A new efficient explicit Deferred Correction framework: analysis and applications to hyperbolic PDEs and adaptivity

L. Micalizzi*and D.Torlo†

October 6, 2022

Abstract

The Deferred Correction is an iterative procedure used to design numerical methods for systems of ODEs, characterized by an increasing accuracy at each iteration. The main advantage of this framework is the automatic way of getting arbitrarily high order methods, which can be put in Runge-Kutta form, based on the definition of subtimenodes in each timestep. The drawback is a larger computational cost with respect to the most used Runge-Kutta methods. To reduce such cost, in an explicit setting, we propose an efficient modification: we remove the unnecessary subtimenodes in all the iterations, introducing interpolation processes between them. We provide the Butcher tableaux of the novel methods and we study their stability, showing that in some cases the computational advantage does not affect the stability. The flexibility of the novel modification allows nontrivial applications to PDEs and construction of adaptive methods. The good performances of the introduced methods are broadly tested on several benchmarks both in the ODE and PDE settings.

1 Introduction

A huge amount of phenomena in many different fields, e.g. engineering, physics, chemistry, biology or social sciences, can be modeled through ordinary and partial differential equations (ODEs and PDEs) whose analytical solution is usually not available. Hence, many numerical methods have been developed to approximate such solutions in a very accurate way. With modern computers and technologies the speed of the simulations has dramatically dropped, but this is still not enough for very complicate problems, for which computational costs are also nowadays too expensive. That is why any effort in reducing the computational costs for numerical methods is of paramount importance.

A classical way of reducing them is the adoption of high order methods. Such methods allow to reach lower errors within way coarser discretizations. Moreover, in the context of PDEs, they allow to introduce less diffusion and to quickly catch complicated structures that low order methods struggle in capturing. For ODEs there is a vast collection of high order methods, e.g. Runge-Kutta (RK) methods [12, 31], multistep methods [31] and predictor-corrector methods [28]. Also for PDEs several techniques have been proposed to obtain high order accuracy either by separating

*Affiliation: Institute of Mathematics, University of Zurich, Winterthurerstrasse 190, Zurich, 8057, Switzerland. Email: lorenzo.micalizzi@math.uzh.ch.

†Affiliation: SISSA mathLab, SISSA, via Bonomea 265, Trieste, 34136, Italy. Email: davide.torlo@sissa.it.

the spatial and time discretizations via the method of lines, e.g. finite element methods (FEM) [22, 34, 17], finite volume (FV) [42, 23, 61, 29], finite difference (FD) [41, 29] and spectral methods [30], or through more involved space-time methods, e.g. ADER-DG [7, 27, 10], Deferred Correction-Residual Distribution methods [5, 2, 4], space-time parallel solvers [57].

A wide series of arbitrarily high order methods is based on the Deferred Correction (DeC) approach. The original method has been firstly introduced in [24] in a simple prediction-correction time integrator framework. A more elegant explicit version based on spectral elements in time was introduced in 2000 [21], characterized by an iterative procedure allowing to increase the order of accuracy by one at each iteration. In 2003 [47], Minion generalized the DeC framework to obtain an implicit-explicit arbitrarily high order method, with various applications to ODEs and PDEs [48, 40, 35, 46, 58]. Later on, the DeC approach has been generalized by Abgrall [5] to solve hyperbolic PDEs with high order continuous Galerkin (CG) FEM spatial discretizations, overcoming the burden related to the mass matrix which is typical in this context. This allowed to build arbitrarily high order FEM for hyperbolic PDEs with computational costs comparable to the FV or DG methods, leading to numerous applications in the hyperbolic field [2, 4, 45, 16]. The DeC has been also modified in order to preserve physical structures (positivity, entropy) [49, 3]. Finally, in [32] it has been pointed out that DeC and ADER methods are very similar iterative time integrators and, when restricted to ODEs, they can be written as RK schemes, see also [37, 60].

The clear advantage of the DeC framework is the possibility to easily increase the order of accuracy, the drawback is the expensive computational cost, due to the iterations and the high order reconstruction for each time-step. Both of them scale as the aimed order of accuracy and this makes the computational costs increase quadratically in the order. To alleviate the cost, the *ladder* strategy was proposed in implicit DeC algorithms [47, 40, 58], where the reconstruction in time increases the accuracy at each iteration. Between the iterations, an interpolation procedure links the different reconstructions. Though being the idea used in some works, it has never been deeply studied and analyzed, in particular, for the purely explicit DeC.

Inspired by this idea, in this work, we provide a detailed description of two novel explicit families of efficient DeC methods, where the interpolation process is applied either on the state solutions or on their evolution operators. These DeC methods can be interpreted as RK methods by explicitly constructing their Butcher tableaux and studying their stability, we show that in some cases the efficient version and the classical one have the same stability functions. Moreover, we exploit the novel framework to build an adaptive DeC that, given a certain tolerance, automatically chooses the order of accuracy to reach such error in the most efficient way. We also see how this framework can be applied to the CG DeC framework [5] in the context of hyperbolic PDEs with different ways to avoid the costs induced by the mass matrix in the timestepping.

The structure of this work is the following. We start by introducing the DeC method in an abstract framework in section 2 and as a general strategy to numerically solve systems of ordinary differential equations in section 3. In section 4, we introduce the new families of efficient DeC methods. Then, we show their Butcher tableaux in section 5 and in section 6 we study in detail their linear stability. In section 7, we describe the application to the numerical solution of hyperbolic problems with CG spatial discretizations avoiding mass matrices. We propose an adaptive and efficient version of the DeC that automatically selects the order of accuracy in section 8. In section 9, we present numerical results on ODEs and hyperbolic PDEs with various comparisons with the classical DeC methods. Section 10 is dedicated to the conclusions.

2 DeC in the abstract framework

The DeC has been originally introduced in 1949 in [24], afterward studied in different works [52, 25, 26, 59, 56] and then proposed again by Dutt et al. in 2000 in [21] in a spectral version as an iterative procedure to numerically solve systems of ordinary differential equations with arbitrary high order accuracy. Many modifications have been proposed in the following years to apply it to different fields [38, 47, 11, 48, 35, 14, 15, 33, 58, 8, 5, 9, 2, 4, 49, 16, 32, 3].

We will first introduce it in an abstract context as proposed in [5]. Assume that we have two general operators depending on a parameter Δ between two normed vector spaces $(X, \|\cdot\|_X)$ and $(Y, \|\cdot\|_Y)$

$$\mathcal{L}^1_\Delta, \mathcal{L}^2_\Delta : X \longrightarrow Y. \tag{1}$$

Remark 2.1. *Even if we are still in an abstract context and not in the specific case of an evolution problem (like for example an ordinary differential equation) it is useful, in order to make the concepts clearer, to give an idea of the objects we are working with. Imagine that we want to numerically solve a Cauchy problem for a system of ODEs, then $\mathcal{L}^2_\Delta$ is a high order implicit operator and $\mathcal{L}^1_\Delta$ is an explicit low order operato, for example obtained by using an explicit Euler approximation. We would like to solve $\mathcal{L}^2_\Delta$ i.e. finding $\underline{\boldsymbol{u}} \in X$ such that $\mathcal{L}^2_\Delta(\underline{\boldsymbol{u}}) = \mathbf{0}_Y$ but this is not so easy given the implicit nature of the operator. On the other hand the explicit operator $\mathcal{L}^1_\Delta$ is very easy to solve (more in general it is easy to solve $\mathcal{L}^1_\Delta(\underline{\boldsymbol{u}}) = \underline{\boldsymbol{r}}$ with $\underline{\boldsymbol{r}} \in Y$ given) but it is a low order operator. In such context the parameter Δ is the step size, while it represents the characteristic mesh size in the application to hyperbolic systems of balance laws.*

In the next theorem we will provide a recipe to get an arbitrary high order approximation of the solution of $\mathcal{L}^2_\Delta$ in an explicit way by combining the operators $\mathcal{L}^1_\Delta$ and $\mathcal{L}^2_\Delta$ through an iterative procedure.

Theorem 2.1 (DeC accuracy)**.** *Let the following hypotheses hold*

1. ***Existence of a unique solution to $\mathcal{L}^2_\Delta$***
   $\exists!\, \underline{\boldsymbol{u}}_\Delta \in X$ *solution of* $\mathcal{L}^2_\Delta$ *such that* $\mathcal{L}^2_\Delta(\underline{\boldsymbol{u}}_\Delta) = \mathbf{0}_Y$*;*

2. ***Coercivity-like property of $\mathcal{L}^1_\Delta$***
   $\exists\, \alpha_1 \geq 0$ *independent of* Δ *s.t.*

$$\left\|\mathcal{L}^1_\Delta(\underline{\boldsymbol{v}}) - \mathcal{L}^1_\Delta(\underline{\boldsymbol{w}})\right\|_Y \geq \alpha_1 \left\|\underline{\boldsymbol{v}} - \underline{\boldsymbol{w}}\right\|_X, \ \forall \underline{\boldsymbol{v}}, \underline{\boldsymbol{w}} \in X; \tag{2}$$

3. ***Lipschitz-continuity-like property of $\mathcal{L}^1_\Delta - \mathcal{L}^2_\Delta$***
   $\exists\, \alpha_2 \geq 0$ *independent of* Δ *s.t.*

$$\left\|\left(\mathcal{L}^1_\Delta(\underline{\boldsymbol{v}}) - \mathcal{L}^2_\Delta(\underline{\boldsymbol{v}})\right) - \left(\mathcal{L}^1_\Delta(\underline{\boldsymbol{w}}) - \mathcal{L}^2_\Delta(\underline{\boldsymbol{w}})\right)\right\|_Y \leq \alpha_2 \Delta \left\|\underline{\boldsymbol{v}} - \underline{\boldsymbol{w}}\right\|_X, \ \forall \underline{\boldsymbol{v}}, \underline{\boldsymbol{w}} \in X. \tag{3}$$

Then, if we iteratively define $\underline{\boldsymbol{u}}^{(p)}$ as the solution of

$$\mathcal{L}^1_\Delta(\underline{\boldsymbol{u}}^{(p)}) = \mathcal{L}^1_\Delta(\underline{\boldsymbol{u}}^{(p-1)}) - \mathcal{L}^2_\Delta(\underline{\boldsymbol{u}}^{(p-1)}), \quad p = 1, \ldots, P, \tag{4}$$

we have that

$$\left\|\underline{\boldsymbol{u}}^{(P)} - \underline{\boldsymbol{u}}_\Delta\right\|_X \leq \left(\Delta \frac{\alpha_2}{\alpha_1}\right)^P \left\|\underline{\boldsymbol{u}}^{(0)} - \underline{\boldsymbol{u}}_\Delta\right\|_X. \tag{5}$$

Proof. By using the coercivity-like property of $\mathcal{L}^1_\Delta$ and the definition of $\mathcal{L}^1_\Delta(\underline{\boldsymbol{u}}^{(p)})$ given in (4) we have

$$\left\|\underline{\boldsymbol{u}}^{(P)} - \underline{\boldsymbol{u}}_\Delta\right\|_X \leq \frac{1}{\alpha_1}\left\|\mathcal{L}^1_\Delta(\underline{\boldsymbol{u}}^{(P)}) - \mathcal{L}^1_\Delta(\underline{\boldsymbol{u}}_\Delta)\right\|_Y \tag{6}$$

$$= \frac{1}{\alpha_1}\left\|\mathcal{L}^1_\Delta(\underline{\boldsymbol{u}}^{(P-1)}) - \mathcal{L}^2_\Delta(\underline{\boldsymbol{u}}^{(P-1)}) - \mathcal{L}^1_\Delta(\underline{\boldsymbol{u}}_\Delta)\right\|_Y. \tag{7}$$

Since $\underline{\boldsymbol{u}}_\Delta$ is the solution of $\mathcal{L}^2_\Delta$ we have that $\mathcal{L}^2_\Delta(\underline{\boldsymbol{u}}_\Delta) = \mathbf{0}_Y$ and we can add it inside the norm in (7) and get

$$\left\|\underline{\boldsymbol{u}}^{(P)} - \underline{\boldsymbol{u}}_\Delta\right\|_X \leq \frac{1}{\alpha_1}\left\|\left[\mathcal{L}^1_\Delta(\underline{\boldsymbol{u}}^{(P-1)}) - \mathcal{L}^2_\Delta(\underline{\boldsymbol{u}}^{(P-1)})\right] - \left[\mathcal{L}^1_\Delta(\underline{\boldsymbol{u}}_\Delta) - \mathcal{L}^2_\Delta(\underline{\boldsymbol{u}}_\Delta)\right]\right\|_Y. \tag{8}$$

Now, by applying the Lipschitz-continuity-like property we get

$$\left\|\underline{\boldsymbol{u}}^{(P)} - \underline{\boldsymbol{u}}_\Delta\right\|_X \leq \Delta\frac{\alpha_2}{\alpha_1}\left\|\underline{\boldsymbol{u}}^{(P-1)} - \underline{\boldsymbol{u}}_\Delta\right\|_X. \tag{9}$$

By repeating these calculations recursively we get the thesis. □

Let us remark that, due to the explicit nature of the operator $\mathcal{L}^1_\Delta$, the updating formula (4) represents a simple explicit recipe to approximate arbitrarily well the solution $\underline{\boldsymbol{u}}_\Delta$ of $\mathcal{L}^2_\Delta$. The convergence for $P \to +\infty$ is ensured independently of the starting vector $\underline{\boldsymbol{u}}^{(0)}$ provided that $\Delta\frac{\alpha_2}{\alpha_1} < 1$.

Remark 2.2. *If the solution $\underline{\boldsymbol{u}}_\Delta$ of $\mathcal{L}^2_\Delta$ is an R-order approximation of the exact solution $\underline{\boldsymbol{u}}^{ex}$ of a more general problem then an R-order approximation $\tilde{\underline{\boldsymbol{u}}}$ of $\underline{\boldsymbol{u}}_\Delta$ is an R-order approximation of the exact solution $\underline{\boldsymbol{u}}^{ex}$ as well. Applying the triangular inequality leads to*

$$\|\tilde{\underline{\boldsymbol{u}}} - \underline{\boldsymbol{u}}^{ex}\|_X \leq \|\tilde{\underline{\boldsymbol{u}}} - \underline{\boldsymbol{u}}_\Delta\|_X + \|\underline{\boldsymbol{u}}_\Delta - \underline{\boldsymbol{u}}^{ex}\|_X \leq O(\Delta^{R+1}). \tag{10}$$

Thus, in the optimal case, the number of iterations to perform in the context of the DeC must be the minimal amount which allows to match the accuracy of the solution of the operator $\mathcal{L}^2_\Delta$ with respect to the exact solution of the original problem, any extra iterations in approximating $\underline{\boldsymbol{u}}_\Delta$ would be formally useless. We will be more precise about the accuracy and the number of iterations needed to reach the highest possible accuracy later in the applications.

Now that we have introduced and proved the DeC in a general framework, the only thing that is left to do to apply it in a more specific context is to characterize the operators ($\mathcal{L}^1_\Delta$ and $\mathcal{L}^2_\Delta$) and the normed spaces (X and Y) and verify that the three hypotheses are satisfied.

3 The DeC for systems of ODEs

In this section we will focus on the explicit DeC methods for systems of ordinary differential equations (ODEs). In particular we will focus on the general Cauchy problem

$$\begin{cases} \frac{d}{dt}\boldsymbol{u}(t) = \boldsymbol{G}(t, \boldsymbol{u}(t)), \quad t \in [0, T], \\ \boldsymbol{u}(0) = \boldsymbol{z}, \end{cases} \tag{11}$$

with $\boldsymbol{u} : \mathbb{R}_0^+ \to \mathbb{R}^Q$, $\boldsymbol{z} \in \mathbb{R}^Q$ and $\boldsymbol{G} : \mathbb{R}_0^+ \times \mathbb{R}^Q \to \mathbb{R}^Q$ a continuous map Lipschitz continuous with respect to $\boldsymbol{u}$ uniformly with respect to t with a Lipschitz constant L. This ensures the existence of a unique solution for the system of ODEs (11). Up to a simple translation, the initial time can be set at $t = 0$ without loss of generality.

We will present two DeC methods for the numerical solution of the problem (11)

- bDeC, which was introduced originally in [43] in a more general family of schemes, but fully exploited for its simplicity only starting from [5] in the context of Galerkin solvers for hyperbolic PDEs without mass matrix.
- sDeC, which has a longer history [21] and more developments [47, 39, 35, 58] and it can actually be interpreted as a high order modification of the previous method as we will see.

Then, we will consider a general family of DeC methods, αDeC, depending on one parameter α, which contains both the previously described formulations as particular cases as described in [43]. Finally, we will present a compact matrix-formulation of such methods that will be quite useful in the following sections to recast them as RK methods and to construct the related Butcher tableaux.

Remark 3.1. *The names bDeC and sDeC come from the fact that in both cases the definition of the methods is based on the integration of the initial ODE over some intervals which are "bigger" in the context of bDeC and "smaller" in the context of sDeC.*

We will assume a classical one-step method setting: we discretize the time domain $[0, T]$ by introducing $N + 1$ time nodes t_n, which are such that $0 = t_0 < t_1 < \cdots < t_N = T$ and therefore inducing N intervals $[t_n, t_{n+1}]$, we denote by $\boldsymbol{u}_n$ an approximation of the exact solution $\boldsymbol{u}(t_n)$ at the time t_n and we look for a recipe to compute $\boldsymbol{u}_{n+1}$ by knowing $\boldsymbol{u}_n$ for each $n = 0, 1, \ldots, N - 1$.

We will focus on the generic time interval $[t_n, t_{n+1}]$ with $\Delta t = t_{n+1} - t_n$ and, as in the context of a general consistency analysis, we will assume $\boldsymbol{u}_n = \boldsymbol{u}(t_n)$.

In this context, the parameter Δ of the DeC is the step size Δt.

3.1 bDeC

In the generic time step $[t_n, t_n + \Delta t]$, we introduce $M + 1$ subtimenodes $t^0, \ldots, t^M$ such that $t_n = t^0 < t^1 < \cdots < t^M = t_n + \Delta t$. In literature, there are different choice of subtimenodes t^m, but for the following discussion we will consider equispaced nodes. In the numerical tests, we will also present results obtained with Gauss–Lobatto nodes [50, 32, 21], which can obtain high order accuracy with less subtimenodes.

We will refer to $\boldsymbol{u}(t^m)$ as the exact solution in the node t^m and to $\boldsymbol{u}^m$ as the approximation of the solution in the same node. Just for the first node we set $\boldsymbol{u}^0 := \boldsymbol{u}_n$ and, in the accuracy study, we will consider it to be exact, i.e., $\boldsymbol{u}^0 = \boldsymbol{u}(t^0) = \boldsymbol{u}(t_n) = \boldsymbol{u}_n$.

The bDeC method is based on the integral version of the ODE (11) in each interval $[t^0, t^m]$, which reads

$$\boldsymbol{u}(t^m) - \boldsymbol{u}^0 - \int_{t^0}^{t^m} \boldsymbol{G}(t, \boldsymbol{u}(t))dt = \boldsymbol{0}, \quad m = 1, \ldots, M \tag{12}$$

and, starting from this formulation, defines two discrete operators.

3.1.1 Definition of $\mathcal{L}^2_\Delta$

The definition of the $\mathcal{L}^2_\Delta$ operator is based on a high order discretization of (12). We obtain it by approximating the function $\boldsymbol{G}$ in (12) with a high order interpolation through the Lagrange polynomials ψ^ℓ of degree M associated to the $M+1$ subtimenodes

$$\boldsymbol{u}^m - \boldsymbol{u}^0 - \Delta t \sum_{\ell=0}^{M} \theta_\ell^m \boldsymbol{G}(t^\ell, \boldsymbol{u}^\ell) = \boldsymbol{0}, \quad \forall m = 1, \dots, M. \tag{13}$$

The normalized coefficients θ_ℓ^m are given by

$$\theta_\ell^m := \frac{1}{\Delta t} \int_{t^0}^{t^m} \psi^\ell(t) dt = \int_0^{\frac{t^m - t^0}{\Delta t}} \psi^\ell \left(t^0 + \Delta t s\right) ds \tag{14}$$

and do not depend on Δt but only on the number and distribution of the subtimenodes. By collecting all the components (13) related to all the subtimenodes but the first one in which the solution is known we can define the operator $\mathcal{L}^2_\Delta : \mathbb{R}^{(M\times Q)} \to \mathbb{R}^{(M\times Q)}$ as

$$\mathcal{L}^2_\Delta(\underline{\boldsymbol{u}}) = \begin{pmatrix} \boldsymbol{u}^1 - \boldsymbol{u}^0 - \Delta t \sum_{\ell=0}^{M} \theta_\ell^1 \boldsymbol{G}(t^\ell, \boldsymbol{u}^\ell) \\ \vdots \\ \boldsymbol{u}^m - \boldsymbol{u}^0 - \Delta t \sum_{\ell=0}^{M} \theta_\ell^m \boldsymbol{G}(t^\ell, \boldsymbol{u}^\ell) \\ \vdots \\ \boldsymbol{u}^M - \boldsymbol{u}^0 - \Delta t \sum_{\ell=0}^{M} \theta_\ell^M \boldsymbol{G}(t^\ell, \boldsymbol{u}^\ell) \end{pmatrix}, \text{ with } \underline{\boldsymbol{u}} = \begin{pmatrix} \boldsymbol{u}^1 \\ \vdots \\ \boldsymbol{u}^m \\ \vdots \\ \boldsymbol{u}^M \end{pmatrix}. \tag{15}$$

Let us remark that $\mathcal{L}^2_\Delta$ is defined on M components $\boldsymbol{u}^m \in \mathbb{R}^Q$ while $\boldsymbol{u}^0$ is an intrinsic datum of the operator. This leads to the definition of the spaces $X = Y := \mathbb{R}^{M\times Q}$ of section 2. The $\mathcal{L}^2_\Delta$ operator is a high order accurate discretization of the left hand side of (12) and what is interesting for us is that the solution of the equation $\mathcal{L}^2_\Delta(\underline{\boldsymbol{u}}_\Delta) = 0$ is a high order accurate discretization of the exact solution. For equispaced subtimenodes we can state the following result.

Proposition 3.1. *Let $\boldsymbol{u}^m$ be the m-th component of the solution of* (15)*. Then, $\boldsymbol{u}^m$ is an $(M+1)$-order accurate approximation of $\boldsymbol{u}(t^m)$.*

The proof is based on a fixed-point argument and can be found in the supplementary material. It is worth noting that $\mathcal{L}^2_\Delta(\underline{\boldsymbol{u}}_\Delta) = 0$ coincides with an implicit RK method with M stages, e.g. when choosing Gauss–Lobatto points one obtains the LobattoIIIA methods.

3.1.2 Definition of $\mathcal{L}^1_\Delta$

The auxiliary operator that we introduce in this section is a first order explicit discretization of (12) and it will serve to approximate the solution of $\mathcal{L}^2_\Delta = 0$. If we apply the Euler method to approximate (12) we have

$$\boldsymbol{u}^m - \boldsymbol{u}^0 - \Delta t \beta^m \boldsymbol{G}(t^0, \boldsymbol{u}^0) = \boldsymbol{0}, \tag{16}$$

where $\beta^m = \frac{t^m - t^0}{\Delta t}$. Let us remark, that also in this case β^m are determined only by the distribution of the subtimenodes but are independent of Δt. The Euler method is well known to provide a first order approximation of the exact solution.

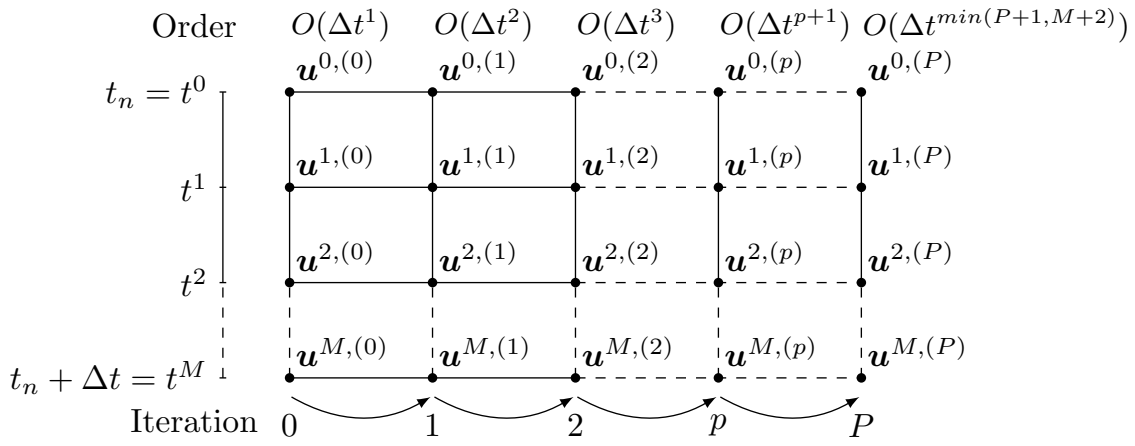

Figure 1: Sketch of the DeC iterative process for equispaced subtimenodes

Proposition 3.2. *Let $\boldsymbol{u}^m$ be the solution of* (16)*, then $\boldsymbol{u}^m$ is first order accurate, i.e., $\boldsymbol{u}(t^m) - \boldsymbol{u}^m = O(\Delta t^2)$.*

Directly from (16) we get our explicit, low order operator $\mathcal{L}^1_\Delta : \mathbb{R}^{(M\times Q)} \to \mathbb{R}^{(M\times Q)}$ defined as

$$\mathcal{L}^1_\Delta(\underline{\boldsymbol{u}}) = \begin{pmatrix} \boldsymbol{u}^1 - \boldsymbol{u}^0 - \Delta t \beta^1 \boldsymbol{G}(t^0, \boldsymbol{u}^0) \\ \vdots \\ \boldsymbol{u}^m - \boldsymbol{u}^0 - \Delta t \beta^m \boldsymbol{G}(t^0, \boldsymbol{u}^0) \\ \vdots \\ \boldsymbol{u}^M - \boldsymbol{u}^0 - \Delta t \beta^M \boldsymbol{G}(t^0, \boldsymbol{u}^0) \end{pmatrix} \text{ with } \underline{\boldsymbol{u}} = \begin{pmatrix} \boldsymbol{u}^1 \\ \vdots \\ \boldsymbol{u}^m \\ \vdots \\ \boldsymbol{u}^M \end{pmatrix}. \tag{17}$$

So, the operator $\mathcal{L}^1_\Delta$ is first order accurate and the system $\mathcal{L}^1_\Delta(\underline{\boldsymbol{u}}) = \underline{\boldsymbol{r}}$ is easy to solve and explicit for every given datum $\underline{\boldsymbol{r}} \in \mathbb{R}^{M\times Q}$.

