SWEET - Shallow Water Equation Environment for Tests v1.0

Keerthi Gaddameedi[1], François Hamon[2], Dominik Huber[1], Thibaut Lunet[3], Pedro S. Peixoto[4], João Guilherme Caldas Steinstraesser[4], Martin Schreiber[5,6,1], and Valentina Schüller[7]

[1]Technical University of Munich, Garching, 85748, Germany
[2]TotalEnergies E&P Research & Technology USA, LLC, United States
[3]Chair Computational Mathematics, Institute of Mathematics, Hamburg University of Technology, 21073 Hamburg, Germany
[4]Applied Mathematics, Universidade de Sao Paulo, São Paulo, São Paulo, CEP 05508-090, Brazil
[5]Université Grenoble Alpes / Inria / Laboratoire Jean Kuntzmann / INP / CNRS, Saint-Martin-d'Hères, 38400, France
[6]Inria AIRSEA team, Grenoble, 38058, France
[7]Lund University, Lund, Sweden

Correspondence: João Guilherme Caldas Steinstraesser (joao.steinstraesser@usp.br)

Abstract. SWEET is an open-source software for numerical simulation of differential equations discretized with global spectral methods, both on the bi-periodic plane and the sphere. Although not directly restricted to it, its main focus is on numerical developments for the shallow water equations (SWE) since they play a crucial role in developing new numerical methods for climate and weather simulations.

SWEET's main purpose is to bridge the gap between the development of new time integration methods for atmospheric dynamical cores and high-performance computing. This is done by providing a fast and efficient environment for developing and analyzing time discretization methods for the SWE while reducing spatial errors to a minimum due to the utilization of global spectral methods. In addition, the performance of new time integration methods can be assessed on HPC systems. Regarding the numerics, this is achieved through a versatile implementation, allowing the user to quickly run and combine different time-stepping schemes, and flexibly choose the different terms of the governing equations (i.e., the different physical processes) to be considered in the time integration in a composable way through command line arguments, allowing a rapid exploration of time integration methods. Concerning HPC, SWEET supports various ways to explore parallel-in-time integration methods on large-scale HPC clusters. To analyze the results, SWEET also contains many benchmark tests, including standard test cases relevant to atmospheric modeling research. These features make SWEET a robust and powerful tool for researching temporal schemes for atmospheric circulation models.

This paper summarizes the main features of SWEET and provides some numerical examples illustrating its application.

1 Introduction

As a main component of Earth Systems Models (ESMs), the accuracy and feasibility of numerical simulations for atmospheric circulation models are capital aspects in the context of climate modeling and numerical weather prediction. Continuous research efforts have been made to develop and improve discretization schemes that are able to better represent physical processes within reasonable memory costs and computing times. The evolution of computational technology, with increasing memory and processing capacities along with the advent of parallel computing, constantly opens new possibilities, allowing for more accurate and detailed simulations by enabling finer spatial and temporal discretizations and incorporating additional physical processes (see *e.g.*, (Williamson, 2007; Randall et al., 2018) for reviews on the evolution of global circulation models and ESMs).

The large variety of existing time integration schemes, both for research and operational purposes, illustrates this strongly dynamic development in atmospheric modeling. Examples include explicit and implicit methods and different discretization approaches for different physical processes, *e.g.*, the horizontal and vertical dynamics; also, both Eulerian and semi-Lagrangian paradigms for the spatio-temporal discretization are used. Mengaldo et al. (2018) provides a detailed review of time integration schemes used by climate prediction and numerical weather prediction agencies. Moreover, other classes of time-stepping methods for atmospheric modeling have raised an increasing interest in research works, *e.g.*, exponential (Schreiber and Loft, 2018) and semi-Lagrangian exponential methods (Peixoto and Schreiber, 2019), and spectral deferred correction methods (Hamon et al., 2019). Parallel-in-time methods, such as Parareal (Lions et al., 2001), MGRIT (Friedhoff et al., 2013), and PFASST (Emmett and Minion, 2012), which aim to replace the serial time integration approach by the simultaneous computation of several time steps through an iterative procedure, have also been the subject of increasing research interest.

