A Generic Framework for Time-Stepping PDEs: general linear methods, object-orientated implementation and application to fluid problems

Peter E.J. Vos[a,b], Sehun Chun[a], Alessandro Bolis[a],
Claes Eskilsson[c], Robert M. Kirby[d] and Spencer J. Sherwin[a,*]

[a] *Dept. of Aeronautics, Imperial College London, South Kensington Campus, London, SW7 2AZ, UK*
[b] *Flemish Institute for Technological Research (Vito), Boeretang 200, BE-2400 Mol, Belgium*
[c] *Dept. of Shipping and Marine Technology, Chalmers Univ. of Tech., SE-412 96 Gothenburg, Sweden*
[d] *School of Computing, Univ. of Utah, 50 S. Central Campus Drive, Salt Lake City, UT 84112, USA*

October 24, 2010

Abstract

Time-stepping algorithms and their implementations are a critical component within the solution of time-dependent partial differential equations (PDEs). In this paper we present a generic framework – both in terms of algorithms and implementations – that allows an almost seamlessly switch between various explicit, implicit and implicit-explicit (IMEX) time-stepping methods. We put particular emphasis on how to incorporate time-dependent boundary conditions, an issue that go beyond classical ODE theory but which play an important role in the time-stepping of the PDEs arising in computational fluid dynamics. Our algorithm is based upon J.C. Butcher's unifying concept of *General Linear Methods* that we have extended to accommodate the family IMEX schemes that are often used in engineering practice. In the paper we discuss design considerations and presents an object-orientated implementation. Finally we illustrate the use of the framework by applications to model problem as well as to more complex fluid problems.

1 Introduction

In the development of simulation software into which numerical approximation strategies for solving time-dependent partial differential equations (PDEs) are utilised, the time-stepping method and its implementation typically receive a subordinate role to the modelling and spatial discretisation choices. There exist a myriad of reasons why this partitioning of effort exists and is justified. In part, the Method of Lines (MoL), which is commonly employed to help simplify the discretisation process, focuses one's attention on distilling the partial differential equations down to a collection of coupled ordinary differential equations

*Corresponding author. Email: s.sherwin@imperial.ac.uk

(ODEs) to which classic time-stepping methods can be applied (see *e.g.* [25] for discussions and examples of the MoL approach). Tremendous effort is invested into the distillation process of modelling and spatial discretisation, and often is the final ODE integration stage viewed as a straightforward process requiring little concentrated focus.

When ready to start time-stepping the semi-discretized PDE the multi-stage/multi-step divide is encountered – whether to use multi-step methods like Adams-Bashforth and Adams-Moulton, which typically require more memory but have an economy of floating-point operations, or to use multi-stage methods like Runge-Kutta (RK), which typically have larger stability regions and require less memory. Whichever selection is made might require further reworking of the simulation software to accommodate either the memory needs or evaluation needs of the family of schemes selected. This serves further to discourage fully exploiting all the advances that have been made in the numerical solution of ODEs and discourages doing verification studies in which the interplay between spatial and temporal discretisation errors (beyond just leading order-of-accuracy statements) are quantified.

The goal of this effort was to develop a generic framework, both in terms of algorithms and software implementations, which allows an almost seamlessly switch between various explicit and implicit time-stepping methods. The first challenge we encountered was the question of how to span the multi-stage/multi-step divide. By basing our algorithms on J.C. Butcher's unifying *General Linear Methods* (GLM), as originally introduced by [6], we are able to accommodate a wide range of the time-stepping schemes used in engineering practice, which not only encourages the judicious use of the plethora of different methods that exist, but also facilitates time-discretisation verification studies.

General linear methods, see e.g. [3, 4, 16, 5, 17] and the numerous references therein, unify the analysis of ODEs with respect to consistency, stability and convergence. In addition to GLM covering many of the classical methods, it also includes methods such as the 'two-step Runge-Kutta methods' [18], 'almost Runge-Kutta methods' [9, 23], 'diagonally implicit multistage integration methods' [7, 8] and 'methods with inherent Runge-Kutta stability' [30]. Examples of existing GLM based ODE codes are due to [28, 29, 10, 15, 30]. These codes are highly specialised implementations of a single sub-class of GLMs. We note that our objective is quite different as we want to use the unifying property of the GLMs as foundation for building a generic time-stepping framework.

However, the ODE concept of GLM currently does not encompass the family of implicit-explicit (IMEX) schemes that are often used to time-integrate PDEs, see *e.g.* [2, 1, 19]. In order to treat these schemes in a similar way, we have shown that through a small modification, these schemes as well can be formulated as a general linear method.

Although the MoL approach in principle abstracts away the spatial discretization part of the PDE, there are some specific issues arising during this procedure that have a decisive influence on the design of a generic PDE time-stepping framework. In particular, the question how to deal with time-dependent boundary conditions in a generic and computationally efficient manner forms the second major challenge of this work.

Finally, a remark regarding terminology when dealing with time-stepping schemes that are formally explicit from an ODE point of view. Spatially discretising a PDE using a Galerkin approach generally leads to a ODE system

which involves the "inversion" of a global system regardless of the fact that we are using an explicit time-stepping scheme. This situation can be referred to as an *indirect* explicit method in contrast to the *direct* explicit method resulting from, for example, a standard finite difference discretization.

1.1 Design Considerations

Based upon the aforementioned motivations, we set the following as the objectives of our time-stepping framework:

- It should facilitate both explicit/implicit time-stepping and multi-step and multi-stage schemes, as well as allowing implementation of more elaborate partitioning schemes, *e.g.* IMEX-RK. However, the framework is restricted to only incorporate implicit schemes in which the stage values can be computed in a decoupled fashion. This includes all implicit multi-step schemes and the diagonally implicit multi-stage schemes such as the Diagonally Implicit Runge-Kutta (DIRK) schemes. Fully implicit multi-stage methods, which are rarely adopted in engineering practice, do not fit into the presented framework.
- It should be designed anticipating that the MoL has been used on a PDE to yield a system of coupled ODEs. Therefore the framework should work for both static and time-dependent, as well as for both weakly and strongly enforced boundary conditions.
- It should work independent of the spatial discretisation choice (*i.e.* it should work with continuous Galerkin and discontinuous Galerkin methods, as well as with finite difference and finite volume methods).
- It should provide an efficient solution for time-stepping PDEs, *i.e.* the computational cost should be comparable to scheme-specific implementations.

1.2 Outline

In this paper, we document the objectives of our framework, provide a brief overview of general linear methods, and explain how we design and implement a software solution written in an object-oriented language (OOL) that meets our objectives. The paper is organized as follows. We begin in Section 2 by presenting Butcher's idea of general linear methods and we show how IMEX schemes can also be worked into this framework. Section 3 describes the design, algorithms and implementation of the ODE solving framework. In Section 4, we then present how this ODE framework can be modified into a generic PDE time-stepping framework meeting the objectives. Therefore, we introduce the MoL decomposition of a model problem, and we explain how to deal with time-dependent boundary conditions in a generic and computationally efficient way. In Section 5, we demonstrate the capabilities of the presented framework by presenting some examples and in Section 6 we summarise and conclude the presented work.

2 General Linear Methods

General linear methods (GLM) have emerged as an effort to connect two main types of time integration schemes: linear multi-step method and linear multi-stage method. Linear multi-step methods, such as Adams family of schemes, use the collection of r input parameters from the previous time-levels to obtain the solution at the next time-level. On the other hand, linear multi-stage methods such as Runge-Kutta methods approximate the solution at the new time-level by linearly combining the solution at s intermediate stages.

To begin, the standard *initial value problem* in autonomous form is represented by the ODE,

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(\boldsymbol{y}), \quad \boldsymbol{y}(t_0) = \boldsymbol{y}_0, \tag{1}$$

where $\boldsymbol{f} : \mathbb{R}^N \to \mathbb{R}^N$. The n^{th} step of the general linear method comprised of r steps and s stages is then formulated as [3]:

$$\boldsymbol{Y}_i = \Delta t \sum_{j=1}^{s} a_{ij} \boldsymbol{F}_j + \sum_{j=1}^{r} u_{ij} \boldsymbol{y}_j^{[n-1]}, \qquad 1 \le i \le s, \tag{2a}$$

$$\boldsymbol{y}_i^{[n]} = \Delta t \sum_{j=1}^{s} b_{ij} \boldsymbol{F}_j + \sum_{j=1}^{r} v_{ij} \boldsymbol{y}_j^{[n-1]}, \qquad 1 \le i \le r, \tag{2b}$$

where $\boldsymbol{Y}_i$ are called the stage values and $\boldsymbol{F}_i$ are called the stage derivatives. Both quantities are related by the differential equation:

$$\boldsymbol{F}_i = \boldsymbol{f}(\boldsymbol{Y}_i). \tag{2c}$$

The matrices $A = [a_{ij}]$, $U = [u_{ij}]$, $B = [b_{ij}]$, $V = [v_{ij}]$ are characteristic of a specific method, and as a result, each scheme can be uniquely defined by the partitioned $(s + r) \times (s + r)$ matrix

$$\begin{bmatrix} A & U \\ B & V \end{bmatrix}. \tag{3}$$

For a more concise notation, it is convenient to define the vectors $\boldsymbol{Y}, \boldsymbol{F} \in \mathbb{R}^{sN}$ and $\boldsymbol{y}_i^{[n-1]}, \boldsymbol{y}_i^{[n]} \in \mathbb{R}^{rN}$ as follows:

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{Y}_1 \\ \boldsymbol{Y}_2 \\ \vdots \\ \boldsymbol{Y}_s \end{bmatrix}, \quad \boldsymbol{F} = \begin{bmatrix} \boldsymbol{F}_1 \\ \boldsymbol{F}_2 \\ \vdots \\ \boldsymbol{F}_s \end{bmatrix}, \quad \boldsymbol{y}^{[n-1]} = \begin{bmatrix} \boldsymbol{y}_1^{[n-1]} \\ \boldsymbol{y}_2^{[n-1]} \\ \vdots \\ \boldsymbol{y}_r^{[n-1]} \end{bmatrix}, \text{ and } \boldsymbol{y}^{[n]} = \begin{bmatrix} \boldsymbol{y}_1^{[n]} \\ \boldsymbol{y}_2^{[n]} \\ \vdots \\ \boldsymbol{y}_r^{[n]} \end{bmatrix}. \tag{4}$$

Using these vectors, it is possible to write Eq. (2a) and Eq. (2b) in the form

$$\begin{bmatrix} \boldsymbol{Y} \\ \boldsymbol{y}^{[n]} \end{bmatrix} = \begin{bmatrix} A \otimes I_N & U \otimes I_N \\ B \otimes I_N & V \otimes I_N \end{bmatrix} \begin{bmatrix} \Delta t \boldsymbol{F} \\ \boldsymbol{y}^{[n-1]} \end{bmatrix}, \tag{5}$$

where I_N is the identity matrix of dimension $N \times N$ and $\otimes$ denotes the Kronecker product. Note that it is the first element of the input vector $\boldsymbol{y}^{[n-1]}$ and output vector $\boldsymbol{y}^{[n]}$ which represents the solution at the corresponding time-level, *i.e.*

$\boldsymbol{y}_1^{[n]} = \boldsymbol{y}_n = \boldsymbol{y}(t_0 + n\Delta t)$. The other subvectors $\boldsymbol{y}_i^{[n]}$ $(2 \leq i \leq r)$ refer to the approximation of an auxiliary set of values inherent to the scheme. These values, in general, are comprised of either solutions or derivatives at earlier time-levels or a combination hereof. Many well-known, as well as lesser known, schemes can be cast as a GLM, see Appendices A.1-A.3.

2.1 Implicit-explicit general linear methods

In this section, we extend the idea of GLM to accommodate in addition implicit-explicit (IMEX) schemes. IMEX schemes [2, 1] were introduced to time-integrate ODEs of the form

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(\boldsymbol{y}) + \boldsymbol{g}(\boldsymbol{y}), \quad \boldsymbol{y}(t_0) = \boldsymbol{y}_0, \tag{6}$$

where $\boldsymbol{f} : \mathbb{R}^N \to \mathbb{R}^N$ typically is a non-linear function and $\boldsymbol{g} : \mathbb{R}^N \to \mathbb{R}^N$ is a stiff term (or where $\boldsymbol{f}$ and $\boldsymbol{g}$ have disparate time-scales). The idea behind IMEX methods is to combine two different type of schemes: one would like to use an implicit scheme for the stiff term in order to avoid an excessively small time-step. At the same time, explicit integration of the non-linear term is preferred to avoid its expensive inversion.

