F(:,s) + gamma(l)*(Ul(1:d,(l-1)*d+1:l*d) \ (Ll(1:d,(l-1)*d+1:l*d)\f(:,s))
    );
        end
        y(:,n) = y(:,n-1) + sum(kron(b,ones(d,1)).*F,2);
    end
    Cf = cputime;
end

```

```
92 CPUtime = Cf - C;
93 yT = y(:,end);
94 end
```

5.2 Description of the code

Let us describe the lines of code of the function `TASERK.m`.

- From line 3 to line 18: we select the TASE-RK methods chosen by the user; method 20 corresponds to the case $s = p = 2$, using the midpoint rule as underlying explicit RK; method 30 corresponds to the case $s = p = 3$, using the Ralston's method as underlying explicit RK; method 40 corresponds to the case $s = p = 4$, using the Kutta's method as underlying explicit RK.

- From line 20 to line 30: we compute the values of γ_j, storing them in a vector `gamma`, starting from the `alpha` ($\boldsymbol{\alpha}$) vector according to Equation (12).

- From line 32 to line 38: we define the time grid `t`, the step-size `h`, the Identity matrix `Id` of size `d` (i.e. the size of the problem); we also initialize the time step `n`, the matrix `y`, which, at the end, will contain the numerical solution (in the columns) at all the discrete points, and the CPU time `C`.

- From line 40 to line 64: if `jacup`$= 0$, i.e. we want to fix J_n avoiding updating it at each step, we first call the function `Jac.m`, which returns in this case a suitable constant approximation of J_n; then we apply the TASE-RK method as described below.

- From line 43 to line 45: using the `lu` command, we compute the LU factorizations of the matrices $I_d - \alpha_j h J_n$, $j = 1, \ldots, p$, whose summed inverses define the TASE operator T_p according to Equation (12); we allocate all the lower and upper triangular matrices L and U of size d thus obtained in successive blocks of the matrices `Ll` and `Ul`, respectively, which have dimension $d \times (pd)$; note that the matrices `Ll` and `Ul` are only computed here, and are not updated within the method being $I_d - \alpha_j h J_n$, $j = 1, \ldots, p$, constants.

- From line 46 to line 64: we compute the numerical solution at all grid points using the TASE-RK method; we explain below the operations performed here.

  - From line 47 to line 50: we define the matrix `Y` which, at the end of the current step, will contain all the stages $Y_{n,j}$, $j = 1, \ldots, s$, in columns; the matrix `f` which, at the end of the current step, will contain all the function evaluations $hf(t_n + c_j h, Y_{n,j})$, $j = 1, \ldots, s$, in columns; the matrix F which, at the end of the current step, will contain the products between T_p and $hf(t_n + c_j h, Y_{n,j})$, $j = 1, \ldots, s$, in columns.
  - From line 51 to line 61: we compute all the stages storing them in `Y`; note that the products between T_p and $hf(t_n + c_j h, Y_{n,j})$ are calculated by solving linear systems of the form $\tilde{A}x = \tilde{b}$ by means of the `backslash` command, with $\tilde{A} = I_d - \alpha_j h J_n$, $\tilde{b} = hf(t_n + c_j h, Y_{n,j})$, through the LU factorizations of the coefficient matrices stored in the arrays `Ll` and `Ul`.
  - Line 62: we compute the solution y_{n-1}, which we store in the `n`-th column of the matrix `y`; the index `n` is shifted by one with respect to n as the initial condition y_0 is stored in the first column of `y` (in MATLAB, array indexes start at 1).

- From line 65 to line 90: if `jacup`$= 1$, we update J_n and therefore the LU factorizations of the matrices $I_d - \alpha_j h J_n$, $j = 1, \dots, p$, at each step; in this case, the function `Jac.m` returns the exact Jacobian of the problem evaluated at the desired grid point; note that, of course, lines 72–89 correspond to lines 47–64 (in fact, the only change with respect to the case `jacup`$= 0$ is due to the fact that J_n and the matrix factorizations must be updated at each step and therefore appear inside the loop).
- From line 92 to line 94: we compute the total CPU time employed by the method and store in the vector `yT` the numerical solution at the final point of the grid.

Note that, of course, setting J_n constant, the number of operations required by the method drops drastically. In fact, for:

- `jacup`$= 0$, we have to compute only p LU factorizations, then using them in the solution of sp linear systems per step; thus, at the end we have p LU factorizations plus $N(sp)$ linear systems;
- `jacup`$= 1$, we have to compute p LU factorizations per step, using them in the solution of sp linear systems; thus, at the end we have Np LU factorizations plus $N(sp)$ linear systems.

Obviously, we underline that the cost of solving a linear system with already factorized matrix is considerably reduced. In particular, given d the size of the problem, an LU factorization costs $O(d^3/3)$, and solving a linear system with an already factorized matrix costs $O(d^2)$. Therefore, the approach we propose in the code is especially convenient for problems of big dimensions, and when a large number N of time grid points (i.e. small h) is required.

6 Examples of application

We report below two examples of application of the code `TASERK.m`. The first is quite simple and concerns a system of ODEs. The second concerns the numerical solution of the famous Burgers' PDE.

6.1 Euler's problem

In this subsection, we show the application of the function `TASERK.m` in solving the well known Euler's problem, given by the following system of coupled ODEs:

$$\begin{cases} \dfrac{dy_1}{dt} = -2y_2y_3, \\ \dfrac{dy_2}{dt} = \dfrac{5}{4}y_1y_3, \\ \dfrac{dy_3}{dt} = -\dfrac{1}{2}y_1y_2, \end{cases} \qquad t \in [t_0, t_e]. \tag{13}$$

This model is related to the rotational motion of solid bodies. We take $t_0 = 0$, $t_e = 10$, and $y_0 = (1, 0, 0.9)$. Easily, note that

$$J_f = \begin{pmatrix} 0 & -2y_3 & -2y_2 \\ \dfrac{5}{4}y_3 & 0 & \dfrac{5}{4}y_1 \\ \dfrac{-1}{2}y_2 & \dfrac{-1}{2}y_1 & 0 \end{pmatrix}. \tag{14}$$

Since we can use a fixed approximation of the Jacobian to lower the computational cost of TASE-RK methods preserving their order of consistency, if `jacup`$= 0$ we impose that the function `Jac` returns the matrix J_f evaluated at the initial point (t_0, y_0). Then, in this case we evaluate the Jacobian only at the initial grid point, without updating it during the integration.

We report and describe below the main code, which we have called `exampleEuler.m`. In line 2 we define the function `Fun`, which corresponds to `funEuler.m`. In line 3 we choose the TASE-RK methods to use; we apply in this case the method with $p = s = 4$. From line 5 to line 10, we choose the `jacup` parameter and fix the function `Jac` according to its value. From line 13 to line 16, we set the time grid, the initial conditions and the number `N` of discrete intervals. From line 19 to line 21 we also compute a reference solution by means of the MATLAB `ode15s` function. In line 23 we apply the selected TASE-RK method to the Euler's model. Finally, we print the error at the final time grid point and the employed CPU time.

Code 2: main code `exampleEuler.m`.

```
1  %% Main code: exampleEuler.m
2  Fun = @funEuler;
3  Tmethod = [40]; % Select the TASE-RK method
4  nTmethods = length(Tmethod);
5  jacup = 0; % We want 'constant Jacobian'
6  if (jacup == 0)
7      Jac = @jacEulerfix;
8  elseif (jacup == 1)
9      Jac = @jacEuler;
10 end
11
12 % Initial conditions
13 global y0
14 tspan = [0 10];
15 y0 = [1;0;0.9];
16 N = 5000; % Number of grid intervals
17
18 % Compute a reference solution with ode15s
19 options = odeset('RelTol',5e-14,'AbsTol',5e-14);
20 [tode15s,yode15s] = ode15s(@funEuler,tspan,y0,options);
21 YrefT = yode15s(end,:)';
22
23 [yTTRK,yTRK,t,CPUtimeTRK] = TASERK(N,tspan,y0,Fun,Jac,Tmethod,jacup);
24
25 % Print results
26 format short e
27 errT_TRK = norm(yTTRK-YrefT,inf) % Error
28 CPUtimeTRK % CPU time
```

We also report the functions `funEuler.m`, `jacEulerfix.m`, `jacEuler.m`, respectively, which are recalled in the main algorithm. Obviously, `funEuler.m` returns the (column) vector field f given in Equation (13) evaluated at a point `(t,y)`. Furthermore, `jacEulerfix.m` returns the Jacobian (14) evaluated at the initial grid point. Finally, `jacEuler.m` returns the exact Jacobian (14) evaluated at a point `(t,y)`.

Code 3: function `funEuler.m`.

```
1  function yp = funEuler(t,y)
2
3  yp(1) = -2*y(2)*y(3);
4  yp(2) = 5/4*y(3)*y(1);
5  yp(3) = -1/2*y(1)*y(2);
6  yp = [yp(1);yp(2);yp(3)];
7
8  end
```

Code 4: function `jacEulerfix.m`.

```
1  function J = jacEulerfix()
```

```
2
3  global y0
4  J = [0 -2*y0(3) -2*y0(2);
5      5/4*y0(3) 0 5/4*y0(1);
6      -1/2*y0(2) -1/2*y0(1) 0];
7
8  end
```

Code 5: function `jacEuler.m`.