In this context of dynamic research in atmospheric model simulation, it is of great value to be able to develop, validate, and test new time integration schemes in a fast and straightforward way. This is precisely the primary purpose of the software SWEET (Shallow Water Equation Environment for Tests), which we present in this paper. As stated in its name, the software focuses on shallow water equations (SWE), both on the sphere and on the biperiodic plane. Despite being a two-dimensional, simplified model, the SWE are of great importance in atmospheric modeling research since they facilitate the study and handling of the difficulties and properties of the numerical schemes related to horizontal discretization, which are found in more complex, three-dimensional models used in operational contexts (Williamson et al., 1992). Other simpler models, such as the advection equation and both scalar and vector ordinary differential equations (ODEs), are also available in SWEET. All the available models consider a global spectral discretization in space, using Fourier basis functions (on the plane) or spherical harmonics (on the sphere). The main computational parts of SWEET are coded in C++ for computational performance purposes with benchmark studies on HPC machines, but non-critical parts of the simulation such as pre- and postprocessing scripts, are performed in Python. Other open-source framework are available to prototype and evaluate new numerical schemes for atmospheric model simulation, like FIREDRAKE (Rathgeber et al., 2016), GUSTO (Shipton et al., 2020) or DEDALUS (Burns et al., 2020), but they focus primarily on space discretization and modeling rather than investigating time-integration schemes, which is the main focus of SWEET.

The key feature of SWEET is the possibility of easily defining different time-stepping schemes for the dynamical core and how to use them for solving the equations of interest. Through simple command-line arguments, it is possible to define a scheme by composing building blocks, which correspond to elementary time integration methods; therefore, the user is able to choose which terms of the dynamical core (*i.e.*, which physical processes) will be numerically integrated, and which method will be used for integrating each of them. This approach provides a very versatile and powerful tool for studies around numerical methods for atmospheric circulation. Therefore, in the context of model hierarchies for atmospheric circulation as defined by

(Maher et al., 2019), although the SWE are a model of intermediate complexity in the dynamical hierarchy, SWEET allows it to be studied as a relatively complex model in the process hierarchy. Indeed, different atmospheric processes can be isolated, providing a better understanding on their behavior and on how they are affected by discretization choices.

Several works using SWEET have already been published. Hamon et al. (2019) and Peixoto and Schreiber (2019) studied, respectively, multi-level spectral deferred correction (MLSDC) for the SWE on the rotating sphere and semi-Lagrangian exponential methods for the SWE on the rotating plane. In the context of parallel-in-time methods, Schreiber et al. (2017); Schreiber and Loft (2018) studied rational approximation exponential integrators (REXI) for the linear SWE, Schmitt et al. (2018) studied a semi-Lagrangian Parareal method for the viscous Burgers equation, and Schreiber et al. (2019), Hamon et al. (2020), and Caldas Steinstraesser et al. (2024) studied, respectively REXI, PFASST and Parareal/MGRIT for the nonlinear SWE on the rotating sphere. Finally, Raphaldini et al. (2022) studied precession resonance phenomena in the barotropic vorticity equation. The present article provides the first detailed overview of SWEET, which may contribute to its utilization in further research.

This article is organized as follows: in Section 2, we present an overview of the governing equations and benchmark tests implemented in SWEET, focusing on the SWE on the rotating sphere; Section 3 describes the spatial discretization and parallelization; Section 4 is devoted to a detailed description of the temporal discretization and the composability of time integration schemes; an overview of the usage of SWEET, including compilation, execution and pre-and postprocessing, is presented in Section 5; numerical examples exploring some capabilities of SWEET are presented in Section 6; and a discussion and conclusions are drawn in Section 7.