Following the same underlying idea as discussed in the previous sections, IMEX linear multi-step schemes [2] and IMEX Runge-Kutta schemes [1] can be unified into an IMEX general linear method formulation, *i.e.*

$$\boldsymbol{Y}_i = \Delta t \sum_{j=1}^{s} a_{ij}^{\mathrm{IM}} \boldsymbol{G}_j + \Delta t \sum_{j=1}^{s} a_{ij}^{\mathrm{EX}} \boldsymbol{F}_j + \sum_{j=1}^{r} u_{ij} \boldsymbol{y}_j^{[n-1]}, \qquad 1 \leq i \leq s, \tag{7a}$$

$$\boldsymbol{y}_i^{[n]} = \Delta t \sum_{j=1}^{s} b_{ij}^{\mathrm{IM}} \boldsymbol{G}_j + \Delta t \sum_{j=1}^{s} b_{ij}^{\mathrm{EX}} \boldsymbol{F}_j + \sum_{j=1}^{r} v_{ij} \boldsymbol{y}_j^{[n-1]}, \qquad 1 \leq i \leq r, \tag{7b}$$

where the stage derivatives $\boldsymbol{F}_i$ and $\boldsymbol{G}_i$ are defined as

$$\boldsymbol{F}_i = \boldsymbol{f}(\boldsymbol{Y}_i), \quad \boldsymbol{G}_i = \boldsymbol{g}(\boldsymbol{Y}_i), \tag{7c}$$

and where the superscripts IM and EX are used to denote implicit and explicit respectively. Adopting a matrix formulation similar to that shown in Eq. (5), this can be written in the form

$$\left[\begin{array}{c} \boldsymbol{Y} \\ \hline \boldsymbol{y}^{[n]} \end{array}\right] = \left[\begin{array}{c:c|c} A^{\mathrm{IM}} \otimes I_N & A^{\mathrm{EX}} \otimes I_N & U \otimes I_N \\ \hline B^{\mathrm{IM}} \otimes I_N & B^{\mathrm{EX}} \otimes I_N & V \otimes I_N \end{array}\right] \left[\begin{array}{c} \Delta t \boldsymbol{G} \\ \Delta t \boldsymbol{F} \\ \hline \boldsymbol{y}^{[n-1]} \end{array}\right]. \tag{8}$$

To further illustrate the formulation of IMEX schemes as a GLM, a few examples are given in Appendix A.4.

3 A generic ODE solving framework

Just as Butcher's general linear methods provide a general framework to study the basic properties such as consistency, stability and convergence of different families of numerical methods for ODEs, it can also serve as a starting point for a unified numerical implementation. For maximum generality we base our

implementation on the IMEX-GLM formulation described in Section 2.1: for purely explicit methods we simply define A^{IM}, B^{IM} as well as $\boldsymbol{g}(\boldsymbol{y})$ equal to zero. For purely implicit schemes we analogously set A^{EX}, B^{EX} and $\boldsymbol{f}(\boldsymbol{y})$ to be zero.

3.1 Evaluation of general linear methods

Inspecting Eq. (7) it can be appreciated that a single step from level $n-1$ to n for an IMEX-GLM formulation can be evaluated through the following algorithm:

input : the vector $\boldsymbol{y}^{[n-1]}$

output: the vector $\boldsymbol{y}^{[n]}$

`// Calculate stage values` $\boldsymbol{Y}_i$ `and the stage derivatives` $\boldsymbol{F}_i$ `and` $\boldsymbol{G}_i$

for $i = 1$ **to** s **do**

`// calculate the temporary variable` $\boldsymbol{x}_i$

(A1.1) $\quad \boldsymbol{x}_i = \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{IM}} \boldsymbol{G}_j + \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{EX}} \boldsymbol{F}_j + \sum_{j=1}^{r} u_{ij} \boldsymbol{y}_j^{[n-1]}$

`// calculate the stage value` $\boldsymbol{Y}_i$

(A1.2) $\quad$ solve $\left(\boldsymbol{Y}_i - a_{ii}^{\mathrm{IM}} \Delta t \boldsymbol{g}(\boldsymbol{Y}_i) \right) = \boldsymbol{x}_i$

`// calculate the explicit stage derivative` $\boldsymbol{F}_i$

(A1.3) $\quad \boldsymbol{F}_i = \boldsymbol{f}(\boldsymbol{Y}_i)$

`// calculate the implicit stage derivative` $\boldsymbol{G}_i$

(A1.4) $\quad \boldsymbol{G}_i = \boldsymbol{g}(\boldsymbol{Y}_i) = \frac{1}{a_{ii}^{\mathrm{IM}} \Delta t} \left(\boldsymbol{Y}_i - \boldsymbol{x}_i \right)$

end

`// Calculate the output vector` $\boldsymbol{y}^{[n]}$

for $i = 1$ **to** r **do**

`// calculate` $\boldsymbol{y}_i^{[n]}$

(A1.5) $\quad \boldsymbol{y}_i^{[n]} = \Delta t \sum_{j=1}^{s} b_{ij}^{\mathrm{IM}} \boldsymbol{G}_j + \Delta t \sum_{j=1}^{s} b_{ij}^{\mathrm{EX}} \boldsymbol{F}_j + \sum_{j=1}^{r} v_{ij} \boldsymbol{y}_j^{[n-1]}$

end

Algorithm 1: A GLM-based ODE solving algorithm.

Here we first observe that the algorithm, by virtue of the GLM framework, is independent of the actual numerical scheme used – only the *values* of the coefficients a and b change for different methods. Further, if we are using a purely explicit scheme then $a_{ii}^{\mathrm{IM}} = 0$ and the stage value is equal to the the temporary value computed in step (A1.1), *i.e.* step (A1.2) is greatly simplified to $\boldsymbol{Y}_i = \boldsymbol{x}_i$. It is also worth noting that the stage derivative $\boldsymbol{G}_i = \boldsymbol{g}(\boldsymbol{Y}_i)$ in (A1.4) need not be explicitly evaluated, but is given by already computed values, as seen by reordering Eq. (A1.2). Indeed, steps (A1.2) and (A1.3) are the only instances in the algorithm where specific information from the ODE is required, all other steps in the algorithm simply involve linear combinations of precomputed information. These two steps are to be considered *external* parts

representing the ODE rather than being a part of the numerical GLM algorithm. We thus need to define the following external functions:

- If $A^{\mathrm{IM}} \neq 0$ we must supply a routine for solving a system of form (A1.2),

$$(\boldsymbol{Y} - \lambda \boldsymbol{g}(\boldsymbol{Y})) = \boldsymbol{x}, \tag{9}$$

  for $\boldsymbol{Y} \in \mathbb{R}^N$, given as input the vector $\boldsymbol{x} \in \mathbb{R}^N$ and the scalar $\lambda \in \mathbb{R}$. $\boldsymbol{g} : \mathbb{R}^N \to \mathbb{R}^N$ denotes the function prescribing the terms of the ODE that are to be implicitly evaluated. In general, the solution or fixed point of this system can be found through root finding algorithms. In the case $\boldsymbol{g}$ is a linear operator, one may opt for a direct solution method to solve this system through the inverse operator $(\boldsymbol{\mathcal{I}} - \lambda \boldsymbol{g})^{-1}$, where $\boldsymbol{\mathcal{I}} : \mathbb{R}^N \to \mathbb{R}^N$ is the identity function.

- If $A^{\mathrm{EX}} \neq 0$ we must also supply a routine for evaluating (A.1.3), *i.e.* a function $\boldsymbol{f} : \mathbb{R}^N \to \mathbb{R}^N$ that maps the stage values to the stage derivatives for the terms of the ODE that are explicitly evaluated

$$\boldsymbol{F} = \boldsymbol{f}(\boldsymbol{Y}). \tag{10}$$

The external functions are specific for the ODE under consideration and have to be specified to the framework. The *decoupling* of the external components from the GLM algorithm naturally leads to a level of abstraction allowing a generic object-oriented implementation, in a programming language such as C++, to be discussed in the next section.

3.2 Encapsulation of key concepts

As outlined in the introduction, it is our goal to implement an ODE solving toolbox where switching between numerical schemes is as simple as changing an input parameter. To accomplish this, we have encapsulated the key concepts observed in the previous section into a set of C++ type classes, depicted in Fig. 1. It is not our intention to necessarily advocate using only C++ but rather to highlight how any OOL could be used to encapsulate the concepts. We acknowledge that many OOL exist which could be used for this implementation stage.

3.2.1 The class TimeIntegrationOperators

This class provides a general interface to the *external* components (see Section 3.1) needed for time marching. As a result, this class can be seen as the abstraction of the ODE. As data members, it contains two objects which can be thought of as *function pointers*: `m_explicitEvaluate` should point to the implementation of (10) and `m_implicitSolve` should point to the implementation of solving system (9). Note that the actual routines pointed to by the function pointers are not part of the framework and must be provided as they are specific for each ODE under consideration. The function pointers can be linked to these implementations by means of the methods `DefineExplicitEvaluate` and `DefineImplicitSolve`. The encapsulation of these functions into another

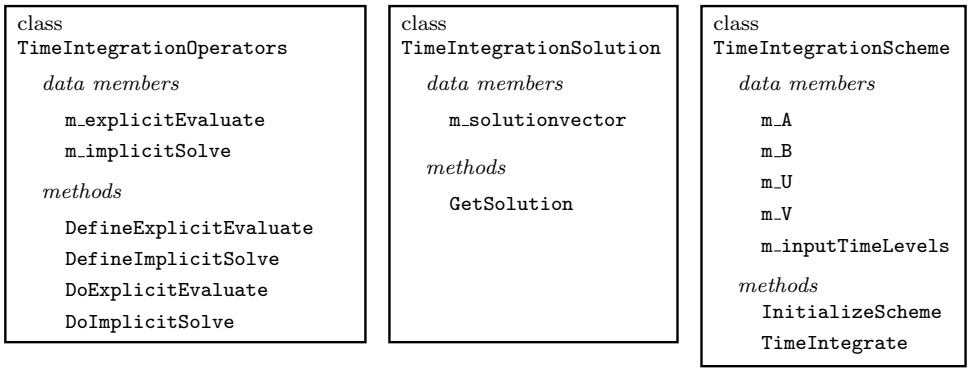


Figure 1: Overview of the classes in the implementation of the generic ODE solving framework.

class is needed to ensure that both these functions can be accessed from within the time-stepping algorithms in a unified fashion, independent of which ODE is being solved. Therefore, the class `TimeIntegrationOperators` also contains the methods `DoExplicitEvaluate` and `DoImplicitSolve` for internal use.

3.2.2 The class TimeIntegrationSolution

This class is the abstraction of the vector $\boldsymbol{y}^{[n]}$ as defined in Eq. (4). One can think of it as an array of arrays. The first entry $\boldsymbol{y}_1^{[n]} = \boldsymbol{y}_n$ represents the approximate solution at time-level n, and can be obtained by means of the method `GetSolution`.

3.2.3 The class TimeIntegrationScheme

This class can be considered as the main class as it is the abstraction of a general linear method. As each scheme is uniquely defined by the partitioned coefficient matrix (3), the sub-matrices A, B, U and V are core data members of this class, implemented respectively as `m_A`, `m_B`, `m_U` and `m_V`. In addition, this class contains the data member `m_inputTimeLevels` which reflect the structure of the input/output vector $\boldsymbol{y}^{[n]}$ associated to the scheme. Based upon the fact that all input vectors can be ordered such that the stage values are listed first before the explicit stage derivatives and the implicit stage derivatives, `m_inputTimeLevels` can be seen as an array existing of three parts that indicate the time-level at which the values/derivatives are evaluated. As an example, consider the second-order Crank-Nicholson/Adams-Bashforth scheme with input vector $\boldsymbol{y}^{[n]}$ as defined in Eq. (73). The data member `m_inputTimeLevels` is then defined as

$$\texttt{m\_inputTimeLevels} = \left[\begin{array}{c} 0 \\ \hline 0 \\ \hline 0 \\ 1 \end{array}\right]. \tag{11}$$

Furthermore, this class is equipped with the two methods needed for the actual time-marching. The function `InitializeScheme` converts the initial value $\boldsymbol{y}_0$ in

an object of the class `TimeIntegrationSolution`. This object is then going to be advanced in time using the method `TimeIntegrate`. It is the `TimeIntegrate` method which actually implements the GLM algorithm in Section 3.1 and hence integrates the ODE for a single time-step. Note that internally, this method calls the functions `DoExplicitEvaluate` and `DoImplicitSolve` of the class `TimeIntegrationOperators` in order to evaluate Eq. (10) and solve Eq. (9) respectively.