3.1.3 Updating formula

The operators $\mathcal{L}^1_\Delta$ and $\mathcal{L}^2_\Delta$ fulfill the hypotheses required to apply the DeC, the proofs can be found in the supplementary material. Now, let us characterize the update formula (4) in the bDeC context:

$$\mathcal{L}^1_\Delta(\underline{\boldsymbol{u}}^{(p)}) = \mathcal{L}^1_\Delta(\underline{\boldsymbol{u}}^{(p-1)}) - \mathcal{L}^2_\Delta(\underline{\boldsymbol{u}}^{(p-1)}), \quad p = 1, \dots, P, \tag{18}$$

where $\underline{\boldsymbol{u}}^{(p)} \in \mathbb{R}^{(M\times Q)}$ and $\underline{\boldsymbol{u}}^{(p)}$ denotes the p-th iteration approximation at all the subtimenodes. The approximation at the subtimenode t^m will be denoted as $\boldsymbol{u}^{m,(p)} \in \mathbb{R}^Q$. We define the starting vector $\underline{\boldsymbol{u}}^{(0)}$ for our iteration process as the solution at the beginning of the timestep, i.e., $\boldsymbol{u}^{m,(0)} := \boldsymbol{u}_n$ for all $m = 0, \dots, M$. Moreover, as one can also observe in figure 1, we introduce also the values $\boldsymbol{u}^{0,(p)} = \boldsymbol{u}_n$ for all p that will be useful to write a general formula. Finally, we set the value of the next timestep as $\boldsymbol{u}_{n+1} := \boldsymbol{u}^{M,(P)}$.

Now, in (18) only $\underline{\boldsymbol{u}}^{(p)}$ is unknown, hence, it can be solved explicitly, as $\mathcal{L}^1_\Delta$ is an explicit operator, iteratively for p. The iterative formula (18) can be characterized for the generic p-th

iteration and the m-th subtimenodes as

$$\begin{aligned} \boldsymbol{u}^{m,(p)} - \boldsymbol{u}^0 - \Delta t \beta^m \boldsymbol{G}(t^0, \boldsymbol{u}^0) = \boldsymbol{u}^{m,(p-1)} - \boldsymbol{u}^0 - \Delta t \beta^m \boldsymbol{G}(t^0, \boldsymbol{u}^0) \\ - \left(\boldsymbol{u}^{m,(p-1)} - \boldsymbol{u}^0 - \Delta t \sum_{\ell=0}^{M} \theta_\ell^m \boldsymbol{G}(t^\ell, \boldsymbol{u}^{\ell,(p-1)}) \right), \end{aligned} \tag{19}$$

which simplifies to

$$\boldsymbol{u}^{m,(p)} = \boldsymbol{u}^0 + \Delta t \sum_{\ell=0}^{M} \theta_\ell^m \boldsymbol{G}(t^\ell, \boldsymbol{u}^{\ell,(p-1)}). \tag{20}$$

A graphical sketch of the updating process is shown in figure 1. In the following, we provide the minimal number of iterations P one needs to perform to obtain the optimal order of accuracy.

As already anticipated in remark 2.2, the DeC converges to the solution of $\mathcal{L}_\Delta^2 = 0$ at each subtimenode t^m and not directly to the ODE solution. Considering the solution of $\mathcal{L}_\Delta^2 = 0$ to be an $(M+1)$-th order accurate approximation of the exact solution, it suffice to do a number of iterations P that matches this accuracy.

Proposition 3.3 (Optimal number of iterations)**.** *Let $\underline{\boldsymbol{u}}_\Delta$ be the solution of $\mathcal{L}_\Delta^2 = 0$ and $\underline{\boldsymbol{u}}^{ex}$ be the exact solution at all subtimenodes. Then, the accuracy of $\underline{\boldsymbol{u}}^{(P)}$ given by* (18)*, with respect to $\underline{\boldsymbol{u}}^{ex}$, is* $\min\{P, M+1\}$.

Proof. As seen in (10), we can apply the triangular inequality to the error of $\underline{\boldsymbol{u}}^{(P)}$ to obtain

$$\left\| \underline{\boldsymbol{u}}^{(P)} - \underline{\boldsymbol{u}}^{ex} \right\|_\infty \leq \left\| \underline{\boldsymbol{u}}^{(P)} - \underline{\boldsymbol{u}}_\Delta \right\|_\infty + \left\| \underline{\boldsymbol{u}}_\Delta - \underline{\boldsymbol{u}}^{ex} \right\|_\infty \tag{21}$$

Since the operator $\mathcal{L}_\Delta^2$ is $(M+1)$-order accurate we have that $\left\| \underline{\boldsymbol{u}}_\Delta - \underline{\boldsymbol{u}}^{ex} \right\|_\infty = O(\Delta t^{M+2})$. From (5) we know that

$$\left\| \underline{\boldsymbol{u}}^{(P)} - \underline{\boldsymbol{u}}_\Delta \right\|_\infty = C \Delta t^P \left\| \underline{\boldsymbol{u}}^{(0)} - \underline{\boldsymbol{u}}_\Delta \right\|_\infty, \tag{22}$$

but $\left\| \underline{\boldsymbol{u}}^{(0)} - \underline{\boldsymbol{u}}_\Delta \right\|_\infty = O(\Delta t)$, since $\underline{\boldsymbol{u}}_\Delta$ is a consistent approximation of $u(t^m)$ and $t^m - t^0 \leq \Delta t$, hence,

$$\left\| \underline{\boldsymbol{u}}^{(P)} - \underline{\boldsymbol{u}}_\Delta \right\|_\infty = C \Delta t^P \left\| \underline{\boldsymbol{u}}^{(0)} - \underline{\boldsymbol{u}}_\Delta \right\|_\infty = O(\Delta t^{P+1}). \tag{23}$$

Thus, we can write

$$\left\| \underline{\boldsymbol{u}}^{(P)} - \underline{\boldsymbol{u}}^{ex} \right\|_\infty \leq O(\Delta t^{M+2}) + O(\Delta t^{P+1}), \tag{24}$$

which is the thesis. □

From this proposition a simple recipe follows on the choice of P, i.e., $P = M + 1$. Further iterations would not increase the order of accuracy of the method, though they might slightly improve the accuracy. We remark that this criterion is valid for equispaced distributed subtimenodes, while for other distributions less subtimenodes might be needed, e.g. M Gauss–Lobatto nodes lead to an order of $2M$ of the $\mathcal{L}_\Delta^2$ operator and, hence, to the optimal choice of $P = 2M$. On the other hand, if we fix the order to be P, the most efficient way to achieve it is to take $M = P - 1$ if we use equispaced subtimenodes or $M = \left\lceil \frac{P}{2} \right\rceil$ if we use Gauss–Lobatto subtimenodes and perform P iterations.

3.2 sDeC

The definition of the operators for this DeC method will be similar to the one seen in the context of the bDeC method but we will consider the integration of the original system of ODEs over intervals which are "smaller". Strictly speaking, it is not possible to put this formulation into the described framework by proving the properties of the operators, nevertheless it is possible to show that this new DeC formulation is actually a perturbation of the previous one providing the same order of accuracy. The proof of this fact can be found in the supplementary material.

3.2.1 Definition of $\mathcal{L}^2_\Delta$

We consider the usual $M+1$ nodes t^m with $m = 0, 1, \dots, M$ in the interval $[t_n, t_n + \Delta t]$ and we assume the same definition of $\boldsymbol{u}^m$ and $\boldsymbol{u}(t^m)$ as in the previous section. By repeating the same steps seen in the previous section but focusing on the integration of the system of ODEs over the "smaller" intervals $[t^{m-1}, t^m]$ we get

$$\boldsymbol{u}^m - \boldsymbol{u}^{m-1} - \Delta t \sum_{\ell=0}^{M} \delta_\ell^m \boldsymbol{G}(t^\ell, \boldsymbol{u}^\ell) = \boldsymbol{0}, \quad \forall m = 1, \dots, M, \tag{25}$$

where

$$\int_{t^{m-1}}^{t^m} \psi^\ell(t) dt = \Delta t \int_{\frac{t^{m-1}-t^0}{\Delta t}}^{\frac{t^m-t^0}{\Delta t}} \psi^\ell\left(\Delta t s + t^0\right) ds = \Delta t \delta_\ell^m \tag{26}$$

with normalized coefficients δ_ℓ^m, depending just on the number and on the distribution of the nodes t^m but not on Δt, defined as

$$\delta_\ell^m := \int_{\frac{t^{m-1}-t^0}{\Delta t}}^{\frac{t^m-t^0}{\Delta t}} \psi^\ell\left(\Delta t s + t^0\right) ds. \tag{27}$$

Our implicit, $(M+1)$-order accurate operator $\mathcal{L}^2_\Delta : \mathbb{R}^{(M\times Q)} \to \mathbb{R}^{(M\times Q)}$ is therefore defined as

$$\mathcal{L}^2_\Delta(\underline{\boldsymbol{u}}) = \begin{pmatrix} \boldsymbol{u}^1 - \boldsymbol{u}^0 - \Delta t \sum_{\ell=0}^{M} \delta_\ell^1 \boldsymbol{G}(t^\ell, \boldsymbol{u}^\ell) \\ \vdots \\ \boldsymbol{u}^m - \boldsymbol{u}^{m-1} - \Delta t \sum_{\ell=0}^{M} \delta_\ell^m \boldsymbol{G}(t^\ell, \boldsymbol{u}^\ell) \\ \vdots \\ \boldsymbol{u}^M - \boldsymbol{u}^{M-1} - \Delta t \sum_{\ell=0}^{M} \delta_\ell^M \boldsymbol{G}(t^\ell, \boldsymbol{u}^\ell) \end{pmatrix}, \text{ with } \underline{\boldsymbol{u}} = \begin{pmatrix} \boldsymbol{u}^1 \\ \vdots \\ \boldsymbol{u}^m \\ \vdots \\ \boldsymbol{u}^M \end{pmatrix}. \tag{28}$$

3.2.2 Definition of $\mathcal{L}^1_\Delta$

If we apply the explicit Euler method in the subinterval $[t^{m-1}, t^m]$ we get

$$\boldsymbol{u}^m - \boldsymbol{u}^{m-1} - \Delta t \gamma^m \boldsymbol{G}(t^{m-1}, \boldsymbol{u}^{m-1}) = \boldsymbol{0}, \tag{29}$$

where $\gamma^m = \frac{t^m - t^{m-1}}{\Delta t}$ are normalized coefficients. The explicit, first-order order operator $\mathcal{L}^1_\Delta : \mathbb{R}^{(M\times Q)} \to \mathbb{R}^{(M\times Q)}$ is defined as

$$\mathcal{L}^1_\Delta(\underline{\boldsymbol{u}}) = \begin{pmatrix} \boldsymbol{u}^1 - \boldsymbol{u}^0 - \Delta t\gamma^1 \boldsymbol{G}(t^0, \boldsymbol{u}^0) \\ \vdots \\ \boldsymbol{u}^m - \boldsymbol{u}^{m-1} - \Delta t\gamma^m \boldsymbol{G}(t^{m-1}, \boldsymbol{u}^{m-1}) \\ \vdots \\ \boldsymbol{u}^M - \boldsymbol{u}^{M-1}\Delta t\gamma^M \boldsymbol{G}(t^{M-1}, \boldsymbol{u}^{M-1}) \end{pmatrix}, \text{ with } \underline{\boldsymbol{u}} = \begin{pmatrix} \boldsymbol{u}^1 \\ \vdots \\ \boldsymbol{u}^m \\ \vdots \\ \boldsymbol{u}^M \end{pmatrix}. \tag{30}$$

Let us make some useful remarks.

Remark 3.2. *The operator $\mathcal{L}^1_\Delta$ is explicit and easy to solve but just first-order accurate. Differently from the analogous operator of the previous formulation, in this case we cannot solve the operator $\mathcal{L}^1_\Delta$ in all its components at the same time but we have to do it component by component from $\boldsymbol{u}^1$ to $\boldsymbol{u}^M$. The problem $\mathcal{L}^1_\Delta(\underline{\boldsymbol{u}}) = \underline{\boldsymbol{0}}$ must be solved by firstly determining $\boldsymbol{u}^1$ from the related component associated to the subtimenode $m = 1$, then we proceed computing $\boldsymbol{u}^2$ and we continue iteratively until the last component. In the same fashion it is possible to explicitly solve the general problem $\mathcal{L}^1_\Delta(\underline{\boldsymbol{u}}) = \underline{\boldsymbol{r}}$ for a fixed $\underline{\boldsymbol{r}} \in \mathbb{R}^{(M\times Q)}$. At each iteration of the DeC method associated to this second formulation we will have to solve recursively such problem in order to compute $\underline{\boldsymbol{u}}^{(p)}$.*

3.2.3 Updating formula

Let us characterize the updating formula to this context. Again, we recall it for the sake of clarity

$$\mathcal{L}^1_\Delta(\underline{\boldsymbol{u}}^{(p)}) = \mathcal{L}^1_\Delta(\underline{\boldsymbol{u}}^{(p-1)}) - \mathcal{L}^2_\Delta(\underline{\boldsymbol{u}}^{(p-1)}), \quad p = 1, \dots, P, \tag{31}$$

where we adopt the usual notation for $\underline{\boldsymbol{u}}^{(p)}$, unknown involved in the p-th iteration, made by M components $\boldsymbol{u}^{m,(p)}$ corresponding to the approximations of the solution in the different subtimenodes t^m $m = 1, \dots, M$.

Also in this case, the explicit character of the operator $\mathcal{L}^1_\Delta$ leads to an explicit recipe for the computation of $\underline{\boldsymbol{u}}^{(p)}$ whose components, in this case, must be computed iteratively one by one from $\boldsymbol{u}^{1,(p)}$ to $\boldsymbol{u}^{M,(p)}$ as already anticipated in remark 3.2. Actually, one can easily check that the updating formula (31) is in the form $\mathcal{L}^1_\Delta(\underline{\boldsymbol{u}}^{(p)}) = \underline{\boldsymbol{r}}$ with $\underline{\boldsymbol{r}} \in \mathbb{R}^{(M\times Q)}$ fixed and we already explained how to solve this problem.

We will try now to characterize better the updating formula (31). The computation of generic m-th component of $\underline{\boldsymbol{u}}^{(p)}$ in p-th iteration, in particular, reads

$$\begin{aligned} &\boldsymbol{u}^{m,(p)} - \boldsymbol{u}^{m-1,(p)} - \Delta t\gamma^m \boldsymbol{G}(t^{m-1}, \boldsymbol{u}^{m-1,(p)}) \\ =&\boldsymbol{u}^{m,(p-1)} - \boldsymbol{u}^{m-1,(p-1)} - \Delta t\gamma^m \boldsymbol{G}(t^{m-1}, \boldsymbol{u}^{m-1,(p-1)}) \\ -&\boldsymbol{u}^{m,(p-1)} - \boldsymbol{u}^{m-1,(p-1)} - \Delta t \sum_{\ell=0}^{M} \delta^m_\ell \boldsymbol{G}(t^\ell, \boldsymbol{u}^{\ell,(p-1)}), \end{aligned} \tag{32}$$

from which we get

$$\begin{aligned} \boldsymbol{u}^{m,(p)} = \boldsymbol{u}^{m-1,(p)} &+ \Delta t\gamma^m \left(\boldsymbol{G}(t^{m-1}, \boldsymbol{u}^{m-1,(p)}) - \boldsymbol{G}(t^{m-1}, \boldsymbol{u}^{m-1,(p-1)}) \right) \\ &+ \Delta t \sum_{\ell=0}^{M} \delta^m_\ell \boldsymbol{G}(t^\ell, \boldsymbol{u}^{\ell,(p-1)}). \end{aligned} \tag{33}$$

By direct computation, recursively explicitly writing $\boldsymbol{u}^{m-k,(p)}$ until $k = m-1$ in the previous formula and recalling the fact that $\boldsymbol{u}^{0,(p)} = \boldsymbol{u}^0$, we get

$$\begin{aligned}\boldsymbol{u}^{m,(p)} = \boldsymbol{u}^0 &+ \Delta t \sum_{\ell=0}^{m-1} \gamma^{\ell+1} \left(\boldsymbol{G}(t^\ell, \boldsymbol{u}^{\ell,(p)}) - \boldsymbol{G}(t^\ell, \boldsymbol{u}^{\ell,(p-1)}) \right) \\ &+ \Delta t \sum_{r=1}^{m} \sum_{\ell=0}^{M} \delta_\ell^r \boldsymbol{G}(t^\ell, \boldsymbol{u}^{\ell,(p-1)}).\end{aligned} \tag{34}$$

Now, let us focus on the last term of (34). Exchanging the sums over r and ℓ and observing that, thanks to the definitions of the coefficients δ_ℓ^r and of the coefficients θ_ℓ^m, we have

$$\Delta t \sum_{r=1}^{m} \delta_\ell^r = \sum_{r=1}^{m} \int_{t^{r-1}}^{t^r} \psi^\ell(t) dt = \int_{t^0}^{t^m} \psi^\ell(t) dt = \Delta t \theta_\ell^m. \tag{35}$$

So, we can rewrite (34) as

$$\begin{aligned}\boldsymbol{u}^{m,(p)} = \boldsymbol{u}^0 &+ \Delta t \sum_{\ell=0}^{m-1} \gamma^{\ell+1} \left(\boldsymbol{G}(t^\ell, \boldsymbol{u}^{\ell,(p)}) - \boldsymbol{G}(t^\ell, \boldsymbol{u}^{\ell,(p-1)}) \right) \\ &+ \Delta t \sum_{\ell=0}^{M} \theta_\ell^m \boldsymbol{G}(t^\ell, \boldsymbol{u}^{\ell,(p-1)}),\end{aligned} \tag{36}$$

which allows to explicitly compute all the components $\boldsymbol{u}^{m,(p)}$ in sequence from $m = 1$ to $m = M$. Note that all the components $\boldsymbol{u}^{\ell,(p)}$ with $\ell = 0, \ldots, m-1$ are actually used in the computation of $\boldsymbol{u}^{m,(p)}$, this is why the components must be computed serially on m (in opposition to the bDeC where a parallel strategy is usable). Let us remark also that $\boldsymbol{u}^{0,(p)} = \boldsymbol{u}^0 = \boldsymbol{u}_n$ is known. For what concerns the starting vector $\underline{\boldsymbol{u}}^{(0)}$, the $(M+1)$-th order of accuracy and the optimal number of iterations $P = M+1$ in the equispaced setting, the $2M$-th order of accuracy and the optimal number of iterations $P = 2M$ in the Gauss–Lobatto setting, we can refer to what already said in the context of the bDeC formulation.

3.3 A general family of DeC methods, αDeC

We can easily construct a family of schemes dependent on a single parameter α by combining the two presented formulations. In particular we consider a convex combination of the updating formulas (20) and (36) through the parameter $\alpha \in [0, 1]$ which gives

$$\begin{aligned}\boldsymbol{u}^{m,(p)} = \boldsymbol{u}^0 &+ \Delta t \sum_{\ell=0}^{M} \theta_\ell^m \boldsymbol{G}(t^\ell, \boldsymbol{u}^{\ell,(p-1)}) \\ &+ \alpha \left[\Delta t \sum_{\ell=0}^{m-1} \gamma^{\ell+1} \left(\boldsymbol{G}(t^\ell, \boldsymbol{u}^{\ell,(p)}) - \boldsymbol{G}(t^\ell, \boldsymbol{u}^{\ell,(p-1)}) \right) \right].\end{aligned} \tag{37}$$

Through (37), it is possible to explicitly compute iteration by iteration the different components $\boldsymbol{u}^{m,(p)}$ starting from $m = 1$ until M. Of course when $\alpha = 0$ we retrieve the first formulation, when $\alpha = 1$ we get, instead, the second one. Thanks to considerations analogous to the ones which hold for the sDeC, it is also possible to interpret (37) as a weighted perturbation through the coefficient α of the bDeC.

3.3.1 Matrix formulation

We will now introduce a compact matrix-formulation of the presented methods. Before doing it, we specify here a general guideline concerning the notation that will be very useful and broadly used in the following. It is clear at this point that we have approximations of $\boldsymbol{u}(t)$ in certain time-nodes t^ℓ in the interval $[t_n, t_n + \Delta t]$. These approximations have been often collected in vectors, like $\underline{\boldsymbol{u}}^{(p)}$, containing as components the quantities related to all the time-nodes but the initial one because as remarked several times the quantity $\boldsymbol{u}^0$ associated to the initial time is known. For practical reasons, we will now introduce the vectors containing as components the quantities related to all the time-nodes including the initial one. In order to avoid confusion, we will always refer to the vectors not containing the component associated to the initial subtimenode with the small letter and to the vectors containing it with the capital letter

$$\underline{\boldsymbol{u}}^{(p)} = \begin{pmatrix} \boldsymbol{u}^{1,(p)} \\ \vdots \\ \boldsymbol{u}^{m,(p)} \\ \vdots \\ \boldsymbol{u}^{M,(p)} \end{pmatrix}, \quad \underline{\boldsymbol{U}}^{(p)} = \begin{pmatrix} \boldsymbol{u}^0 \\ \underline{\boldsymbol{u}}^{(p)} \end{pmatrix}. \tag{38}$$

In order to light the notation we commit a little abuse of notation definining

$$\underline{\boldsymbol{G}}(\underline{\boldsymbol{u}}^{(p)}) = \begin{pmatrix} \boldsymbol{G}(t^1, \boldsymbol{u}^{1,(p)}) \\ \vdots \\ \boldsymbol{G}(t^m, \boldsymbol{u}^{m,(p)}) \\ \vdots \\ \boldsymbol{G}(t^M, \boldsymbol{u}^{M,(p)}) \end{pmatrix}, \quad \underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(p)}) = \begin{pmatrix} \boldsymbol{G}(t^0, \boldsymbol{u}^0) \\ \underline{\boldsymbol{G}}(\underline{\boldsymbol{u}}^{(p)}) \end{pmatrix} \tag{39}$$

i.e., when $\boldsymbol{G}$ is applied to the vectors $\underline{\boldsymbol{u}}^{(p)}$ and $\underline{\boldsymbol{U}}^{(p)}$, it is meant to be applied component-wise with the values t^ℓ chosen accordingly with $\boldsymbol{u}^{\ell,(p)}$. With the previous definitions it is possible to recast the general updating formula (37) in the following compact form

$$\begin{aligned} \underline{\boldsymbol{U}}^{(p)} &= \underline{\boldsymbol{U}}^{(0)} + \Delta t \Theta \underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(p-1)}) + \Delta t \alpha \Gamma (\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(p)}) - \underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(p-1)})) \\ &= \underline{\boldsymbol{U}}^{(0)} + \Delta t (\Theta - \alpha\Gamma) \underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(p-1)}) + \Delta t \alpha \Gamma \underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(p)}), \end{aligned} \tag{40}$$

where the vector $\underline{\boldsymbol{U}}^{(0)} \in \mathbb{R}^{((M+1)\times Q)}$ and the matrices $\Theta, \Gamma \in \mathbb{R}^{(M+1)\times(M+1)}$ are defined as

$$\underline{\boldsymbol{U}}^{(0)} = \begin{pmatrix} \boldsymbol{u}_n \\ \boldsymbol{u}_n \\ \vdots \\ \boldsymbol{u}_n \end{pmatrix}, \Theta = \begin{pmatrix} 0 & 0 & \dots & 0 \\ \theta_0^1 & \theta_1^1 & \dots & \theta_M^1 \\ \theta_0^2 & \theta_1^2 & \dots & \theta_M^2 \\ \vdots & \vdots & \ddots & \vdots \\ \theta_0^M & \theta_1^M & \dots & \theta_M^M \end{pmatrix}, \Gamma = \begin{pmatrix} 0 & 0 & \dots & 0 & 0 \\ \gamma^1 & 0 & \dots & 0 & 0 \\ \gamma^1 & \gamma^2 & \ddots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \gamma^1 & \gamma^2 & \dots & \gamma^M & 0 \end{pmatrix}, \tag{41}$$

with the matrix Γ being strictly lower-triangular as a consequence of the fact that the resulting scheme is fully explicit. Let us observe that the first component $\boldsymbol{u}^0$ of $\underline{\boldsymbol{U}}^{(p)}$ is never updated. This is coherent with what we have said so far.