```
1  function J = jacEuler(t,y)
2
3  J = [0 -2*y(3) -2*y(2);
4      5/4*y(3) 0 5/4*y(1);
5      -1/2*y(2) -1/2*y(1) 0];
6
7  end
```

To conclude this subsection, we also report the outputs obtained.

Code 6: outputs of the main code `exampleEuler.m`.

```
1  >> exampleEuler
2  errT_TRK =
3     3.3776e-08
4  CPUtimeTRK =
5     6.4062e-01
```

6.2 Burgers' equation

In this subsection, we show the application of the function `TASERK.m` in solving the Burgers' equation [6, 9], which can be expressed as follows:

$$\frac{\partial u}{\partial t} = \varepsilon \frac{\partial^2 u}{\partial x^2} - u\frac{\partial u}{\partial x}, \qquad (x,t) \in [x_0, X] \times [t_0, t_e]. \tag{15}$$

The function u represents the speed of the fluid at the considered spatial (x) and temporal (t) coordinates, ε is related to a constant physical property of the fluid, generally the viscosity or something similar to it. When the diffusion term is absent, this PDE becomes the inviscid Burgers' equation. Furthermore, Equation (15) can also be expressed in the following conservative form:

$$\frac{\partial u}{\partial t} = \varepsilon \frac{\partial^2 u}{\partial x^2} - \frac{1}{2}\frac{\partial u^2}{\partial x}, \qquad (x,t) \in [x_0, X] \times [t_0, t_e]. \tag{16}$$

Here, we consider the numerical solution of Equation (16). Regarding the initial conditions, the spatial semi-discretization and the boundary conditions we mainly refer to [12, Section 3.2]. In particular, we consider as initial conditions the function

$$y(x,0) = \begin{cases} 1, & x \in [0, \pi], \\ 0, & x \in (\pi, 2\pi]. \end{cases}$$

Regarding the spatial semi-discretization of the equation, we take centred finite differences of order four for both the first and second order spatial derivatives. Then, by fixing the uniform spatial grid $\{x_n = x_0 + m\Delta x; m = 0, \ldots, M; x_M = X\}$, the semi-discretized Burgers' equation (16) reads

$$y'(t) = \varepsilon L_1 y(t) - \frac{1}{2} L_2 y(t)^2. \tag{17}$$

By calling with $(d_{-2}, d_{-1}, d, d_1, d_2)$ the significant entries of the sub-, main-, and over-diagonals, L_1 and L_2 are the following pentadiagonal Toeplitz matrices:

$$L_1 = \frac{1}{12\Delta x^2}(-1, 16, -30, 16, -1), \quad L_2 = \frac{1}{12\Delta x}(1, -8, 0, 8, -1).$$

The Jacobian is in this case given by the non-constant matrix

$$J_f = \varepsilon L_1 - L_2 Y(t). \tag{18}$$

Here, $Y(t)$ is a square matrix of size M, having in each column the vector $y(t)$. For semi-discretized PDEs of this form, the diffusion part is mainly responsible for the stiffness of the problem. Thus, in our code, since we can use a fixed approximation of the Jacobian to lower the computational cost while maintaining order of TASE-RK methods, if `jacup`$= 0$ the function `Jac` returns the constant matrix εL_1.

We report and describe below the main code, which we have called `exampleBurgers.m`. Note that, unlike the Euler's problem, we now apply all the schemes reported in Table 2. Furthermore, we perform the numerical integration using several values of N; in particular we use $N = 2^8, 2^9, \ldots, 2^{12}$ time grid points, as can be seen from line 11. From line 13 to line 25, we set the parameters, initial conditions and spatial semi-discretization of Burgers' equation. In the example, we have used $\varepsilon = 1/10$, $M = 32$ spatial grid points, $t_0 = x_0 = 0$, $X = 2\pi$, $t_e = 4$. From line 28 to line 30 we also compute, like before, a reference solution by means of the MATLAB `ode15s` function. From line 32 to line 46 we apply the selected TASE-RK methods to the chosen problem, and store the obtained results in the matrices `errT_TRK`, `CPUtime_TRK`, `pest_TRK`. In particular, the first two matrices contain in column i the absolute errors (at the last time grid point) and CPU times, respectively, of the TASE-RK method with $p = s = i + 1$. Each row corresponds to a different value of N. The first row corresponds to the case $N = 2^8$, the last to $N = 2^{12}$. The matrix `pest_TRK` is constructed in the same way and contains the estimated order of the methods. However, it has one row less since it is not possible to estimate the order for the first value of N.

Code 7: main code `exampleBurgers.m`.

```
1  %% Main code: exampleBurgers.m
2  Fun = @funBurgers;
3  Tmethod = [20 30 40]; % Select the TASE-RK methods
4  nTmethods = length(Tmethod);
5  jacup = 0; % We want 'constant Jacobian'
6  if (jacup == 0)
7      Jac = @jacBurgersfix;
8  elseif (jacup == 1)
9      Jac = @jacBurgers;
10 end
11 inN = 8; finN = 12; % We do the time integration using 2^(inN),...,2^(finN) time grid points
12
13 global epsilon M L1 L2
14 epsilon = 1/10; M = 32; % Spatial grid points
15 xspan = [0 2*pi]; tspan = [0 4];
16 Deltax = (xspan(2)-xspan(1))/M; % Spatial step-size
17
18 % Initial conditions
19 y0 = []; y0(1:M/2,1) = ones(M/2,1); y0(M/2+1:M,1) = zeros(M/2,1);
20
21 % Space-discetization: order-four FD and periodic BC
22 e = ones(M,1); r = 1/(12*Deltax^2);
23 L1 = spdiags([-r*e 16*r*e -30*r*e 16*r*e -r*e], -2:2, M, M); L1(1,M-1) = -r; L1(M-1,1) = -r;
       L1(M,2) = -r; L1(2,M) = -r; L1(1,end) = 16*r; L1(end,1) = 16*r;
24 r = 1/(12*Deltax);
25 L2 = spdiags([r*e -8*r*e 0*r*e 8*r*e -r*e], -2:2, M, M); L2(1,M-1) = r; L2(M-1,1) = -r; L2(M
       ,2) = -r; L2(2,M) = r; L2(1,end) = -8*r; L2(end,1) = 8*r;
```

```
26
27 % Compute a reference solution with ode15s
28 options = odeset('RelTol',5e-14,'AbsTol',5e-14);
29 [tode15s,yode15s] = ode15s(@funBurgers,tspan,y0,options);
30 YrefT = yode15s(end,:)';
31
32 for nm = 1:nTmethods % We apply all the selected TASE-RK
33     i = 1;
34     for N = 2.^(inN:finN) % For the simulations done
35         [yTTRK,yTRK,t,CPUtimeTRK] = TASERK(N,tspan,y0,Fun,Jac,Tmethod(nm),jacup);
36         errT_TRK(i,nm) = norm(yTTRK-YrefT,inf); % Error
37         cd_TRK(i,nm) = -log10(errT_TRK(i,nm));
38         CPUtime_TRK(i,nm) = CPUtimeTRK; % CPU time
39         i = i + 1;
40     end
41
42     l = length(cd_TRK(:,nm));
43     for i = 2:l % Compute the estimated order
44         pest_TRK(i-1,nm) = (cd_TRK(i,nm)-cd_TRK(i-1,nm))/log10(2);
45     end
46 end
47
48 % Print results
49 format short e
50 errT_TRK
51 CPUtime_TRK
52 format short
53 pest_TRK
```

We also report below the function `funBurgers.m`, which obviously returns the vector field of the semi-discretized Burgers' equation (17). Furthermore, we report the functions `jacBurgersfix.m`, `jacBurgers.m`. The latter computes the exact Jacobian, given in Equation (18). The first returns only the diffusion part.

Code 8: function `funBurgers.m`.

```
1 function yp = funBurgers(t,y)
2
3 global epsilon L1 L2
4 yp = epsilon*L1*y - (1/2)*L2*(y.^2);
5
6 end
```

Code 9: function `jacBurgersfix.m`.

```
1 function J = jacBurgersfix()
2
3 global epsilon L1
4 J = epsilon*L1;
5
6 end
```

Code 10: function `jacBurgers.m`.

```
1 function J = jacBurgers(t,y)
2
3 global epsilon L1 L2 M
4 J = epsilon*L1-L2.*repmat(y,1,M)';
5
6 end
```

Finally, we report below the outputs obtained by running `exampleBurgers.m`. Note that, although we used `jacup`$=0$, the methods maintain their order, as expected from the consistency analysis done in the paper. This can be seen by looking at the columns of the matrices `errT_TRK` and `pest_TRK`. Indeed, we recall that the first column corresponds to the TASE-RK of order $p = s = 2$, the second to the method of order $p = s = 3$ and the third to the scheme with $p = s = 4$. Looking at the matrix with the CPU times, it is evident that the more the number of

grid points N increases and the more the number of stages is high, the greater the computational effort.

Code 11: outputs of the main code `exampleBurgers.m`.