3.3 Use of the framework

As mentioned earlier the functionality described by Eqs. (9) and (10) are ODE specific and must be implemented and provided to the framework according to the prototype defined in Code 1.

Code 1 Prototype of functions required to be provided by the user

```
double* ExplicitEvaluate(double* x)
{
    ... // ODE specific implementation
}

double* ImplicitSolve(double* x, double lambda)
{
    ... // ODE specific implementation
}
```

These functions, together with the classes of the toolbox, can then be used to numerically solve the ODE. A typical example of this is shown in the example Code 2. In this particular example, the constructor call

```
scheme = TimeIntegrationScheme(FORWARD_EULER)
```

loads the object with the coefficient matrices of the forward Euler scheme. However, none of the external ODE specific implementation changes when selecting a more advanced time-stepping method. Other schemes can simply be loaded by changing the input argument (*e.g.* from `FORWARD_EULER` to `CLASSICAL_RK_4`). This example illustrates how the presented framework can be used to numerically solve an ODE in a unified fashion, independent of the chosen scheme.

3.3.1 Initiating multi-step schemes

For multi-step schemes, a slight modification is required to properly start-up the system. In the framework we use the classic start-up procedure of employing the first $k-1$ step of a k-step multi-step scheme using a kth order multi-stage scheme, as illustrated for the third-order Adams-Bashforth scheme in the example Code 3. For non-stiff problem we use an explicit RK scheme while for stiff problems the start-up could be performed by means of a DIRK scheme.

Underneath, this starting-up procedure is founded upon the data member `m_inputTimeLevels`. When making the call

Code 2 Example demonstrating how the framework can be used to solve an ODE

```
TimeIntegrationOperators  ode;
TimeIntegrationSolution   y_n;
TimeIntegrationScheme     scheme;

ode.DefineExplicitEvaluate(&ExplicitEvaluate);
ode.DefineImplicitSolve(&ImplicitSolve);

scheme = TimeIntegrationScheme(FORWARD_EULER);

y_n = scheme->InitializeScheme(dt,y_0,ode);

for(n = 0; n < nsteps; ++n)
{
    scheme->TimeIntegrate(dt,y_n,ode);
}
```

Code 3 Example demonstrating the initiation of multi-step schemes.

```
scheme          = TimeIntegrationScheme(ADAMS_BASHFORTH_ORDER3);
startup_scheme  = TimeIntegrationScheme(RK_ORDER3);

y_n = scheme->InitializeScheme(dt,y_0,ode);

startup_scheme->TimeIntegrate(dt,y_n,ode);   // step n = 0
startup_scheme->TimeIntegrate(dt,y_n,ode);   // step n = 1

for(n = 2; n < nsteps; ++n)
{
    scheme->TimeIntegrate(dt,y_n,ode);
}
```

```
startup_scheme1->TimeIntegrate(dt,y_n,ode)
```

the `TimeIntegrate` routine recognises that the input vector `y_n` is initialised according to another scheme. It is therefore going first to construct an input vector according to the start-up scheme, and it will map the information from the vector `y_n` to the newly constructed input vector, thereby making use of the data member `m_inputTimeLevels`. If the start-up scheme requires information in its input vector that is not available in the provided input vector `y_n` it will simply assume zero for these stage values or derivatives. Once the solution is advanced in time for a single time-step using the start-up scheme, the output vector is mapped back to the vector `y_n`, again making use of the information in `m_inputTimeLevels`.

The start-up procedure effectively integrates the ODE $k-1$ steps. If this is not desirable then the derivatives in $y^{[0]}$ must be estimated by a more elaborate starting procedure, see *e.g.* [28, 30, 17].

4 Time-dependent partial differential equations

Ordinary differential equations are generally used to model *initial value problems*. However, many physical processes can be regarded as *initial boundary value problems* which are described by partial differential equations. A first step in solving time-dependent PDEs consists of reducing the PDE to a system of ODEs through the MoL approach. For this procedure, which involves the discretisation of the spatial dimensions, we will primarily adopt the spectral/*hp* element method [21] in this work. We will show that the application of the MoL introduces some typical issues which prevent the straightforward application of the ODE framework discussed before. We distinguish the following issues:

- strongly enforced essential boundary conditions, and
- computational efficiency.

To facilitate the discussion we will use the scalar advection-diffusion equation as an illustrative example throughout this section. It is given by

$$\frac{\partial u}{\partial t} + \nabla \cdot \boldsymbol{F}(u) = \nabla^2 u, \qquad \text{in } \Omega \times [0, \infty), \tag{12a}$$

$$u(\boldsymbol{x}, t) = g_D(\boldsymbol{x}, t), \qquad \text{on } \partial\Omega_D \times [0, \infty), \tag{12b}$$

$$\frac{\partial u}{\partial n}(\boldsymbol{x}, t) = g_N(\boldsymbol{x}, t) \cdot \boldsymbol{n}, \qquad \text{on } \partial\Omega_N \times [0, \infty), \tag{12c}$$

$$u(\boldsymbol{x}, 0) = u_0(\boldsymbol{x}), \qquad \text{in } \Omega, \tag{12d}$$

where Ω is a bounded domain of $\mathbb{R}^d$ with boundary $\partial\Omega = \partial\Omega_D \bigcup \partial\Omega_N$ and $\boldsymbol{n}$ denotes the outward normal to the boundary $\partial\Omega$. Furthermore, we will abbreviate the advection term as $f(u) = -\nabla \cdot \boldsymbol{F}(u)$ in the following sections.

4.1 The Method of Lines

We start with a (possibly high-order) finite element approach to reduce the advection-diffusion equation (12) to a system of ODEs using the MoL. Following the standard Galerkin formulation we multiply Eq. (12a) by a smooth test function $v(\boldsymbol{x})$, which by definition is zero on all Dirichlet boundaries. Integrating over the entire spatial domain leads to the following variational formulation: Find $u \in \mathcal{U}$ such that

$$\int_\Omega v \frac{\partial u}{\partial t} d\boldsymbol{x} - \int_\Omega v f(u) d\boldsymbol{x} = \int_\Omega v \nabla^2 u d\boldsymbol{x}, \quad \forall v \in \mathcal{V}, \tag{13}$$

where $\mathcal{U}$ and $\mathcal{V}$ are suitably chosen trial and test spaces respectively. We obtain the weak form of the diffusion operator by applying the divergence theorem to the right-hand-side term yielding: Find $u \in \mathcal{U}$ such that

$$\int_\Omega v \frac{\partial u}{\partial t} d\boldsymbol{x} - \int_\Omega v f(u) d\boldsymbol{x} = -\int_\Omega \nabla v \cdot \nabla u d\boldsymbol{x} + \int_{\partial\Omega} v \nabla u \cdot \boldsymbol{n} d\boldsymbol{x}, \quad \forall v \in \mathcal{V}. \tag{14}$$

As $v(\partial\Omega_D)$ is equal to zero, only Neumann conditions will give contributions to the boundary integral, and we enforce the conditions weakly through substituting $\nabla u = \boldsymbol{g}_N$ in the boundary integral. In order to impose Dirichlet boundary

conditions one can choose to adopt a lifting strategy where the solution is decomposed into a known function, u^D and an unknown homogeneous function u^H, *i.e.*

$$u(\boldsymbol{x},t) = u^H(\boldsymbol{x},t) + u^D(\boldsymbol{x},t). \tag{15}$$

Here u^D satisfies the Dirichlet boundary conditions, $u^D(\partial\Omega_D) = g_D$, and the homogeneous function is equal to zero on the Dirichlet boundary, $u^H(\partial\Omega_D) = 0$. The weak form (14) can then be formulated as: Find $u^D \in \mathcal{U}_0$ such that,

$$\begin{aligned}\int_\Omega v\frac{\partial(u^H+u^D)}{\partial t}d\boldsymbol{x} - \int_\Omega v f(u^H+u^D)d\boldsymbol{x} = &-\int_\Omega \nabla v\cdot(\nabla u^H + \nabla u^D)d\boldsymbol{x} \\ &+\int_{\partial\Omega_N} v\boldsymbol{g}_N\cdot\boldsymbol{n}d\boldsymbol{x}, \quad \forall v\in\mathcal{V}.\end{aligned} \tag{16}$$

Following a finite element discretisation procedure, the solution is expanded in terms of a globally C^0-continuous expansion basis Φ_i that spans the finite dimensional solution space $\mathcal{U}^\delta$. We also decompose this expansion basis into the homogeneous basis functions Φ_i^H and the basis functions Φ_i^D having support on the Dirichlet boundary such that

$$u^\delta(\boldsymbol{x},t) = \sum_{i\in\mathcal{N}^H}\Phi_i^H(\boldsymbol{x})\hat{u}_i^H(t) + \sum_{i\in\mathcal{N}^D}\Phi_i^D(\boldsymbol{x})\hat{u}_i^D(t). \tag{17}$$

Finally, employing the same expansion basis Φ_i^H to span the test space $\mathcal{V}$, Eq. (16) leads to the semi-discrete system of ODEs

$$\left[\begin{array}{cc}\boldsymbol{M}^{HD} & \boldsymbol{M}^{HH}\end{array}\right]\frac{d}{dt}\left[\begin{array}{c}\hat{\boldsymbol{u}}^D\\ \hat{\boldsymbol{u}}^H\end{array}\right] = -\left[\begin{array}{cc}\boldsymbol{L}^{HD} & \boldsymbol{L}^{HH}\end{array}\right]\left[\begin{array}{c}\hat{\boldsymbol{u}}^D\\ \hat{\boldsymbol{u}}^H\end{array}\right] + \boldsymbol{\Gamma}^H + \hat{\boldsymbol{f}}^H \tag{18}$$

where

$$\begin{aligned}
\boldsymbol{M}^{HH}[i][j] &= \int_\Omega \Phi_i^H\Phi_j^H d\boldsymbol{x} && i\in\mathcal{N}^H, j\in\mathcal{N}^H,\\
\boldsymbol{M}^{HD}[i][j] &= \int_\Omega \Phi_i^H\Phi_j^D d\boldsymbol{x} && i\in\mathcal{N}^H, j\in\mathcal{N}^D,\\
\boldsymbol{L}^{HH}[i][j] &= \int_\Omega \nabla\Phi_i^H\cdot\nabla\Phi_j^H d\boldsymbol{x} && i\in\mathcal{N}^H, j\in\mathcal{N}^H,\\
\boldsymbol{L}^{HD}[i][j] &= \int_\Omega \nabla\Phi_i^H\cdot\nabla\Phi_j^D d\boldsymbol{x} && i\in\mathcal{N}^H, j\in\mathcal{N}^D,\\
\hat{\boldsymbol{f}}^H[i] &= \int_\Omega \Phi_i^H f(u) d\boldsymbol{x} && i\in\mathcal{N}^H\\
\boldsymbol{\Gamma}^H[i] &= \int_{\partial\Omega_N}\Phi_i^H\boldsymbol{g}_N\cdot\boldsymbol{n}d\boldsymbol{x} && i\in\mathcal{N}^H.
\end{aligned}$$

This can be rewritten in terms of the unknown variable $\hat{\boldsymbol{u}}^H$ as

$$\frac{d\hat{\boldsymbol{u}}^H}{dt} = \left(\boldsymbol{M}^{HH}\right)^{-1}\left\{-\left[\begin{array}{cc}\boldsymbol{L}^{HD} & \boldsymbol{L}^{HH}\end{array}\right]\left[\begin{array}{c}\hat{\boldsymbol{u}}^D\\ \hat{\boldsymbol{u}}^H\end{array}\right] + \boldsymbol{\Gamma}^H + \hat{\boldsymbol{f}}^H - \boldsymbol{M}^{HD}\frac{d\hat{\boldsymbol{u}}^D}{dt}\right\}, \tag{19}$$

which, in the absence of Dirichlet boundary conditions, simplifies to

$$\frac{d\hat{\boldsymbol{u}}}{dt} = -\boldsymbol{M}^{-1}\left(\boldsymbol{L}\hat{\boldsymbol{u}} - \boldsymbol{\Gamma}\right) + \boldsymbol{M}^{-1}\hat{\boldsymbol{f}} \tag{20}$$

4.2 Use of the ODE framework for time integrating PDEs

At first sight, it may seem feasible to apply ODE framework of Section 3 to time-integrate Eq. (19) (or Eq. (20)). However, this straightforward approach appears to lead to two problems.