2 Governing equations and benchmark tests

2.1 Governing equations

SWEET supports the simulation of various PDEs using global spectral methods. Its primary focus, in the context of atmospheric circulation modeling, is on the shallow-water equations on the rotating sphere, which read

$$\frac{\partial \boldsymbol{U}}{\partial t} = L_G(\boldsymbol{U}) + L_C(\boldsymbol{U}) + N_A(\boldsymbol{U}) + N_R(\boldsymbol{U}) + L_\nu(\boldsymbol{U}) + \boldsymbol{b}. \tag{1}$$

The unknown $\boldsymbol{U} = \boldsymbol{U}(t,\lambda,\theta) := (\xi,\delta,\Phi)$ is a function of time t and the longitude-latitude coordinates (λ,θ). It is described in terms of the vorticity $\xi := \boldsymbol{k}\cdot(\nabla\times\boldsymbol{u})$, the divergence $\delta := \nabla\cdot\boldsymbol{u}$ and the geopotential $\Phi = \Phi_0 + \Phi' = gh$. Therein, $\boldsymbol{u} = (u,v)$ is the horizontal velocity vector, $\boldsymbol{k}$ is the unit vector in the vertical direction, Φ_0 and Φ' are respectively the mean geopotential and the geopotential perturbation, h is the fluid height and g is the gravitation. The individual terms in the right-hand side of (1) read

$$L_G(\boldsymbol{U}) := \begin{pmatrix} 0 \\ -\nabla^2\Phi \\ -\Phi_0\delta \end{pmatrix}, \qquad L_C(\boldsymbol{U}) := \begin{pmatrix} -\nabla\cdot(f\boldsymbol{u}) \\ \boldsymbol{k}\nabla\times(f\boldsymbol{u}) \\ 0 \end{pmatrix}, \qquad L_\nu(\boldsymbol{U}) := (-1)^{\frac{q}{2}+1}\nu \begin{pmatrix} \nabla^q\xi \\ \nabla^q\delta \\ \nabla^q\Phi' \end{pmatrix}$$

$$N_A(\boldsymbol{U}) := \begin{pmatrix} -\nabla\cdot(\xi\boldsymbol{u}) \\ -\nabla^2\left(\frac{\boldsymbol{u}\cdot\boldsymbol{u}}{2}\right) + \boldsymbol{k}\cdot\nabla\times(\xi\boldsymbol{u}) \\ -\boldsymbol{u}\cdot\nabla\Phi \end{pmatrix}, \qquad N_R(\boldsymbol{U}) := \begin{pmatrix} 0 \\ 0 \\ -\Phi'\delta \end{pmatrix}, \qquad \boldsymbol{b} := \begin{pmatrix} 0 \\ 0 \\ -g\nabla^2 b \end{pmatrix}$$

where $f = 2\Omega\sin(\theta)$ is the Coriolis coefficient, Ω is the Earth's rotation's angular velocity and $b = b(\lambda,\theta)$ is the bathymetry profile. The linear terms L_G, L_C and L_ν account respectively for phenomena related to gravity, the Earth's rotation and (hyper-)viscosity (of order q even and coefficient $\nu \geq 0$), while the nonlinear terms N_A accounts for the advection and N_R contains the rest of the nonlinear terms; moreover, SWEET also takes into account the term $\boldsymbol{b}$ for the bathymetry (topography) effects the SWE on the sphere.

Despite being a two-dimensional model, the SWE are widely used in the early stages of atmospheric modeling research, since they enable meaningful studies and insights related to the horizontal discretization of more complex, three-dimensional models. In addition to these equations, SWEET also supports the simulation of:

- the SWE on the biperiodic plane, considering an $f-$plane approximation, which facilitates insightful studies on geophysical dynamics in a simplified framework;
- linear and nonlinear advection equations, providing the first stepping stone for time integration of hyperbolic problems;
- the Dahlquist ODE, or Dahlquist test equation ($y' = \lambda y$), providing initial and fast development and debugging for time-stepping schemes, which will be used in more complex equations.