```
>> exampleBurgers
errT_TRK =
   3.2141e-04   2.5591e-05   8.8510e-06
   8.9912e-05   3.9132e-06   9.0181e-07
   2.3923e-05   5.4871e-07   7.5195e-08
   6.1825e-06   7.2968e-08   5.5087e-09
   1.5724e-06   9.4195e-09   3.7483e-10
CPUtime_TRK =
   3.1250e-02   4.6875e-02   6.2500e-02
   4.6875e-02   1.0938e-01   1.2500e-01
   1.4062e-01   1.5625e-01   2.6562e-01
   2.3438e-01   2.9688e-01   5.0000e-01
   3.4375e-01   6.5625e-01   9.8438e-01
pest_TRK =
    1.8378    2.7092    3.2949
    1.9101    2.8342    3.5841
    1.9521    2.9107    3.7708
    1.9752    2.9535    3.8774
```

7 Conclusions

In this manuscript, we have analyzed in detail the consistency and stability properties of the recently introduced TASE-RK methods. Taking advantage of this analysis, we have proposed a MATLAB implementation of these methods allowing the user to make a flexible choice of the approximation of J_n to be used. Furthermore, by formulating TASE-RK schemes as linearly implicit methods, we have employed some MATLAB functions to efficiently solve the underlying linear systems, by computing a-priori the LU factorizations of the matrices involved. The proposed implementation is not limited to the cases of TASE-RK methods with $p = s \leq 4$, but can allow the user to test methods with a higher number of stages and such that $p \neq s$.

We were motivated by the growing interest in TASE-RK methods, which are quite promising in solving large stiff problems, especially coming from the spatial semi-discretization of PDEs. The results reported on the Euler's problem and Burgers' equation testify the correctness of the theoretical analysis and the functioning of the proposed MATLAB code.

Acknowledgments

The authors are members of the GNCS group. This work is supported by GNCS-INDAM project and by the Italian Ministry of University and Research (MUR), through the PRIN 2020 project (No. 2020JLWP23) "Integrated Mathematical Approaches to Socio–Epidemiological Dynamics" (CUP: E15F21005420006) and the PRIN 2017 project (No. 2017JYCLSF) "Structure preserving approximation of evolutionary problems".

Declaration of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] A. Abdi, G. Hojjati, Z. Jackiewicz, H. Podhaisky, and M. Sharifi. *On the implementation of explicit two-step peer methods with Runge-Kutta stability*. Appl. Numer. Math., vol. 186, (2023), 213–227. ISSN 0168-9274. URL http://dx.doi.org/10.1016/j.apnum.2023.01.015

[2] L. Aceto, D. Conte, and G. Pagano. *On a generalization of time-accurate and highly-stable explicit operators for stiff problems*. Appl. Numer. Math. URL http://dx.doi.org/10.1016/j.apnum.2023.04.001

[3] S. M. Allen and J. W. Cahn. *Ground State Structures in Ordered Binary Alloys with Second Neighbor Interactions*. Acta Metall., vol. 20, 3, (1972), 423–433. URL http://dx.doi.org/10.1016/0001-6160(72)90037-5

[4] D. Aregba-Driollet, F. Diele, and R. Natalini. *A mathematical model for the sulphur dioxide aggression to calcium carbonate stones: numerical approximation and asymptotic analysis*. SIAM J. Appl. Math., vol. 64, 5, (2004), 1636–1667. ISSN 0036-1399. URL http://dx.doi.org/10.1137/S003613990342829X

[5] M. Bassenne, L. Fu, and A. Mani. *Time-accurate and highly-stable explicit operators for stiff differential equations*. J. Comput. Phys., vol. 424, (2021), Paper No. 109847, 24. ISSN 0021-9991. URL http://dx.doi.org/10.1016/j.jcp.2020.109847

[6] H. Bateman. *Some recent researches on the motion of fluids*. Monthly Weather Review, vol. 43, 3, (1915), 163–170. URL http://dx.doi.org/10.1175/1520-0493(1915)43<163:SRROTM>2.0.CO;2

[7] M. A. Budroni, G. Pagano, D. Conte, B. Paternoster, R. D'Ambrosio, S. Ristori, A. Abou-Hassan, and F. Rossi. *Synchronization scenarios induced by delayed communication in arrays of diffusively coupled autonomous chemical oscillators*. Phys. Chem. Chem. Phys., vol. 23, 32, (2021), 17606–17615. URL http://dx.doi.org/10.1039/d1cp02221k

[8] M. A. Budroni, K. Torbensen, S. Ristori, A. Abou-Hassan, and F. Rossi. *Membrane Structure Drives Synchronization Patterns in Arrays of Diffusively Coupled Self-Oscillating Droplets*. J. Phys. Chem. Lett., vol. 11, 6, (2020), 2014–2020. URL http://dx.doi.org/10.1021/acs.jpclett.0c00072

[9] J. M. Burgers. *A mathematical model illustrating the theory of turbulence*. *Advances in Applied Mechanics*, (Academic Press, Inc., New York, N.Y.1948). 171–199. Edited by Richard von Mises and Theodore von Kármán, URL http://dx.doi.org/10.1016/S0065-2156(08)70100-5

[10] J. C. Butcher. *The numerical analysis of ordinary differential equations*. A Wiley-Interscience Publication, (John Wiley & Sons, Ltd., Chichester1987). ISBN 0-471-91046-5. Runge-Kutta and general linear methods, URL http://dx.doi.org/10.1137/1031147

[11] J. C. Butcher. *Numerical methods for ordinary differential equations*, (John Wiley & Sons, Ltd., Chichester2008), second edn. ISBN 978-0-470-72335-7. URL http://dx.doi.org/10.1002/9780470753767

[12] M. Calvo, J. I. Montijano, and L. Rández. *A note on the stability of time-accurate and highly-stable explicit operators for stiff differential equations*. J. Comput. Phys., vol. 436, (2021), Paper No. 110316, 13. ISSN 0021-9991. URL http://dx.doi.org/10.1016/j.jcp.2021.110316

[13] A. Cardone, R. D'Ambrosio, and B. Paternoster. *Exponentially fitted IMEX methods for advection-diffusion problems*. J. Comput. Appl. Math., vol. 316, (2017), 100–108. ISSN 0377-0427. URL http://dx.doi.org/10.1016/j.cam.2016.08.025

[14] A. Cardone, Z. Jackiewicz, A. Sandu, and H. Zhang. *Extrapolation-based implicit-explicit general linear methods*. Numer. Algorithms, vol. 65, 3, (2014), 377–399. ISSN 1017-1398. URL http://dx.doi.org/10.1007/s11075-013-9759-y

[15] D. Conte and G. Frasca-Caccia. *A Matlab code for the computational solution of a phase field model for pitting corrosion*. Dolomites Res. Notes Approx., vol. 15, Special Issue SA2022—Software for Approximation 2022, (2022), 47–65. URL http://dx.doi.org/10.14658/PUPJ-DRNA-2022-2-5

[16] D. Conte, G. Pagano, and B. Paternoster. *Two-step peer methods with equation-dependent coefficients*. Comput. Appl. Math., vol. 41, 4, (2022), Paper No. 140, 21. ISSN 2238-3603. URL http://dx.doi.org/10.1007/s40314-022-01844-z

[17] D. Conte, G. Pagano, and B. Paternoster. *Nonstandard finite differences numerical methods for a vegetation reaction-diffusion model*. J. Comput. Appl. Math., vol. 419, (2023), Paper No. 114790, 17. ISSN 0377-0427. URL http://dx.doi.org/10.1016/j.cam.2022.114790

[18] D. Conte, G. Pagano, and B. Paternoster. *Time-accurate and highly-stable explicit peer methods for stiff differential problems*. Commun. Nonlinear Sci. Numer. Simul., vol. 119, (2023), Paper No. 107136, 20. ISSN 1007-5704. URL http://dx.doi.org/10.1016/j.cnsns.2023.107136

[19] L. Eigentler and J. A. Sherratt. *Metastability as a Coexistence Mechanism in a Model for Dryland Vegetation Patterns*. Bull. Math. Biol., vol. 81, (2019), 2290–2322. URL http://dx.doi.org/10.1007/s11538-019-00606-z

[20] G. Frasca-Caccia and P. E. Hydon. *Locally conservative finite difference schemes for the modified KDV equation*. J. Comput. Dyn., vol. 6, 2, (2019), 307–323. ISSN 2158-2491. URL http://dx.doi.org/10.3934/jcd.2019015

[21] G. Frasca-Caccia and P. E. Hydon. *Simple bespoke preservation of two conservation laws*. IMA J. Numer. Anal., vol. 40, 2, (2020), 1294–1329. ISSN 0272-4979. URL http://dx.doi.org/10.1093/imanum/dry087

[22] S. González-Pinto, D. Hernández-Abreu, G. Pagano, and S. Pérez-Rodríguez. *Generalized TASE-RK methods for stiff problems*. Appl. Numer. Math., vol. 188, (2023), 129–145. ISSN 0168-9274. URL http://dx.doi.org/10.1016/j.apnum.2023.03.007

[23] S. González-Pinto, D. Hernández-Abreu, and S. Pérez-Rodríguez. *Rosenbrock-type methods with inexact AMF for the time integration of advection-diffusion-reaction PDEs*. J. Comput. Appl. Math., vol. 262, (2014), 304–321. ISSN 0377-0427. URL http://dx.doi.org/10.1016/j.cam.2013.10.050