4.2.1 Computational efficiency

Considering Eq. (20) in the context of the IMEX algorithm of the ODE framework (Algorithm 1), it can be appreciated that for the explicit advection term, step (A.1.4) requires the calculation of the term

$$\boldsymbol{M}^{-1}\hat{\boldsymbol{f}}, \tag{21}$$

while for the implicit diffusion term, step (A.1.2) would require solving a system of the form

$$\left(\boldsymbol{I} + \Delta t \boldsymbol{M}^{-1}\boldsymbol{L}\right)\hat{\boldsymbol{u}} = \hat{\boldsymbol{x}}. \tag{22}$$

It appears that next to the implicit term, the explicit term now also requires a global matrix inverse due to $\boldsymbol{M}^{-1}$. This means that the generic ODE time-stepping algorithm would require two global matrix inverses at every timestep/timestage. For comparison, let us consider the (single-stage) first-order Backward Euler/Forward Euler IMEX scheme given by Eq. (70). A scheme-specific implementation of this method (that is, not making use of the proposed framework) can integrate Eq. (20) for a single time-step as

$$\hat{\boldsymbol{u}}_n = \left(\boldsymbol{M} + \Delta t \boldsymbol{L}\right)^{-1}\left(\boldsymbol{M}\hat{\boldsymbol{u}}_{n-1} + \Delta t \hat{\boldsymbol{f}}_{n-1} + \Delta t \boldsymbol{\Gamma}_n\right). \tag{23}$$

Clearly, this only involves a single global matrix inversion, *i.e.* $\left(\boldsymbol{M} + \Delta t \boldsymbol{L}\right)^{-1}$. Such global matrix inversions can be assumed to be the critical cost of the time-integration process as they typically –especially for three-dimensional simulations– require an iterative solution method which induce a far bigger cost than the other "forward" operations. Because of this substantial performance penalty, using the ODE framework to time-integrate PDEs can be argued to be impractical.

4.2.2 Time-dependent Dirichlet boundary conditions

The second complication with applying the framework to time integrate Eq. (19) or (20) arises from the term $\frac{d\hat{\boldsymbol{u}}^D}{dt}$ in Eq. (19) which is due to a strong imposition of the Dirichlet boundary conditions. Although the value $\hat{\boldsymbol{u}}^D(t)$ of the Dirichlet boundary conditions would typically be given for arbitrary t, a prescription of its time rate-of-change $\frac{d\hat{\boldsymbol{u}}^D}{dt}$ is not usually available. This again prevents a straightforward application of the presented ODE framework.

4.3 A generic PDE time-stepping framework

In order to alleviate both the issues of efficiency and time-dependent boundary conditions, we propose a modified framework designed to time-integrate PDEs in a generic *and* efficient manner. The new framework is largely founded on the fact that a finite or spectral/*hp* element approximation $u^\delta(\boldsymbol{x}, t)$ can be described not only by a set of global degrees of freedom $\hat{\boldsymbol{u}}$ (in *coefficient space*), but also

by a set of nodal values $\boldsymbol{u}$ (in *physical space*). These nodal values represent the spectral/*hp* solution at a set of quadrature points $\boldsymbol{x}_i$ (or *collocation points*), such that they can be related to the global coefficients as

$$\boldsymbol{u}[i] = u^\delta(\boldsymbol{x}, t) = \sum_{j \in \mathcal{N}} \Phi_j(\boldsymbol{x}_i)\hat{u}_i(t), \tag{24}$$

which, in matrix notation, can be written as $\boldsymbol{u} = \boldsymbol{B}\hat{\boldsymbol{u}}$, where $\boldsymbol{B}[i][j] = \Phi_j(\boldsymbol{x}_i)$. In case of a lifted Dirichlet solution, this becomes

$$\boldsymbol{u} = \begin{bmatrix} \boldsymbol{B}^D & \boldsymbol{B}^H \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{u}}^D \\ \hat{\boldsymbol{u}}^H \end{bmatrix}, \tag{25}$$

with $\boldsymbol{B}^D[i][j] = \Phi_j^D(\boldsymbol{x}_i)$ and $\boldsymbol{B}^H[i][j] = \Phi_j^H(\boldsymbol{x}_i)$.

As commonly is the case in finite or spectral/*hp* methods, we will also use this nodal interpretation for the explicit treatment of the (non-linear) advection term. The term $\hat{\boldsymbol{f}}$ in Eq. (20) will then be computed as

$$\hat{\boldsymbol{f}} = \boldsymbol{B}^\top \boldsymbol{W} \boldsymbol{f}, \tag{26}$$

where $\boldsymbol{f}$ represents the original advection term evaluated at the quadrature points, *i.e.* $\boldsymbol{f}[i] = f(u(\boldsymbol{x}_i))$, and $\boldsymbol{W}$ is a diagonal matrix containing the quadrature weights needed for an appropriate numerical evaluation of the integral. Such a collocation approach is also known as the *pseudo-spectral* method [14].

4.3.1 The Helmholtz problem and the projection problem

Before we derive the new framework, we will first introduce the following two concepts which will facilitate the derivation.

The Galerkin projection Consider the discrete solution space $\mathcal{U}^\delta(\Omega, t)$ of C^0-continuous piecewise polynomial functions that satisfy the (possibly time-dependent) Dirichlet boundary conditions. In a finite or spectral/*hp* methods we typically define the projection of an arbitrary function $f(\boldsymbol{x})$, denoted as

$$u = \mathcal{P}(f, t), \tag{27}$$

as the L^2 projection of f onto $\mathcal{U}^\delta(\Omega)$.

This projection is equivalent to solving the following minimisation problem using a traditional Galerkin finite element approach: Find $u \in \mathcal{U}^\delta(\Omega)$ such that $||u - f||_{L^2}$ is minimal. In a nodal/collocated context, this projection can be computed as:

- in case of strongly enforced Dirichlet boundary conditions

$$\boldsymbol{u} = \boldsymbol{\mathcal{P}}(\boldsymbol{f}, t) = \begin{bmatrix} \boldsymbol{B}^D & \boldsymbol{B}^H \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{u}}^D(t) \\ \left(\boldsymbol{M}^{HH}\right)^{-1} \left\{ \left(\boldsymbol{B}^H\right)^\top \boldsymbol{W}\boldsymbol{f} - \boldsymbol{M}^{HD}\hat{\boldsymbol{u}}^D(t) \right\} \end{bmatrix}, \tag{28}$$

- which in the absence of strongly enforced Dirichlet boundary conditions, simplifies to

$$\boldsymbol{u} = \boldsymbol{\mathcal{P}}(\boldsymbol{f}, t) = \boldsymbol{B}\boldsymbol{M}^{-1}\boldsymbol{B}^\top \boldsymbol{W}\boldsymbol{f}. \tag{29}$$

The Galerkin Helmholtz problem Given an arbitrary function f, we define the Helmholtz problem as finding the Galerkin finite element solution to the (steady) elliptic Helmholtz equation

$$u - \lambda \nabla^2 u = f, \qquad \text{in } \Omega, \tag{30a}$$
$$u(\boldsymbol{x}) = g_D(\boldsymbol{x}), \qquad \text{on } \partial\Omega_D, \tag{30b}$$
$$\frac{\partial u}{\partial n}(\boldsymbol{x}) = \boldsymbol{g}_N(\boldsymbol{x}) \cdot \boldsymbol{n}, \qquad \text{on } \partial\Omega_N. \tag{30c}$$

We will also denote this problem as

$$u = \mathcal{H}(f, \lambda, t). \tag{31}$$

Once again evaluating the solution at nodal/collocation points, this problem can be discretely evaluated as

- in case of strongly enforced Dirichlet boundary conditions

$$\boldsymbol{u} = \boldsymbol{\mathcal{H}}(\boldsymbol{f}, \lambda, t) = \begin{bmatrix} \boldsymbol{B}^D & \boldsymbol{B}^H \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{u}}^D(t) \\ \left(\boldsymbol{H}^{HH}\right)^{-1} \left\{ \left(\boldsymbol{B}^H\right)^\top \boldsymbol{W}\boldsymbol{f} + \lambda \boldsymbol{\Gamma}^H(t) - \boldsymbol{H}^{HD}\hat{\boldsymbol{u}}^D(t) \right\} \end{bmatrix}, \tag{32}$$

- which in the absence of strongly enforced Dirichlet boundary conditions, simplifies to

$$\boldsymbol{u} = \boldsymbol{\mathcal{H}}(\boldsymbol{f}, \lambda, t) = \boldsymbol{B}\boldsymbol{H}^{-1}\left(\boldsymbol{B}^\top \boldsymbol{W}\boldsymbol{f} + \lambda \boldsymbol{\Gamma}(t)\right). \tag{33}$$

In the expressions above, the matrix $\boldsymbol{H}$ represents the Helmholtz matrix defined as

$$\boldsymbol{H}[i][j] = \int_\Omega \Phi_i \Phi_j + \lambda \nabla \Phi_i \cdot \nabla \Phi_j d\boldsymbol{x} \quad i, j \in \mathcal{N}. \tag{34}$$

Properties We will use the following properties of the operators $\boldsymbol{\mathcal{P}}$ and $\boldsymbol{\mathcal{H}}$ in the subsequent sections:

- In case $\lambda = 0$, the operator $\boldsymbol{\mathcal{H}}$ reduces to the projection operator $\boldsymbol{\mathcal{P}}$, *i.e.*

$$\boldsymbol{\mathcal{H}}(\boldsymbol{f}, 0, t) = \boldsymbol{\mathcal{P}}(\boldsymbol{f}, t). \tag{35}$$

- The operators can be shown to have the following properties:

$$\boldsymbol{\mathcal{P}}(\boldsymbol{\mathcal{P}}(\boldsymbol{f}, t_m), t_n) = \boldsymbol{\mathcal{P}}(\boldsymbol{f}, t_n), \tag{36}$$
$$\boldsymbol{\mathcal{H}}(\boldsymbol{\mathcal{P}}(\boldsymbol{f}, t_m), t_n) = \boldsymbol{\mathcal{H}}(\boldsymbol{f}, t_n) \tag{37}$$
$$, \boldsymbol{\mathcal{P}}(\boldsymbol{g} + \boldsymbol{\mathcal{P}}(\boldsymbol{f}, t_m), t_n) = \boldsymbol{\mathcal{P}}(\boldsymbol{g} + \boldsymbol{f}, t_n), \text{ and} \tag{38}$$
$$\boldsymbol{\mathcal{H}}(\boldsymbol{g} + \boldsymbol{\mathcal{P}}(\boldsymbol{f}, t_m), t_n) = \boldsymbol{\mathcal{H}}(\boldsymbol{g} + \boldsymbol{f}, t_n). \tag{39}$$

4.3.2 Derivation of the framework

According to Eq. (7a), the calculation of the i^{th} stage (for convenience of notation denoted as $\hat{\boldsymbol{u}}_i^H$) of an arbitrary GLM applied to Eq. (19) can be represented as

$$\begin{aligned}\hat{\boldsymbol{u}}_i^H =& \Delta t \sum_{j=1}^{i} a_{ij}^{\mathrm{IM}} \left[\left(\boldsymbol{M}^{HH} \right)^{-1} \left(\hat{\boldsymbol{g}}_j^H - \boldsymbol{M}^{HD} \frac{d\hat{\boldsymbol{u}}^D}{dt}\bigg|_j \right) \right] \\ &+ \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{EX}} \left[\left(\boldsymbol{M}^{HH} \right)^{-1} \hat{\boldsymbol{f}}_j^H \right] + \sum_{j=1}^{r} u_{ij} \hat{\boldsymbol{u}}_j^{H[n-1]}, \end{aligned} \tag{40}$$

where for simplicity we have used the notation

$$\hat{\boldsymbol{g}}_j^H = - \begin{bmatrix} \boldsymbol{L}^{HD} & \boldsymbol{L}^{HH} \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{u}}_j^D \\ \hat{\boldsymbol{u}}_j^H \end{bmatrix} + \boldsymbol{\Gamma}_j^H. \tag{41}$$