We highlight that implementing new governing equations in SWEET is a relatively straightforward task, due to the adopted code architecture. In particular, the composable time stepping approach, combined with the exact evaluation of spatial derivatives resulting from the spectral discretization (see Sections 3 and 4), allows for an easy definition of time integration schemes for individual terms of the governing equations, each one defined in a C++ class. In addition to this, each set of equations also comprises classes representing data containers, benchmark tests, etc., and implementing new equations requires mostly an adaptation of existing classes defined for the same geometry. For example, the 2D Navier-Stokes on the sphere could be easily implemented based on the SWE on the sphere.

2.2 Benchmark tests

A large variety of benchmark test cases are implemented in SWEET, which can be easily chosen through a command-line argument. This also includes the Dahlquist equation, mainly for debugging purposes. This allows us to first develop and assess the quality of time integration methods in this simplified environment. Once the time integration methods are tested, they can be directly 1:1 applied to different kinds of PDEs in SWEET and, if possible, also combined with other time integrators without additional programming efforts.

For the SWE, they notably include most of the tests proposed by Williamson et al. (1992) and the unstable jet test proposed by Galewsky et al. (2004), which are standard configurations for the research and development of numerical schemes for the SWE on the rotating sphere. Therefore, SWEET can be used to conduct numerical studies based on standardized benchmarks in the context of atmospheric circulation modeling. Analytical solutions are also available for the linear SWE on the plane, which is of great importance for testing time integration schemes before applying them *e.g.*, for the SWE on the rotating sphere.

3 Space discretization

SWEET strictly follows a software design targeting a clear separation of space and time discretization. This section describes the different spatial-related components of SWEET. These components are broken down into fundamental building blocks that can then be reused in a highly modular way to be reutilized by time integration methods.

3.1 Cartesian and spherical data in grid and spectral space

In SWEET, the state variables are either represented in grid or spectral space. In spectral space, global spectral methods are used, with two-dimensional Fourier transforms on the plane and spherical harmonics (Fourier-Legendre) transforms on the sphere. These transforms are performed respectively by the efficient FFTW (Frigo and Johnson, 1998) and SHTns libraries (Schaeffer, 2013). On the sphere, a triangular truncation of the spectral coefficients is adopted – which corresponds to truncation at a given length-scale – with spectral resolution M.

To provide access to the grid or spectral data, 2D Cartesian and 2D spherical data storage are available; in both cases, both grid and spectral representation are available, including support of complex-valued data in grid and spectral space, which is necessitated by the REXI time integration methods. Various operations are provided directly based on these data storage: conversion between grid and spectral space, (linear) vector space operations by operator overloading (e.g., scaling by a factor, adding two vectors), reduction operations for norm computations, etc.

Operators: In addition to these basic operations on the data containers, additional features are required for explicitly and implicitly time-integrating linear PDEs.

Explicitly evaluated spatial differential operators: SWEET uses spectral methods since they allow us to evaluate spatial derivatives exactly, up to the truncation of the spectral expansion, since Fourier basis functions and spherical harmonics are eigenfunctions of spatial derivatives on the plane and spherical Laplacian, respectively. Such operations in spectral space are provided by a dedicated class to separate the data storage itself from the operations applied to it. In addition, operations might have additional preprocessing requirements (e.g., computing particular identity coefficients for spherical harmonics).

For explicit evaluations, computing derivatives in various forms (along a single dimension, Laplace operator, etc.) is provided as a member function of the operator class. Applying a derivative to a spectral data container returns a new data container with the new spectral data. For spherical harmonics, special operators computing the transformation from velocity (in grid space) to vorticity-divergence (in spectral space) and vice versa are also made available.

Implicit solvers for linear equations: SWEET also supports the implicit evaluation of linear terms of the governing equations, through the solution of Helmholtz equations arising from the exact computation of derivatives using the spectral approach. The implicit evaluation of linear terms through the solution of pentadiagonal systems arising in semi-implicit discretizations of the shallow water equations (Temperton, 1997) is also supported by SWEET.