[24] S. González-Pinto, D. Hernández-Abreu, and S. Pérez-Rodríguez. *W-methods to stabilize standard explicit Runge-Kutta methods in the time integration of advection-diffusion-reaction PDEs*. J. Comput. Appl. Math., vol. 316, (2017), 143–160. ISSN 0377-0427. URL http://dx.doi.org/10.1016/j.cam.2016.08.026

[25] S. González-Pinto, D. Hernández-Abreu, S. Pérez-Rodríguez, and R. Weiner. *A family of three-stage third order AMF-W-methods for the time integration of advection diffusion reaction PDEs*. Appl. Math. Comput., vol. 274, (2016), 565–584. ISSN 0096-3003. URL http://dx.doi.org/10.1016/j.amc.2015.10.013

[26] E. Hairer and G. Wanner. *Solving ordinary differential equations. II*, *Springer Series in Computational Mathematics*, vol. 14, (Springer-Verlag, Berlin1996), second edn. ISBN 3-540-60452-9. Stiff and differential-algebraic problems, URL http://dx.doi.org/10.1007/978-3-642-05221-7

[27] I. Higueras and T. Roldán. *Starting algorithms for some DIRK methods*. Numer. Algorithms, vol. 23, 4, (2000), 357–369. ISSN 1017-1398. URL http://dx.doi.org/10.1023/A:1019112419829

[28] A. Iserles. *A first course in the numerical analysis of differential equations*. Cambridge Texts in Applied Mathematics, (Cambridge University Press, Cambridge1996). ISBN 0-521-55376-8; 0-521-55655-4. URL http://dx.doi.org/10.1017/CBO9780511995569

[29] Z. Jackiewicz. *General linear methods for ordinary differential equations*, (John Wiley & Sons, Inc., Hoboken, NJ2009). ISBN 978-0-470-40855-1. URL http://dx.doi.org/10.1002/9780470522165

[30] W. Ji, W. Qiu, Z. Shi, S. Pan, and S. Deng. *Stiff-PINN: Physics-Informed Neural Network for Stiff Chemical Kinetics*. J. Phys. Chem. A, vol. 125, 36, (2021), 8098–8106. URL http://dx.doi.org/10.1021/acs.jpca.1c05102

[31] B. A. Schmitt and R. Weiner. *Parallel two-step W-methods with peer variables*. SIAM J. Numer. Anal., vol. 42, 1, (2004), 265–282. ISSN 0036-1429. URL http://dx.doi.org/10.1137/S0036142902411057

[32] B. A. Schmitt, R. Weiner, and H. Podhaisky. *Multi-implicit peer two-step W-methods for parallel time integration*. BIT, vol. 45, 1, (2005), 197–217. ISSN 0006-3835. URL http://dx.doi.org/10.1007/s10543-005-2635-y

[33] H. Soomro, N. Zainuddin, H. Daud, J. Sunday, N. Jamaludin, A. Abdullah, M. Apriyanto, and E. Kadir. *Variable Step Block Hybrid Method for Stiff Chemical Kinetics Problems*. Appl. Sci., vol. 19, 9, (2022), Paper No. 4484. URL http://dx.doi.org/10.3390/app12094484

[34] T. Steihaug and A. Wolfbrandt. *An attempt to avoid exact Jacobian and nonlinear equations in the numerical solution of stiff differential equations*. Math. Comp., vol. 33, 146, (1979), 521–534. ISSN 0025-5718. URL http://dx.doi.org/10.2307/2006293

[35] R. Weiner, K. Biermann, B. A. Schmitt, and H. Podhaisky. *Explicit two-step peer methods*. Comput. Math. Appl., vol. 55, 4, (2008), 609–619. ISSN 0898-1221. URL http://dx.doi.org/10.1016/j.camwa.2007.04.026

[36] H. Zhang, X. Qian, J. Xia, and S. Song. *Unconditionally maximum-principle-preserving parametric integrating factor two-step Runge-Kutta schemes for parabolic sine-Gordon equations*. CSIAM Trans. Appl. Math., vol. 4, 1, (2023), 177–224. ISSN 2708-0560. URL http://dx.doi.org/10.4208/csiam-am.so-2022-0019

J. Approx. Softw., Vol. 1, No. 1 (2024) 5

Journal of Approximation Software

Numerical quadrature for integrals involving oscillating functions

E. Denich [1,⋆] and P. Novati [1]

[1] *Dipartimento di Matematica, Informatica e Geoscienze, Università degli Studi di Trieste, Via Valerio 12/b - 34127, Trieste, Italy.*

Received: 21/03/2024 – Published: 23/07/2024
Communicated by: R. Cavoretto and A. De Rossi

Abstract

This paper deals with the construction of a coupled Gaussian rule for weight functions involving powers, exponentials and trigonometric functions. Starting from a recursive relation for the moments, nodes and weights are computed by using the Chebyshev algorithm together with the Golub and Welsch method. An a posteriori approximation of the quadrature error by means of the generalized averaged Gaussian rules is also considered. Several numerical examples are provided.

Keywords: Gaussian quadrature, Fourier type integral, averaged Gaussian rule (MSC2020: 65D32, 33C45)

1 Introduction

This work deals with the computation of

$$J(g) = \int_0^{+\infty} g(x) x^{\alpha-1} e^{-\beta x} \cos(\omega x)\, dx, \quad \alpha > 0,\ \beta > 0,\ \omega > 0, \tag{1}$$

where g is a smooth function. The above integral can be also interpreted as the cosine transform (see [16]) of the function $g(x)x^{\alpha-1}e^{-\beta x}$ and it is typically referred to as a Fourier type integral, with a broad range of applications in signal processing. More specifically, integrals of type (1) arise for instance in geophysical electromagnetic survey (see e.g. [15]). In particular, the electromagnetic fields induced by an infinite line of electric current placed above the earth surface can be expressed as in (1), in which the function g encodes the reflection and refraction of the electromagnetic waves and depends on the properties of the subsoil structure. Such kind of source is used to simulate a long grounded wire or one side of a large rectangular loop.

⋆ Corresponding author: eleonora.denich@phd.units.it

By using the simple change of variable $\omega x = t$, we reformulate the problem in the evaluation of integrals of type

$$I(f) = \int_0^{+\infty} f(t) t^{\alpha-1} e^{-ct} \cos t \, dt = \omega^{\alpha} J(g), \tag{2}$$

with $c = \frac{\beta}{\omega}$, $f(t) = g\left(\frac{t}{\omega}\right)$. In this way the frequency is inherited by the scale factor c. Working with formulation (2), in this paper we construct a coupled Gaussian rule, by first considering the positive weight function

$$w(t) = t^{\alpha-1} e^{-ct} \left(\cos t + 1\right), \tag{3}$$

and, then, by rewriting integral (2) as

$$I(f) = I^C(f) - I^L(f),$$

with

$$I^C(f) = \int_0^{+\infty} f(t) t^{\alpha-1} e^{-ct} \left(\cos t + 1\right) dt, \tag{4}$$

and

$$I^L(f) = \int_0^{+\infty} f(t) t^{\alpha-1} e^{-ct} dt = \frac{1}{c^{\alpha}} \int_0^{+\infty} f\left(\frac{y}{c}\right) y^{\alpha-1} e^{-y} dy. \tag{5}$$

We notice that the integral $I^L(f)$ can be accurately computed by using the generalized Gauss-Laguerre formula, that we denote by $I_n^L(f)$. Therefore, we focus on the construction of a Gaussian quadrature rule with respect to the weight function (3). Having at disposal such a formula, denoted by $I_n^C(f)$, we then consider the approximation

$$I(f) = \left(I_n^C(f) - I_n^L(f)\right) + E_n(f), \tag{6}$$

where $E_n(f)$ is the quadrature error.

In this setting, since we do not have at disposal the explicit expression of the orthogonal polynomials π_k, $k \geq 0$, relative to $w(t)$ as in (3), we need to employ a numerical scheme to compute the coefficients of the three-term recursion

$$\begin{gathered} \pi_{k+1}(t) = (t - \alpha_k)\pi_k(t) - \beta_k \pi_{k-1}(t), \quad k \geq 0, \\ \pi_{-1}(t) = 0, \quad \pi_0(t) = 1, \end{gathered}$$

with $\beta_k > 0$. This can be done by evaluating the associated moments

$$\mu_k = \int_0^{+\infty} t^k w(t) dt, \quad k \geq 0, \tag{7}$$

and then by using the Chebyshev algorithm [4, sect. 2.3]. The coefficients α_k, β_k, $k \geq 0$, define the tridiagonal symmetric Jacobi matrix, whose eigenvalue decomposition provides abscissas and weights of the quadrature rule. This final step is efficiently implemented by the famous Golub and Welsch algorithm [7].

In order to approximate the quadrature error, we consider the corresponding generalized averaged Gaussian rules (see [14, 12, 13, 3]). These formulas provide an a posteriori estimate of the error. Moreover, they are easy to construct and typically lead to quite accurate approximations (see [12]).

We point out that all the results presented in the paper can be easily extended to the case of integrals as in (1), with the cosine replaced by the sine function (see also Remark 2).