For generality we will assume a GLM with an input/output vector of the form

$$\hat{\boldsymbol{u}}^{H[n]} = \begin{bmatrix} \hat{\boldsymbol{u}}_n^H \\ \hat{\boldsymbol{u}}_{n-1}^H \\ \Delta t \boldsymbol{G}_n \\ \Delta t \boldsymbol{G}_{n-1} \\ \Delta t \boldsymbol{F}_n \\ \Delta t \boldsymbol{F}_{n-1} \end{bmatrix}, \tag{42}$$

which applied to the advection-diffusion example under consideration, leads to

$$\hat{\boldsymbol{u}}^{H[n]} = \begin{bmatrix} \hat{\boldsymbol{u}}_n^H \\ \hat{\boldsymbol{u}}_{n-1}^H \\ \Delta t \left(\boldsymbol{M}^{HH} \right)^{-1} \left(\hat{\boldsymbol{g}}_n^H - \boldsymbol{M}^{HD} \frac{d\hat{\boldsymbol{u}}^D}{dt}\Big|_n \right) \\ \Delta t \left(\boldsymbol{M}^{HH} \right)^{-1} \left(\hat{\boldsymbol{g}}_{n-1}^H - \boldsymbol{M}^{HD} \frac{d\hat{\boldsymbol{u}}^D}{dt}\Big|_{n-1} \right) \\ \Delta t \left(\boldsymbol{M}^{HH} \right)^{-1} \hat{\boldsymbol{f}}_n^H \\ \Delta t \left(\boldsymbol{M}^{HH} \right)^{-1} \hat{\boldsymbol{f}}_{n-1}^H \end{bmatrix}. \tag{43}$$

In order to deal with the time-derivative of the Dirichlet boundary condition, we first would like to note that we have chosen to treat the term involving $\dfrac{d\hat{\boldsymbol{u}}^D}{dt}$ implicitly in Eq. (40). However, this is an arbitrary choice and we could equally well have chosen to treat this term explicitly, leading to exactly the same framework. If we then acknowledge that the variable $\hat{\boldsymbol{u}}^D$ can be understood to satisfy the ODE

$$\left(\hat{\boldsymbol{u}}^D \right)' = \frac{d\hat{\boldsymbol{u}}^D}{dt}, \tag{44}$$

we can apply the same GLM to this ODE as the one we have used for the original ODE in terms of $\hat{\boldsymbol{u}}^H$, *i.e.* Eq. (40), to arrive at

$$\hat{\boldsymbol{u}}_i^D = \Delta t \sum_{j=1}^{i} a_{ij}^{\mathrm{IM}} \left. \frac{d\hat{\boldsymbol{u}}^D}{dt} \right|_j + \sum_{j=1}^{r} u_{ij} \hat{\boldsymbol{u}}_j^{D[n-1]}. \tag{45}$$

There are no explicit stage derivatives $\boldsymbol{F}_j$ appearing in the equation above (or more precisely, $\boldsymbol{F}_j = 0$) due to the fact that we also choose to treat the right-hand-side term $\dfrac{d\hat{\boldsymbol{u}}^D}{dt}$ in Eq. (44) implicitly. As a result, the input/output vector of the GLM under consideration, see Eq. (42), applied to Eq. (44) takes the form

$$\hat{\boldsymbol{u}}^{D[n]} = \begin{bmatrix} \hat{\boldsymbol{u}}_n^D \\ \hat{\boldsymbol{u}}_{n-1}^D \\ \Delta t \left.\dfrac{d\hat{\boldsymbol{u}}^D}{dt}\right|_n \\ \Delta t \left.\dfrac{d\hat{\boldsymbol{u}}^D}{dt}\right|_{n-1} \\ 0 \\ 0 \end{bmatrix}. \tag{46}$$

To eliminate the Dirichlet derivative in Eq. (40), we substitute Eq. (45) into Eq. (40), yielding

$$\begin{aligned} \hat{\boldsymbol{u}}_i^H =& \Delta t \sum_{j=1}^{i} a_{ij}^{\mathrm{IM}} \left[\left(\boldsymbol{M}^{HH} \right)^{-1} \hat{\boldsymbol{g}}_j^H \right] + \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{EX}} \left[\left(\boldsymbol{M}^{HH} \right)^{-1} \hat{\boldsymbol{f}}_j^H \right] \\ &+ \left(\boldsymbol{M}^{HH} \right)^{-1} \boldsymbol{M}^{HD} \left[\sum_{j=1}^{r} u_{ij} \hat{\boldsymbol{u}}_j^{D[n-1]} - \hat{\boldsymbol{u}}_i^D \right] + \sum_{j=1}^{r} u_{ij} \hat{\boldsymbol{u}}_j^{H[n-1]}, \end{aligned} \tag{47}$$

which after rearranging and multiplication with $\boldsymbol{M}^{HH}$ leads to

$$\begin{aligned} \boldsymbol{M}^{HH} \hat{\boldsymbol{u}}_i^H + \boldsymbol{M}^{HD} \hat{\boldsymbol{u}}_i^D =& \Delta t \sum_{j=1}^{i} a_{ij}^{\mathrm{IM}} \hat{\boldsymbol{g}}_j^H + \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{EX}} \hat{\boldsymbol{f}}_j^H \\ &+ \sum_{j=1}^{r} u_{ij} \left[\boldsymbol{M}^{HH} \hat{\boldsymbol{u}}_j^{H[n-1]} + \boldsymbol{M}^{HD} \hat{\boldsymbol{u}}_j^{D[n-1]} \right], \end{aligned} \tag{48}$$

or

$$\begin{aligned} \boldsymbol{H}^{HH} \hat{\boldsymbol{u}}_i^H + \boldsymbol{H}^{HD} \hat{\boldsymbol{u}}_i^D =& \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{IM}} \hat{\boldsymbol{g}}_j^H + \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{EX}} \hat{\boldsymbol{f}}_j^H \\ &+ \sum_{j=1}^{r} u_{ij} \left[\boldsymbol{M}^{HH} \hat{\boldsymbol{u}}_j^{H[n-1]} + \boldsymbol{M}^{HD} \hat{\boldsymbol{u}}_j^{D[n-1]} \right] + a_{ii}^{\mathrm{IM}} \Delta t \boldsymbol{\Gamma}_i^H, \end{aligned} \tag{49}$$

where $\boldsymbol{H}$ is the Helmholtz matrix, see Eq. (34), with $\lambda = a_{ii}^{\mathrm{IM}} \Delta t$. This elimination of $\dfrac{d\hat{\boldsymbol{u}}^D}{dt}$ appears to give rise to a modified input/output vector $\boldsymbol{M}^{HH} \hat{\boldsymbol{u}}_j^{H[n-1]} + \boldsymbol{M}^{HD} \hat{\boldsymbol{u}}_j^{D[n-1]}$, which after combining Eq. (43) and Eq. (46)

can be appreciated to be equal to

$$\boldsymbol{M}^{HH}\hat{\boldsymbol{u}}^{H[n]} + \boldsymbol{M}^{HD}\hat{\boldsymbol{u}}^{D[n]} = \begin{bmatrix} \boldsymbol{M}^{HH}\hat{\boldsymbol{u}}_n^H + \boldsymbol{M}^{HD}\hat{\boldsymbol{u}}_n^D \\ \boldsymbol{M}^{HH}\hat{\boldsymbol{u}}_{n-1}^H + \boldsymbol{M}^{HD}\hat{\boldsymbol{u}}_{n-1}^D \\ \Delta t\hat{\boldsymbol{g}}_n^H \\ \Delta t\hat{\boldsymbol{g}}_{n-1}^H \\ \Delta t\hat{\boldsymbol{f}}_n^H \\ \Delta t\hat{\boldsymbol{f}}_{n-1}^H \end{bmatrix}. \tag{50}$$

Following a collocation approach to calculate the advection term, see Eq. (26), and adopting a nodal interpretation for the solution values at the time-levels n and $n-1$ according to Eq. (25), this input/output vector can be considered as the inner product of an input/output vector $\boldsymbol{u}^{[n]}$ in physical space, *i.e.*

$$\boldsymbol{M}^{HH}\hat{\boldsymbol{u}}^{H[n]} + \boldsymbol{M}^{HD}\hat{\boldsymbol{u}}^{D[n]} = \left(\boldsymbol{B}^H\right)^\top \boldsymbol{W}\boldsymbol{u}^{[n]} = \left(\boldsymbol{B}^H\right)^\top \boldsymbol{W} \begin{bmatrix} \boldsymbol{u}_n \\ \boldsymbol{u}_{n-1} \\ \Delta t\boldsymbol{g}_n \\ \Delta t\boldsymbol{g}_{n-1} \\ \Delta t\boldsymbol{f}_n \\ \Delta t\boldsymbol{f}_{n-1} \end{bmatrix}, \tag{51}$$

where we have made use of the relationship $\boldsymbol{M}^{HD} = \left(\boldsymbol{B}^H\right)^\top \boldsymbol{W}\boldsymbol{B}^D$. In addition, we have also adopted a nodal interpretation $\boldsymbol{g}_n$ of the implicit stage value $\hat{\boldsymbol{g}}_n^H$ which will be further discussed in the next section. Making use of this formulation in physical space, Eq. (49) can be written as

$$\begin{aligned} \boldsymbol{H}^{HH}\hat{\boldsymbol{u}}_i^H + \boldsymbol{H}^{HD}\hat{\boldsymbol{u}}_i^D = \left(\boldsymbol{B}^H\right)^\top \boldsymbol{W} \left\{ \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{IM}} \boldsymbol{g}_j + \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{EX}} \boldsymbol{f}_j + \sum_{j=1}^{r} u_{ij}\boldsymbol{u}_j \right\} \\ + a_{ii}^{\mathrm{IM}} \Delta t \boldsymbol{\Gamma}_i^H. \end{aligned} \tag{52}$$

Finally also adopting a nodal interpretation $\boldsymbol{u}_i$ for the solution at stage i, the calculation of the i^{th} stage value can be recognised as

$$\boldsymbol{u}_i = \boldsymbol{\mathcal{H}} \left(\Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{IM}} \boldsymbol{g}_j + \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{EX}} \boldsymbol{f}_j + \sum_{j=1}^{r} u_{ij}\boldsymbol{u}_j, a_{ii}^{\mathrm{IM}} \Delta t, t_n \right). \tag{53}$$

This formulation allows for a well defined procedure to advance the solution in time as the calculation of the stage values only involves solving the associated Helmholtz problem. Note that for a pure explicit method, the Helmholtz problem reduces to the L^2 projection. This solution procedure is very attractive in particularly for the following three reasons:

- uniform treatment of Dirichlet boundary conditions (*i.e.* the Dirichlet boundary conditions only come into play when enforcing them while solving the global matrix system),
- only one global matrix inverse is required per stage, and
- it is sufficiently generic to be extended to the entire range of GLMs (see next section).

Note that the derivation of this framework was founded on the following two steps:

- adopting a consistent-order discretisation of the Dirichlet derivative consistent to the discretisation of the original ODE, and
- formulating the GLM algorithm in physical space.