Remark 3.3. *The matrices Θ and Γ that we have defined are referred to a scalar ODE ($Q = 1$). In case we want to adapt them to a vectorial problem we have to block-expand them.*

4 Two novel families of novel DeC methods

In this section, we will show how to construct two new families of efficient DeC methods by introducing a modification in the αDeC methods. We will first assume equispaced subtimenodes and we will extend it to Gauss–Lobatto subtimenodes. The modification is based on the following observation: in the context of a standard DeC method, at any iteration $p < M + 1$, we get a solution $\underline{\boldsymbol{u}}^{(p)}$ which is p-th order accurate using $M + 1$ subtimenodes even though only p would be formally sufficient to provide such accuracy. In other words, the number of subtimenodes is fixed a priori for all iterations in order to get the desired order of accuracy. These subtimenodes are always used although the formal order of accuracy, for which such nodes are required, is reached only in the final iteration. This represents indeed a huge waste of computational resources.

The modification consists in starting with only two subtimenodes and increasing their number, iteration by iteration, so to match the order of accuracy of the specific iteration. In particular, we introduce an intermediate interpolation process between the iterations in order to retrieve the needed quantities in the new subtimenodes.

The idea has been introduced in [47] for implicit methods, but without a systematic theory and related analytical study. We will present here two possible interpolation strategies which will lead to the definition of two general families of efficient DeC methods. In the next sections, we will show how they can be reinterpreted as RK methods by explicitly constructing their Butcher tableaux and we will investigate their linear stability.

We remind here some general guidelines concerning the notation: we adopt the convention introduced in section 3.3.1 for the definition of the matrix formulation of the original methods. Therefore, we use the capital letter to refer to the vectors containing the component of the initial subtimenode, i.e., $\underline{\boldsymbol{U}}^{(p)}$, and with the small letter to the ones not containing it, i.e., $\underline{\boldsymbol{u}}^{(p)}$, and when $\boldsymbol{G}$ is applied to vectors it is meant to be applied component-wise. In addition, we will use the asterisk to refer to the vectors got through the interpolation process. The number of subtimenodes will change iteration by iteration and it is useful to define the vector $\underline{t}^{(p)} := \left(t^{0,(p)}, \dots, t^{p,(p)}\right)^T$ of the subtimenodes in which we have the approximations of the solution at the p-th iteration, with $t^{0,(p)} = t_n$ and $t^{p,(p)} = t_{n+1}$, and the vector $\underline{t}^{*(p)} := \underline{t}^{(p+1)}$ of the interpolation subtimenodes at the iteration $p + 1$. Their role will be explained in the following.

4.1 αDeCu

The αDeCu methods are obtained from the αDeC methods by introducing an intermediate interpolation process between the iterations. In particular, the interpolated quantity in this case is the solution $\boldsymbol{u}(t)$. We will describe here the method and characterize the updating formulas. For convenience, we will formulate the methods in terms of the vectors $\underline{\boldsymbol{U}}^{(p)}$ containing the component $\boldsymbol{u}^0 = \boldsymbol{u}_n$ associated to the initial subtimenode. It must be clear at this point that such component is fixed and that, in the updating, the operators $\mathcal{L}^1_\Delta$ and $\mathcal{L}^2_\Delta$ act on the vectors $\underline{\boldsymbol{u}}^{(p)}$.

4.1.1 Method

We start with two subtimenodes associated to t_n and $t_n + \Delta t$ and so $\underline{\boldsymbol{U}}^{(0)} \in \mathbb{R}^{(2\times Q)}$. We perform the first iteration of the αDeC method with $\mathcal{L}^1_\Delta$ and $\mathcal{L}^2_\Delta$ associated to two subtimenodes thus getting $\underline{\boldsymbol{U}}^{(1)} \in \mathbb{R}^{(2\times Q)}$ which is first-order accurate and so $O(\Delta t^2)$-accurate.

Up to now, we have in the two components of $\underline{\boldsymbol{U}}^{(1)}$ an $O(\Delta t^2)$-accurate approximation of the values of the solution in t_n and $t_n + \Delta t$ which are sufficient to get an $O(\Delta t^2)$-accurate global reconstruction of the solution in the interval $[t_n, t_n + \Delta t]$ through Lagrange interpolation. We can therefore perform an interpolation to get $\underline{\boldsymbol{U}}^{*(1)} \in \mathbb{R}^{(3\times Q)}$ corresponding to three equispaced nodes in the interval $[t_n, t_n+\Delta t]$. Then, we perform the second iteration passing from $\underline{\boldsymbol{U}}^{*(1)}$ to $\underline{\boldsymbol{U}}^{(2)} \in \mathbb{R}^{(3\times Q)}$ which is $O(\Delta t^3)$-accurate. From $\underline{\boldsymbol{U}}^{(2)}$, we can construct a second order accurate reconstruction of the solution in $[t_n, t_n + \Delta t]$. We continue in this fashion, alternating iterations of the αDeC method with interpolations increasing the order of accuracy and the number of subtimenodes.

In general, when performing the p-th iteration we have $\underline{\boldsymbol{U}}^{(p-1)} \in \mathbb{R}^{(p\times Q)}$ which is $O(\Delta t^p)$-accurate; through interpolation we get $\underline{\boldsymbol{U}}^{*(p-1)} \in \mathbb{R}^{((p+1)\times Q)}$, still $O(\Delta t^p)$-accurate, from which we get $\underline{\boldsymbol{U}}^{(p)} \in \mathbb{R}^{((p+1)\times Q)}$ which is $O(\Delta t^{p+1})$-accurate. In particular, the interpolation can be easily performed through a matrix product involving the interpolation matrix $H^{(p-1)}$

$$\underline{\boldsymbol{U}}^{*(p-1)} = H^{(p-1)}\underline{\boldsymbol{U}}^{(p-1)}. \tag{42}$$

Then, we perform the p-th iteration and get $\underline{\boldsymbol{U}}^{(p)} \in \mathbb{R}^{((p+1)\times Q)}$, which is $O(\Delta t^{p+1})$-accurate. Of course the operators $\mathcal{L}^1_\Delta$ and $\mathcal{L}^2_\Delta$ change at each iteration accordingly to the number of subtimenodes; therefore, at the general iteration p we will not have fixed matrices Θ and Γ, but rather $\Theta^{(p)}$ and $\Gamma^{(p)}$.

The final number of subtimenodes is not fixed a priori and we could in principle continue adding nodes improving the accuracy of the approximation, this fact will be exploited in section 8 to design adaptive methods. For the moment, we consider a given number of final subtimenodes $M + 1$, as in the original method. In such case, it is useful to observe that $\underline{\boldsymbol{U}}^{(p)} \in \mathbb{R}^{((p+1)\times Q)}$, got at the p-th iteration, is $O(\Delta t^{p+1})$-accurate and associated to $p + 1$ subtimenodes. Actually, $p + 1$ time-nodes could reach an $O(\Delta t^{p+2})$-accuracy, for this reason, it is useful to perform $M+1$ iterations including a final iteration without interpolation to get the optimal order of accuracy $O(\Delta t^{M+2})$ associated to $M + 1$ subtimenodes. In this way, we have that the interpolation is performed at each iteration except the first and the last.

A useful sketch of the algorithm is represented in figure 2 following the indication of αDeCu. We focus on the interval $[t_n, t_n + \Delta t]$ and we use the dots to indicate the subtimenodes in which, at each iteration p, we have approximations of the solution, while we use the crosses to express the location in time of the interpolated values.

4.1.2 Updating formula

We start with two subtimenodes associated to t_n and $t_n + \Delta t$

$$\underline{\boldsymbol{U}}^{(0)} = \begin{pmatrix} \boldsymbol{u}_n \\ \boldsymbol{u}_n \end{pmatrix} \in \mathbb{R}^{(2\times Q)} \tag{43}$$

and we perform the first iteration

$$\underline{\boldsymbol{U}}^{(1)} = \underline{\boldsymbol{U}}^{(0)} + \Delta t(\Theta^{(1)} - \alpha\Gamma^{(1)})\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(0)}) + \Delta t\alpha\Gamma^{(1)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(1)}). \tag{44}$$

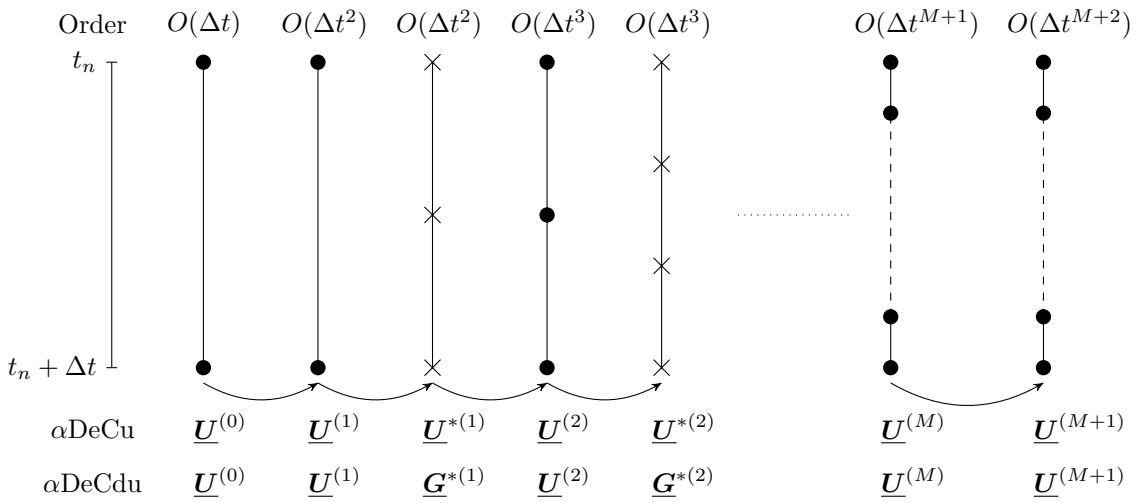

Figure 2: αDeCu and αDeCdu, sketches

Now, we perform the first interpolation getting

$$\begin{aligned}\underline{\boldsymbol{U}}^{*(1)} &= H^{(1)}\underline{\boldsymbol{U}}^{(1)} \\ &= H^{(1)}\left[\underline{\boldsymbol{U}}^{(0)} + \Delta t(\Theta^{(1)} - \alpha\Gamma^{(1)})\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(0)}) + \Delta t\alpha\Gamma^{(1)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(1)})\right] \\ &= \underline{\boldsymbol{U}}_3^{(0)} + \Delta t H^{(1)}(\Theta^{(1)} - \alpha\Gamma^{(1)})\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(0)}) + \Delta t\alpha H^{(1)}\Gamma^{(1)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(1)}).\end{aligned} \tag{45}$$

where the last equality is due to the fact that, by consistency, the sum of the elements on the rows of the interpolation matrices $H^{(p)}$ is equal to 1. The subscript 3 has been added to $\underline{\boldsymbol{U}}_3^{(0)} \in \mathbb{R}^{3\times Q}$ to distinguish it from the initial $\underline{\boldsymbol{U}}^{(0)} \in \mathbb{R}^{2\times Q}$.

Then, we perform the second iteration

$$\underline{\boldsymbol{U}}^{(2)} = \underline{\boldsymbol{U}}_3^{(0)} + \Delta t(\Theta^{(2)} - \alpha\Gamma^{(2)})\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{*(1)}) + \Delta t\alpha\Gamma^{(2)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(2)}). \tag{46}$$

We then perform the interpolation getting

$$\begin{aligned}\underline{\boldsymbol{U}}^{*(2)} &= H^{(2)}\underline{\boldsymbol{U}}^{(2)} \\ &= H^{(2)}\left[\underline{\boldsymbol{U}}_3^{(0)} + \Delta t(\Theta^{(2)} - \alpha\Gamma^{(2)})\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{*(1)}) + \Delta t\alpha\Gamma^{(2)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(2)})\right] \\ &= \underline{\boldsymbol{U}}_4^{(0)} + \Delta t H^{(2)}(\Theta^{(2)} - \alpha\Gamma^{(2)})\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{*(1)}) + \Delta t\alpha H^{(2)}\Gamma^{(2)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(2)})\end{aligned} \tag{47}$$

from which we can get $\underline{\boldsymbol{U}}^{(3)}$. Iteratively, at the p-th iteration we have

$$\begin{aligned}\underline{\boldsymbol{U}}^{*(p-1)} = \underline{\boldsymbol{U}}_{p+1}^{(0)} &+ \Delta t H^{(p-1)}(\Theta^{(p-1)} - \alpha\Gamma^{(p-1)})\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{*(p-2)}) \\ &+ \Delta t\alpha H^{(p-1)}\Gamma^{(p-1)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(p-1)}),\end{aligned} \tag{48}$$

$$\underline{\boldsymbol{U}}^{(p)} = \underline{\boldsymbol{U}}_{p+1}^{(0)} + \Delta t(\Theta^{(p)} - \alpha\Gamma^{(p)})\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{*(p-1)}) + \Delta t\alpha\Gamma^{(p)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(p)}). \tag{49}$$

We remark that, if the final number of nodes is fixed to be $M+1$, it is more "efficient" to perform M iterations with interpolation and the last one without interpolation between $\underline{\boldsymbol{U}}^{(M)}$ and $\underline{\boldsymbol{U}}^{(M+1)}$ to achieve the optimal accuracy with the minimal number of iterations thus getting

$$\underline{\boldsymbol{U}}^{(M+1)} = \underline{\boldsymbol{U}}^{(0)}_{M+1} + \Delta t(\Theta^{(M)} - \alpha\Gamma^{(M)})\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(M)}) + \Delta t\alpha\Gamma^{(M)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(M+1)}). \tag{50}$$

This is due to the fact that this last iteration allows to reach the maximal accuracy associated to $M+1$ nodes without requiring any additional structure. In fact, the matrices $\Theta^{(M)}$ and $\Gamma^{(M)}$ have been already used for the M-th iteration.

4.2 αDeCdu

Like the αDeCu methods, the αDeCdu methods are based on the introduction of an interpolation process between the iterations. In this case, the interpolated quantity is the function $\boldsymbol{G}(t, \boldsymbol{u}(t))$. The name is due to the fact that formally we interpolate $\frac{d}{dt}\boldsymbol{u}(t) = \boldsymbol{G}(t, \boldsymbol{u}(t))$.

4.2.1 Method

We start with two subtimenodes associated to t_n and $t_n + \Delta t$ and $\underline{\boldsymbol{U}}^{(0)} \in \mathbb{R}^{(2\times Q)}$. We perform the first iteration of the αDeC method getting $\underline{\boldsymbol{U}}^{(1)} \in \mathbb{R}^{(2\times Q)}$ which is $O(\Delta t^2)$-accurate. Then, we can compute $\boldsymbol{G}(t^0, \boldsymbol{u}^{0,(1)}) = \boldsymbol{G}(t_n, \boldsymbol{u}_n)$ and $\boldsymbol{G}(t^1, \boldsymbol{u}^{1,(1)})$ thanks to which we can get an $O(\Delta t^2)$-accurate global reconstruction of $\boldsymbol{G}(t, \boldsymbol{u}(t))$ in the interval $[t_n, t_n + \Delta t]$ through Lagrange interpolation. We thus perform an interpolation to get approximated values of $\boldsymbol{G}(t, \boldsymbol{u}(t))$ in three equispaced nodes in the interval $[t_n, t_n + \Delta t]$ which are used to compute $\underline{\boldsymbol{U}}^{(2)} \in \mathbb{R}^{(3\times Q)}$, which is $O(\Delta t^3)$-accurate. From the components of $\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(2)})$ we can get an $O(\Delta t^3)$-accurate reconstruction of $\boldsymbol{G}(t, \boldsymbol{u}(t))$ in four nodes in the interval $[t_n, t_n + \Delta t]$ and perform another iteration of the DeC. We can iteratively continue and get, at each iteration, $\underline{\boldsymbol{U}}^{(p)} \in \mathbb{R}^{((p+1)\times Q)}$, which is $O(\Delta t^{p+1})$-accurate. It is useful to define, for $p \geq 1$, the vector $\underline{\boldsymbol{G}}^{*(p)}$, containing the interpolated values of $\boldsymbol{G}(t, \boldsymbol{u}(t))$ in the interpolation subtimenodes $\underline{t}^{*(p)} = \underline{t}^{(p+1)}$. In particular, at the iteration p we have

$$\underline{\boldsymbol{G}}^{*(p-1)} = H^{(p-1)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(p-1)}). \tag{51}$$

Also in this case, the reader is referred to figure 2 for a better understanding.

4.2.2 Updating formula

We start with two subtimenodes associated to t_n and $t_n + \Delta t$, and therefore $\underline{\boldsymbol{U}}^{(0)} \in \mathbb{R}^{(2\times Q)}$. The first iteration is identical to (44)

$$\underline{\boldsymbol{U}}^{(1)} = \underline{\boldsymbol{U}}^{(0)} + \Delta t(\Theta^{(1)} - \alpha\Gamma^{(1)})\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(0)}) + \Delta t\alpha\Gamma^{(1)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(1)}). \tag{52}$$

Then, we interpolate $\boldsymbol{G}$ through the interpolation matrix $H^{(1)}$ getting

$$G^{*(1)} = H^{(1)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(1)}) \tag{53}$$

and perform the second iteration, which reads

$$\begin{aligned}\underline{\boldsymbol{U}}^{(2)} &= \underline{\boldsymbol{U}}^{(0)}_3 + \Delta t(\Theta^{(2)} - \alpha\Gamma^{(2)})G^{*(1)} + \Delta t\alpha\Gamma^{(2)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(2)}) \\ &= \underline{\boldsymbol{U}}^{(0)}_3 + \Delta t(\Theta^{(2)} - \alpha\Gamma^{(2)})H^{(1)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(1)}) + \Delta t\alpha\Gamma^{(2)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(2)}).\end{aligned} \tag{54}$$

Iteratively, we get the p-th iteration:

$$\underline{\boldsymbol{U}}^{(p)} = \underline{\boldsymbol{U}}^{(0)}_{p+1} + \Delta t(\Theta^{(p)} - \alpha\Gamma^{(p)})H^{(p-1)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(p-1)}) + \Delta t\alpha\Gamma^{(p)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(p)}). \tag{55}$$

Also in this case, if the final number of nodes is fixed to be $M+1$, the optimal choice is to make M iterations with the interpolation and the last one without interpolation, i.e.,

$$\underline{\boldsymbol{U}}^{(M+1)} = \underline{\boldsymbol{U}}^{(0)}_{M+1} + \Delta t(\Theta^{(M)} - \alpha\Gamma^{(M)})\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(M)}) + \Delta t\alpha\Gamma^{(M)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(M+1)}). \tag{56}$$

See figure 2 for a visual sketch of the method, following line αDeCdu.

4.3 αDeCu and αDeCdu with Gauss–Lobatto subtimenodes

As already pointed out, in this context $M+1$ Gauss–Lobatto subtimenodes provide an accuracy equal to $2M$. In this case, we start with two subtimenodes and we alternate iterations of the αDeC method and interpolations, adding one subtimenode at each iteration, until reaching a fixed number of subtimenodes, say $M+1$. Then, we continue with normal iterations of the αDeC until $P = 2M$ to get the maximal order of accuracy associated to such nodes, i.e., order $2M$. The updating formulas are identical to the ones already presented. The interpolation is not performed at the first iteration and from the $(M+1)$-th iteration on. Let us conclude this section with a final observation: in the context of a scheme of order P with Gauss–Lobatto subtimenodes, the most efficient choice is given by a final number of subtimenodes equal to $M+1$ with $M = \lceil \frac{P}{2} \rceil$ and P iterations.

Remark 4.1 (On the interpolation accuracy)**.** *A natural question that could arise in this context is why the interpolation processes should take place in the early iterations only and not rather when the maximal order of accuracy associated to a given number of Gauss–Lobatto subtimenodes is reached, assuming that a Gauss–Lobatto distribution is used also in the first iterations. The answer lies in the mismatch between the $(2p)$-th order of accuracy of the operator $\mathcal{L}^2_\Delta$ with $p+1$ Gauss–Lobatto subtimenodes and the $(p+1)$-th order of accuracy of the interpolation associated to the same number of subtimenodes. The interpolation represents a bottleneck which makes useless to perform more than one iteration with the same subtimenodes before the final distribution is settled.*

5 The DeC as RK

A general explicit RK method with S stages applied in the interval $[t_n, t_{n+1}]$, with $t_{n+1} = t_n + \Delta t$, reads

$$\begin{cases} \boldsymbol{y}^1 = \boldsymbol{u}_n, \\ \boldsymbol{y}^s = \boldsymbol{u}_n + \Delta t \sum_{r=1}^{s-1} a_{sr}\boldsymbol{G}(t_n + c_r\Delta t, \boldsymbol{y}^r), \quad \text{for } s = 2, \dots, S, \\ \boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \Delta t \sum_{r=1}^{S} b_r\boldsymbol{G}(t_n + c_r\Delta t, \boldsymbol{y}^r). \end{cases} \tag{57}$$

We adopt here the notation already introduced denoting by $\boldsymbol{u}_k$ an approximation of $\boldsymbol{u}(t_k)$, the exact solution to the ODEs system at the time t_k. The coefficients a_{sr}, c_r and b_r uniquely characterize

the RK method and they are stored in the Butcher tableau

$$\begin{array}{c|ccccc} c_1 & & & & & \\ c_2 & a_{2,1} & & & & \\ c_3 & a_{3,1} & a_{3,2} & & & \\ \vdots & \vdots & \vdots & \ddots & & \\ c_S & a_{S,1} & a_{S,2} & \cdots & a_{S,S-1} & \\ \hline & b_1 & b_2 & \cdots & b_{S-1} & b_S \end{array} . \tag{58}$$

The non-specified entries are assumed to be zero. It is also useful to define the matrix A and the vectors $\boldsymbol{b}$ and $\boldsymbol{c}$ in which we collect the related RK coefficients.

It is well known, as presented in [43, 37, 32], that DeC methods can be written into RK form. This also holds for the new methods, αDeCu and αDeCdu; we will show this fact in this section by explicitly constructing their Butcher tableaux.

Before doing this, let us introduce a simple notation to denote slices of matrices. Let $M \in \mathbb{R}^{(D_1+1)\times(D_2+1)}$ defined as

$$M = \begin{pmatrix} M_{0,0} & M_{0,1} & \dots & M_{0,D_2} \\ M_{1,0} & M_{1,1} & \dots & M_{1,D_2} \\ \vdots & \vdots & & \vdots \\ M_{D_1,0} & M_{D_1,1} & \dots & M_{D_1,D_2} \end{pmatrix}. \tag{59}$$

We define the slice from the i-th row to the j-th row (included) and from the k-th column to the ℓ-th column (included) as $M_{i:j,k:\ell}$. We omit the last index in case we want to include all the entries from the first index until the end, e.g. $M_{1:,1:}$ indicates the whole matrix without the first row and the first column. The same notation holds for vectors. Furthermore, we denote by $M_{i,:}$ and $M_{:,j}$ respectively the i-th row and the j-th column of the matrix M.

In order make the Butcher tableaux as compact as possible, the computation of the solution in the different subtimenodes at the first iteration will be always made through the explicit Euler method. This little modification has no impact on the formal accuracy, since the first iteration is meant to provide a first order approximation of the solution. In particular, when $\alpha = 0$ and we are in the context of autonomous systems, i.e., when the function $\boldsymbol{G}$ does not explicitly depend on t, the modification has no effect.

Moreover, it is useful to define here the vector $\underline{\beta}^{(p)} = \left(0, \frac{t^{1,(p)}-t_n}{\Delta t}, \dots, \frac{t^{p,(p)}-t_n}{\Delta t}\right)^T$ of the β^m coefficients in the different iterations. Of course, in the context of the original αDeC methods, this has no dependence on the iteration and, in such context, we have $\underline{\beta} = \left(0, \frac{t^1-t_n}{\Delta t}, \dots, \frac{t^M-t_n}{\Delta t}\right)^T$.

Finally, we will focus on the methods got by using equispaced subtimenodes. The extension to the Gauss–Lobatto case is trivial: it suffices to repeat for the needed number of times, $M-1$ in the optimal case, the block without interpolation, which is the one related to the final iteration of the standard method, in the Butcher tableaux.