The Matlab codes for the computation of integrals of type (2) by using the approximation (6) are available at https://github.com/EleonoraDe/Fourier-type-integrals.

The paper is organized as follows. In Section 2 we derive a recursive relation for the evaluation of the moments and show how to construct the coupled Gaussian formula. Some numerical experiments, in which we compare the rule with other methods, are provided in Section 3. In Section 4 we employ the generalized averaged Gaussian formulas to obtain an a posteriori estimate of the quadrature error. Concluding remarks can be found in Section 5.

2 Construction of the Gaussian formula

In order to develop the Gaussian quadrature rule, relative to the weight function (3), first of all we need to compute the moments

$$\mu_k = \int_0^{+\infty} t^{\alpha+k-1} e^{-ct} (\cos t + 1)\, dt, \quad k \geq 0. \tag{8}$$

Proposition 1. The following recursion holds

$$\begin{aligned} \mu_0 &= \frac{\Gamma(\alpha)}{c^\alpha} \left(\cos(\alpha\varphi)(\cos\varphi)^\alpha + 1\right), \\ \mu_k &= \frac{k-1+\alpha}{c} \frac{\cos((k+\alpha)\varphi)(\cos\varphi)^{k+\alpha} + 1}{\cos((k-1+\alpha)\varphi)(\cos\varphi)^{k-1+\alpha} + 1} \mu_{k-1}, \quad k \geq 1, \end{aligned} \tag{9}$$

with $\varphi = \arctan\frac{1}{c}$ and where Γ is the Gamma function.

Proof. First of all, from [8, p.490, 3.944, n.6], for the so called core moments (see [6, sect. 2.1]), defined by

$$\mu_{k,0} = \int_0^{+\infty} t^{\alpha+k-1} e^{-ct} \cos t\, dt, \quad k \geq 0, \tag{10}$$

we have that

$$\mu_{k,0} = \frac{\Gamma(k+\alpha)}{(1+c^2)^{\frac{k+\alpha}{2}}} \cos((k+\alpha)\varphi), \quad k \geq 0,\ \varphi = \arctan\frac{1}{c}.$$

Then, by definitions (8)-(10) and by using [8, sect. 3.381, n.4], for the moments μ_k, $k \geq 0$, we have

$$\begin{aligned} \mu_k &= \mu_{k,0} + \int_0^{+\infty} t^{\alpha+k-1} e^{-ct} dt \\ &= \Gamma(k+\alpha) \left(\frac{\cos((k+\alpha)\varphi)}{(1+c^2)^{\frac{k+\alpha}{2}}} + \frac{1}{c^{k+\alpha}} \right) \\ &= \Gamma(k+\alpha) \frac{\cos((k+\alpha)\varphi) c^{k+\alpha} + (1+c^2)^{\frac{k+\alpha}{2}}}{(c(1+c^2))^{\frac{k+\alpha}{2}}} \end{aligned} \tag{11}$$

Defining, for simplicity of notation,

$$d_k := \frac{\cos((k+\alpha)\varphi) c^{k+\alpha} + (1+c^2)^{\frac{k+\alpha}{2}}}{(c(1+c^2))^{\frac{k+\alpha}{2}}}, \quad k \geq 0,$$

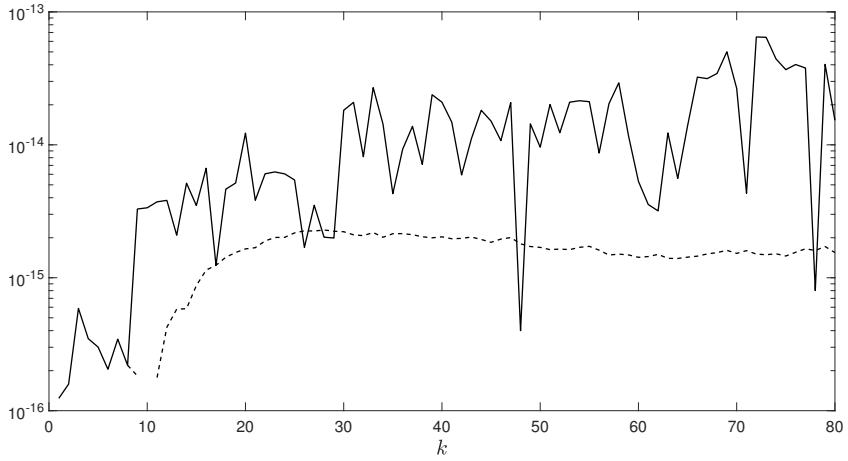


Figure 1: Comparison between formula (9) (dashed line) and formula (11) (solid line) for the computation of the first 80 moments, with $\alpha = 0.7$ and $c = 0.1$. The plots show the relative errors with respect to a reference value computed by employing formula (9) with extended precision arithmetic (50 digits).

so that $\mu_k = \Gamma(k+\alpha)d_k$, after some computations, we have that

$$\frac{d_k}{d_{k-1}} = \frac{1}{c}\frac{\cos((k+\alpha)\varphi)(\cos\varphi)^{k+\alpha}+1}{\cos((k-1+\alpha)\varphi)(\cos\varphi)^{k-1+\alpha}+1}.$$

By using the above relation and formula (11), we obtain the result. □

We notice that (11) gives an explicit expression for the computation of the moments μ_k, $k \geq 0$. Anyway, this formula involves the evaluation of the Gamma function and, from our numerical experiments, for growing k it seems a little less stable than the recursive relation (9) (see Figure 1).

Remark 2. A similar result can be derived also in the case the cosine in (2) is replaced by the sine function. In this situation, for the moments

$$\tilde{\mu}_k = \int_0^{+\infty} t^{\alpha+k-1}e^{-ct}\,(\sin t + 1)\,dt, \quad k \geq 0,$$

it holds

$$\tilde{\mu}_0 = \frac{\Gamma(\alpha)}{c^\alpha}\,(\sin(\alpha\varphi)(\cos\varphi)^\alpha + 1)\,,$$
$$\tilde{\mu}_k = \frac{k-1+\alpha}{c}\frac{\sin((k+\alpha)\varphi)(\cos\varphi)^{k+\alpha}+1}{\sin((k-1+\alpha)\varphi)(\cos\varphi)^{k-1+\alpha}+1}\mu_{k-1}, \quad k \geq 1,$$

with $\varphi = \arctan\frac{1}{c}$ (see [8, p.490, 3.944, n.5]).

At this point, for the computation of the coefficients α_k and β_k of the recurrence relation

$$\pi_{k+1}(t) = (t-\alpha_k)\pi_k(t) - \beta_k\pi_{k-1}(t), \quad k \geq 0,$$
$$\pi_{-1}(t) = 0, \quad \pi_0(t) = 1,$$

with $\beta_k > 0$, we employ the Chebyshev algorithm (see [4, sect. 2.3] and [5]). Given the first $2n$ moments $\mu_0, \ldots, \mu_{2n-1}$, this algorithm uniquely determines the first n recurrence coefficients α_k and β_k, $k = 0, \ldots, n-1$, by using the mixed moments defined as

$$\sigma_{kl} = \int_0^\infty \pi_k(t) t^l w(t) dt, \quad k, l \geq -1.$$

The Chebyshev algorithm is summarized in Algorithm 3.

Algorithm 3. Initialization

$$\begin{aligned} \alpha_0 &= \frac{\mu_1}{\mu_0}, \ \beta_0 = \mu_0, \\ \sigma_{-1,l} &= 0, \quad l = 1, 2, \ldots, 2n-2, \\ \sigma_{0,l} &= \mu_l, \quad 0, 1, \ldots, 2n-1, \end{aligned}$$

for $k = 1, 2, \ldots, n-1$

for $\quad l = k, k+1, \ldots, 2n-k-1$

$$\begin{aligned} \sigma_{k,l} &= \sigma_{k-1,l+1} - \alpha_{k-1}\sigma_{k-1,l} - \beta_{k-1}\sigma_{k-2,l}, \\ \alpha_k &= \frac{\sigma_{k,k+1}}{\sigma_{k,k}} - \frac{\sigma_{k-1,k}}{\sigma_{k-1,k-1}}, \quad \beta_k = \frac{\sigma_{k,k}}{\sigma_{k-1,k-1}}. \end{aligned}$$

After the computation of α_k, β_k, $k = 0, \ldots, n-1$, the eigendecomposition of the corresponding Jacobi matrix

$$J_n = \begin{bmatrix} \alpha_0 & \sqrt{\beta_1} & & & 0 \\ \sqrt{\beta_1} & \alpha_1 & \sqrt{\beta_2} & & \\ & \sqrt{\beta_2} & \alpha_2 & \ddots & \\ & & \ddots & \ddots & \sqrt{\beta_{n-1}} \\ 0 & & & \sqrt{\beta_{n-1}} & \alpha_{n-1} \end{bmatrix} \in \mathbb{R}^{n \times n}, \tag{12}$$

provides the nodes $t_i^{(n)}$ and weights $w_i^{(n)}$, $i = 1, \ldots, n$, of the n-point Gaussian rule (see [1]). Then, for the computation of integral (4) we use the approximation

$$I^C(f) = I_n^C(f) + E_n^C(f) = \sum_{i=1}^n w_i^{(n)} f\left(t_i^{(n)}\right) + E_n^C(f). \tag{13}$$

As for integral (5), denoting by $\lambda_i^{(n)}$, $\xi_i^{(n)}$, $i = 1, \ldots, n$, the nodes and weights of the n-point generalized Gauss-Laguerre rule, we consider the approximation

$$I^L(f) = I_n^L(f) + E_n^L(f) = \frac{1}{c^\alpha} \sum_{i=1}^n \xi_i^{(n)} f\left(\frac{\lambda_i^{(n)}}{c}\right) + E_n^L(f). \tag{14}$$

In (13) and (14) $E_n^C(f)$ and $E_n^L(f)$ denote the corresponding quadrature errors. Finally, for integral (2) we obtain the rule (6), in which $E_n(f) = E_n^C(f) - E_n^L(f)$, that we call coupled Gaussian formula.