4.3.3 Algorithm

A PDE time-stepping algorithm that time-integrates the advection-diffusion equation (12) from time-level $n-1$ to n can then be formulated as:

input : the vector $\boldsymbol{u}^{[n-1]}$

output: the vector $\boldsymbol{u}^{[n]}$

`// Calculate stage values` $\boldsymbol{U}_i$ `and the stage derivatives` $\boldsymbol{F}_i$ `and` $\boldsymbol{G}_i$

for $i = 1$ **to** s **do**

`// calculate the temporary variable` $\boldsymbol{x}_i$

(A2.1) $\quad \boldsymbol{x}_i = \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{IM}} \boldsymbol{G}_j + \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{EX}} \boldsymbol{F}_j + \sum_{j=1}^{r} u_{ij} \boldsymbol{u}_j^{[n-1]}$

`// calculate the stage value` $\boldsymbol{U}_i$

(A2.2) $\quad \boldsymbol{U}_i = \boldsymbol{\mathcal{H}}\left(\boldsymbol{x}_i, a_{ii}^{\mathrm{IM}} \Delta t, t_i\right)$

`// calculate the explicit stage derivative` $\boldsymbol{F}_i$

(A2.3) $\quad \boldsymbol{F}_i = \boldsymbol{f}(\boldsymbol{U}_i)$

`// calculate the implicit stage derivative` $\boldsymbol{G}_i$

(A2.4) $\quad \boldsymbol{G}_i = \frac{1}{a_{ii}^{\mathrm{IM}} \Delta t} \left(\boldsymbol{U}_i - \boldsymbol{x}_i\right)$

end

`// Calculate the output vector` $\boldsymbol{u}^{[n]}$

if *last stage equals new solution* **then**

(A2.5) $\quad \boldsymbol{u}_n = \boldsymbol{U}_s$

else

(A2.6) $\quad \boldsymbol{u}_1^{[n]} = \boldsymbol{u}_1^{[n]} =$
$\boldsymbol{\mathcal{P}}\left(\Delta t \sum_{j=1}^{s} b_{1j}^{\mathrm{IM}} \boldsymbol{G}_j + \Delta t \sum_{j=1}^{s} b_{1j}^{\mathrm{EX}} \boldsymbol{F}_j + \sum_{j=1}^{r} v_{1j} \boldsymbol{u}_j^{[n-1]}, t_n\right)$

end

for $i = 2$ **to** r **do**

(A2.7) $\quad \boldsymbol{u}_i^{[n]} = \Delta t \sum_{j=1}^{s} b_{ij}^{\mathrm{IM}} \boldsymbol{G}_j + \Delta t \sum_{j=1}^{s} b_{ij}^{\mathrm{EX}} \boldsymbol{F}_j + \sum_{j=1}^{r} v_{ij} \boldsymbol{u}_j^{[n-1]}$

end

Algorithm 2: A GLM based PDE time-stepping algorithm

The only subtlety that arises in comparison with the backward/forward Euler example of the previous section is due to the term $\boldsymbol{G}_i$. To formally fit into the framework, a proper definition of $\boldsymbol{G}_i$, denoted as $\bar{\boldsymbol{G}}_i$, should be

$$\bar{\boldsymbol{G}}_i = \boldsymbol{B}^H \left(\boldsymbol{M}^{HH}\right)^{-1} \hat{\boldsymbol{g}}_i^H, \tag{54}$$

with $\hat{\boldsymbol{g}}_i^H$ as defined in Eq. (41). By recombining the terms in the Helmholtz problem associated to step (A2.2) it can be demonstrated that this term can be evaluated as

$$\bar{\boldsymbol{G}}_i = \boldsymbol{\mathcal{P}}\left(\frac{\boldsymbol{U}_i - \boldsymbol{x}_i}{a_{ii}^{\mathrm{IM}}\Delta t}, t_i\right), \tag{55}$$

which corresponds to the L^2 projection of our initial definition of $\boldsymbol{G}_i$, *i.e.* $\bar{\boldsymbol{G}}_i = \boldsymbol{\mathcal{P}}(\boldsymbol{G}_i, t)$ However, do due the properties (38-39) it can be appreciated that using $\boldsymbol{G}_i$ in steps (A2.1), (A2.6) and (A2.7) is equivalent to the use of $\bar{\boldsymbol{G}}_i$, thereby keeping the number of global system inverses per stage to one.

Comparing this PDE time-stepping algorithm with the original ODE algorithm (Algorithm 1), one can identify the following differences:

- While step (A1.2) of the original algorithm essentially is a pure algebraic problem (that is, for schemes with an implicit component), step (A2.2) also as an broader analytical interpretation in the sense that it is the solution of the elliptic Helmholtz partial differential equation.

- It has been indicated before that for purely explicit time-stepping methods (*i.e.* $a_{II}^{\mathrm{IM}} = 0$), the evaluation of step (A1.1) actually vanishes as it is reduced to $\boldsymbol{Y}_i = \boldsymbol{x}_i$. However, in the new algorithm, step (A2.2) now also requires a global system inverse, as the Helmholtz problem is reduced to the projection problem for explicit schemes.

- In addition, step (A2.6) of the new algorithm also requires a L^2 projection. This is necessary to ensure that the solution is in the solution space, as the right-hand-side cannot be guaranteed to be in this space. Note that this requires an additional global system inverse.

- In order to possibly eliminate the cost associated to this additional global system inverse in step (A2.6), we have included an optimisation check in step (A2.5). In case the last row of coefficient matrices A and U is equal to the first row of respectively the matrices B and V, the last stage $\boldsymbol{U}_s$ is equal to the new solution $\boldsymbol{u}_n$ And the (expensive) evaluation of step (A2.6) can be omitted. Fortunately, all multi-step schemes – and many more methods as can be seen from the GLM tableau's (66,68,69,73) – can be formulated to satisfy this condition. As a result, evaluating any linear multi-step method for a single time-step based upon the proposed algorithm only requires a single global system inverse.

  Similar to the optimisation check in step (A2.5), the framework can be equipped with another optimisation feature which checks whether the first stage value $\boldsymbol{U}_1$ is equal to the solution $\boldsymbol{u}_{n-1}$ at the previous time-level, a condition which holds if the first row of the coefficients matrices A and U consists solely of zeros, except for the first entry U which should be $U[1][1] = 1$. Examples include the explicit Runge-Kutta schemes such as *e.g.* given by Eq. (63). For such schemes, steps (A2.1) and (A2.2) can be omitted for $i = 1$, thereby avoiding any unnecessary global system inverses.

In order to evaluate this PDE time-stepping algorithm for an arbitrary general linear method, the framework should be supplied with the following three *external* routines:

- a routine that evaluates the advection term $f(u,t)$ according to the spatial discretisation scheme at the quadrature/collocation points (in order to evaluate step (A.2.3)),
- a routine which solves the projection problem of Section 4.3.1 (in order to evaluate step (A.2.2) in case $a_{ii}^{\mathrm{IM}} = 0$ and step (A.2.6)), and
- a routine that solves the Helmholtz equation of Section 4.3.1 (in order to evaluate step (A.2.2) in case $a_{ii}^{\mathrm{IM}} \neq 0$).

Recall that all three routines should be defined in *physical space*, *i.e.* both input and output arrays correspond to functions evaluated at the quadrature/collocation points.

Remark 1 *Although the framework has been derived by means of the advection-diffusion equation, it is also applicable for other PDEs as well (see also Section 5). The advection term $f(u)$ could in principal also represent a possible reaction term, while the diffusion term $\nabla^2 u$ could be replaced by a more general term $g(u)$ to be treated implicitly. In the latter case, the user should supply the framework with a routine that solves the PDE $u - \lambda g(u) = f$ rather than the Helmholtz equation.*

Remark 2 *Because the algorithm is formulated in physical space, the proposed framework is applicable in case of finite volume or finite difference methods as it is natural to think of these methods in physical space. Is it also possible to verify that the algorithm is valid in case of a Discontinuous Galerkin discretisation (see also Section 5). In this case, the projection operator $\boldsymbol{\mathcal{P}}$ even reduces to the identity operator (because the Dirichlet boundary conditions are weakly imposed).*

4.3.4 Object-oriented implementation

The class structure presented in Section 3.2 that implements the ODE framework can be slightly modified in order to accommodate this PDE time-stepping framework. An overview of the required classes is shown in Figure 2. The underlying idea remains identical and only the class `TimeIntegrationOperators` should altered to take into account the projection operator. Therefore, this class should now be equipped with an additional function pointer `m_project` that points to the implementation of the L^2 projection operator, *i.e.* Eq. (28). In order to set and evaluate this function pointer, the class now also contains the methods `DefineProject` and `DoProject`.

5 Computational Examples

In this section we present examples demonstrating the capabilities of the PDE time-stepping framework presented in the previous section. First we verify the algorithm by considering an advection-diffusion problem and then we apply the framework to two fluid problems: standing waves in shallow water and vortex shedding around a cylinder. In all our examples we use the spectral/*hp* element method [21] for the spatial discretisation. More specifically we will use the modal and hierarchical expansion basis based upon modified Jacobi polynomials, see [21]. If not mentioned otherwise, we use the standard C^0-continuous Galerkin version in the following.

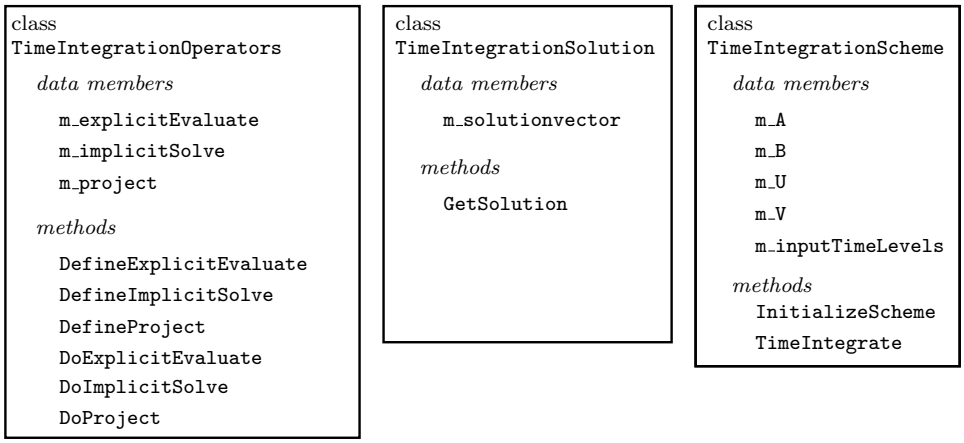


Figure 2: Overview of the classes in the implementation of the generic PDE time-stepping framework.

5.1 Linear advection-diffusion equation

As a first example, we investigate the popular linear advection-diffusion problem of a Gaussian hill convected with a velocity $\boldsymbol{V}$ while spreading isotropically with a diffusivity ν [11, 22]. The analytical solution is given by

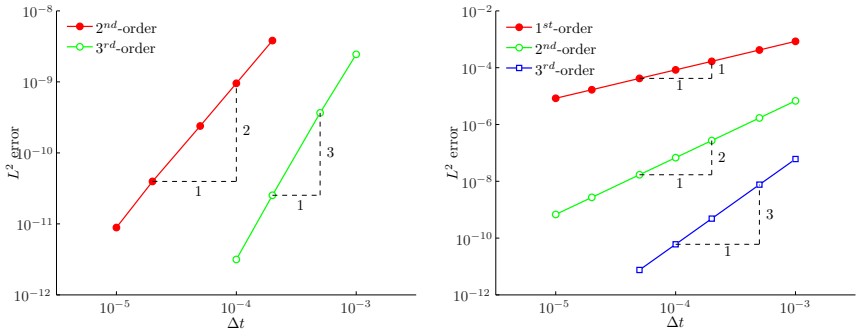
$$u(\boldsymbol{x},t) = \frac{1}{4t+1}\exp\left(-\left(\frac{x-x_0-V_x t}{\nu(4t+1)}\right)-\left(\frac{y-y_0-V_y t}{\nu(4t+1)}\right)\right). \qquad (56)$$

The problem is defined in the domain $\boldsymbol{x} \in [-1\,,1] \times [-1\,,1]$ and is discretised in space on an unstructured triangular mesh of 84 elements using a 12^{th}-order spectral/hp element expansion. The Gaussian hill is initially located a $\boldsymbol{x}_0 = [-0.5\,,-0.5]$ and is convected with a velocity $\boldsymbol{V} = [1\,,1]$ for one time unit and the diffusivity is set to $\nu = 0.05$. Time-dependent Dirichlet boundary conditions given by the analytical solution are imposed on the domain boundaries.

We have applied the time-stepping framework on this equation both for multi-stage and multi-step IMEX time-integration schemes. Therefore, we have supplied the framework with the three necessary external routines as explained in Section 4.3.3, *i.e.* a function that evaluates the linear advection term, a projection operator and a Helmholtz solver. In order to verify that the framework integrates the PDE correctly, we have checked the order of convergence in function of the time-step Δt. Fig. 3(a) confirms the correct convergence rate for the 2^{nd}- and the 3^{rd}-order IMEX-DIRK schemes as respectively presented in [2, 1] (see also Eq. (74)). In Fig. 3(b), we can observe that the L^2 error converges according to the expected rate for the multi-step stiffly stable schemes introduced in Section 5.3 when using the framework.

5.2 Shallow water equations

The shallow water equations (SWE) are frequently used for simulating flows in shallow coastal regions and rivers, for example storm surges, tsunamis and river



(a) Multi-stage IMEX-DIRK schemes (b) Multi-step stiffly stable IMEX schemes

Figure 3: Error convergence in function of Δt for various IMEX schemes when using the PDE time-stepping framework of Section 4.3.

flooding. The SWE can be formulated as

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot \boldsymbol{F}(\boldsymbol{u}) = 0. \tag{57}$$

For an appropriate definition of the flux term $\boldsymbol{F}$ the reader is referred to *e.g.* [13]. Due to their hyperbolic nature, explicit methods are typically employed for time-stepping the SWE. As a result, the user should only provide a proper implementation of the term $\nabla \cdot \boldsymbol{F}(\boldsymbol{u})$ together with a projection method in order to use the framework.