5.1 αDeC

We recall the general updating formula of the αDeC methods in matricial form

$$\underline{\boldsymbol{U}}^{(p)} = \underline{\boldsymbol{U}}^{(0)} + \Delta t(\Theta - \alpha\Gamma)\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(p-1)}) + \Delta t\alpha\Gamma\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(p)}).$$

$\boldsymbol{c}$	$\boldsymbol{u}^0$	$\underline{\boldsymbol{u}}^{(1)}$	$\underline{\boldsymbol{u}}^{(2)}$	$\underline{\boldsymbol{u}}^{(3)}$	$\cdots$	$\underline{\boldsymbol{u}}^{(M)}$	$\underline{\boldsymbol{u}}_{:M-1}^{(M+1)}$	A
0	0							$\boldsymbol{u}^0$
$\underline{\beta}_{1:}$	$\underline{\beta}_{1:}$	$\underline{\underline{0}}$						$\underline{\boldsymbol{u}}^{(1)}$
$\underline{\beta}_{1:}$	$\Theta_{1:,0}$	$(\Theta-\alpha\Gamma)_{1:,1:}$	$\alpha\Gamma_{1:,1:}$	$\underline{\underline{0}}$				$\underline{\boldsymbol{u}}^{(2)}$
$\underline{\beta}_{1:}$	$\Theta_{1:,0}$	$\underline{\underline{0}}$	$(\Theta-\alpha\Gamma)_{1:,1:}$	$\alpha\Gamma_{1:,1:}$	$\underline{\underline{0}}$			$\underline{\boldsymbol{u}}^{(3)}$
	$\vdots$	$\vdots$		$\ddots$	$\ddots$			$\vdots$
	$\vdots$	$\vdots$			$\ddots$	$\ddots$		$\vdots$
$\underline{\beta}_{1:M-1}$	$\Theta_{1:M-1,0}$	$\underline{\underline{0}}$	$\cdots$	$\cdots$	$\underline{\underline{0}}$	$(\Theta-\alpha\Gamma)_{1:M-1,1:}$	$\alpha\Gamma_{1:M-1,1:M-1}$	$\underline{\boldsymbol{u}}_{:M-1}^{(M+1)}$
$\boldsymbol{b}$	$\Theta_{M,0}$	$\underline{\underline{0}}$	$\cdots$	$\cdots$	$\underline{\underline{0}}$	$(\Theta-\alpha\Gamma)_{M,1:}$	$\alpha\Gamma_{M,1:M-1}$	$\underline{\boldsymbol{u}}^{M,(M+1)}$

Table 1: RK structures for the original αDeC with equispaced subtimenodes, $\boldsymbol{c}$ at the left $\boldsymbol{b}$ at the bottom, A in the middle. We add on top and right sides the reference to the iteration steps

$\boldsymbol{c}$	$\boldsymbol{u}^0$	$\underline{\boldsymbol{u}}^{(1)}$	$\underline{\boldsymbol{u}}^{(2)}$	$\underline{\boldsymbol{u}}^{(3)}$	$\cdots$	$\underline{\boldsymbol{u}}^{(M-1)}$	$\underline{\boldsymbol{u}}^{(M)}$	A
0	0							$\boldsymbol{u}^0$
$\underline{\beta}_{1:}$	$\underline{\beta}_{1:}$	$\underline{\underline{0}}$						$\underline{\boldsymbol{u}}^{(1)}$
$\underline{\beta}_{1:}$	$\Theta_{1:,0}$	$\Theta_{1:,1:}$	$\underline{\underline{0}}$					$\underline{\boldsymbol{u}}^{(2)}$
$\underline{\beta}_{1:}$	$\Theta_{1:,0}$	$\underline{\underline{0}}$	$\Theta_{1:,1:}$	$\underline{\underline{0}}$				$\underline{\boldsymbol{u}}^{(3)}$
	$\vdots$	$\vdots$		$\ddots$	$\ddots$			$\vdots$
	$\vdots$	$\vdots$			$\ddots$	$\ddots$		$\vdots$
$\underline{\beta}_{1:}$	$\Theta_{1:,0}$	$\underline{\underline{0}}$	$\cdots$	$\cdots$	$\underline{\underline{0}}$	$\Theta_{1:,1:}$	$\underline{\underline{0}}$	$\underline{\boldsymbol{u}}^{(M)}$
$\boldsymbol{b}$	$\Theta_{M,0}$	$\underline{\underline{0}}$	$\cdots$	$\cdots$	$\cdots$	$\underline{\underline{0}}$	$\Theta_{M,1:}$	$\underline{\boldsymbol{u}}^{M,(M+1)}$

Table 2: RK structures for the original bDeC with equispaced subtimenodes, $\boldsymbol{c}$ at the left $\boldsymbol{b}$ at the bottom, A in the middle. We add on top and right sides the reference to the iteration steps

If we align each iteration one after the other and we consider each subtimenode of each iteration as a RK stage, we can pass to the RK formulation. In order to have a more compact formulation, we do not repeat the redundant states, i.e., all the $\boldsymbol{u}^{0,(p)}$ and we keep only $\boldsymbol{u}^0$ as representative of all of them. We remark that the first iteration is replaced by a simple Euler approximation for the sake of efficiency. This leads to the formulation (57) with coefficients A, $\boldsymbol{b}$ and $\boldsymbol{c}$ as defined in table 1. Note that Γ is a strictly lower triangular matrix, hence, even if there are blocks on the diagonal of A in table 1, the diagonal of A and all the upper triangular terms are actually 0, so the scheme is explicit. The number of stages of this formulation amounts to $S = M \times P$ holding for any distribution of subtimenodes. Therefore, we have $S = M^2 + M$ for equispaced subtimenodes with order $P = M + 1$ and $S = 2M^2$ for Gauss–Lobatto subtimenodes with order $P = 2M$.

If $\alpha = 0$, the αDeC method reduces to the bDeC method and the general table simplifies to table 2. In such case, we observe that we do not need the whole vector $\underline{\boldsymbol{u}}^{(P)}$, but we can just compute the component associated to the final subtimenode only with $\underline{\boldsymbol{u}}^{(P-1)}$, leading to a total number of RK stages equal to $S = M(P-1)+1$ and so $S = M^2+1$ if we have equispaced subtimenodes and $S = 2M^2 - M + 1$ if we have Gauss–Lobatto subtimenodes.

5.2 bDeCu

Let us recall the general updating formulas of the αDeCu methods in matricial form

$$\begin{aligned}\underline{\boldsymbol{U}}^{*(p-1)} = \underline{\boldsymbol{U}}^{(0)}_{p+1} &+ \Delta t H^{(p-1)}(\Theta^{(p-1)} - \alpha\Gamma^{(p-1)})\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{*(p-2)}) \\ &+ \Delta t\alpha H^{(p-1)}\Gamma^{(p-1)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(p-1)}),\end{aligned} \tag{60}$$

$$\underline{\boldsymbol{U}}^{(p)} = \underline{\boldsymbol{U}}^{(0)}_{p+1} + \Delta t(\Theta^{(p)} - \alpha\Gamma^{(p)})\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{*(p-1)}) + \Delta t\alpha\Gamma^{(p)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(p)}), \tag{61}$$

to which we need to add an initial iteration made with Euler and either a final iteration or, in the context of Gauss–Lobatto subtimenodes, some final iterations (M in the optimal case) of the standard αDeC method performed without interpolation. It is indeed possible to construct the Butcher tableaux of this family of methods. In particular, the stages of the RK method are given by all the components of the vectors $\underline{\boldsymbol{U}}^{(p)}$ and $\underline{\boldsymbol{U}}^{*(p)}$ (excluding the redundant states). Nevertheless, from easy computations one can see that for $\alpha \neq 0$ the number of stages of the αDeCu method coincides with the number of stages of the αDeC method without computational advantage under this point of view, see remark 5.1. For this reason and for the sake of compactness, we focus directly on the bDeCu method ($\alpha = 0$), for which we have a substantial computational advantage in terms of reduction of the number of stages with respect to the original bDeC. In such case, the updating formulas (60) and (61) reduce to

$$\underline{\boldsymbol{U}}^{*(p-1)} = \underline{\boldsymbol{U}}^{(0)}_{p+1} + \Delta t H^{(p-1)}\Theta^{(p-1)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{*(p-2)}), \tag{62}$$

$$\underline{\boldsymbol{U}}^{(p)} = \underline{\boldsymbol{U}}^{(0)}_{p+1} + \Delta t\Theta^{(p)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{*(p-1)}). \tag{63}$$

Let us observe that the updating formulae (62) and (63) involve only the evaluation of the flux of $\underline{\boldsymbol{U}}^*$ and, if we look at (62), we observe that the update of $\underline{\boldsymbol{U}}^{*(p-1)}$ only depends on $\underline{\boldsymbol{U}}^{*(p-2)}$, meaning that the scheme can essentially be rewritten only in terms of the vectors $\underline{\boldsymbol{U}}^{*(p)}$ (plus $\underline{\boldsymbol{U}}^{M,(P)}$) drastically reducing the number of stages. The RK coefficients are reported in table 3 in which we have

$$W^{(p)} := \begin{cases} H^{(p)}\Theta^{(p)} \in \mathbb{R}^{(p+2)\times(p+1)}, & \text{if } p = 2, \dots, M-1, \\ \Theta^{(M)} \in \mathbb{R}^{(M+1)\times(M+1)}, & \text{if } p \geq M. \end{cases} \tag{64}$$

We remark that the size of the vectors increases at each iteration. By a direct computation, we have that the total number of RK stages is given by $S = M \cdot (P-1) + 1 - \frac{(M-1)(M-2)}{2}$, so $\frac{(M-1)(M-2)}{2}$ less with respect to the original method. The formula is general and holds for both equispaced ($P = M+1$) and Gauss–Lobatto ($P = 2M$) subtimenodes.

Remark 5.1 (On the relation between stages and computational cost)**.** *The number of stages is not completely explanatory of the computational costs of the new algorithms. In the context of the novel methods, the cost associated to the computation of the different stages is not homogeneous as some of them are "properly" computed through the updating formula* (37) *of the original scheme, while the others are got through an interpolation process which is much cheaper. As an example,* (60) *can be computed as* $\underline{\boldsymbol{U}}^{*(p-1)} = H^{(p-1)}\underline{\boldsymbol{U}}^{(p-2)}$*. In particular, as already specified, the novel* α*DeCu methods for* $\alpha \neq 0$ *are characterized by the same number of stages as the original* α*DeC, nevertheless, roughly half of them is computed through interpolation. For this reason, they have been numerically investigated for* $\alpha = 1$*.*

$\boldsymbol{c}$	$\boldsymbol{u}^0$	$\underline{\boldsymbol{u}}^{*(1)}$	$\underline{\boldsymbol{u}}^{*(2)}$	$\underline{\boldsymbol{u}}^{*(3)}$	$\cdots$	$\underline{\boldsymbol{u}}^{*(M-2)}$	$\underline{\boldsymbol{u}}^{*(M-1)}$	$\underline{\boldsymbol{u}}^{(M)}$	A	dim
0	0								$\boldsymbol{u}^0$	1
$\underline{\beta}^{(2)}_{1:}$	$\underline{\beta}^{(2)}_{1:}$	$\underline{\underline{0}}$							$\underline{\boldsymbol{u}}^{*(1)}$	2
$\underline{\beta}^{(3)}_{1:}$	$W^{(2)}_{1:,0}$	$W^{(2)}_{1:,1:}$	$\underline{\underline{0}}$						$\underline{\boldsymbol{u}}^{*(2)}$	3
$\underline{\beta}^{(4)}_{1:}$	$W^{(3)}_{1:,0}$	$\underline{\underline{0}}$	$W^{(3)}_{1:,1:}$	$\underline{\underline{0}}$					$\underline{\boldsymbol{u}}^{*(3)}$	4
	$\vdots$	$\vdots$		$\ddots$	$\ddots$				$\vdots$	$\vdots$
	$\vdots$	$\vdots$			$\ddots$	$\ddots$			$\vdots$	$\vdots$
$\underline{\beta}^{(M)}_{1:}$	$W^{(M-1)}_{1:,0}$	$\underline{\underline{0}}$	$\cdots$	$\cdots$	$\underline{\underline{0}}$	$W^{(M-1)}_{1:,1:}$	$\underline{\underline{0}}$	$\underline{\underline{0}}$	$\underline{\boldsymbol{u}}^{*(M-1)}$	M
$\underline{\beta}^{(M)}_{1:}$	$W^{(M)}_{1:,0}$	$\underline{\underline{0}}$	$\cdots$	$\cdots$	$\cdots$	$\underline{\underline{0}}$	$W^{(M)}_{1:,1:}$	$\underline{\underline{0}}$	$\underline{\boldsymbol{u}}^{(M)}$	M
$\boldsymbol{b}$	$W^{(M+1)}_{M,0}$	$\underline{\underline{0}}$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\underline{\underline{0}}$	$W^{(M+1)}_{M,1:}$	$\underline{\boldsymbol{u}}^{M,(M+1)}$	

Table 3: RK structures for the bDeCu method, $\boldsymbol{c}$ at the left $\boldsymbol{b}$ at the bottom, A in the middle. We add on top and right sides the reference to the iteration steps

5.3 αDeCdu

Now, we apply the same technique on the methods that interpolate $\frac{du}{dt}$. Again, we start by recalling the updating formula of the method in matricial form

$$\underline{\boldsymbol{U}}^{(p)} = \underline{\boldsymbol{U}}^{(0)}_{p+1} + \Delta t(\Theta^{(p)} - \alpha\Gamma^{(p)})H^{(p-1)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(p-1)}) + \Delta t\alpha\Gamma^{(p)}\underline{\boldsymbol{G}}(\underline{\boldsymbol{U}}^{(p)}). \tag{65}$$

plus an initial Euler step and a final iteration or, for Gauss–Lobatto subtimenodes, at most M final iterations of the standard αDeC method without interpolation. One of the main differences with respect to the αDeCu methods is that this formulation does not require any extra interpolated state $\underline{\boldsymbol{U}}^{*(p)}$ as, here, the interpolation process involves the function $\boldsymbol{G}$. This results in a great advantage from a computational point of view also for $\alpha \neq 0$. The RK coefficients are reported in table 4, in which we have

$$X^{(p)} := \begin{cases} (\Theta^{(p)} - \alpha\Gamma^{(p)})H^{(p-1)} \in \mathbb{R}^{(p+1)\times p}, & \text{if } p = 1, \dots, M, \\ \Theta^{(M)} - \alpha\Gamma^{(M)} \in \mathbb{R}^{(M+1)\times(M+1)}, & \text{if } p > M, \end{cases} \tag{66}$$

$$Y^{(p)} := \begin{cases} \alpha\Gamma^{(p)} \in \mathbb{R}^{(p+1)\times(p+1)}, & \text{if } p = 1, \dots, M, \\ \alpha\Gamma^{(M)} \in \mathbb{R}^{(M+1)\times(M+1)}, & \text{if } p > M. \end{cases} \tag{67}$$

The number of stages in this case amounts to $S = MP - \frac{M(M-1)}{2}$, with a computational advantage of $\frac{M(M-1)}{2}$ with respect to the original method.

Also in this case, it is worth giving a particular attention to the method given by $\alpha = 0$. Again, the computation of $\underline{\boldsymbol{u}}^{M,(P)}$ without the need of the other components of $\underline{\boldsymbol{u}}^{(P)}$ can further reduce the number of stages to $S = M(P-1)+1-\frac{M(M-1)}{2}$. The Butcher tableaux of the bDeCdu method is given in table 5, where we have set

$$Z^{(p)} := \begin{cases} \Theta^{(p)}H^{(p-1)} \in \mathbb{R}^{(p+1)\times p}, & \text{if } p = 1, \dots, M, \\ \Theta^{(M)} \in \mathbb{R}^{(M+1)\times(M+1)}, & \text{if } p > M. \end{cases} \tag{68}$$

$\boldsymbol{c}$	$\boldsymbol{u}^0$	$\underline{\boldsymbol{u}}^{(1)}$	$\underline{\boldsymbol{u}}^{(2)}$	$\underline{\boldsymbol{u}}^{(3)}$	$\cdots$	$\underline{\boldsymbol{u}}^{(M-2)}$	$\underline{\boldsymbol{u}}^{(M-1)}$	$\underline{\boldsymbol{u}}^{(M)}$	$\underline{\boldsymbol{u}}_{:M-1}^{(M+1)}$	A	dim
0	0									$\boldsymbol{u}^0$	1
$\underline{\beta}_{1:}^{(1)}$	$\underline{\beta}_{1:}^{(1)}$	$\underline{\underline{0}}$								$\underline{\boldsymbol{u}}^{(1)}$	1
$\underline{\beta}_{1:}^{(2)}$	$X_{1:,0}^{(2)}$	$X_{1:,1:}^{(2)}$	$Y_{1:,1:}^{(2)}$							$\underline{\boldsymbol{u}}^{(2)}$	2
$\underline{\beta}_{1:}^{(3)}$	$X_{1:,0}^{(3)}$	$\underline{\underline{0}}$	$X_{1:,1:}^{(3)}$	$Y_{1:,1:}^{(3)}$						$\underline{\boldsymbol{u}}^{(3)}$	3
	$\vdots$	$\vdots$		$\ddots$	$\ddots$					$\vdots$	$\vdots$
	$\vdots$	$\vdots$			$\ddots$	$\ddots$				$\vdots$	$\vdots$
$\underline{\beta}_{1:}^{(M-1)}$	$X_{1:,0}^{(M-1)}$	$\underline{\underline{0}}$	$\cdots$	$\cdots$	$\underline{\underline{0}}$	$X_{1:,1:}^{(M-1)}$	$Y_{1:,1:}^{(M-1)}$	$\underline{\underline{0}}$		$\underline{\boldsymbol{u}}^{(M-1)}$	$M-1$
$\underline{\beta}_{1:}^{(M)}$	$X_{1:,0}^{(M)}$	$\underline{\underline{0}}$	$\cdots$	$\cdots$	$\cdots$	$\underline{\underline{0}}$	$X_{1:,1:}^{(M)}$	$Y_{1:,1:}^{(M)}$		$\underline{\boldsymbol{u}}^{(M)}$	M
$\underline{\beta}_{1:M-1}^{(M)}$	$X_{1:M-1,0}^{(M+1)}$	$\underline{\underline{0}}$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\underline{\underline{0}}$	$X_{1:M-1,1:}^{(M+1)}$	$Y_{1:M-1,1:M-1}^{(M+1)}$	$\underline{\boldsymbol{u}}_{1:M-1}^{(M+1)}$	$M-1$
$\boldsymbol{b}$	$X_{M,0}^{(M+1)}$	$\underline{\underline{0}}$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\underline{\underline{0}}$	$X_{M,1:}^{(M+1)}$	$Y_{M,1:M-1}^{(M+1)}$	$\underline{\boldsymbol{u}}^{M,(M+1)}$	

Table 4: RK structures for the αDeCdu method with equispaced subtimenodes, $\boldsymbol{c}$ at the left $\boldsymbol{b}$ at the bottom, A in the middle. We add on top and right sides the reference to the iteration steps

$\boldsymbol{c}$	$\boldsymbol{u}^0$	$\underline{\boldsymbol{u}}^{(1)}$	$\underline{\boldsymbol{u}}^{(2)}$	$\underline{\boldsymbol{u}}^{(3)}$	$\cdots$	$\underline{\boldsymbol{u}}^{(M-2)}$	$\underline{\boldsymbol{u}}^{(M-1)}$	$\underline{\boldsymbol{u}}^{(M)}$	A	dim
0	0								$\boldsymbol{u}^0$	1
$\underline{\beta}_{1:}^{(1)}$	$\underline{\beta}_{1:}^{(1)}$	$\underline{\underline{0}}$							$\underline{\boldsymbol{u}}^{(1)}$	1
$\underline{\beta}_{1:}^{(2)}$	$Z_{1:,0}^{(2)}$	$Z_{1:,1:}^{(2)}$	$\underline{\underline{0}}$						$\underline{\boldsymbol{u}}^{(2)}$	2
$\underline{\beta}_{1:}^{(3)}$	$Z_{1:,0}^{(3)}$	$\underline{\underline{0}}$	$Z_{1:,1:}^{(3)}$	$\underline{\underline{0}}$					$\underline{\boldsymbol{u}}^{(3)}$	3
	$\vdots$	$\vdots$		$\ddots$	$\ddots$				$\vdots$	$\vdots$
	$\vdots$	$\vdots$			$\ddots$	$\ddots$			$\vdots$	$\vdots$
$\underline{\beta}_{1:}^{(M-1)}$	$Z_{1:,0}^{(M-1)}$	$\underline{\underline{0}}$	$\cdots$	$\cdots$	$\underline{\underline{0}}$	$Z_{1:,1:}^{(M-1)}$	$\underline{\underline{0}}$	$\underline{\underline{0}}$	$\underline{\boldsymbol{u}}^{(M-1)}$	$M-1$
$\underline{\beta}_{1:}^{(M)}$	$Z_{1:,0}^{(M)}$	$\underline{\underline{0}}$	$\cdots$	$\cdots$	$\cdots$	$\underline{\underline{0}}$	$Z_{1:,1:}^{(M)}$	$\underline{\underline{0}}$	$\underline{\boldsymbol{u}}^{(M)}$	M
$\boldsymbol{b}$	$Z_{M,0}^{(M+1)}$	$\underline{\underline{0}}$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\underline{\underline{0}}$	$Z_{M,1:}^{(M+1)}$	$\underline{\boldsymbol{u}}^{M,(M+1)}$	

Table 5: RK structures for the bDeCdu method with equispaced subtimenodes, $\boldsymbol{c}$ at the left $\boldsymbol{b}$ at the bottom, A in the middle. We add on top and right sides the reference to the iteration steps

P=order	M	αDeC	bDeC	bDeCu	αDeCdu	bDeCdu
2	1	2	2	2	2	2
3	2	6	5	5	5	4
4	3	12	10	9	9	7
5	4	20	17	14	14	11
6	5	30	26	20	20	16
7	6	42	37	27	27	22
8	7	56	50	35	35	29
9	8	72	65	44	44	37
10	9	90	82	54	54	46
11	10	110	101	65	65	56
12	11	132	122	77	77	67
13	12	156	145	90	90	79

Table 6: Number of stages for the original (αDeC, bDeC) and novel (bDeCu, αDeCdu, bDeCdu) methods with equispaced subtimenodes: for αDeC and bDeC we have respectively $S = M \cdot P$ and $S = M \cdot (P-1) + 1$; for bDeCu we have $S = M \cdot (P-1) + 1 - \frac{(M-1)(M-2)}{2}$; for αDeCdu and bDeCdu we have respectively $S = M \cdot P - \frac{M(M-1)}{2}$ and $S = M \cdot (P-1) + 1 - \frac{M(M-1)}{2}$.

We conclude this section with two tables, table 6 and table 7, containing the number of stages of the original methods and of the novel ones respectively for equispaced and Gauss–Lobatto subtimenodes up to order 13.

6 Stability analysis

In this section, we study the stability of the novel DeC schemes. We will prove two original results. First, the stability functions (and regions) of the bDeCu and bDeCdu methods coincide with the ones of the original bDeC methods and do not depend on the distribution of the subtimenodes but only on the order. Second, if we fix the subtimenodes distribution and the order, the αDeCdu methods coincide with the αDeCu methods on linear problems. For all the schemes, we will show the stability region using some symbolical and numerical tools.

Let us start by reviewing some known results for RK methods [12, 62]. The linear stability of a RK scheme is tested on Dahlquist's problem

$$u' = \lambda u \tag{69}$$

with $\Re(\lambda) < 0$. Being the RK schemes linear, we can write them as $u_{n+1} = R(\lambda \Delta t) u_n$, with R the stability function of the method. The stability function for an RK scheme is defined as

$$R(z) = 1 + z\boldsymbol{b}^T (I - zA)^{-1} \mathbf{1} \tag{70}$$

where $\mathbf{1}$ is a vector with all the entries equal to 1. We have that the scheme is stable if $|R(\lambda \Delta t)| < 1$. The set of complex numbers z such that $|R(z)| < 1$ is called stability region. We remark that the stability function for explicit RK methods is a polynomial, indeed the inverse of $(I - zA)$ can be

P=order	M	αDeC	bDeC	bDeCu	αDeCdu	bDeCdu
2	1	2	2	2	2	2
3	2	6	5	5	5	4
4	2	8	7	7	7	6
5	3	15	13	12	12	10
6	3	18	16	15	15	13
7	4	28	25	22	22	19
8	4	32	29	26	26	23
9	5	45	41	35	35	31
10	5	50	46	40	40	36
11	6	66	61	51	51	46
12	6	72	67	57	57	52
13	7	91	85	70	70	64

Table 7: Number of stages for the original (αDeC, bDeC) and novel (bDeCu, αDeCdu, bDeCdu) methods with Gauss-Lobatto subtimenodes: for αDeC and bDeC we have respectively $S = M \cdot P$ and $S = M \cdot (P-1) + 1$; for bDeCu we have $S = M \cdot (P-1) + 1 - \frac{(M-1)(M-2)}{2}$; for αDeCdu and bDeCdu we have respectively $S = M \cdot P - \frac{M(M-1)}{2}$ and $S = M \cdot (P-1) + 1 - \frac{M(M-1)}{2}$.

written in Taylor expansion as

$$(I - zA)^{-1} = \sum_{r=0}^{\infty} z^r A^s = I + zA + z^2A^2 + \dots, \tag{71}$$

and, since A is strictly lower triangular, it is nilpotent. This means that there exists an integer r such that $A^r = \underline{\underline{0}}$ and the minimum of these natural numbers $\mathcal{N}$ is called degree of nilpotence. By definition of L, it is clear that $A^{\mathcal{N}+r} = \underline{\underline{0}}$ for all $r \geq 0$. Moreover, it is also clear that $\mathcal{N} \leq S$, where S is the number of stages of the RK methods and the dimension of the matrix A. Hence, $R(z)$ is a polynomial in z with degree at most equal to S.