3 Numerical experiments

In this section we provide some numerical examples in which we test the performances of the developed coupled Gaussian rule. We start by observing that, by using the change of variable $x = ct$ in (2), which leads to

$$I(f) = I_c(h) = \frac{1}{c^\alpha} \int_0^{+\infty} h(x) x^{\alpha-1} e^{-x} dx, \quad h(x) = f\left(\frac{x}{c}\right) \cos\left(\frac{x}{c}\right),$$

and by considering as weight function

$$w^{GL}(x) = x^{\alpha-1} e^{-x},$$

integral (2) can be evaluated by employing the n-point generalized Gauss-Laguerre formula, that is,

$$I_n^{GL}(h) = \frac{1}{c^\alpha} \sum_{i=1}^{n} h\left(\lambda_i^{(n)}\right) \xi_i^{(n)}, \tag{15}$$

so that $I(f) = I_n^{GL}(f) + E_n^{GL}(h)$, where $E_n^{GL}(h)$ denotes the quadrature error. As for the computation of the nodes and weights of the generalized Gauss-Laguerre rule, we have used the Matlab routine `lagpts.m` of the Chebfun package (see. [2]) In this view, in Figures 2-3-4 we compare the behavior of the coupled Gaussian rule (6) with (15), with respect to a reference solution. In other words, as a first set of experiments we compare formula (6) with the Laguerre rule in which the oscillating term is not part of the weight function. Different sets of parameters and functions f are considered. All the computations are carried out in Matlab by using extended precision arithmetic. Indeed, it is known that the computation of the coefficients α_k, β_k is a severely ill-conditioned problem, even for k not too large (see e.g. [5]).

We remark that formula (6) requires a double set of points and therefore a double number of function evaluations. Nevertheless, by looking at the figures, we observe that this formula is typically more accurate than the generalized Gauss-Laguerre formula, especially for small values of the parameter c (see Figures 2a-3-4a). Recalling that $c = \frac{\beta}{\omega}$ in (2), this parameter handles the scale and, therefore, the frequency of oscillations (cf. (1)-(2)). For large c method (6) is less effective since the Laguerre rule appears to be reliable for slow oscillations (see Figures 2b-4b).

As already mentioned in the Introduction, integral (1) can be interpreted as the cosine transform

$$I(F, \omega) = \int_0^{+\infty} F(x) \cos(\omega x) dx, \tag{16}$$

with $F(x) = g(x) x^{\alpha-1} e^{-\beta x}$. An efficient method for the computation of (16) where the function F is slowly decaying is based on the use of the double exponential transformation (see [10, 9])

$$x = \frac{M\Phi\left(t - \frac{\pi}{2M}\right)}{\omega}, \quad M > 0, \quad \Phi(\zeta) = \frac{\zeta}{1 - e^{-2\pi \sinh \zeta}},$$

that leads to

$$I(F, \omega) = \frac{M}{\omega} \int_{-\infty}^{+\infty} F\left(\frac{M\Phi\left(t - \frac{\pi}{2M}\right)}{\omega}\right) \cos\left(M\Phi\left(t - \frac{\pi}{2M}\right)\right) \Phi'\left(t - \frac{\pi}{2M}\right) dt.$$

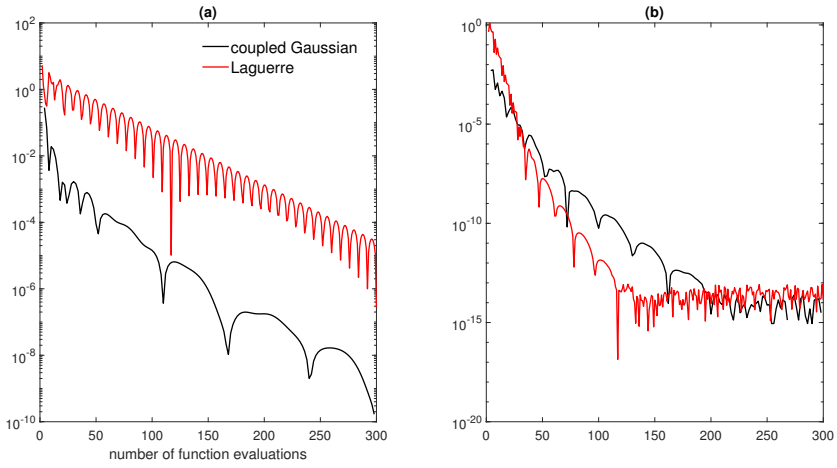


Figure 2: Comparison between the absolute error obtained by using the coupled Gaussian approach (6) and the Gauss-Laguerre formula (15) for $\alpha = 1.1$, $c = 0.2$ (left) and $\alpha = 0.5$, $c = 0.4$ (right). In both cases $f(t) = \frac{1}{1+e^{-t}}$.

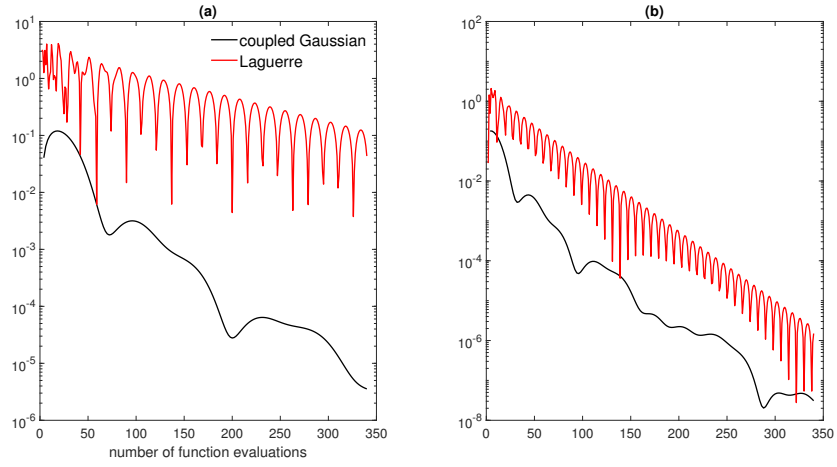


Figure 3: Comparison between the absolute error obtained by using the coupled Gaussian approach (6) and the Gauss-Laguerre formula (15) for $\alpha = 1.5$, $c = 0.05$ (left) and $\alpha = 1.3$, $c = 0.1$ (right). In both cases $f(t) = \frac{1}{1+t}$.

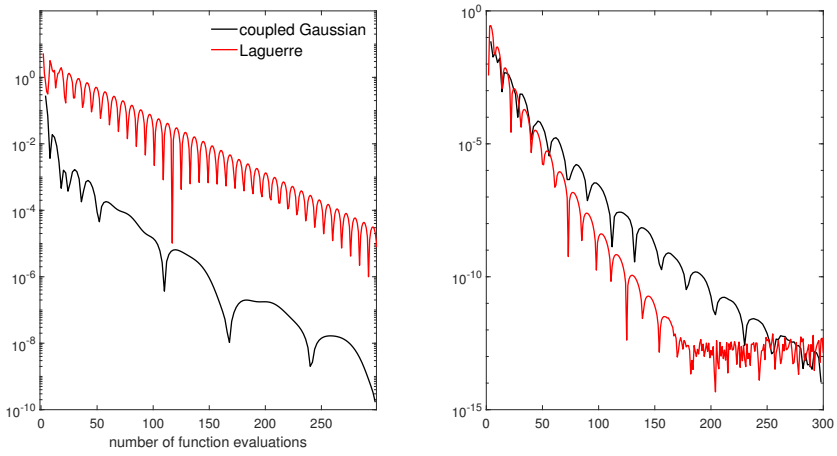


Figure 4: Comparison between the absolute error obtained by using the coupled Gaussian approach (6) and the Gauss-Laguerre formula (15) for $\alpha = 0.5$, $c = 0.2$ (left) and $\alpha = 1.3$, $c = 0.7$ (right). In both cases $f(t) = e^{-0.5t^2}$.