The objective of this example is twofold:

- We would like to demonstrate that the framework can be applied with different spatial discretisation techniques. Therefore, we solve the SWE using both the discontinuous Galerkin (DG) method and the C^0 continuous Galerkin (CG) method. Note that in the former case, the user-supplied function that evaluates the flux term should include the numerical flux term typical to the DG method.

- We also want to compare the computational efficiency of the framework. Therefore, we will compare the run-time of solving the SWE using the framework with a specialised implementation of the chosen time-stepping schemes.

Within the DG community the third-order three-stage Strong-Stability-Preserving (SSP) RK schemes are popular and we use the third-order four-stage SSP-RK scheme [24] for the SWE test-case.

We compute the simple case of two super-positioned standing linear waves with wavelengths L (one wave aligned in the x_1 direction and wave aligned in the x_2 direction) in a basin $\boldsymbol{x} \in [0\,, L] \times [0\,, L]$ of constant depth with slip conditions at the wall boundaries. For the DG method the boundary conditions are implemented weakly through the use of the numerical flux using standard mirroring technique, while for the CG method we apply $u = 0$, $\partial v/\partial n = 0$, $\partial \zeta/\partial n = 0$ at the north/south boundaries and $\partial u/\partial n = 0$, $v = 0$, $\partial \zeta/\partial n = 0$ at the east/west boundaries, respectively.

We are using a 3rd order spectral/hp element method on a mesh of 16 quadrilateral elements and we solve the problem for $t \in [0\,, 100\,T]$ wave where T denotes the wave period. Table 1 present the required time step, the measured run-time and memory usage (based upon the heap profiler Massif of Valgrind's tool suite) for obtaining an error less than 1×10^{-4}. First of all, it can be concluded that the framework has been successfully applied for both the spatial discretisation techniques. We can also see from Table 1 that the run-time overhead of using the framework is in the order of a percent when compared to the scheme-specific implementations while the memory usage is equal. As a result, we can conclude that next to the high-level of generality, the framework also is competitive with scheme-specific implementations in terms of performance.

Table 1: The SWE for standing waves. Computational results for obtaining an L^2 error less than 1×10^{-4}.

		DG method	CG method
	Δt	$T/69$	$T/70$
	L^2 error	9.88E-05	9.73E-06
GLM	Run-time	16.8 s	13.7 s
framework	Storage	15.7 MB	8.0 MB
Specialised	Run-time	16.7 s	13.5 s
implementation	Storage	15.7 MB	8.0 MB

5.3 Incompressible Navier-Stokes equation

As an illustrative example of the use of the time-integration framework to solve more complex fluid dynamics problems, we consider a DNS simulation of the two-dimensional flow past a circular cylinder in a free stream. The incompressible Navier-Stokes (NS) equations are solved using a spectral/hp element discretisation in space combined with the high-order stiffly stable splitting scheme of [20] for the discretisation in time. In this splitting scheme, each time step is subdivided into three substeps, and the solution of the discretised Navier-Stokes equation is advanced from time-step $n-1$ to time-step n as follows:

$$\frac{\breve{\boldsymbol{u}} - \sum_{q=1}^{J_i} \alpha_q \boldsymbol{u}^{n-q}}{\Delta t} = -\sum_{q=1}^{J_e-1} \beta_q \left[(\boldsymbol{u}\cdot\nabla)\,\boldsymbol{u}\right]^{n-q}, \tag{58a}$$

$$\nabla^2 p^n = \nabla\cdot\left(\frac{\breve{\boldsymbol{u}}}{\Delta t}\right), \tag{58b}$$

$$\frac{\gamma_0 \boldsymbol{u}^n - \breve{\boldsymbol{u}}}{\Delta t} = \nu\nabla^2\boldsymbol{u}^n - \nabla p^n. \tag{58c}$$

The splitting scheme decouples the velocity field $\boldsymbol{u}$ from the pressure p, leading to an explicit treatment of the advection term and an implicit treatment of the pressure and the diffusion term. J_i is the integration order for the implicit terms and J_e is the integration order for the explicit terms. The values of the

coefficients γ_0 , α_q and β_q of this multi-step IMEX scheme are given in Table 2 for different orders. In order to use the PDE time-stepping framework of Section

Table 2: Stiffly stable splitting scheme coefficients

	1^{st}-order	2^{nd}-order	3^{rd}-order
γ_0	1	3/2	11/6
α_0	1	2	3
α_1	0	$-1/2$	$-3/2$
α_2	0	0	1/3
β_0	1	2	3
β_1	0	-1	-3
β_2	0	0	1

4, we first formulate the stiffly stable scheme as a general linear method. For the second-order variant for example, this yields

$$\left[\begin{array}{c:c|c} A^{\mathrm{IM}} & A^{\mathrm{EX}} & U \\ \hline B^{\mathrm{IM}} & B^{\mathrm{EX}} & V \end{array}\right] = \left[\begin{array}{c:c|cccc} \frac{2}{3} & 0 & \frac{4}{3} & -\frac{1}{3} & \frac{4}{3} & -\frac{2}{3} \\ \hline \frac{2}{3} & 0 & \frac{4}{3} & -\frac{1}{3} & \frac{4}{3} & -\frac{2}{3} \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right] \quad \text{with} \quad \boldsymbol{y}^{[n]} = \left[\begin{array}{c} \boldsymbol{y}_n \\ \boldsymbol{y}_{n-1} \\ \Delta t \boldsymbol{F}_n \\ \Delta t \boldsymbol{F}_{n-1} \end{array}\right]. \tag{59}$$

where the values in the first two rows have been scaled with γ_0 compared to the values in Table 2. Furthermore, we need to properly define the external functions needed for the time-stepping framework:

- For the explicit term, this should be a function $\boldsymbol{f}$ that evaluates the advection term, *i.e.*

$$\boldsymbol{f}(\boldsymbol{u}) = -\left(\boldsymbol{u} \cdot \nabla\right) \boldsymbol{u}. \tag{60}$$

  Because we will follow a pseudo-spectral approach for the advection term, this term should simply be evaluated at the quadrature/collocation points.

- The projection operator to be provided to the system is identical to the one defined in Section 4.3.1.

- For the implicit part of the scheme, a routine that solves the following problem is required. Given an arbitrary function $\boldsymbol{f}$, a scalar λ and a time-level t, find the velocity field u such that

$$\nabla^2 p = \nabla \cdot (\frac{\boldsymbol{f}}{\lambda}), \tag{61a}$$

$$\boldsymbol{u} - \nu\lambda\nabla^2\boldsymbol{u} = \boldsymbol{f} - \lambda\nabla p, \tag{61b}$$

  and subject to the appropriate boundary conditions. It can be observed that this problem involves the consecutive solution of three elliptic problems: a Poisson problem and two (in 2D) scalar Helmholtz problems. This routine can also be used to solve the *unsteady Stokes* equations.

Figure 4 shows the vorticity field of a flow past a cylinder with a $Re = 100$ after solving the NS equations with the framework presented in Section 4. The solution, which highlights the vortex shedding, has been obtained using the 3^{rd} order stiffly stable splitting scheme with $\Delta t = 0.0001s$ and 7^{th} order spectral/hp expansion on a mesh of 1500 quadrilaterals. The cylinder has a diameter $D = 0.4$ and the domain is defined by a rectangle $\boldsymbol{x} \in [-4\,,16] \times [-5\,,5]$ as shown in Figure 4. Boundary conditions are of Dirichlet type at the inflow, where a constant velocity in x-direction is imposed ($u = 1$ and $v = 0$) and of Neumann type (homogeneous) at the outflow and on the upper and lower domain limits. Similar results have been obtained by different authors with other techniques, *e.g.* [26, 27].

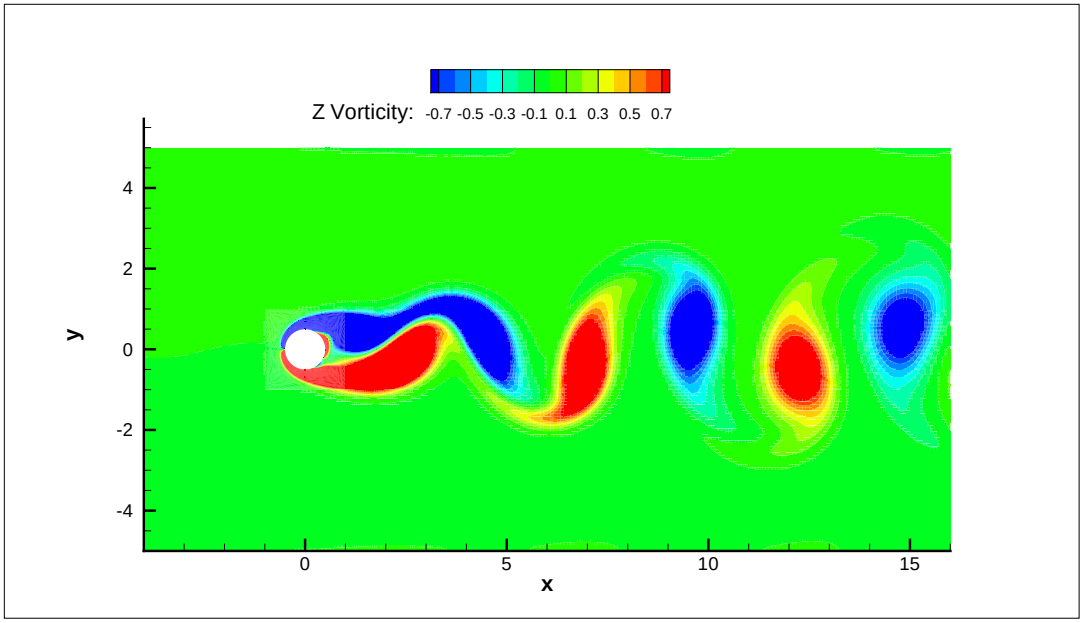


Figure 4: Vorticity field for a flow past a cylinder at $Re = 100$

6 Summary

We proposed a generic framework, both in terms of algorithms and implementations, that facilitates the application of a broad range of time-stepping schemes in a uniform way. We based our algorithm on Butcher's unifying theory of General Linear Methods, a concept widely accepted within the ODE community but little known from a PDE point of view. The framework should allow CFD users, who often tend to limit themselves to a single (family of) schemes, to explore the plethora of different methods that exist – implicit versus explicit, multi-stage versus muli-step – without any additional effort. We illustrated that the abstract character of the framework allows for an object-oriented implementation where switching between different schemes is as simple as changing an input parameter. Although we first presented an generic ODE solving framework, the main emphasis of this work is on time-integrating PDEs. Therefore, we first

showed how IMEX schemes – a family of time-stepping schemes popular within the CFD community – can be formulated as a General Linear Method. Then we demonstrated how through some modifications, the framework can be adapted to accommodate the time-integration of PDEs, characterised by some typical features such as time-dependent boundary conditions, in a generic and computationally efficient way. Overall we believe the paper provides the essential building blocks for time-integrating PDEs in a unified way. Finally, please note that even though we mainly followed a finite element procedure for the spatial discretisation the presented techniques are general and can be used within a finite volume or finite difference context.

Acknowledgment

The authors would like to acknowledge the insightful input of Professor Ray Spiteri of the University of Saskatchewan. SJS would like to acknowledge the support under an EPSRC Advance Research Fellowship, SC would like to acknowledge support from the CardioMath initiative of the Institute of Mathematical Science at Imperial College London, and RMK would like to acknowledge support under the Leverhulme Foundation Trust.

References

[1] U.M. Ascher, S.J. Ruuth, and R.J. Spiteri. Implicit–explicit Runge–Kutta methods for time-dependent partial differential equations. *Appl. Numer. Math.*, 25(2–3):151–167, 1997.

[2] U.M. Ascher, S.J. Ruuth, and B.T.R. Wetton. Implicit-explicit methods for time-dependent partial differential equations. *SIAM J. Numer. Anal.*, 32(3):797–823, 1995.

[3] K. Burrage and J. C. Butcher. Non-linear stability of a general class of differential equation methods. *BIT*, 20:185–203, 1980.

[4] J. C. Butcher. *The Numerical Analysis of Ordinary Differential Equations: Runge-Kutta and General Linear Methods*. Wiley, Chichester, 1987.

[5] J. C. Butcher. General linear methods. *Acta Numerica*, 15:157–256, 2006.

[6] J.C. Butcher. On the convergence of numerical solutions of ordinary differential equations. *Math. Comp.*, pages 1–10, 1966.