The following theorem is proved in [62].

Theorem 6.1. *If the RK method is of order P, then*

$$R(z) = 1 + z + \frac{z^2}{2!} + \dots + \frac{z^P}{P!} + O(z^{P+1}). \tag{72}$$

So, we know the first $P+1$ terms of the stability functions $R(\cdot)$ for all the DeCs we presented above of order P.

Theorem 6.2. *The stability function of any bDeC, bDeCu and bDeCdu method of order P is*

$$R(z) = \sum_{r=0}^{P} \frac{z^r}{r!} = 1 + z + \frac{z^2}{2!} + \dots + \frac{z^P}{P!} \tag{73}$$

and does not depend on the distribution of the subtimenodes.

Proof. The proof of this theorem relies only on the block structure of the matrix A for such schemes. In all these cases, the matrix A can be written as

$$A = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 & 0 \\ \star & 0 & 0 & \dots & 0 & 0 \\ \star & \star & 0 & \dots & 0 & 0 \\ \star & 0 & \star & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \star & 0 & 0 & \cdots & \star & 0 \end{pmatrix}, \tag{74}$$

where $\star$ are some non-zero block matrices and the 0 are some zero block matrices of different sizes. The number of blocks in each line and row of these matrices is P, the order of the scheme. By induction, we can prove that A^k has zeros in the upper triangular part, in the main block diagonal, and in all the $k-1$ block diagonals below the main diagonal, i.e., $(A^k)_{i,j} = 0$ if $i < j+k$, where the indexes here refer to the blocks. Indeed, it is true that $A_{i,j} = 0$ if $i < j+1$. Now, let us consider the entry $(A^{k+1})_{i,j}$ with $i < j+k+1$, i.e., $i-k < j+1$. It is defined as

$$(A^{k+1})_{i,j} = \sum_w (A^k)_{i,w} A_{w,j}. \tag{75}$$

Now, we can prove that all the terms of the sum are 0. Let $w < j+1$, then $A_{w,j} = 0$ because of the structure of A; while, if $w \geq j+1 > i-k$, we have that $i < w+k$, so $(A^k)_{i,w} = 0$ by induction.

In particular, this means that $A^P = \underline{0}$, because i is always smaller than $j+P$ as P is the number of the block matrices that we have. Hence,

$$(I - zA)^{-1} = \sum_{r=0}^{\infty} z^r A^s = \sum_{r=0}^{P-1} z^r A^s = I + zA + z^2A^2 + \cdots + z^{P-1}A^{P-1}. \tag{76}$$

Plugging this result into (70), we can state that the stability function $R(z)$ is a polynomial of degree P, the order of the scheme. Since all the terms of order lower or equal to P must agree with the expansion of the exponential function as stated in theorem 6.1, the stability function must be

$$R(z) = \sum_{r=0}^{P} \frac{z^r}{r!} = 1 + z + \frac{z^2}{2!} + \cdots + \frac{z^P}{P!}. \tag{77}$$

Finally, let us notice that no assumption has been done on the distribution of the subtimenodes, hence, the result is general for any distribution. □

In the following, we will show that, given a certain order P and a distribution of subtimenodes, the αDeCu and αDeCdu methods are equivalent on linear problems and, as a consequence, they share the same stability functions. This is interesting when comparing the two methods as we can always choose the most efficient one, without losing stability properties.

Theorem 6.3 (Equivalence on linear problems)**.** *Given an order P, a distribution of subtimenodes and $\alpha \in [0,1]$, the schemes αDeCu and αDeCdu applied to linear systems are equivalent.*

Proof. Without loss of generality, we can focus on Dahlquist's equation $u' = \lambda u$. Since the schemes are linear, the same arguments would apply component-wise also on linear systems of equations.

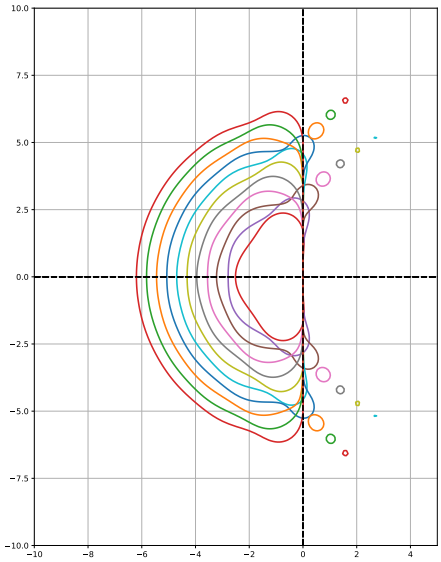

Figure 3: Stability regions of bDeC, bDeCu and bDeCdu (equivalent on linear problems) for any distribution of subtimenodes with order from 3 to 13

Let us start by explicitly writing down the general updating formula (55) of the αDeCdu method for Dahlquist's equation:

$$\underline{\boldsymbol{U}}^{(p)} = \underline{\boldsymbol{U}}^{(0)}_{p+1} + \omega(\Theta^{(p)} - \alpha\Gamma^{(p)})H^{(p-1)}\underline{\boldsymbol{U}}^{(p-1)} + \omega\alpha\Gamma^{(p)}\underline{\boldsymbol{U}}^{(p)}, \tag{78}$$

where we have set $\omega := \Delta t\lambda$. For the αDeCu method, the updating formula (49) becomes

$$\underline{\boldsymbol{U}}^{(p)} = \underline{\boldsymbol{U}}^{(0)}_{p+1} + \omega(\Theta^{(p)} - \alpha\Gamma^{(p)})\underline{\boldsymbol{U}}^{*(p-1)} + \omega\alpha\Gamma^{(p)}\underline{\boldsymbol{U}}^{(p)}, \tag{79}$$

now, using the definition of $\underline{\boldsymbol{U}}^{*(p-1)} = H^{(p-1)}\underline{\boldsymbol{U}}^{(p-1)}$, we obtain

$$\underline{\boldsymbol{U}}^{(p)} = \underline{\boldsymbol{U}}^{(0)}_{p+1} + \omega(\Theta^{(p)} - \alpha\Gamma^{(p)})H^{(p-1)}\underline{\boldsymbol{U}}^{(p-1)} + \omega\alpha\Gamma^{(p)}\underline{\boldsymbol{U}}^{(p)}, \tag{80}$$

which coincides with (78). In fact, apart from these direct computations, another way to show the equivalence is observing that on linear problems interpolating $\boldsymbol{u}$ and then applying $\boldsymbol{G}$ is formally equivalent to interpolating $\boldsymbol{G}$.

This means that each iteration of the two modified schemes coincide. Moreover, $\underline{\boldsymbol{U}}^{(1)}$ is a simple forward Euler step for both algorithms. Finally, the final update does not involve any interpolation for both schemes. Therefore, the two schemes coincide on linear problems. □

As stated above, the immediate consequence is that the stability functions (and regions) of the αDeCu and the αDeCdu methods are identical.

In figure 3, we depict the stability region of the classical bDeC method and of the new methods bDeCu and bDeCdu from order 3 until order 13, we remark that there is no difference in terms of stability between bDeC, bDeCu and bDeCdu, nor dependence on the distribution of the subtimenodes. In figure 4, instead, we plot the stability regions of the classical sDeC method and of the new methods sDeCu and sDeCdu, we remark that sDeCu and sDeCdu have the same stability regions.

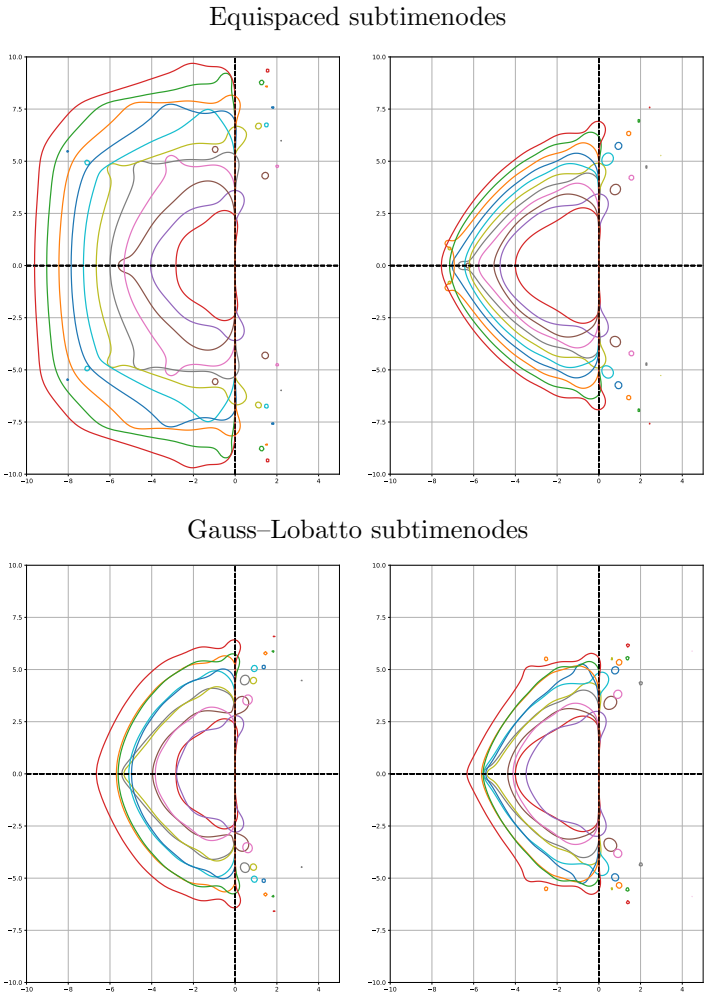

Figure 4: Stability regions of sDeC, sDeCu and sDeCdu with order from 3 to 13: sDeC on the left, sDeCu and sDeCdu (equivalent on linear problems) on the right

7 Application to hyperbolic PDEs

In this section, we apply the novel explicit efficient DeC techniques to hyperbolic PDEs. It is straightforward to do it when the PDE is discretized with the method of lines, i.e., the spatial and the time discretization are handled separately. This is the case for many finite volume (FV) and discontinuous Galerkin Finite Element (DG) methods, where there is no mass matrix or it is block diagonal and easy to invert. On the other side, the application of the novel efficient DeC technique to a CG framework is more complicated, because the mass matrix has only a sparse structure and, hence, global linear systems should be solved at each time step, wasting computational time. We will focus on the latter case, which is more challenging, and, in particular, on two strategies that

allow to avoid the solution of global linear systems. The first one is to use particular basis functions like the Cubature elements introduced in [19] and studied in [36, 51, 44, 45] which, combined with a suitable choice of the quadrature formula, allow to achieve a high-order lumping of the mass matrix. The diagonal mass matrix allows to get a system of ODEs like the one described in the previous section and, hence, the novel methods can be applied in a straightforward way. The second strategy, instead, has been introduced by Abgrall in [5] and it is based on the concept of residual [1, 53, 2, 4]. It consists in introducing, together with the low-order forward Euler approximation in time, a first-order lumping in the mass matrix of the operator $\mathcal{L}^1_\Delta$ to make the resulting scheme fully explicit.

In this section, we will present the general semidiscrete formulation of a CG scheme and we will describe the two mentioned strategies to numerically solve it avoiding global linear systems. As the first strategy leads to an ODE system identical to the one presented in the previous section, we will dedicate more attention to the second one by defining the operators $\mathcal{L}^1_\Delta$ and $\mathcal{L}^2_\Delta$, characterizing the updating formulas and discussing the problem of modifying the method, via interpolation processes, to make it more efficient. Without loss of generality, we will consider the bDeC formulation. The proofs of the properties of the operators of the original formulation for ODEs cannot be extended to this context in a straightforward way. Additional hypotheses and a careful choice of the norms on the spaces X and Y are required. We provide the proofs in the supplementary material. To our knowledge, this is the only formal proof alternative to the one presented in [5].

7.1 Continuous Galerkin FEM

The general form of a hyperbolic system of balance laws reads

$$\frac{\partial}{\partial t}\boldsymbol{u}(\boldsymbol{x},t) + \mathrm{div}_{\boldsymbol{x}}\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{x},t)) = \boldsymbol{S}(\boldsymbol{x},\boldsymbol{u}(\boldsymbol{x},t)) \quad (\boldsymbol{x},t) \in \Omega \times \mathbb{R}^+_0 \tag{81}$$

where $\boldsymbol{u} : \Omega \times \mathbb{R}^+_0 \to \mathbb{R}^Q$ with some initial condition $\boldsymbol{u}(\boldsymbol{x},0) = \boldsymbol{u}_0(\boldsymbol{x})$ on the space domain $\Omega \subseteq \mathbb{R}^D$, which we assume here to be bounded, and boundary conditions on $\partial\Omega$. In particular, we would like to find $\boldsymbol{u}_h$, an approximation of $\boldsymbol{u}$, in $\overline{\Omega} \times [0,T]$ of the weak-entropy solution of our problem. Basically, the Galerkin FEM consists in a projection of the analytical weak formulation in space of our initial problem over a finite dimensional space. We consider a tessellation $\mathcal{T}_h$ of the closure of the spatial domain with characteristic length h, i.e., a finite set of D-dimensional nonoverlapping convex polytopal closed subsets K of $\overline{\Omega}$, which cover $\overline{\Omega}$ exactly and such that $\sup_{K\in\mathcal{T}_h} \mathrm{diam}(K) \leq h$. We introduce the finite dimensional space V_h of continuous piecewise polynomial functions defined as

$$V_h = \{g \in C^0(\overline{\Omega}) \quad s.t. \quad g|_K \in \mathbb{P}_M(K) \quad \forall K \in \mathcal{T}_h\} \tag{82}$$

with $\mathbb{P}_M(K)$ denoting the space of polynomials of degree M defined on the element K. We choose a basis $\{\varphi_i\}_{i=1,\dots,I}$ of V_h, for example the Lagrange polynomials or the Bernstein polynomials, which is such that each basis function φ_i can be associated to a point $\boldsymbol{x}_i$ located somewhere in the closure of the spatial domain. Such points are usually referred as degrees of freedom (DoFs). Further, we assume that the basis functions are normalized in such a way that $\sum_{i=1}^I \varphi_i(\boldsymbol{x}) \equiv 1 \ \forall \boldsymbol{x} \in \overline{\Omega}$ and that each basis function φ_i has its support contained in the union of the elements containing the node $\boldsymbol{x}_i$ to which it is associated, i.e., $\varphi_i \in C^0_o(\cup_{K\in K_i} K)$, where

$$K_i = \{K \in \mathcal{T}_h \ s.t. \ \boldsymbol{x}_i \in K\} \tag{83}$$

is the set of the elements containing the DoF $\boldsymbol{x}_i$. We project the weak formulation of the original PDE problem onto V_h, i.e., we look for $\boldsymbol{u}_h(\boldsymbol{x},t) = \sum_{j=1}^{I} \boldsymbol{c}_j(t)\varphi_j(\boldsymbol{x}) \in V_h^Q$ such that for any $i = 1, \dots, I$

$$\int_\Omega \left(\frac{\partial}{\partial t}\boldsymbol{u}_h(\boldsymbol{x},t) + \mathrm{div}_{\boldsymbol{x}}\boldsymbol{F}(\boldsymbol{u}_h(\boldsymbol{x},t)) - \boldsymbol{S}(\boldsymbol{x},\boldsymbol{u}_h(\boldsymbol{x},t)) \right) \varphi_i(\boldsymbol{x})d\boldsymbol{x} + \boldsymbol{ST}_i(\boldsymbol{u}_h) = \boldsymbol{0}, \tag{84}$$

where the divergence in space is meant in the weak sense and the stabilization term $\boldsymbol{ST}_i(\boldsymbol{u}_h)$ is added to avoid the instabilities associated to central schemes. Actually, by the argument that each basis function has support contained in the union of the elements containing the DoF to which it is associated, it is possible to recast (84) as

$$\sum_{K \in K_i} \sum_{\boldsymbol{x}_j \in K} \frac{d}{dt}\boldsymbol{c}_j(t) \int_K \varphi_i(\boldsymbol{x})\varphi_j(\boldsymbol{x})d\boldsymbol{x} + \boldsymbol{\phi}_i(\boldsymbol{c}(t)) = \boldsymbol{0}, \quad \forall i = 1, \dots, I, \tag{85}$$

where, in order to have a more compact notation, we collected the terms not strictly related to the temporal evolution in (84) in a single term, to which we will refer to as "space residual", defined as

$$\boldsymbol{\phi}_i(\boldsymbol{c}(t)) = \sum_{K \in K_i} \int_K \left(\mathrm{div}_{\boldsymbol{x}}\boldsymbol{F}(\boldsymbol{u}_h(\boldsymbol{x},t)) - \boldsymbol{S}(\boldsymbol{x},\boldsymbol{u}_h(\boldsymbol{x},t))\right) \varphi_i(\boldsymbol{x})d\boldsymbol{x} + \boldsymbol{ST}_i(\boldsymbol{u}_h). \tag{86}$$

It is not difficult to show that (85), after the temporal discretization, is a huge system of ODEs, in the unknowns $\boldsymbol{c}_i(t)$, characterized by a huge global mass matrix. We would like to avoid solving linear systems in the numerical solution of (85).

One possibility is given by a careful choice of the basis functions and of the quadrature points through the Cubature elements [19, 36]. The key idea is to find a high order quadrature formula with positive weights and use its quadrature points to define the bases through Lagrangian polynomials. Adopting such elements and the induced quadrature formula, one obtains a spectral formulation characterized by a diagonal mass matrix without spoiling the high order accuracy. In practice, this allows to get a system of ODEs like the one presented in the previous section and the DeC formulations that we have introduced can be applied without any problem to such system. Examples of such basis functions are given by the Lagrange polynomials associated to the Gauss–Lobatto points in one-dimensional domains and the Cubature elements introduced in [19] in two-dimensional domains.

The alternative, introduced in [5], involves a modification of the original method for ODEs and it is explained in the following.

For what follows, it is useful to define here the vector $\boldsymbol{c}(t) \in \mathbb{R}^{I \times Q}$, in which we collect all the Q-dimensional components $\boldsymbol{c}_i(t)$ associated to the DoFs.

Remark 7.1 (Link with Residual Distribution)**.** *It is possible to put the Continuous FEM formulation* (85) *in a Residual Distribution framework. The interested reader is referred to [6].*

7.2 DeC for CG

Here, we will define the operators $\mathcal{L}_\Delta^1$ and $\mathcal{L}_\Delta^2$ of the DeC formulation for CG FEM discretizations proposed by Abgrall and we will characterize the updating formula. The key idea is to modify the original method for ODEs by introducing a first order mass lumping in the operator $\mathcal{L}_\Delta^1$. This modification has been originally introduced in the context of a bDeC formulation but can actually

be performed on any general αDeC method. Without loss of generality, here we will consider the bDeC formulation as the reference one. In this context, the parameter Δ of the DeC is the mesh parameter h of the space discretization. We assume CFL conditions on the timestep of the type $\Delta t \leq Ch$ for some constant C not dependent on the discretization. In particular, we will have $\Delta \approx \Delta t \approx h$.

7.2.1 Definition of $\mathcal{L}^2_\Delta$

The operator $\mathcal{L}^2_\Delta$ is the high order implicit operator that we would like to solve. Its definition is not very different from the one seen in the context of the bDeC for ODEs. We introduce the $M+1$ subtimenodes t^m with $m = 0, \dots, M$ in the interval $[t_n, t_n + \Delta t]$ in which we will consider the approximations of the values of the solution to our system of ODEs. We refer to $\boldsymbol{c}(t^m)$ as the exact solution in the node t^m and to $\boldsymbol{c}^m$ as the approximation of the solution in the same node. Clearly, in this case $\boldsymbol{c}(t^m)$ and $\boldsymbol{c}^m$ contain as components all the coefficients corresponding to the spatial DoFs, i.e., respectively the vectors $\boldsymbol{c}_i(t^m)$ of the exact coefficients in the i-th DoF at the time t^m and the vectors $\boldsymbol{c}_i^m$ of the approximated ones. As usual, for the first subtimenode we set $\boldsymbol{c}^0 = \boldsymbol{c}(t^0) = \boldsymbol{c}(t_n) = \boldsymbol{c}_n$ without any approximation. Starting from the exact integration of (85) over $[t^0, t^m]$ and substituting $\boldsymbol{\phi}_i(\boldsymbol{c}(t))$ with its M-order interpolation in time associated to the $M+1$ subtimenodes we get

$$\sum_{K \in K_i} \sum_{\boldsymbol{x}_j \in K} \left(\int_K \varphi_i(\boldsymbol{x}) \varphi_j(\boldsymbol{x}) d\boldsymbol{x} \right) \left(\boldsymbol{c}_j^m - \boldsymbol{c}_j^0 \right) + \Delta t \sum_{\ell=0}^{M} \theta_\ell^m \boldsymbol{\phi}_i(\boldsymbol{c}^\ell) = \boldsymbol{0} \tag{87}$$

for any $i = 1, \dots, I$ and $m = 1, \dots, M$. Therefore, we can define the operator $\mathcal{L}^2_\Delta : \mathbb{R}^{(I \times Q \times M)} \to \mathbb{R}^{(I \times Q \times M)}$ as

$$\mathcal{L}^2_\Delta(\underline{\boldsymbol{c}}) = \left(\mathcal{L}^2_{\Delta,1}(\underline{\boldsymbol{c}}), \mathcal{L}^2_{\Delta,2}(\underline{\boldsymbol{c}}), \dots, \mathcal{L}^2_{\Delta,I}(\underline{\boldsymbol{c}}) \right), \quad \forall \underline{\boldsymbol{c}} \in \mathbb{R}^{(I \times Q \times M)}, \tag{88}$$

where, for any i, we have

$$\mathcal{L}^2_{\Delta,i}(\underline{\boldsymbol{c}}) = \begin{pmatrix} \sum_{K \in K_i} \sum_{\boldsymbol{x}_j \in K} \left(\int_K \varphi_i(\boldsymbol{x}) \varphi_j(\boldsymbol{x}) d\boldsymbol{x} \right) \left(\boldsymbol{c}_j^1 - \boldsymbol{c}_j^0 \right) + \Delta t \sum_{\ell=0}^{M} \theta_\ell^1 \boldsymbol{\phi}_i(\boldsymbol{c}^\ell) \\ \vdots \\ \sum_{K \in K_i} \sum_{\boldsymbol{x}_j \in K} \left(\int_K \varphi_i(\boldsymbol{x}) \varphi_j(\boldsymbol{x}) d\boldsymbol{x} \right) \left(\boldsymbol{c}_j^m - \boldsymbol{c}_j^0 \right) + \Delta t \sum_{\ell=0}^{M} \theta_\ell^m \boldsymbol{\phi}_i(\boldsymbol{c}^\ell) \\ \vdots \\ \sum_{K \in K_i} \sum_{\boldsymbol{x}_j \in K} \left(\int_K \varphi_i(\boldsymbol{x}) \varphi_j(\boldsymbol{x}) d\boldsymbol{x} \right) \left(\boldsymbol{c}_j^M - \boldsymbol{c}_j^0 \right) + \Delta t \sum_{\ell=0}^{M} \theta_\ell^M \boldsymbol{\phi}_i(\boldsymbol{c}^\ell) \end{pmatrix}. \tag{89}$$

The solution $\underline{\boldsymbol{c}}_\Delta$ to $\mathcal{L}^2_\Delta = \boldsymbol{0}$ is $(M+1)$-order accurate in the sense that it would contain as components $(M+1)$-order accurate approximations of the coefficients which represent the exact solution to (85) in all the subtimenodes t^m for $m = 1, \dots, M$. Unfortunately, the problem $\mathcal{L}^2_\Delta(\underline{\boldsymbol{c}}) = \boldsymbol{0}$ is a huge nonlinear system.