The idea was then to consider the trapezoidal rule with step τ and suitable truncation. By setting $M\tau = \pi$ as in [11], the method reads

$$I_N(F) = \frac{\pi}{\omega} \sum_{n=-N}^{N} F\left(\frac{M\Phi\left(\left(n-\frac{1}{2}\right)\frac{\pi}{M}\right)}{\omega}\right) \cos\left(M\Phi\left(\left(n-\frac{1}{2}\right)\frac{\pi}{M}\right)\right) \times \Phi'\left(\left(n-\frac{1}{2}\right)\frac{\pi}{M}\right). \tag{17}$$

This rule can be very efficient but requires the proper selection of τ (M) and N. This corresponds to locate the significant support of the function with respect to the required accuracy and to define a suitable discretization. As for the sine transform

$$I(F,\omega) = \int_0^{+\infty} F(x)\sin(\omega x)dx,$$

the method is almost identical with the only difference in the initial substitution, that now reads

$$x = \frac{M\Phi(t)}{\omega}.$$

Assuming $\omega = 1$ and taking $F(x) = f(x)x^{\alpha-1}e^{-cx}$, we have $I(F,\omega) = I(f)$ (cf. (2)-(16)). In this setting, in Figures 5-6-7 we report some results, where we compare our coupled Gaussian method with the trapezoidal rule for different sets of parameters. In all pictures we consider the results of the trapezoidal rule for $M = 4, 8, 12, \ldots$ in order to reduce the step, and then $N = \frac{M}{2}, M, \frac{3}{2}M$ to work with increasing number of points, that is, by reducing the truncation errors. As for the coupled Gaussian approach, we truncate rules (13)-(14) by neglecting the terms for which the values of the weights $w_i^{(n)}$, $\lambda_i^{(n)}$, $i = 1, \ldots, n$, are less than $1e-16$. Depending on the function f and on the parameters, the trapezoidal rule appears to be extremely sensitive with respect to the choice of N. For small N it shows a very fast initial convergence but also stagnation, while for larger N the attainable accuracy is higher, but the method is slower (see Figure 5). Basically, the Gaussian approach appears preferable for α, c and f such that the

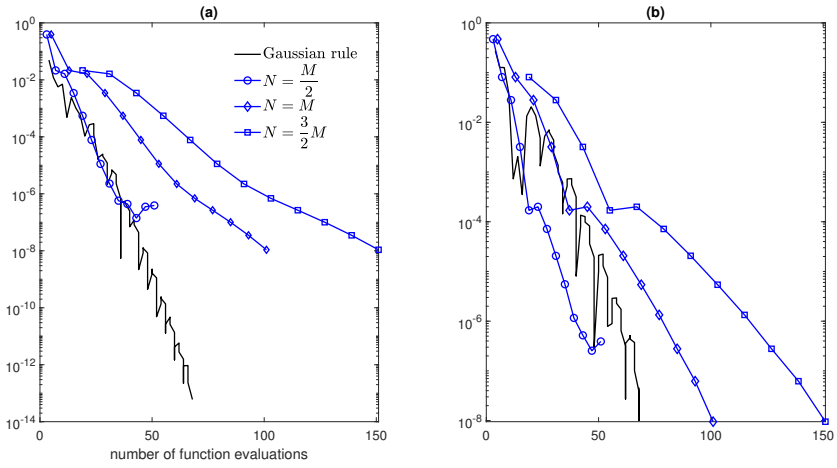


Figure 5: Comparison between the absolute error obtained by using the coupled Gaussian rule (6) and the trapezoidal rule based on the double exponential transform (17) for $\alpha = 1.3$, $c = 0.7$ (left) and $\alpha = 1.3$, $c = 0.3$ (right). In both cases $f(t) = e^{-0.5t^2}$.

support is relatively large (Figures 5-6a), whereas the trapezoidal rule is more effective for functions rapidly decaying (Figure 6b and other experiments non reported). Figure 7 shows a situation in which both methods are not much accurate because of the high frequency of oscillations.

4 A posteriori error estimate

In this section, we construct the generalized averaged Gaussian rules A^C_{2n+1}, A^L_{2n+1} (see [14]), associated with the Gaussian formulas I^C_n, I^L_n, respectively, and employ them to approximate the quadrature error

$$E_n(f) = I(f) - \left(I^C_n(f) - I^L_n(f)\right). \tag{18}$$

The generalized averaged Gaussian formula A_{2n+1}, associated with a generic Gaussian rule I_n is given by (see [12])

$$A_{2n+1}(f) = \frac{b_{n+1}}{b_n + b_{n+1}} I_n(f) + \frac{b_n}{b_n + b_{n+1}} \tilde{A}_{n+1}(f),$$

where the quadrature formula

$$\tilde{A}_{n+1}(f) = \sum_{i=1}^{n+1} \sigma_i^{(n+1)} f\left(\tau_i^{(n+1)}\right) \tag{19}$$

arises from the symmetric tridiagonal matrix $\tilde{J}_{n+1} \in \mathbb{R}^{(n+1)\times(n+1)}$, defined as

$$\tilde{J}_{n+1} = \begin{bmatrix} J_n & e_n\sqrt{b_n + b_{n+1}} \\ e_n^T\sqrt{b_n + b_{n+1}} & a_n \end{bmatrix}, \tag{20}$$

in which $e_n = (0,\ldots,0,1)^T \in \mathbb{R}^n$, J_n is as in (12) and a_k, b_k, $k \geq 0$, are the coefficients of the corresponding three-term recurrence relation of the orthogonal system $\{p_k\}_{k\geq 0}$ associated with

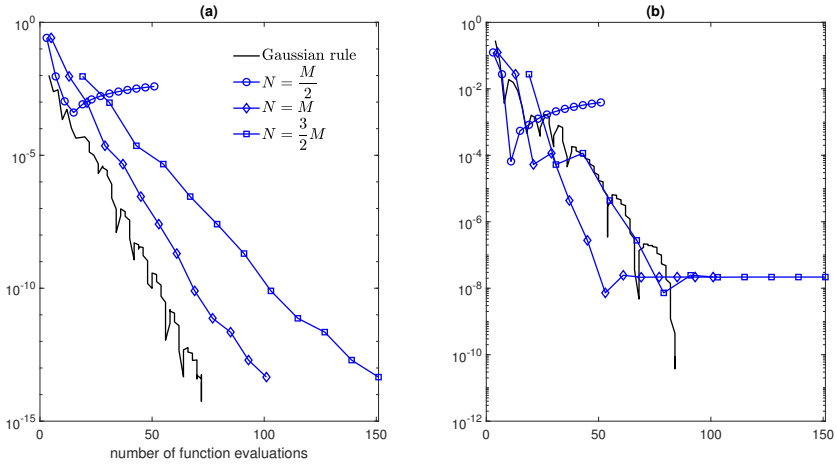


Figure 6: Comparison between the absolute error obtained by using the coupled Gaussian rule (6) and the trapezoidal rule based on the double exponential transform (17) for $\alpha = 0.5$, $c = 0.4$ (left) and $\alpha = 0.5$, $c = 0.1$ (right). In both cases $f(t) = \frac{1}{1+e^{-t}}$.

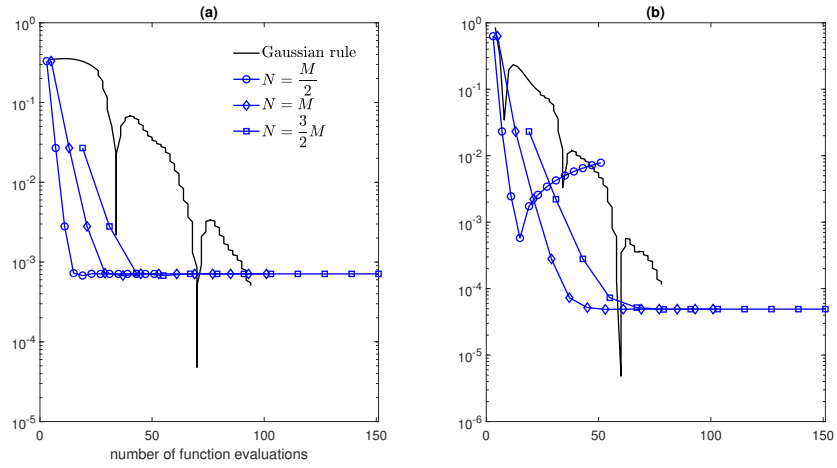


Figure 7: Comparison between the absolute error obtained by using the developed Gaussian rule (6) and the trapezoidal rule based on the double exponential transform (17) for $\alpha = 1.3$, $c = 0.05$ (left) and $\alpha = 0.5$, $c = 0.1$ (right). In both cases $f(t) = \frac{1}{1+t^2}$.