Nonlinear evaluations: In the case of nonlinear equations, the nonlinear terms are computed explicitly using a pseudo-spectral approach, *i.e.*, with the nonlinear product being computed in the physical grid and then transformed back to the spectral space. The resolution of the physical grid is determined by the 3/2-antialiasing approach for quadratic non-linear terms (Durran, 2010).

Vector data: As described in Section 4, SWEET supports spatio-temporal discretization using a Semi-Lagrangian scheme, which requires an iterative estimation of departure points of the Lagrangian trajectories arriving at each Eulerian grid point in each time step. This can be easily handled by storing the points' coordinates in one-dimensional vector structures available in SWEET, accompanied by overloaded operators allowing them to perform operations with spatial coordinates. Similar to the previously described data containers for spatial simulation data, the vector data containers also support all variants of linear (vector) space operations.

Interpolation methods: The Semi-Lagrangian discretization also requires interpolation of the solution of the departure points of the Lagrangian trajectories, which, in general, will not lie on the fixed Eulerian grid point. SWEET provides support for bilinear and bicubic Lagrange interpolation procedures, which are used respectively in the iterative process to estimate the departure points and to obtain the solution at the final approximated departure points; this combination leads to second-order schemes.

3.2 Parallelization

Most of the operations with Cartesian and Spectral data storage are parallelized using OpenMP in a similar fashion as it is the case for operational codes. For the spherical harmonic transform with the SHTns library, we use a parallelization with OpenMP, which allows an exploration of parallelization with SWEET using either the nested parallelism feature of OpenMP, or by calling SHTns from a single thread only. SHTns proved to run highly efficiently on shared memory systems, thanks to its on-the-fly algorithm that trades some memory accesses with efficient vectorized computations (Schaeffer, 2013). Extensions to distributed-memory systems for spatial parallelization are currently the only limiting factor of SWEET with respect to parallelization.

4 Time discretization

The main objective of SWEET is to test, benchmark, and prototype time integration schemes for atmospheric circulation. More specifically, in order to provide a versatile and robust tool for this purpose, the following requirements are met by SWEET:

1. Support treating different terms in the governing equations differently (e.g., linear and nonlinear, stiff and nonstiff terms);
2. Support mix-and-matching time integrators;

3. Support for implementations in SWEET itself, but also be able to interface with external implementations of time integration methods;
4. Support for novel time integration approaches, e.g., exponential and parallel-in-time integration schemes.
5. Support for using different time integration methods in the same execution (e.g., different for each multi-level).

In the following paragraphs, we describe in detail how this versatile temporal discretization is achieved in SWEET. We start by presenting the general template for defining time integration schemes through the composition of building blocks; then, we present a list of the available building blocks and we discuss how different terms of the governing equations can be solved by different blocks. A few examples of composed methods is presented to provide a glimpse on the flexibility and possibilities provided by SWEET, and we also discuss the limitations of the proposed approach. Finally we briefly present the parallel-in-time methods implemented in SWEET, which do not follow the same composability approach but still rely on it, resulting in a very versatile implementation.

4.1 Overview

In SWEET, the time integration based on building blocks can be seen as a tree structure. The interior nodes are elementary time stepping schemes, and the leaf nodes are the terms of the governing equations. The interior and leaf nodes can be composed if they provide matching interfaces, *i.e.*, the time integration method can be used to integrate the given term. Moreover, some of the interior nodes, such as splitting ones, can or need to be interfaced with other interior nodes, resulting in a hierarchical composition.