7.2.2 Definition of $\mathcal{L}^1_\Delta$

As anticipated, in order to define the operator $\mathcal{L}^1_\Delta$ in this context, we couple the forward Euler time discretization with a first order mass lumping, obtaining

$$C_i\left(\boldsymbol{c}_i^m - \boldsymbol{c}_i^0\right) + \Delta t \beta^m \boldsymbol{\phi}_i(\boldsymbol{c}^0) = \mathbf{0}, \quad \forall i = 1, \dots, I, \quad \forall m = 1, \dots, M, \tag{90}$$

where C_i are constant quantities defined as

$$C_i = \int_\Omega \varphi_i(\boldsymbol{x}) d\boldsymbol{x} = \sum_{K \in K_i} \int_K \varphi_i(\boldsymbol{x}) d\boldsymbol{x}, \quad \forall i = 1, \dots, I. \tag{91}$$

Remark 7.2 (Choice of the basis functions)**.** *For any m and i, it is straightforward to explicitly compute $\boldsymbol{c}_i^m$ from (90) if and only if $C_i \neq 0$. This means that the construction of the operator $\mathcal{L}^1_\Delta$ is not always well-posed for any arbitrary basis of polynomials $\{\varphi_i\}_{i=1,\dots,I}$. With Lagrange polynomials it can happen that $\int_\Omega \varphi_i(\boldsymbol{x}) d\boldsymbol{x} = 0$ for some i, for example for degree $M = 2$ on triangular meshes. On the other side, the construction is always well-posed with Bernstein bases for which $C_i > 0$ $\forall i$.*

For the sake of brevity, we prove in the supplementary material that the solution to (90) is a first order approximation of $\boldsymbol{c}_i(t^m)$. Directly from (90), we can define the explicit low order operator $\mathcal{L}^1_\Delta : \mathbb{R}^{(I \times Q \times M)} \to \mathbb{R}^{(I \times Q \times M)}$ as

$$\mathcal{L}^1_\Delta(\underline{\boldsymbol{c}}) = \left(\mathcal{L}^1_{\Delta,1}(\underline{\boldsymbol{c}}), \mathcal{L}^1_{\Delta,2}(\underline{\boldsymbol{c}}), \dots, \mathcal{L}^1_{\Delta,I}(\underline{\boldsymbol{c}})\right), \quad \forall \underline{\boldsymbol{c}} \in \mathbb{R}^{(I \times Q \times M)}, \tag{92}$$

where for any i we have

$$\mathcal{L}^1_{\Delta,i}(\underline{\boldsymbol{c}}) = \begin{pmatrix} C_i\left(\boldsymbol{c}_i^1 - \boldsymbol{c}_i^0\right) + \Delta t \beta^1 \boldsymbol{\phi}_i(\boldsymbol{c}^0) \\ \vdots \\ C_i\left(\boldsymbol{c}_i^m - \boldsymbol{c}_i^0\right) + \Delta t \beta^m \boldsymbol{\phi}_i(\boldsymbol{c}^0) \\ \vdots \\ C_i\left(\boldsymbol{c}_i^M - \boldsymbol{c}_i^0\right) + \Delta t \beta^M \boldsymbol{\phi}_i(\boldsymbol{c}^0) \end{pmatrix}. \tag{93}$$

7.2.3 Updating formula

Let us characterize the iterative formula (4) in this context. We have

$$\mathcal{L}^1_\Delta(\underline{\boldsymbol{c}}^{(p)}) = \mathcal{L}^1_\Delta(\underline{\boldsymbol{c}}^{(p-1)}) - \mathcal{L}^2_\Delta(\underline{\boldsymbol{c}}^{(p-1)}), \quad p = 1, \dots, P, \tag{94}$$

where $\underline{\boldsymbol{c}}^{(p)} \in \mathbb{R}^{(I \times Q \times M)}$, and $\underline{\boldsymbol{c}}^{(p)}$ denotes the approximation obtained at the p-th iteration and it is made by M components $\boldsymbol{c}^{m,(p)}$ corresponding to the approximations of the solution to (85) in the subtimenodes t^m $m = 1, \dots, M$. Each of them contains I components $\boldsymbol{c}_i^{m,(p)}$ with dimension Q. The reader is referred again to figure 1 for a better understanding. On the x-axis we have the iterations while on the y-axis we have the subtimenodes. Just like in the ODE case, the procedure (94) results in an explicit iterative algorithm due to the fact that the operator $\mathcal{L}^1_\Delta$ is explicit. The update of the component associated to the general DoF i in the m-th subtimenode in the p-th iteration reads

$$\begin{aligned} C_i\left(\boldsymbol{c}_i^{m,(p)} - \boldsymbol{c}_i^0\right) + \Delta t \beta^m \boldsymbol{\phi}_i(\boldsymbol{c}^0) &= C_i\left(\boldsymbol{c}_i^{m,(p-1)} - \boldsymbol{c}_i^0\right) + \Delta t \beta^m \boldsymbol{\phi}_i(\boldsymbol{c}^0) \\ - \sum_{K \in K_i} \sum_{\boldsymbol{x}_j \in K} \left(\boldsymbol{c}_j^{m,(p-1)} - \boldsymbol{c}_j^0\right) \int_K \varphi_i(\boldsymbol{x}) \varphi_j(\boldsymbol{x}) d\boldsymbol{x} &+ \Delta t \sum_{\ell=0}^{M} \theta_\ell^m \boldsymbol{\phi}_i(\boldsymbol{c}^{\ell,(p-1)}), \end{aligned} \tag{95}$$

from which we get

$$\boldsymbol{c}_i^{m,(p)} = \boldsymbol{c}_i^{m,(p-1)} - \frac{1}{C_i}\left[\sum_{K\in K_i}\sum_{\boldsymbol{x}_j\in K}\left(\boldsymbol{c}_j^{m,(p-1)} - \boldsymbol{c}_j^0\right)\int_K \varphi_i(\boldsymbol{x})\varphi_j(\boldsymbol{x})d\boldsymbol{x} + \Delta t\sum_{\ell=0}^{M}\theta_\ell^m \boldsymbol{\phi}_i(\boldsymbol{c}^{\ell,(p-1)})\right]. \tag{96}$$

We remark that also in this case we assume $\boldsymbol{c}_i^{m,(p)} = \boldsymbol{c}_i(t_n)$ whenever p or m are equal to 0. Finally, for what concerns the optimal number of iterations, analogous considerations to the ones that we have made in the ODE case hold.

7.2.4 αDeCu for CG

Just like we did in the ODE context, it is possible to modify the original DeC for hyperbolic problems presented above to get a new more efficient method by introducing interpolation processes between the iterations. The underlying idea is the same, we increase the number of subtimenodes as the accuracy of the approximation increases. Among the two presented interpolation strategies, only the first one, involving the interpolation of the solution, can be extended in a straightforward way to this context. At the general iteration p, the interpolation process allows to get $\underline{\boldsymbol{c}}^{*(p-1)}$ from $\underline{\boldsymbol{c}}^{(p-1)}$ and then we perform the iteration via (96) getting

$$\boldsymbol{c}_i^{m,(p)} = \boldsymbol{c}_i^{*m,(p-1)} - \frac{1}{C_i}\left[\sum_{K\in K_i}\sum_{\boldsymbol{x}_j\in K}\left(\boldsymbol{c}_j^{*m,(p-1)} - \boldsymbol{c}_j^0\right)\int_K \varphi_i(\boldsymbol{x})\varphi_j(\boldsymbol{x})d\boldsymbol{x} + \Delta t\sum_{\ell=0}^{M}\theta_\ell^m \boldsymbol{\phi}_i(\boldsymbol{c}^{*l,(p-1)})\right]. \tag{97}$$

Again, we remark that such interpolation processes can be introduced in the context of any αDeC for hyperbolic problems. Finally, it is worth to observe that the resulting new schemes cannot be written in RK form due to the difference of the mass matrices in $\mathcal{L}^1_\Delta$ and $\mathcal{L}^2_\Delta$ (lumped and classic respectively).

8 Application to adaptivity

In this last theoretical section, before going to the numerical results, we will see how to exploit the interpolation processes in the new schemes, αDeCu and αDeCdu, to design adaptive methods. In the context of an original αDeC method with a fixed number of subtimenodes, iteration by iteration, we increase the order of accuracy with respect to the solution $\underline{\boldsymbol{u}}_\Delta$ to the operator $\mathcal{L}^2_\Delta$, for this reason, performing a number of iterations higher than the order of accuracy of the discretization adopted in the construction of the operator $\mathcal{L}^2_\Delta$ is formally useless, as we have already pointed out in remark 2.2 and proposition 3.3. Instead, in the context of an αDeCu or αDeCdu method, we could in principle keep adding subtimenodes, through interpolation, always improving the accuracy of the approximation with respect to the exact solution to (11). In fact, in such case, there is no bottleneck suggesting to limit the number of iterations (and, hence, the final number of subtimenodes).

Therefore, the most natural way to design adaptive schemes out of the αDeCu and αDeCdu methods is to continue performing the iterations, adding subtimenodes, until a convergence condi-

tion on the final component of $\underline{\boldsymbol{u}}^{(p)}$ (always associated to t_{n+1}) is met, for example

$$\frac{\left\|\underline{\boldsymbol{u}}^{p,(p)} - \underline{\boldsymbol{u}}^{p-1,(p-1)}\right\|}{\left\|\underline{\boldsymbol{u}}^{p,(p)}\right\|} \leq \varepsilon \tag{98}$$

with ε a desired tolerance.

9 Numerical results

In this section, we will investigate numerically the new methods showing the computational advantage with respect to the original ones. We remark that the general αDeC method of order P is obtained with P iterations and a number of subtimenodes equal to $M+1$ with $M = P-1$ in case of equispaced subtimenodes or $M = \lceil \frac{P}{2} \rceil$ in case of Gauss–Lobatto subtimenodes. The same holds for the general αDeCu and αDeCdu methods of order P, with $M+1$ being in this context the final number of subtimenodes. Let us observe that all the αDeC, αDeCu and αDeCdu methods of order 2 coincide, therefore, we will focus on the methods from order 3 on.

9.1 ODE tests

In this section, we will assess the properties of the new methods on different tests, checking their computational costs, their errors and the possibility of adaptively choosing the number of steps to reach a certain threshold. In particular, we will focus on the methods got for $\alpha = 0$ (bDeC) and $\alpha = 1$ (sDeC).

9.1.1 Linear system

The first test we study is a very simple 2×2 system of equations.

$$\begin{cases} u' = -5u + v, \\ v' = 5u - v, \end{cases} \qquad \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} = \begin{pmatrix} 0.9 \\ 0.1 \end{pmatrix}, \tag{99}$$

with exact solution $u(t) = u_0 + (1 - e^{-6t})(-5u_0 + v_0)$ and $v(t) = 1 - u(t)$. We assume a final time $T = 1$. First of all, we check that the order of accuracy of the novel methods stays equal to the one of the original methods. Moreover, being this a linear problem, we expect the bDeC, the bDeCu and the bDeCdu to be all equivalent, as they share the same stability function as shown in theorem 6.2. Furthermore, we expect the sDeCu and the sDeCdu to be equivalent due to theorem 6.3.

In figure 5, we plot the error decay for all methods with respect to the time discretization for all orders from 3 to 9. As observed before, the bDeC, bDeCu and bDeCdu methods have the same error as they coincide on linear problems and the observed order of accuracy is the expected one. The sDeC methods show a more irregular behavior and, on average, the errors with the sDeCu and sDeCdu are slightly larger than the one of sDeC for a fixed Δt.

In figure 6, we plot the error against the computational time of the methods to see which one is more efficient. For bDeC methods there is a huge advantage in using the novel methods, in fact, the errors are the same as the schemes are equivalent in this case. We can see that the Pareto front on the bottom left is composed only by the novel methods. For equispaced subtimenodes there is a larger reduction in computational cost than for Gauss–Lobatto ones, as predicted by the

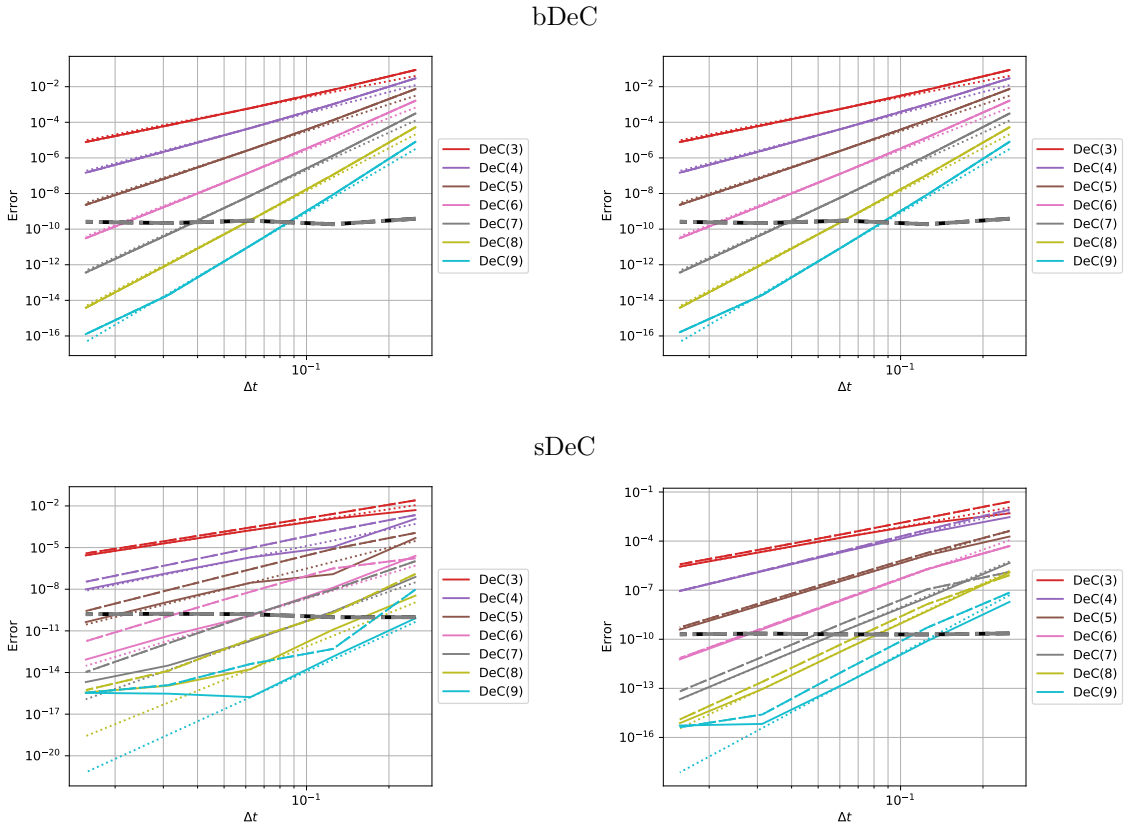

Figure 5: Linear system: Error decay for DeC with continuous line, DeCu with dashed line, DeCdu with dash-dotted line, reference order with dotted line, adaptive DeCu with dashed black line, adaptive DeCdu with dash-dotted gray line. Equispaced subtimenodes on the left and Gauss–Lobatto on the right

theory. For sDeC methods the situation is not as clear as in the bDeC case. We can systematically see a difference between sDeCu and sDeCdu, being the latter more efficient than the former. In the context of Gauss–Lobatto subtimenodes, the sDeCdu is slightly better than the original sDeC method from order 5 on in the mesh refinement.

Then, we analyze the adaptive methods, where at each time step the number of subtimenodes is chosen in order to have a relative error smaller than the tolerance $\varepsilon = 10^{-8}$ as in (98). As we observe in figure 5 the error of these methods (in black and gray) is constant and independent of the time discretization and the required computational time, see figure 6, is comparable to the very high order schemes. In particular, the related lines are close to the Pareto front for very large Δt steps, while, decreasing Δt the computational costs increase obtaining sub-optimal methods. In figure 7, we observe the average number of iterations $\pm$ half standard deviation for different adaptive methods with respect to the time discretization. As expected, the smaller the timestep, the smaller is the number of iterations necessary to reach the expected accuracy. It is also very interesting to

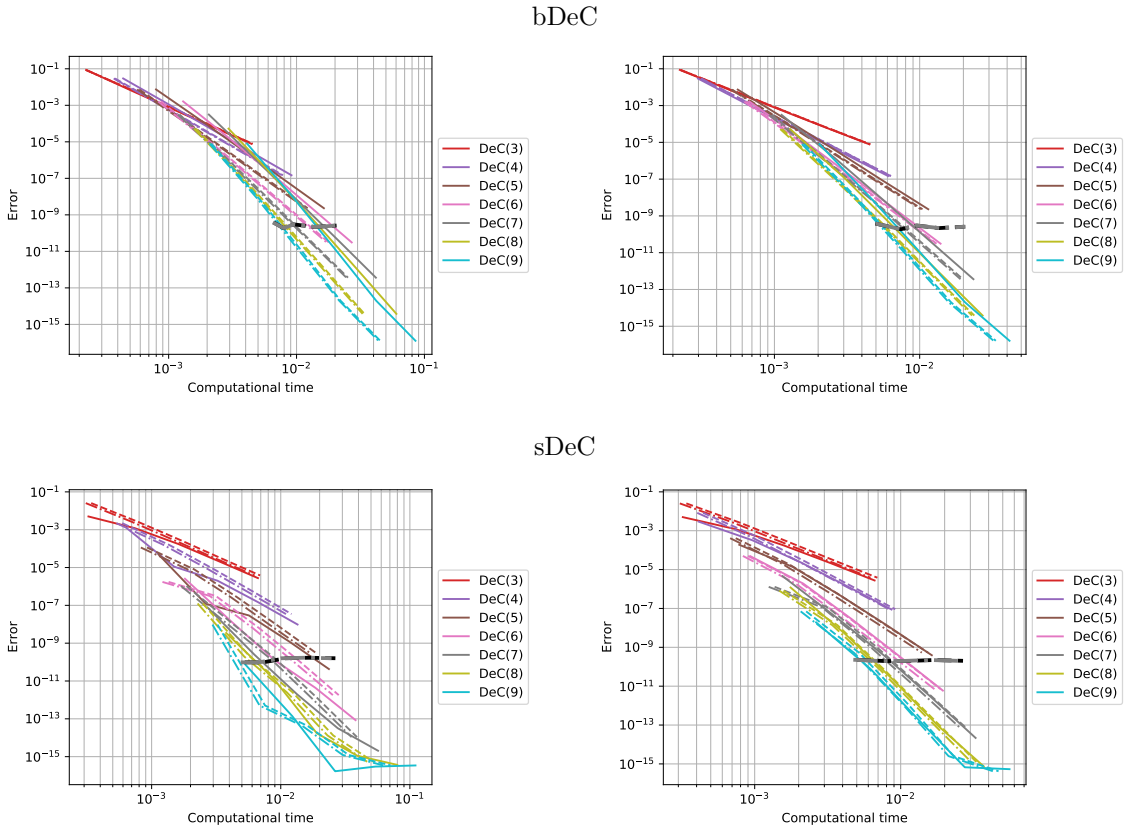

Figure 6: Linear system: Error with respect to computational time for DeC with continuous line, DeCu with dashed line, DeCdu with dash-dotted line, adaptive DeCu with dashed black line, adaptive DeCdu with dash-dotted gray line. Equispaced subtimenodes on the left and Gauss–Lobatto on the right

notice that the number of iterations, for a given Δt, is particularly stable: the standard deviation is almost negligible, meaning that fixing the tolerance and the time step is (informally) equivalent to fixing the order of accuracy. This information could be possibly used to optimize even further the adaptive methods on problems with a uniform regularity of the solution.

In figure 8, we display for different Δt, the speed up factor of the bDeCdu method with respect to the bDeC method computed as the ratio between the computational times required by the bDeCdu and the bDeC method. For equispaced subtimenodes we see that, as the order increases, the interpolation process reduces the computational time by an increasing factor which is almost 2 for order 9. For Gauss–Lobatto subtimenodes the reduction is smaller but still remarkable, close to $\frac{4}{3}$ the asymptotic limit, in particular for odd order methods.

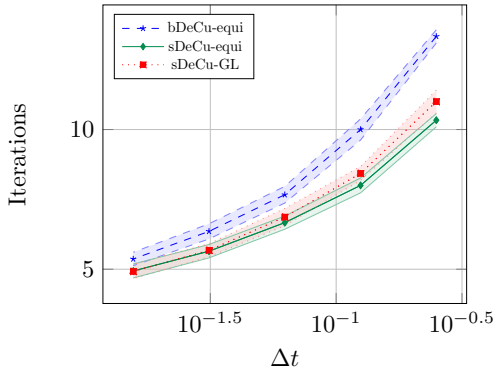

Figure 7: Linear test: Average number of iterations (± half standard deviation) of adaptive DeC for different time steps

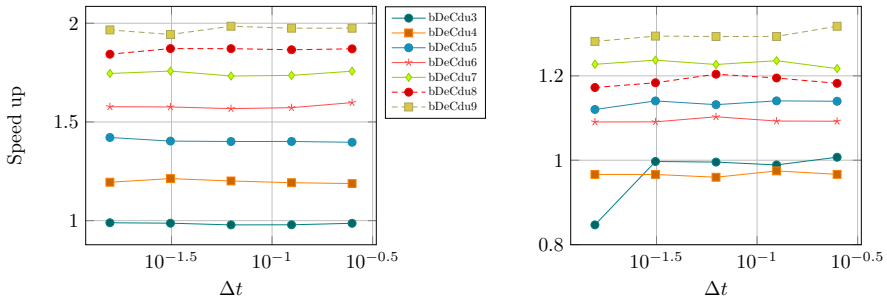

Figure 8: Linear system test: Speed up factor for the bDeCdu method computed as the computational time of the original bDeC method over the computational time of the bDeCdu method. Equispaced subtimenodes on the left and Gauss–Lobatto on the right

9.1.2 Vibrating system

Let us consider a vibrating system defined by the following ODE

$$\begin{cases} my'' + ry' + ky = F\cos(\Omega t + \varphi), \quad t \in \mathbb{R}_0^+, \\ y(0) = A, \\ y'(0) = B, \end{cases} \tag{100}$$

with $m, k, \Omega > 0$, $r, F, \psi \geq 0$. It is possible to find its exact solutions through classical analytical techniques, see for example [13], and it reads $y^{ex}(t) = y_h(t) + y_p(t)$ with $y_p(t) = Y_p \cos(\Omega t + \psi)$ particular solution to the whole equation characterized by

$$Y_p = \frac{F}{\sqrt{(-m\Omega^2 + k)^2 + \Omega^2 r^2}} \tag{101}$$

$$\psi = \varphi - \arg\left(-m\Omega^2 + k + i\Omega r\right) \tag{102}$$

and $y_h(t)$ general solution to the homogeneous equation

$$y_h(t) = \begin{cases} C_1 e^{\lambda_1 t} + C_2 e^{\lambda_2 t}, & \text{if } r > 2\sqrt{km}, \\ C_1 e^{\lambda t} + C_2 t e^{\lambda t}, & \text{if } r = 2\sqrt{km}, \\ e^{-\frac{r}{2m}t} \left(C_1 \cos(\omega t) + C_2 \sin(\omega t) \right), & \text{if } r < 2\sqrt{km}, \end{cases} \tag{103}$$

where $\omega = \frac{1}{2m}\sqrt{4km - r^2}$, λ_1 and λ_2 are the real roots of the characteristics polynomial associated to (100), which are equal to λ when $r = 2\sqrt{km}$. C_1 and C_2 are two constants that can be computed by imposing the initial conditions $y(0) = A$ and $y'(0) = B$ and solving the resulting 2×2 algebraic linear system. The mathematical steps needed to get the solution are reported in the supplementary material, but, for the sake of compactness, are omitted here. The second order scalar ODE (100) can be rewritten in a standard way as a vectorial first order ODE which can be numerically solved through the methods that we have introduced. In the test, we have set $m = 5$, $r = 2$, $k = 5$, $F = 1$, $\Omega = 2$, $\varphi = 0.1$, $A = 0.5$ and $B = 0.25$ with a final time $T = 4$.

As before, we perform a convergence study for all methods for order from 3 to 9. In figure 9, we show the error decay for all methods. Differently from the linear problems, here the methods are not equivalent. Indeed, already for bDeC, bDeCu and bDeCdu we observe differences. Nevertheless, in terms of errors, they are very similar and, also comparing equispaced and Gauss–Lobatto subtimenodes, we do not observe large deviations. On average the novel schemes are slightly less accurate for a fixed Δt, even if this is not true for all orders of accuracy. For the sDeC, there is a larger difference in the errors between sDeC and sDeCu or sDeCdu, though being the order of accuracy always correct.

These effects are visible also in figure 10. There, for bDeC with equispaced subtimenodes the advantages of using the novel methods are evident: the error is almost the same and the computational time reduces by almost half for high order schemes. For bDeC methods with Gauss–Lobatto subtimenodes the computational advantage of the novel methods is not as big as the one registered in the previous case as expected from theory, see table 6 and table 7, but still pretty visible. For what concerns the sDeC methods with equispaced subtimenodes, the performance of sDeCdu is similar to the one of sDeC until order 5, while, from order 6 on, the novel method is definitely more convenient. The sDeCu method is always less efficient than the sDeCdu one; in particular, only for very high orders it appears to be preferable to the standard method. The general trend of the sDeC methods with Gauss–Lobatto subtimenodes is that the sDeCdu and the sDeCu always perform, respectively, slightly better and slightly worse than the original sDeC.

The results of the adaptive methods for this test are qualitatively similar to the ones seen in the context of the previous test. Also in this case, the threshold for the relative error has been chosen equal to 10^{-8}. The methods produce a constant error for any Δt, as can be seen in figure 10. The analysis of the error with respect to the computational cost, in figure 10, confirms the advantage in using such methods for big values of Δt: the associated lines progressively diverge from the Pareto front as Δt decreases.

The average number of iterations performed with respect to Δt, for different adaptive methods, is represented in figure 11 and, just like in the previous case, an increase of the timestep is, comprehensibly, associated to an increase in the number of iterations to match the threshold. Also in the context of this test, one can observe a strong consistency in the number of iterations performed in the whole simulation for a fixed Δt with a non-relevant standard deviation.