I_n, that is,

$$p_{k+1}(t) = (t-a_k)p_k(t) - b_k p_{k-1}(t), \quad k \geq 0,$$
$$p_{-1}(t) = 0, \quad p_0(t) = 1.$$

By construction, formula (19) has the following properties (see [14]):

1. $\sigma_i^{(n+1)} > 0$, $i = 1, \ldots, n+1$;
2. the nodes $\tau_i^{(n+1)}$ are all real and are interlaced by those of I_n;
3. $\tau_i^{(n+1)} \in [0, +\infty)$, for $i \geq 2$;
4. $\tau_1^{(n+1)} \in [0, \infty)$ if and only if

$$\frac{p_{n+1}(0)}{p_{n-1}(0)} \geq b_{n+1}, \quad n \geq 1. \tag{21}$$

Let A^C_{2n+1} and A^L_{2n+1} be the generalized averaged Gaussian rules corresponding to I^C_n and I^L_n, respectively. In this way error (18) is finally estimated as

$$E_n(f) \approx \left(A^C_{2n+1}(f) - A^L_{2n+1}(f)\right) - \left(I^C_n(f) - I^L_n(f)\right). \tag{22}$$

In Figures 8-9 we show the reliability of the above estimate on some examples. For the Gauss-Laguerre rule I^L_n, the expressions of the corresponding orthogonal polynomials and the values of the recurrence coefficients are explicitly known. Indeed, it has been verified that (21) holds true if and only if $\alpha \geq 2$ (see [13]). As for the Gaussian formula I^C_n, since we do not have at disposal an analytical expression of the corresponding orthogonal polynomials and of the recurrence coefficients, relation (21) can only be verified numerically. The experiments show that it is not always true. Nevertheless, we remark that, even if in some cases for the rules $\tilde{A}^C_{n+1}$, $\tilde{A}^L_{n+1}$ condition (21) does not hold, experimentally (for the functions considered) the resulting formulas appear to provide fairly good approximations of the quadrature error.

5 Conclusion

In this work we have considered the construction of a coupled Gaussian formula for weight functions involving powers, exponentials and oscillating functions. We have compared this new approach with the Laguerre rule and a particular double exponential trapezoidal formula. The results show that in some situations the developed rule improves the other two methods. A practical error estimate, based on the use of the generalized averaged Gaussian rule, has been presented and tested with good results.

Acknowledgments

This work was partially supported by GNCS-INdAM and FRA-University of Trieste. The authors are member of the INdAM research group GNCS. Eleonora Denich thanks the University of Trieste for the support, under grant "Programma Regionale (PR) FSE+ 2021/2027 della Regione Autonoma Friuli Venezia Giulia - PPO 2023 - Programma specifico 22/23 - Avviso emanato con decreto n.17895/GRFVG dd.19.04.2023 s.m.i., Linea C) Sportello 2023".

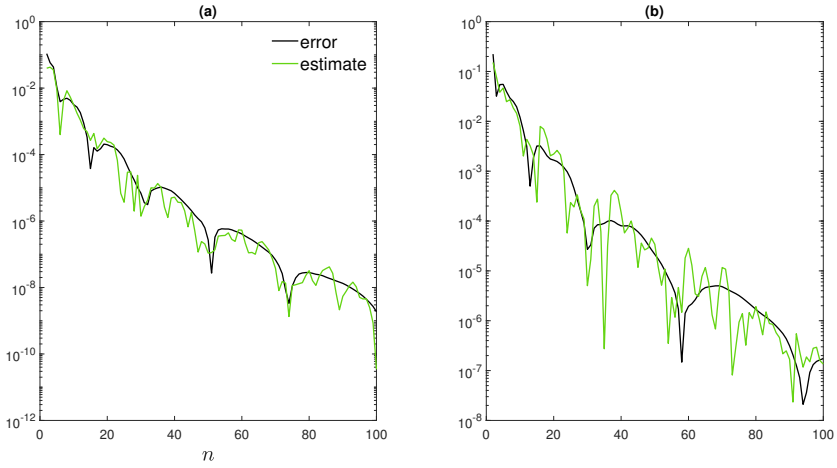


Figure 8: The absolute values of the error $E_n(f)$ and its estimate (22) with $\alpha = 0.7$, $c = 0.5$ (left) and $\alpha = 1.3$, $c = 0.3$. In both cases $f(t) = \frac{1}{1+t^2}$.

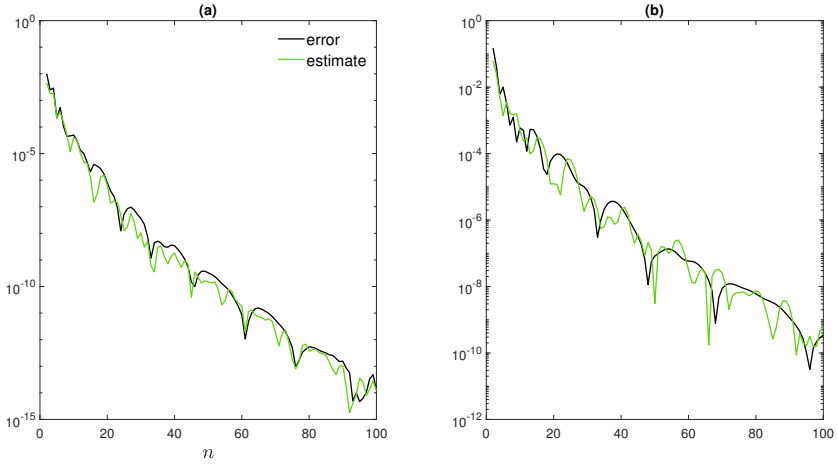


Figure 9: The absolute values of the error $E_n(f)$ and its estimate (22) with $\alpha = 0.5$, $c = 0.4$ (left) and $\alpha = 1.1$, $c = 0.2$. In both cases $f(t) = \frac{1}{1+e^{-t}}$.

Declaration of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] P. J. Davis and P. Rabinowitz. *Methods of numerical integration*. Computer Science and Applied Mathematics, (Academic Press, Inc., Orlando, FL1984), second edn. ISBN 0-12-206360-0

[2] T. A. Driscoll, N. Hale, and L. N. Trefethen. *Chebfun Guide* (2014). URL https://www.chebfun.org/docs/guide/

[3] L. Fermo, L. Reichel, G. Rodriguez, and M. M. Spalević. *Averaged Nyström interpolants for the solution of Fredholm integral equations of the second kind*. Appl. Math. Comput., vol. 467, (2024), Paper No. 128482, 20. ISSN 0096-3003,1873-5649. URL http://dx.doi.org/10.1016/j.amc.2023.128482

[4] W. Gautschi. *On generating orthogonal polynomials*. SIAM J. Sci. Statist. Comput., vol. 3, 3, (1982), 289–317. ISSN 0196-5204. URL http://dx.doi.org/10.1137/0903018

[5] W. Gautschi. *Orthogonal polynomials: applications and computation*. vol. 5, (1996), 45–119. URL http://dx.doi.org/10.1017/S0962492900002622

[6] W. Gautschi. *Computing polynomials orthogonal with respect to densely oscillating and exponentially decaying weight functions and related integrals*. J. Comput. Appl. Math., vol. 184, 2, (2005), 493–504. ISSN 0377-0427,1879-1778. URL http://dx.doi.org/10.1016/j.cam.2005.01.023

[7] G. H. Golub and J. H. Welsch. *Calculation of Gauss quadrature rules*. Math. Comp., vol. 23, (1969), 221–230. ISSN 0025-5718,1088-6842. URL http://dx.doi.org/10.2307/2004418

[8] I. S. Gradshteyn and I. M. Ryzhik. *Table of integrals, series, and products*, (Academic Press [Harcourt Brace Jovanovich, Publishers], New York-London-Toronto1980), enlarged edn. ISBN 0-12-294760-6. Translated from the Russian

[9] M. Mori and T. Ooura. *Double exponential formulas for Fourier type integrals with a divergent integrand*. *Contributions in numerical mathematics*, *World Sci. Ser. Appl. Anal.*, vol. 2, (World Sci. Publ., River Edge, NJ1993). ISBN 981-02-1437-5, 301–308. URL http://dx.doi.org/10.1142/9789812798886_0023

[10] T. Ooura and M. Mori. *The double exponential formula for oscillatory functions over the half infinite interval*. Journal of Computational and Applied Mathematics, vol. 38, 1, (1991), 353–360. ISSN 0377-0427. URL http://dx.doi.org/https://doi.org/10.1016/0377-0427(91)90181-I

[11] T. Ooura and M. Mori. *A robust double exponential formula for Fourier-type integrals* (1999). 229–241. Numerical evaluation of integrals, URL http://dx.doi.org/10.1016/S0377-0427(99)00223-X

[12] L. Reichel and M. M. Spalević. *A new representation of generalized averaged Gauss quadrature rules*. Appl. Numer. Math., vol. 165, (2021), 614–619. ISSN 0168-9274,1873-5460. URL http://dx.doi.org/10.1016/j.apnum.2020.11.016

[13] L. Reichel and M. M. Spalević. *Averaged Gauss quadrature formulas: properties and applications*. J. Comput. Appl. Math., vol. 410, (2022), Paper No. 114232, 18. ISSN 0377-0427,1879-1778. URL http://dx.doi.org/10.1016/j.cam.2022.114232

[14] M. M. Spalević. *On generalized averaged Gaussian formulas*. Math. Comp., vol. 76, 259, (2007), 1483–1492. ISSN 0025-5718,1088-6842. URL http://dx.doi.org/10.1090/S0025-5718-07-01975-8

[15] S. H. Ward and G. W. Hohmann. *4. Electromagnetic Theory for Geophysical Applications*, (Society of Exploration Geophysicists1988). Investigations in Geophysics. ISBN 978-0-931830-51-8ISBN, 130–311. URL http://dx.doi.org/10.1190/1.9781560802631.ch4

[16] E. T. Whittaker and G. N. Watson. *A Course of Modern Analysis*. Cambridge Mathematical Library, (Cambridge University Press1996), 4 edn. URL http://dx.doi.org/10.1017/CBO9780511608759