[7] J.C. Butcher. Diagonally-implicit multi-stage integration methods. *Appl. Numer. Math.*, 11:347–363, 1993.

[8] J.C. Butcher. An introduction to DIMSIMs. *Math. Appl. Comput.*, 14:59–72, 1995.

[9] J.C. Butcher. An introduction to Almost Runge-Kutta methods. *Appl. Numer. Math.*, 24:331–342, 1997.

[10] J.C. Butcher, P. Chartier, and Z. Jackiewicz. Experiments with a variable-order type 1 DIMSIM code. *Numer. Algorithms*, 22:237–261, 1999.

[11] J. Donea, S. Giuliani, H. Laval, and L. Quartapelle. Time-accurate solution of advection-diffusion problems by finite elements. *Comput. Methods Appl. Mech. Engrg.*, 45(1-3):123–145, 1984.

[12] J. Donelson and E. Hansen. Cyclic composite multistep predictor-corrector methods. *SIAM Journal for Numerical Analysis*, 8(1):137–157, 1971.

[13] C. Eskilsson and S. J. Sherwin. A triangular spectral/hp discontinuous galerkin method for modelling 2d shallow water equations. *Int. J. Numer. Meth. Fluids*, 45:605–623, 2004.

[14] D. Gottlieb and S. A. Orszag. *Numerical analysis of spectral methods: theory and applications*. CBMS-NSF. Society for Industrial and Applied Mathematics, Philadelphia, 1977.

[15] Z. Jackiewicz. Implementation of DIMSIMs for stiff differential equations. *Appl. Numer. Math.*, 42:251–267, 2002.

[16] Z. Jackiewicz. Construction and implementation of general linear methods for ordinary differential equations. A review. *J. Sci. Comp.*, 25:29–49, 2005.

[17] Z. Jackiewicz. *General Linear Methods for Ordinary Differential Equations*. Wiley, 2009.

[18] Z. Jackiewicz and S. Tracogna. A general class of two-step Runge-Kutta methods for ordinary differential equations. *SIAM J. Num. Anal.*, 32:1390–1427, 1995.

[19] A. Kanevsky, M.H. Carpenter, D. Gottlieb, and J.S. Hesthaven. Application of implicit-explicit high order Runge-Kutta methods to discontinuous-galerkin schemes. *J. Comp. Phys.*, 225:1753–1781, 2007.

[20] G. E. Karniadakis, M. Israeli, and S. A. Orszag. High-order splitting methods for the incompressible navier-stokes equations. *J. Comput. Phys.*, 97:414–443, 1991.

[21] G. E. Karniadakis and S. J. Sherwin. *Spectral/hp Element Methods for CFD*. Oxford University Press, second edition edition, 2005.

[22] B. J. Noye and H. H. Tan. Finite difference methods for solving the two-dimensional advection-diffusion equation. *Int. J. Numer. Meth. Fluids*, 9(1):75–89, 1989.

[23] N. Rattenbury. *Almost Runge-Kutta methods for stiff and non-stiff problems*. PhD thesis, The University of Auckland, 2005.

[24] S. J. Ruuth and R. J. Spiteri. High-order strong-stability-preserving Runge-Kutta methods with downwind-biased spatial discretizations. *Siam J. Numer. Anal.*, 42(3):974–996, 2004.

[25] W. E. Schiesser. *The Numerical Method of Lines: Integration of Partial Differential Equations*. Academic Press, San Diego, 1991.

[26] C.C.S. Song and M. Yuan. Simulation of vortex-shedding flow about a circular cylinder at high Reynolds numbers. *J. Fluids Eng.*, 112, 1990.

[27] B. Souza Carmo. *On Wake Interference in the Flow around Two Circular Cylinders: Direct Stability Analysis and Flow-Induced Vibrations.* PhD thesis, Department of Aeronautics, Imperial College London, 2009.

[28] J. vanWieren. Using diagonally implicit multistage integrations methods for solving ordinary differential equations. part 1: Introduction and explicit methods. NAWCWPNS TP 8340, Naval Air Warfare Center Weapons Division, 1997.

[29] J. vanWieren. Using diagonally implicit multistage integrations methods for solving ordinary differential equations. part 2: Implicit methods. NAWCWPNS TP 8356, Naval Air Warfare Center Weapons Division, 1997.

[30] W. Wright. *General Linear Methods with Inherent Runge-Kutta Stability.* PhD thesis, University of Auckland, 2002.

A Coefficients of GLM methods

A.1 Common multi-stage methods

Since multi-stage methods consist only of a single step with many stages, they can be represented as a general linear method with $r = 1$. It is sufficient to write $U = [\ 1\ \ 1\ \cdots\ 1\ ]^\top$, $V = [1]$ and to set the coefficient matrices A and B to the matrix A and the single row $b^\top$ of the corresponding *Butcher tableau* [5] respectively. For example, the classic fourth-order Runge-Kutta method with Butcher tableau

$$\begin{array}{c|c} c & A \\ \hline & b^\top \end{array} = \begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}, \tag{62}$$

has the following GLM representation

$$\left[\begin{array}{cc} A & U \\ B & V \end{array}\right] = \left[\begin{array}{cccc|c} 0 & 0 & 0 & 0 & 1 \\ \frac{1}{2} & 0 & 0 & 0 & 1 \\ 0 & \frac{1}{2} & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ \hline \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} & 1 \end{array}\right]. \tag{63}$$

A.2 Common multi-step methods

In contrast to multi-stage methods, multi-step methods have a single stage, but the solution at the new time-level is computed as a linear combination of information at the r previous time-levels. Linear multi-step methods can be formulated to satisfy the relation

$$\boldsymbol{y}_n = \sum_{i=1}^{r} \alpha_i \boldsymbol{y}_{n-i} + \Delta t \sum_{i=0}^{r} \beta_i \boldsymbol{F}_{n-i}. \tag{64}$$

This corresponds to the general linear method with input and output

$$\boldsymbol{y}^{[n-1]} = \begin{bmatrix} \boldsymbol{y}_{n-1} \\ \boldsymbol{y}_{n-2} \\ \vdots \\ \boldsymbol{y}_{n-r} \\ \hdashline \Delta t \boldsymbol{F}_{n-1} \\ \Delta t \boldsymbol{F}_{n-2} \\ \vdots \\ \Delta t \boldsymbol{F}_{n-r} \end{bmatrix}, \quad \boldsymbol{y}^{[n]} = \begin{bmatrix} \boldsymbol{y}_{n} \\ \boldsymbol{y}_{n-1} \\ \vdots \\ \boldsymbol{y}_{n-r+1} \\ \hdashline \Delta t \boldsymbol{F}_{n} \\ \Delta t \boldsymbol{F}_{n-1} \\ \vdots \\ \Delta t \boldsymbol{F}_{n-r+1} \end{bmatrix}, \tag{65}$$

and the partitioned coefficient matrix

$$\begin{bmatrix} A & U \\ B & V \end{bmatrix} = \left[\begin{array}{c|cccccc:cccc} \beta_0 & \alpha_1 & \alpha_2 & \cdots & \alpha_{r-1} & \alpha_r & \beta_1 & \beta_2 & \cdots & \beta_{r-1} & \beta_r \\ \hline \beta_0 & \alpha_1 & \alpha_2 & \cdots & \alpha_{r-1} & \alpha_r & \beta_1 & \beta_2 & \cdots & \beta_{r-1} & \beta_r \\ 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \hdashline 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 1 & 0 \end{array}\right]. \tag{66}$$

Note that in the vectors and matrices above, the solid lines denote the demarcation of the matrices A, B, U and V whereas the dotted lines merely help to highlight the typical structure of a linear multi-step method. As an example of a multi-step scheme consider the well-known third-order Adams-Bashforth scheme

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \Delta t \left(\frac{23}{12} \boldsymbol{f}(\boldsymbol{y}_{n-1}) - \frac{4}{3} \boldsymbol{f}(\boldsymbol{y}_{n-2}) + \frac{5}{12} \boldsymbol{f}(\boldsymbol{y}_{n-3}) \right), \tag{67}$$

which has the following GLM representation

$$\begin{bmatrix} A & U \\ B & V \end{bmatrix} = \left[\begin{array}{c|cccc} 0 & 1 & \frac{23}{12} & -\frac{4}{3} & \frac{5}{12} \\ \hline 0 & 1 & \frac{23}{12} & -\frac{4}{3} & \frac{5}{12} \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{array}\right]. \tag{68}$$

A.3 Beyond common multi-step or multi-stage methods

The general linear methods framework also encompasses methods that do not fit under the conventional Runge-Kutta or linear multi-step headings. This includes, for example, the cyclic composite method of [12]. In addition, the general linear structure of the GLM in itself gave rise to the development of

new numerical methods. An example of one such class of methods is the class of *Almost Runge-Kutta Methods* [9]. To appreciate its typical combined multi-stage multi-step character consider the following third-order scheme due to [23]:

$$\left[\begin{array}{cc} A & U \\ B & V \end{array}\right] = \left[\begin{array}{ccc|ccc} 0 & 0 & 0 & 1 & \frac{1}{3} & \frac{1}{18} \\ \frac{1}{2} & 0 & 0 & 1 & \frac{1}{6} & \frac{1}{18} \\ 0 & \frac{3}{4} & 0 & 1 & \frac{1}{4} & 0 \\ \hline 0 & \frac{3}{4} & 0 & 1 & \frac{1}{4} & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 3 & -3 & 2 & 0 & -2 & 0 \end{array}\right]. \tag{69}$$

A.4 Common implicit-explicit methods

The first-order Backward Euler/Forward Euler IMEX scheme,

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \Delta t \left(\boldsymbol{g}(\boldsymbol{y}_n) + \boldsymbol{f}(\boldsymbol{y}_{n-1})\right), \tag{70}$$

can be written as a general linear method as

$$\left[\begin{array}{c:c|c} A^{\mathrm{IM}} & A^{\mathrm{EX}} & U \\ \hline B^{\mathrm{IM}} & B^{\mathrm{EX}} & V \end{array}\right] = \left[\begin{array}{c:c|cc} 1 & 0 & 1 & 1 \\ \hline 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{array}\right] \quad \text{with} \quad \boldsymbol{y}^{[n]} = \left[\begin{array}{c} \boldsymbol{y}_n \\ \Delta t \boldsymbol{F}_n \end{array}\right]. \tag{71}$$

The second-order Crank-Nicholson/Adams-Bashforth linear multi-step scheme,

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \Delta t \left(\frac{1}{2}\boldsymbol{g}(\boldsymbol{y}_n) + \frac{1}{2}\boldsymbol{g}(\boldsymbol{y}_{n-1}) + \frac{3}{2}\boldsymbol{f}(\boldsymbol{y}_{n-1}) - \frac{1}{2}\boldsymbol{f}(\boldsymbol{y}_{n-1})\right), \tag{72}$$

can be represented as

$$\left[\begin{array}{c:c|cc} A^{\mathrm{IM}} & A^{\mathrm{EX}} & U \\ \hline B^{\mathrm{IM}} & B^{\mathrm{EX}} & V \end{array}\right] = \left[\begin{array}{c:c|cc} \frac{1}{2} & 0 & 1 & \frac{1}{2} & \frac{3}{2} & -\frac{1}{2} \\ \hline \frac{1}{2} & 0 & 1 & \frac{1}{2} & \frac{3}{2} & -\frac{1}{2} \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right] \quad \text{with} \quad \boldsymbol{y}^{[n]} = \left[\begin{array}{c} \boldsymbol{y}_n \\ \Delta t \boldsymbol{G}_n \\ \Delta t \boldsymbol{F}_n \\ \Delta t \boldsymbol{F}_{n-1} \end{array}\right]. \tag{73}$$

The third-order $(2,3,3)$ IMEX Runge-Kutta scheme (see [1]) is represented by the partitioned coefficient matrix where $\gamma = (3+\sqrt{3})/6$:

$$\left[\begin{array}{c:c|c} A^{\mathrm{IM}} & A^{\mathrm{EX}} & U \\ \hline B^{\mathrm{IM}} & B^{\mathrm{EX}} & V \end{array}\right] = \left[\begin{array}{ccc:ccc|c} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & \gamma & 0 & \gamma & 0 & 0 & 1 \\ 0 & 1-2\gamma & \gamma & \gamma-1 & 2(1-\gamma) & 0 & 1 \\ \hline 0 & \frac{1}{2} & \frac{1}{2} & 0 & \frac{1}{2} & \frac{1}{2} & 1 \end{array}\right]. \tag{74}$$