Finally, in figure 12, we display the speed up factor of the new bDeCdu methods with respect to the original bDeC is plotted with respect to the time step: as expected from theory, it increases

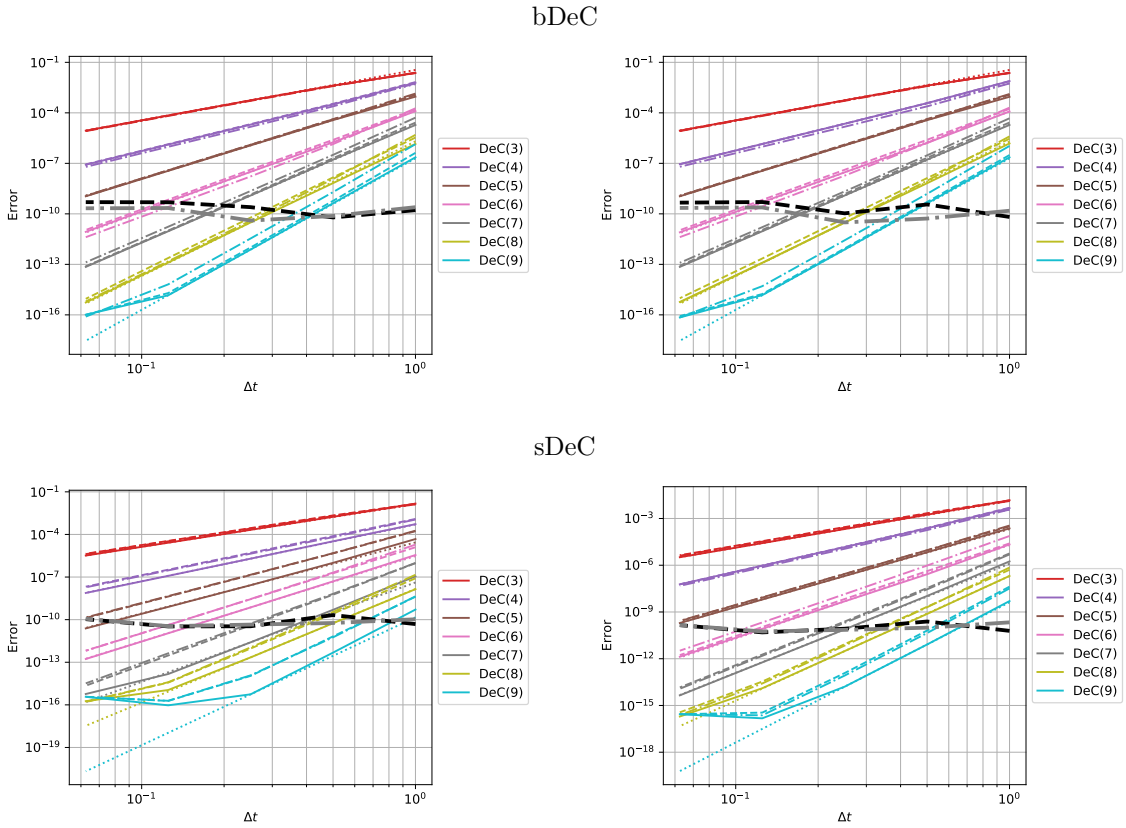

Figure 9: Vibrating system: Error decay for DeC with continuous line, DeCu with dashed line, DeCdu with dash-dotted line, reference order with dotted line, adaptive DeCu with dashed black line, adaptive DeCdu with dash-dotted gray line. Equispaced subtimenodes on the left and Gauss–Lobatto on the right

in accord with the order of accuracy with a computational advantage which is more evident for equispaced subtimenodes.

9.1.3 Three body problem

The last ODE test we study is the three body problem in two dimensions. For each body $i = 1, 2, 3$, we have to solve the following system for position $\boldsymbol{x}_i \in \mathbb{R}^2$ and velocity $\boldsymbol{v}_i \in \mathbb{R}^2$

$$\begin{cases} \frac{d}{dt}\boldsymbol{x}_i = \boldsymbol{v}_i, \\ \frac{d}{dt}\boldsymbol{v}_i = \sum_{j \neq i} \frac{m_j G}{||\boldsymbol{x}_i - \boldsymbol{x}_j||^3}(\boldsymbol{x}_i - \boldsymbol{x}_j), \end{cases} \tag{104}$$

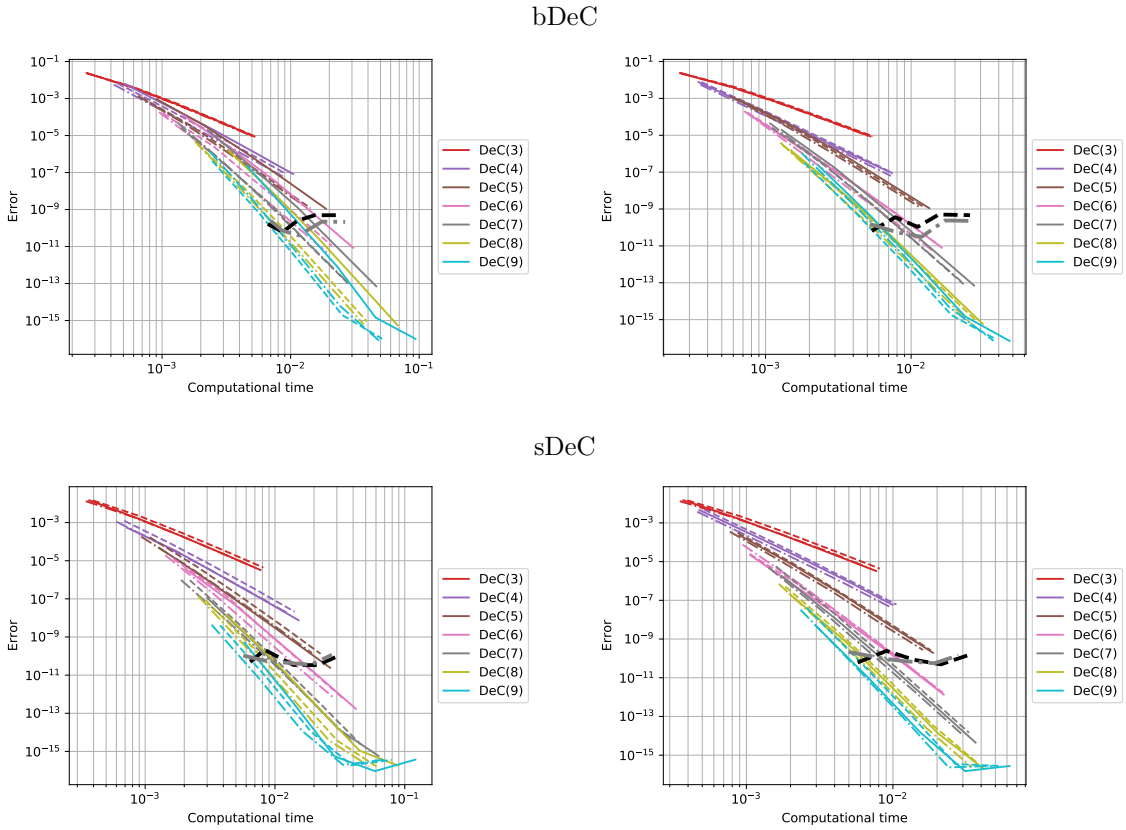

Figure 10: Vibrating system: Error with respect to computational time for DeC with continuous line, DeCu with dashed line, DeCdu with dash-dotted line, adaptive DeCu with dashed black line, adaptive DeCdu with dash-dotted gray line. Equispaced subtimenodes on the left and Gauss–Lobatto on the right

where $G = 6.67 \cdot 10^{-11}$ is the gravitational constant, $m_1 = 1.98892 \cdot 10^{30}$, $m_2 = 5.9722 \cdot 10^{24}$ and $m_3 = 6.4185 \cdot 10^{23}$ are the masses of the three bodies. The initial conditions are the following

$$\boldsymbol{x}_1(0) = (0, 0), \qquad \boldsymbol{v}_1(0) = (0, 0), \tag{105}$$

$$\boldsymbol{x}_2(0) = (149 \cdot 10^9, 0), \qquad \boldsymbol{v}_2(0) = (0, 30 \cdot 10^3), \tag{106}$$

$$\boldsymbol{x}_3(0) = (-226 \cdot 10^9, 0), \qquad \boldsymbol{v}_3(0) = (0, -24 \cdot 10^3), \tag{107}$$

which approximate the behavior of Sun, Earth and Mars.

The analytical solution for such problem is not known, hence, we will proceed with the same study we did in the previous tests, with the difference that, instead of the analytical solution, we will use a solution obtained with a high order method with a very fine grid.

In figure 13, we study the error decay of all the proposed methods. In most of the simulations the error of the original DeC methods and the one of DeCu and DeCdu are comparable. On average

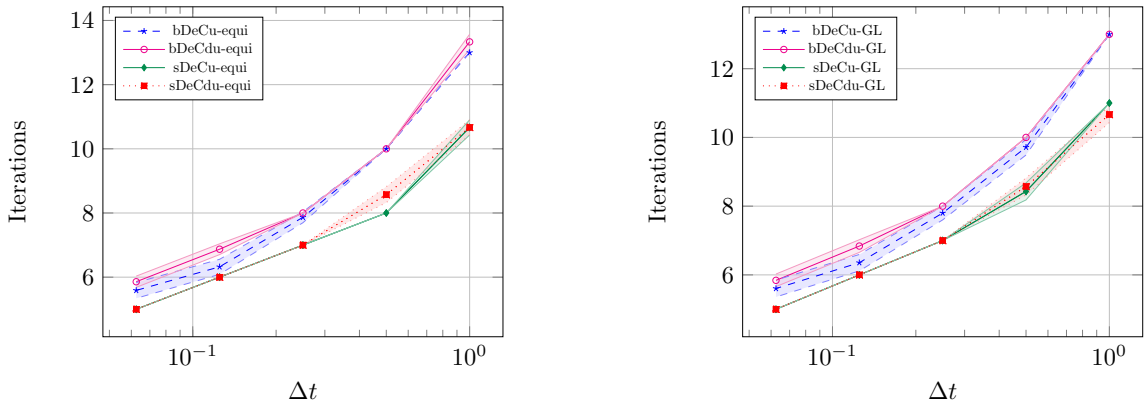

Figure 11: Vibrating system test: Average number of iterations (± half standard deviation) of adaptive DeC for different time steps. Equispaced subtimenodes on the left and Gauss–Lobatto on the right

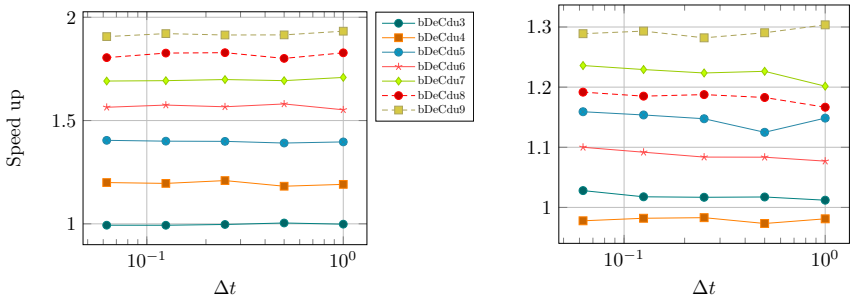

Figure 12: Vibrating system test: Speed up factor for the bDeCdu method computed as the computational time of the original bDeC method over the computational time of the bDeCdu method. Equispaced subtimenodes on the left and Gauss–Lobatto on the right

the error of the classical methods are smaller, but in few simulations the opposite holds.

Looking at performances in figure 14, we see a clear advantage in the new methods with respect to the classic one in the bDeC case, in particular in the equispaced setting. In the Gauss-Lobatto setting, the bDeCu seems the best performing method, while for sDeC methods still the classical methods are slightly better than the novel ones as their errors are smaller. It is remarkable that the adaptive methods, set with a tolerance of the relative error of $\varepsilon = 10^{-8}$ shows very stable results for all Δt. Moreover, they always reach the Pareto front for large Δt.

In figure 15, the average number of iterations ± half standard deviation for the adaptive DeC are depicted for different Δt. As for the previous tests, we see an increment in the number of iterations and increase of the order of accuracy as the timestep increases, keeping the final error constant, and a very stable number of iterations once Δt has been fixed.

Finally, in figure 16 we can see the enormous speed-up we obtain for the bDeCdu with respect to

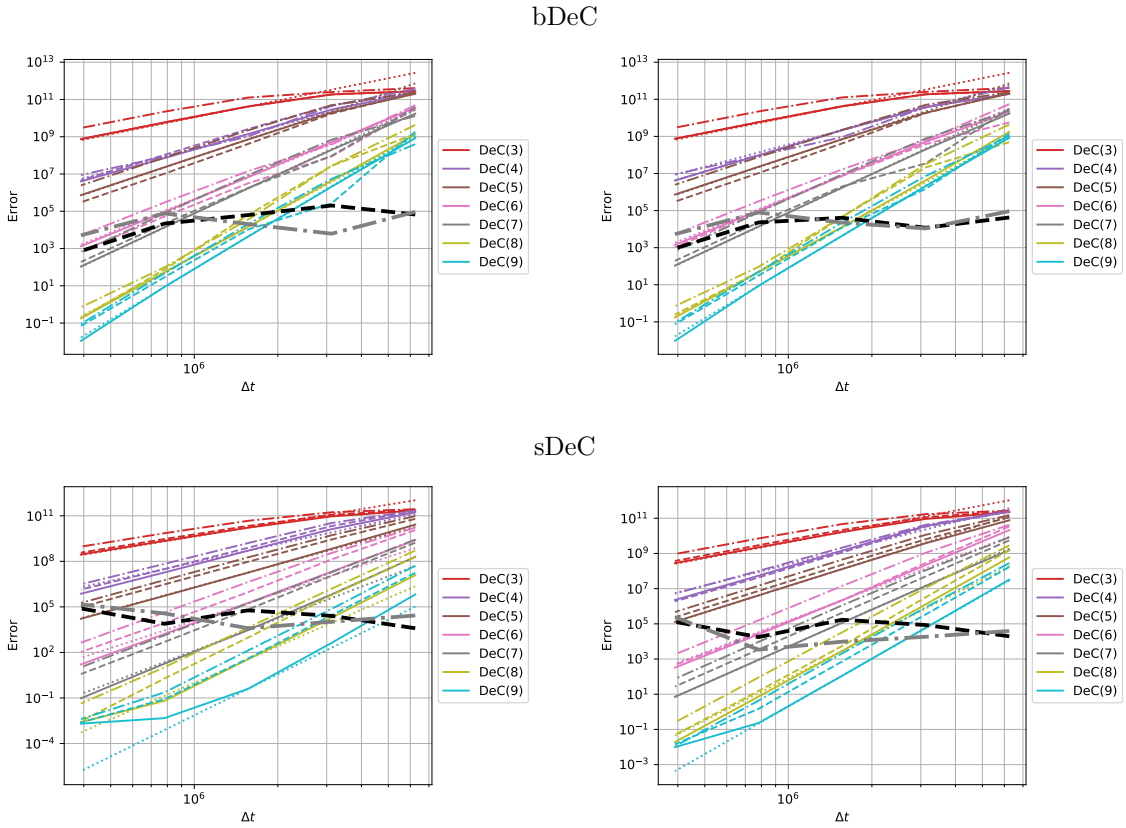

Figure 13: Three body problem: Error decay for DeC with continuous line, DeCu with dashed line, DeCdu with dash-dotted line, reference order with dotted line, adaptive DeCu with dashed black line, adaptive DeCdu with dash-dotted gray line. Equispaced subtimenodes on the left and Gauss–Lobatto on the right

the bDeC, in particular for equispaced subtimenodes, where for order 8 and 9 we reach a speed-up factor of 1.6. For the Gauss-Lobatto nodes, the advantage is less remarkable, but the novel methods are anyway faster, up to a speed-up factor of 1.2.

9.2 Hyperbolic PDE tests

In the context of the numerical results for hyperbolic PDEs, we will focus on the bDeC and the bDeCu methods with equispaced subtimenodes; furthermore, the order of the DeC is chosen accordingly to the one of the space discretization, i.e., for basis functions of degree M we will consider DeC methods of order $M + 1$. We will present two tests: a linear advection equation test in one dimension and a shallow water equations one in two dimensions. We will use two stabilizations respectively introduced in [20] and [18] and deeply studied in [44] and [45]:

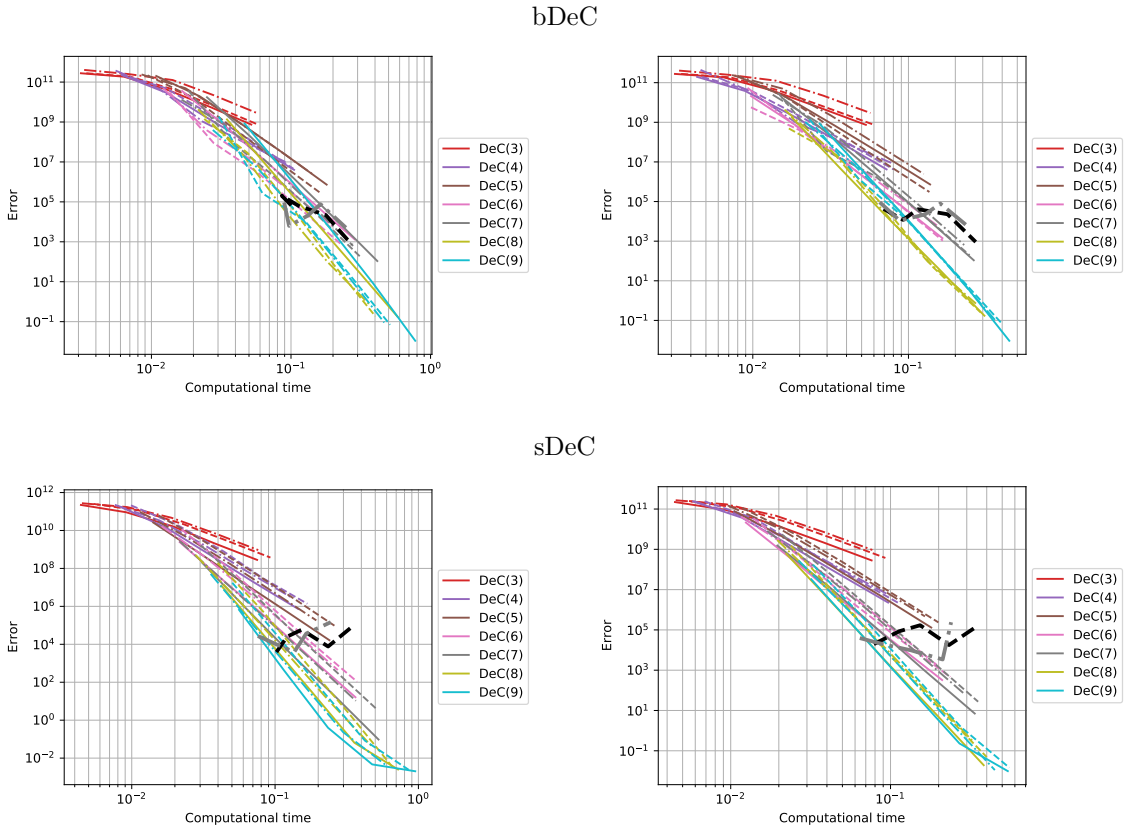

Figure 14: Three body problem: Error with respect to computational time for DeC with continuous line, DeCu with dashed line, DeCdu with dash-dotted line, adaptive DeCu with dashed black line, adaptive DeCdu with dash-dotted gray line. Equispaced subtimenodes on the left and Gauss–Lobatto on the right

- Continuous interior penalty (CIP)

$$\boldsymbol{ST}_i(\boldsymbol{u}_h) = \sum_{f \in \mathcal{F}_h} \alpha_f^{\mathrm{CIP}} \int_f [\![\nabla_{\nu_f} \varphi_i]\!] \cdot [\![\nabla_{\nu_f} \boldsymbol{u}_h]\!] \mathrm{d}\sigma(\boldsymbol{x}), \tag{108}$$

  where $\alpha_f^{\mathrm{CIP}} = \delta^{\mathrm{CIP}} \bar{\rho}_f h_f^2$, $\mathcal{F}_h$ is the set of the $(D-1)$-dimensional faces shared by two elements of $\mathcal{T}_h$, $[\![\cdot]\!]$ is the jump across the face f, ∇_{ν_f} is the partial derivative in the direction ν_f normal to the face f, $\bar{\rho}_f$ is a local reference value for the spectral radius of the normal Jacobian of the flux, h_f is the diameter of f and δ^{CIP} is a parameter that must be tuned;

- Orthogonal subscale stabilization (OSS)

$$\boldsymbol{ST}_i(\boldsymbol{u}_h) = \sum_{K \in \mathcal{T}_h} \alpha_K^{\mathrm{OSS}} \int_K \nabla_{\boldsymbol{x}} \varphi_i \left(\nabla_{\boldsymbol{x}} \boldsymbol{u}_h - \boldsymbol{w}_h \right) \mathrm{d}\boldsymbol{x}, \tag{109}$$

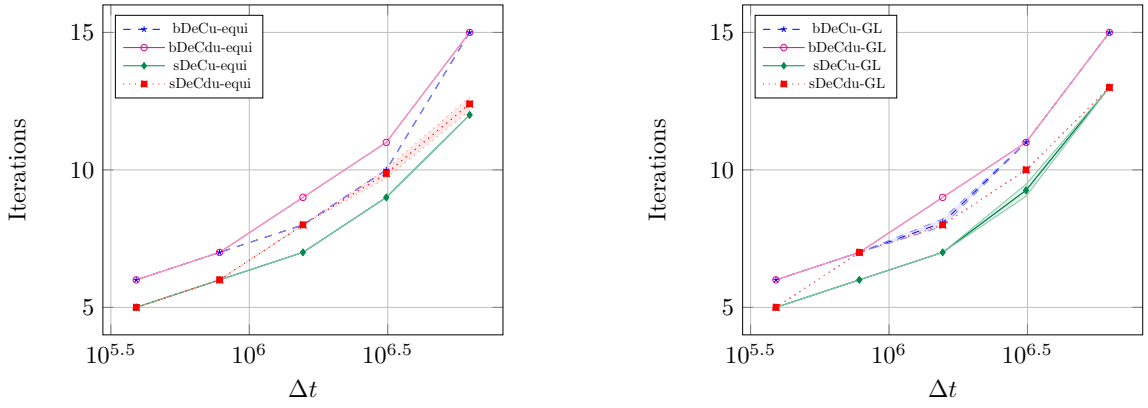

Figure 15: Three body problem: Average number of iterations (± half standard deviation) of adaptive DeC for different time steps. Equispaced subtimenodes on the left and Gauss–Lobatto on the right

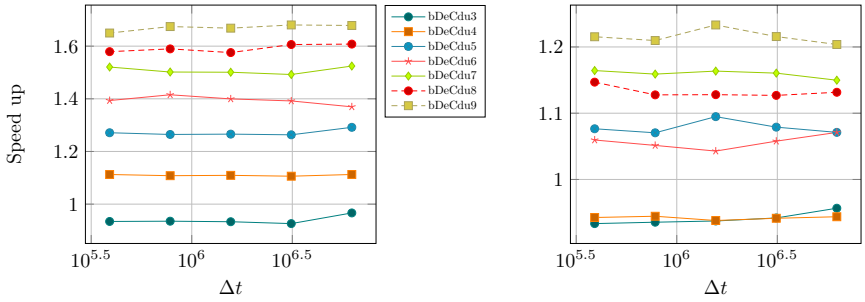

Figure 16: Three body problem: Speed up factor for the bDeCdu method computed as the computational time of the original bDeC method over the computational time of the bDeCdu method. Equispaced subtimenodes on the left and Gauss–Lobatto on the right

where $\alpha_K^{\text{OSS}} = \delta^{\text{OSS}} \bar{\rho}_K h_K$, $\boldsymbol{w}_h$ is the L^2 projection of $\nabla_{\boldsymbol{x}} \boldsymbol{u}_h$ onto $V_h^{Q \times D}$, $\bar{\rho}_K$ is a local reference value for the spectral radius of the normal Jacobian of the flux, h_K is the diameter of K and δ^{OSS} is a parameter that must be tuned.

9.2.1 Linear Advection Equation

The linear advection equation (LAE) in one dimension reads

$$u_t + a u_x = 0, \quad a \in \mathbb{R}. \tag{110}$$

We assume periodic boundary conditions. The exact solution is $u(x,t) = u_0(x - at)$ with $u_0(x)$ being the initial condition [42]. In the numerical tests, we assume $a = 1$, a domain $\Omega = (0,1)$, a final time $T = 1$ and $u_0(x) = \cos(2\pi x)$. For the spatial discretization, we considered three class of

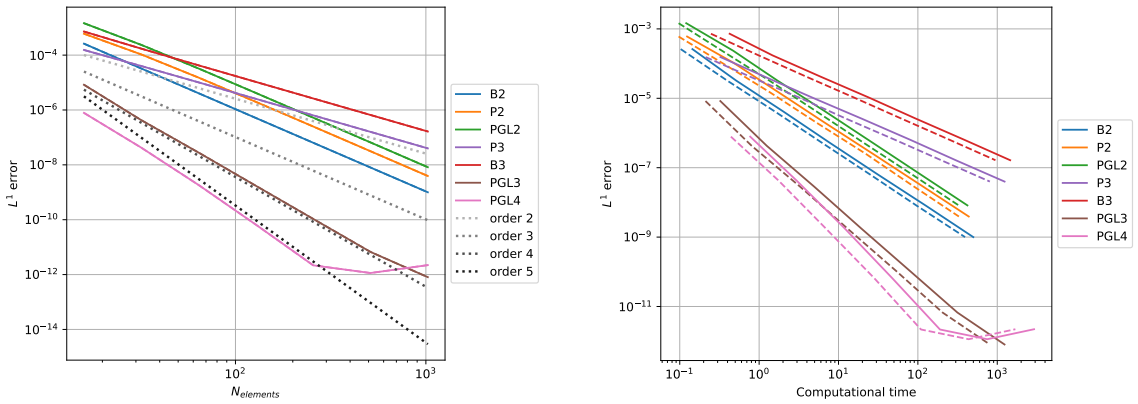

Figure 17: 1D LAE: bDeC with continuous line, bDeCu with dashed line, reference order with dotted line. Convergence analysis on the left and error with respect to computational time on the right

	B2	P2	PGL2	B3	P3*	PGL3	PGL4*
δ^{CIP}	0.016	0.00242	0.00346	0.00702	0.00702	0.000113	0.000113

Table 8: Coefficients δ^{CIP} used for LAE in one dimension. *The coefficients adopted for P3 and PGL4 are not provided in [44].

polynomial basis functions with degree n: Bn, the Bernstein polynomials [2, 5]; Pn, the Lagrange polynomials associated to equispaced nodes; PGLn, the Lagrange polynomials associated to the Gauss–Lobatto nodes [44]. For Bn and Pn, we used the bDeC version for hyperbolic PDEs (97) introduced by Abgrall; for PGLn, we adopted the bDeC formulation for ODEs (20), as, in this case, the choice of the quadrature formula associated to the Lagrangian nodes leads to a high order diagonal mass matrix. For all of them, we used the CIP stabilization (108) with the coefficients δ^{CIP} reported in [44] to minimize the dispersion error, even if, differently from there, we assumed here a constant CFL = 0.1 and, clearly, the modified version of the bDeC. The coefficients are reported in table 8. In particular, since the coefficients for P3 and PGL4 were not provided, we used for the former the same coefficient as for B3, while, for the latter the same coefficient as for PGL3.

The results of the convergence analysis and of the computational cost analysis are displayed in figure 17. For a fixed number of elements, the errors of the bDeC and of the bDeCu methods are identical and it is impossible to distinguish the lines associated to the two methods, as can bee seen from the plot on the left. This leads to a remarkable computational advantage of the novel method with respect to the original bDeC, visible in the plot on the right, where the error against the computational time is depicted. The formal order of accuracy is recovered in all the cases but for B3 and P3 for which we get only second order for both bDeC and bDeCu.

Remark 9.1 (Issues with the DeC for PDEs)**.** *The loss of accuracy for bDeC(4) and B3 elements has been registered in other works, e.g. [44, 45, 55]. Actually, even in the original paper [5], in which the method has been introduced, the author underlines the necessity to perform more iterations*

than what expected from theory for discretizations of order greater than or equal to 4 to recover the formal order of accuracy. According to authors' opinion the problem deserves a particular attention, for this reason, the results related to B3 and P3 have not been omitted. The pathology seems to have effect only in the context of unsteady tests and it is maybe due to a high order weak instability. The phenomenon is currently under investigation; more details can be found in the supplementary material. However, we remark that the problem does not occur for elements which allow a proper mass lumping like PGL (or Cubature in 2D), as can be seen from the numerical results.

The speed up factor of the novel bDeCu with respect to the original method in the one-dimensional tests is reported in figure 18. The speed up factors are higher than the ODE ones, because in the implementation of the DeC for PDEs the major cost is not given by the flux evaluation of previously computed stages, but the evolution of the new stages. This slightly changes the expected and the observed speed up, providing even larger computational advantages.

9.2.2 Shallow Water Equations

The Shallow Water (SW) equations in two dimensions are a system of hyperbolic PDEs, in the form (81), characterized by

$$\boldsymbol{u} = \begin{pmatrix} H \\ H\boldsymbol{v} \end{pmatrix}, \quad \boldsymbol{F}(\boldsymbol{u}) = \begin{pmatrix} H\boldsymbol{v} \\ H\boldsymbol{v} \otimes \boldsymbol{v} + g\frac{H^2}{2}\mathbb{I} \end{pmatrix}, \quad \boldsymbol{S}(\boldsymbol{x}, \boldsymbol{u}) = 0, \tag{111}$$

where H is the water height, $\boldsymbol{v} = (v_1, v_2)^T \in \mathbb{R}^2$ is the vertically averaged speed of the flow, g is the gravitational constant, $\mathbb{I} \in \mathbb{R}^{D\times D}$ is the identity matrix and $D = 2$ is the number of physical dimensions. We consider the computational domain $\Omega = (0, 3) \times (0, 3)$. The test consists of a $C^6(\Omega)$ compactly supported unsteady vortex from the collection presented in [54] given by

$$\boldsymbol{u} = \boldsymbol{u}^\infty + \begin{cases} \boldsymbol{u}_{r_0}(r), & \text{if } r = ||\boldsymbol{x} - \boldsymbol{x}_m(t)||_2 < r_0, \\ 0, & \text{else,} \end{cases} \tag{112}$$

where $\boldsymbol{u}^\infty = (1, 1, 1)^T$, $\boldsymbol{x}_m(t) = \boldsymbol{x}_c + t \cdot (1, 1)^T$ and

$$\boldsymbol{u}_{r_0}(r) = \begin{pmatrix} \frac{1}{g}\left(\frac{\Gamma}{\omega}\right)^2 (\lambda(\omega r) - \lambda(\pi)) \\ \Gamma\left(1 + \cos(\omega r)\right)^2 (x_2 - x_{m,2}) \\ -\Gamma\left(1 + \cos(\omega r)\right)^2 (x_1 - x_{m,1}) \end{pmatrix}, \ \Gamma = \frac{12\pi\sqrt{g\Delta H}}{r_0\sqrt{315\pi^2 - 2048}} \tag{113}$$

with $\omega = \frac{\pi}{r_0}$ and the function λ defined by

$$\begin{aligned} \lambda(s) = {} & \frac{20}{3}\cos(s) + \frac{27}{16}\cos(s)^2 + \frac{4}{9}\cos(s)^3 + \frac{1}{16}\cos(s)^4 + \frac{20}{3}s\sin(s) \\ & + \frac{35}{16}s^2 + \frac{27}{8}s\cos(s)\sin(s) + \frac{4}{3}s\cos(s)^2\sin(s) + \frac{1}{4}s\cos(s)^3\sin(s). \end{aligned} \tag{114}$$

The other parameters are $g = 9.81$, $r_0 = 1$, $\Delta H = 0.1$, $\boldsymbol{x}_c = (1, 1)^T$ with a final time $T = 1$ and Dirichlet boundary conditions.

For the spatial discretization, we considered two basis functions: Bn, the Bernstein polynomials; Cn, the Cubature elements introduced in [19]. The Cubature elements are a generalization

of the PGLn elements in two dimensions. They select Lagrangian nodes which coincide with the quadrature ones, such that the order of accuracy is the correct one and the mass matrix is diagonal. For Cn elements we used the bDeC (20) for ODEs, instead, for Bn we considered the PDE formulation (97). For the Bernstein polynomials we adopted a CIP stabilization (108), for the Cubature elements we adopted the OSS stabilization (109).

The tests with B2 have been run with CFL $= 0.1$ and $\delta^{\text{CIP}} = 0.04$; for C2 elements we have set CFL $= 0.12$ and $\delta^{\text{OSS}} = 0.07$, the optimal coefficients minimizing the dispersion error of the original bDeC according to the linear analysis performed in [45]; for C3 we adopted CFL $= 0.015$ and $\delta^{\text{OSS}} = 0.2$.

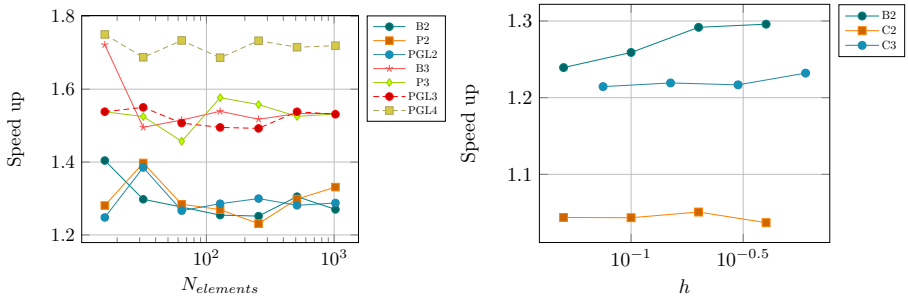

Figure 18: Speed up in the hyperbolic tests of bDeCu with respect to bDeC. 1D LAE on the left and 2D SW on the right

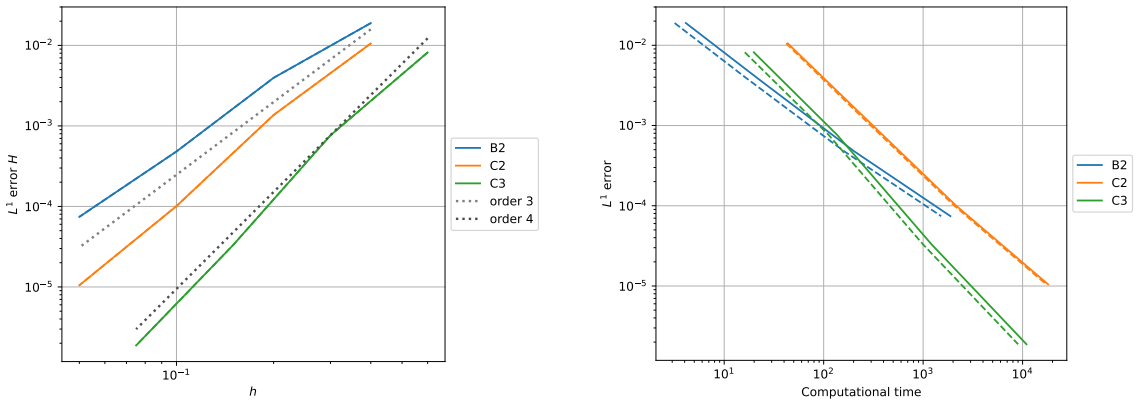

Figure 19: 2D SW: bDeC with continuous line, bDeCu with dashed line, reference order with dotted line. Convergence analysis on the left and error with respect to computational time on the right

The results of the convergence analysis and of the computational cost analysis are displayed in figure 19. From the plot on the left, we can see that, also in this case, the errors produced by the novel and the original bDeC method are very close to the point that the related lines coincide. This is surprising as, differently from before, in this case we are dealing with a nonlinear system.

The resulting computational advantage can be seen in the plot on the right. The formal order of accuracy is recovered in all the cases and the speed up factor, plotted in 18, proves the convenience in using the novel bDeCu formulation instead of the original bDeC.

We conclude this section with one last remark: the computational advantage registered with B2 is much higher with respect to the one of C2 and even of C3. This is due to the fact that we have run the simulations with different codes: the results obtained with B2 are referred to an implementation in Fortran, while, for C2 and C3 we used Parasol, a Python implementation developed by Sixtine Michel for [45] and kindly provided to us. A more careful implementation would increase further the speed up factors associated to such elements.

Remark 9.2 (On the expected computational gain)**.** *Looking at table 6, one can see that the number of stages of bDeC3 and bDeCu3 is identical. Nevertheless, as observed in remark 5.1, the number of stages does not strictly correspond to the computational time. If we do not consider the stages computed via interpolation, we get the theoretical speed up factor* $\frac{5}{4} = 1.25$*, which is what we obtained in the numerical test for B2.*

10 Conclusions and further developments

In this work, we have investigated the analytical and numerical aspects of two novel families of efficient explicit DeC methods. The novel methods are obtained by the introduction of interpolation processes between the iterations, which increase the number of nodes in order to match the accuracy of the approximation associated to the current iteration. In particular, we proved that for some of the novel methods the stability region coincides with the one of the original methods. The novel methods have been tested on classical benchmarks in the ODE context revealing, in most of the cases, a remarkable computational advantage with respect to the original ones. Furthermore, the interpolation strategies have been used to design adaptive schemes. Finally, we successfully proved the good performances of the novel methods in the context of the numerical solution of hyperbolic PDEs with continuous space discretizations. Overall, we believe that this work can alleviate the computational costs of DeC methods, which have been recently used also for complicated problems, but also for other similar schemes that suffer of the same iteration issue. Investigations of other numerical frameworks are planned and, in particular, we are working on applications to hyperbolic PDEs (with FV and ADER schemes), in which also the order of the space reconstruction is increased gradually iteration by iteration. We hope to spread broadly this technique in the community in order to save computational time and energy for computing various ODE and PDE problems, as only little effort is required to embed the novel modification in an existing DeC code.

Supplementary information

The interested reader is referred to the supplementary material for all the proofs omitted in this document for the sake of compactness.

Acknowledgments

L. Micalizzi has been funded by the SNF grant 200020_204917 "Structure preserving and fast methods for hyperbolic systems of conservation laws". D. Torlo has been funded by a SISSA Mathematical Fellowship. The authors warmly acknowledge Sixtine Michel for providing the code Parasol.

Conflict of interest

On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

[1] Rémi Abgrall. Residual distribution schemes: current status and future trends. *Computers & Fluids*, 35(7):641–669, 2006.

[2] Rémi Abgrall, Paola Bacigaluppi, and Svetlana Tokareva. High-order residual distribution scheme for the time-dependent Euler equations of fluid dynamics. *Computers & Mathematics with Applications*, 78(2):274–297, 2019.

[3] Rémi Abgrall, Elise Le Mélédo, Philipp Öffner, and Davide Torlo. Relaxation deferred correction methods and their applications to residual distribution schemes. *accepted in SMAI-JCM, preprint arXiv:2106.05005*, 2022.

[4] Rémi Abgrall and Davide Torlo. High order asymptotic preserving deferred correction implicit-explicit schemes for kinetic models. *SIAM Journal on Scientific Computing*, 42(3):B816–B845, 2020.

[5] Rémi Abgrall. High order schemes for hyperbolic problems using globally continuous approximation and avoiding mass matrices. *J. Sci. Comput.*, 73(2-3):461–494, 2017.

[6] Rémi Abgrall and Mario Ricchiuto. *High order methods for CFD - Encyclopedia of Computational Mechanics Second Edition*. 01 2017.

[7] Dinshaw S Balsara, Tobias Rumpf, Michael Dumbser, and Claus-Dieter Munz. Efficient, high accuracy ADER-WENO schemes for hydrodynamics and divergence-free magnetohydrodynamics. *Journal of Computational Physics*, 228(7):2480–2516, 2009.

[8] Sebastiano Boscarino and Jing-Mei Qiu. Error estimates of the integral deferred correction method for stiff problems. *ESAIM: Mathematical Modelling and Numerical Analysis*, 50(4):1137–1166, 2016.

[9] Sebastiano Boscarino, Jing-Mei Qiu, and Giovanni Russo. Implicit-explicit integral deferred correction methods for stiff problems. *SIAM Journal on Scientific Computing*, 40(2):A787–A816, 2018.

[10] Walter Boscheri, Michael Dumbser, and Elena Gaburro. Continuous finite element subgrid basis functions for Discontinuous Galerkin schemes on unstructured polygonal Voronoi meshes. *arXiv preprint arXiv:2205.14673*, 2022.

[11] Anne Bourlioux, Anita T Layton, and Michael L Minion. High-order multi-implicit spectral deferred correction methods for problems of reactive flow. *Journal of Computational Physics*, 189(2):651–675, 2003.

[12] John Charles Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, Auckland, 2016.

[13] Federico Cheli and Giorgio Diana. *Advanced dynamics of mechanical systems*, volume 2020. Springer, Cham, 2015.

[14] Andrew Christlieb, Benjamin Ong, and Jing-Mei Qiu. Comments on high-order integrators embedded within integral deferred correction methods. *Communications in Applied Mathematics and Computational Science*, 4(1):27–56, 2009.

[15] Andrew Christlieb, Benjamin Ong, and Jing-Mei Qiu. Integral deferred correction methods constructed with high order Runge–Kutta integrators. *Mathematics of Computation*, 79(270):761–783, 2010.

[16] M. Ciallella, L. Micalizzi, P. Öffner, and D. Torlo. An arbitrary high order and positivity preserving method for the shallow water equations. *Computers & Fluids*, page 105630, 2022.

[17] Bernardo Cockburn, George E Karniadakis, and Chi-Wang Shu. The development of discontinuous Galerkin methods. In *Discontinuous Galerkin Methods*, pages 3–50. Springer, Berlin, 2000.

[18] Ramon Codina and Jordi Blasco. A finite element formulation for the stokes problem allowing equal velocity-pressure interpolation. *Computer Methods in Applied Mechanics and Engineering*, 143(3-4):373–391, 1997.

[19] Gary Cohen, Patrick Joly, Jean E. Roberts, and Nathalie Tordjman. Higher order triangular finite elements with mass lumping for the wave equation. *SIAM Journal on Numerical Analysis*, 38(6):2047–2078, 2001.

[20] Jim Douglas and Todd Dupont. Interior penalty procedures for elliptic and parabolic galerkin methods. In *Computing methods in applied sciences*, pages 207–216. Springer, New York, 1976.

[21] Alok Dutt, Leslie Greengard, and Vladimir Rokhlin. Spectral deferred correction methods for ordinary differential equations. *BIT*, 40(2):241–266, 2000.

[22] Alexandre Ern and Jean-Luc Guermond. *Theory and practice of finite elements*, volume 159. Springer, New York, 2004.

[23] Robert Eymard, Thierry Gallouët, and Raphaèle Herbin. Finite volume methods. *Handbook of numerical analysis*, 7:713–1018, 2000.

[24] Leslie Fox and ET Goodwin. Some new methods for the numerical integration of ordinary differential equations. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 45, pages 373–388. Cambridge University Press, 1949.

[25] Reinhard Frank. The method of iterated defect-correction and its application to two-point boundary value problems. *Numerische Mathematik*, 25(4):409–419, 1975.

[26] Reinhard Frank and Christoph W Ueberhuber. Iterated defect correction for the efficient solution of stiff systems of ordinary differential equations. *BIT Numerical Mathematics*, 17(2):146–159, 1977.

[27] Elena Gaburro. A unified framework for the solution of hyperbolic PDE systems using high order direct Arbitrary-Lagrangian–Eulerian schemes on moving unstructured meshes with topology change. *Archives of Computational Methods in Engineering*, 28(3):1249–1321, 2021.

[28] Walter Gautschi. *Numerical analysis*. Springer Science & Business Media, Indiana, 2011.

[29] Edwige Godlewski and Pierre-Arnaud Raviart. *Numerical approximation of hyperbolic systems of conservation laws*, volume 118. Springer Science & Business Media, New York, 2013.

[30] David Gottlieb and Jan S Hesthaven. Spectral methods for hyperbolic problems. *Journal of Computational and Applied Mathematics*, 128(1-2):83–131, 2001.

[31] Ernst Hairer, S.P. Nørsett, and G Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer-Verlag, Berlin, 1987.

[32] Maria Han Veiga, Philipp Öffner, and Davide Torlo. Dec and Ader: similarities, differences and a unified framework. *Journal of Scientific Computing*, 87(1):1–35, 2021.

[33] Anders C Hansen and John Strain. On the order of deferred correction. *Applied Numerical Mathematics*, 61(8):961–973, 2011.

[34] Jan S Hesthaven and Tim Warburton. *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*. Springer Science & Business Media, New York, 2007.

[35] Jingfang Huang, Jun Jia, and Michael Minion. Accelerating the convergence of spectral deferred correction methods. *Journal of Computational Physics*, 214(2):633–656, 2006.

[36] Sébastien Jund and Stéphanie Salmon. Arbitrary High-Order Finite Element Schemes and High-Order Mass Lumping. *International Journal of Applied Mathematics & Computer Science*, 17(3):375–393, 2007.

[37] David Ketcheson and Umair bin Waheed. A comparison of high-order explicit Runge–Kutta, extrapolation, and deferred correction methods in serial and parallel. *Communications in Applied Mathematics and Computational Science*, 9(2):175–200, 2014.

[38] Wendy Kress and Bertil Gustafsson. Deferred correction methods for initial boundary value problems. *Journal of Scientific Computing*, 17(1):241–251, 2002.

[39] Anita T Layton and Michael L Minion. Conservative multi-implicit spectral deferred correction methods for reacting gas dynamics. *Journal of Computational Physics*, 194(2):697–715, 2004.

[40] Anita T Layton and Michael L Minion. Implications of the choice of quadrature nodes for picard integral deferred corrections methods for ordinary differential equations. *BIT Numerical Mathematics*, 45(2):341–373, 2005.

[41] Randall J LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. SIAM, Philadelphia, 2007.

[42] Randall J LeVeque et al. *Finite volume methods for hyperbolic problems*, volume 31. Cambridge university press, Cambridge, 2002.

[43] Yuan Liu, Chi-Wang Shu, and Mengping Zhang. Strong stability preserving property of the deferred correction time discretization. *Journal of Computational Mathematics*, 26(5):633–656, 2008.

[44] Sixtine Michel, Davide Torlo, Mario Ricchiuto, and Rémi Abgrall. Spectral analysis of continuous FEM for hyperbolic PDEs: influence of approximation, stabilization, and time-stepping. *Journal of Scientific Computing*, 89(2):1–41, 2021.

[45] Sixtine Michel, Davide Torlo, Mario Ricchiuto, and Rémi Abgrall. Spectral analysis of high order continuous FEM for hyperbolic PDEs on triangular meshes: influence of approximation, stabilization, and time-stepping. *arXiv preprint arXiv:2206.06150*, 2022.

[46] Michael Minion. A hybrid parareal spectral deferred corrections method. *Communications in Applied Mathematics and Computational Science*, 5(2):265–301, 2011.

[47] Michael L Minion. Semi-implicit spectral deferred correction methods for ordinary differential equations. *Communications in Mathematical Sciences*, 1(3):471–500, 2003.

[48] Michael L Minion. Semi-implicit projection methods for incompressible flow based on spectral deferred corrections. *Applied numerical mathematics*, 48(3-4):369–387, 2004.

[49] Philipp Öffner and Davide Torlo. Arbitrary high-order, conservative and positivity preserving Patankar-type deferred correction schemes. *Applied Numerical Mathematics*, 153:15–34, 2020.

[50] Philipp Öffner and Davide Torlo. Arbitrary high-order, conservative and positivity preserving Patankar-type deferred correction schemes. *Appl. Numer. Math.*, 153:15–34, 2020.

[51] Richard Pasquetti and Francesca Rapetti. Cubature points based triangular spectral elements: An accuracy study. *Journal of Mathematical Study*, 51(1):15–25, 2018.

[52] Victor Pereyra. Iterated deferred corrections for nonlinear boundary value problems. *Numerische Mathematik*, 11(2):111–125, 1968.

[53] Mario Ricchiuto and Remi Abgrall. Explicit Runge–Kutta residual distribution schemes for time dependent problems: second order case. *Journal of Computational Physics*, 229(16):5653–5691, 2010.

[54] Mario Ricchiuto and Davide Torlo. Analytical travelling vortex solutions of hyperbolic equations for validating very high order schemes. 2021.

[55] Abgrall Rémi, Bacigaluppi Paola, and Tokareva Svetlana. High-order residual distribution scheme for the time-dependent euler equations of fluid dynamics. *Computers & Mathematics with Applications*, 78(2):274–297, 2019. Proceedings of the Eight International Conference on Numerical Methods for Multi-Material Fluid Flows (MULTIMAT 2017).

[56] Robert D Skeel. A theoretical framework for proving accuracy results for deferred corrections. *SIAM Journal on Numerical Analysis*, 19(1):171–196, 1982.

[57] Robert Speck, Daniel Ruprecht, Matthew Emmett, Matthias Bolten, and Rolf Krause. A space-time parallel solver for the three-dimensional heat equation. *Parallel Computing: Accelerating Computational Science and Engineering (CSE)*, 25:263–272, 2014.

[58] Robert Speck, Daniel Ruprecht, Matthew Emmett, Michael Minion, Matthias Bolten, and Rolf Krause. A multi-level spectral deferred correction method. *BIT Numerical Mathematics*, 55(3):843–867, 2015.

[59] Hans J Stetter. The defect correction principle and discretization methods. *Numerische Mathematik*, 29(4):425–443, 1978.

[60] Davide Torlo. *Hyperbolic problems: high order methods and model order reduction*. PhD thesis, University Zurich, 2020.

[61] Henk Kaarle Versteeg and Weeratunge Malalasekera. *An introduction to computational fluid dynamics: the finite volume method*. Pearson education, Harlow, 2007.

[62] Gerhard Wanner and Ernst Hairer. *Solving ordinary differential equations II: Stiff and Differential-Algebraic Problems*, volume 375. Springer Berlin Heidelberg, Berlin, 1996.