In order to run a simulation with a user-defined time stepping scheme, a command-line option in the form

```
--timestepping-method=METHOD(...)
```

must be provided. `METHOD` refers to one of the several time integration building blocks available in SWEET, as listed in Section 4.3. The arguments include the accuracy order and the terms of the governing equations to be integrated by `METHOD`; it is possible to integrate the full equation or only part of it, which allows a straightforward simulation of modified equations, since specific terms can be easily neglected or added. Moreover, some of the time integration schemes require specific arguments, *e.g.*, allowing to choose variants of the method, or the interpolation order in the case of semi-Lagrangian schemes, or even another interior node, as mentioned above. Illustrative examples are presented in Section 4.4. We emphasize that this composition is made exclusively via command line, through the argument `--timestepping-method`; therefore, it requires no further programming and compilation. Figure 1 presents an overview of the tree structure of the time integration composition; for the sake of readability, only some of the leaf and interior nodes are shown. As described in detail below, we highlight that some interior nodes have an interface with other interior nodes (*e.g.*, the Strang-splitting (SS) node) and some leaf nodes have leaf nodes (*e.g.*, the ADDT node), which allows a versatile setup of different time stepping schemes and different governing equations.

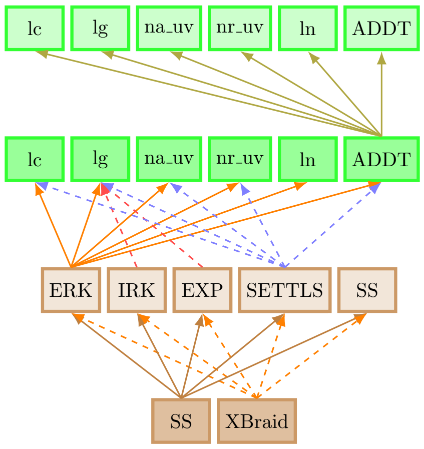


Figure 1. Overview of the tree structure of the time integration composition for the SWE on the rotating sphere; for the sake of readability, only some of the leaf and interior nodes are shown. Arrows indicate that a matching interface exists between nodes. From bottom to top, the first row contains interior nodes with interface to other interior nodes; the second row indicates interior nodes with interface to leaf nodes; and the two top rows indicate leaf nodes.

4.2 Terms of the governing equations (leaf nodes)

One of the arguments to be given to the building blocks is the term or terms of the governing equations to be solved by them. The available leaf nodes depend on the specific PDE or ODE, and they are defined based on the different physical processes they model. For instance, in the case of the SWE on the rotating sphere (Eq. (1)), L_G, L_C, N_A, N_R, L_ν and $\boldsymbol{b}$ are identified respectively by `lg`,`lc`, `na`, `nr`, `visc` and `b`. We also define the terms `l` and `n`, corresponding respectively to $L := L_G + L_C + \boldsymbol{b}$ and $N := N_A + N_R$, and `ln` corresponding to $L + N$. A null term (`null`) is also available, with no tendencies being computed. Moreover, it is possible to combine terms using the operators `ADDT` and `NEGT`, which respectively add and subtract terms, allowing a given time integration building block to solve more than an individual term. Finally, in the case of the SWE on the rotating sphere, two variants of the nonlinear terms are available, namely (`na_uv`,`nr_uv`) and (`na_vd`,`nr_vd`), in which the advected quantities are respectively the velocity field (u, v) or the vorticity and divergence (ξ, δ). The default option is `na` = `na_uv` and `nr` = `nr_uv`.

4.3 Building blocks (interior nodes)

The time integration schemes listed below are available in SWEET. We provide here only a very brief description, and we refer the reader to the respective references for further details. While the more classical schemes, such as Runge-Kutta methods, are simply listed, a brief explanation is given to schemes with which not all readers may be familiar.

Most of the methods are used for the integration of equations in the form

$$\frac{d\boldsymbol{u}}{dt} = f(\boldsymbol{u}) \tag{2}$$

which, in the case of PDEs, is obtained after their spatial discretization. In some cases, it is assumed that $f(\boldsymbol{u}) = L\boldsymbol{u}$ is linear, or $f(\boldsymbol{u}) = L\boldsymbol{u} + N(\boldsymbol{u})$ is given by the sum of a linear and a nonlinear term. In the case of semi-Lagrangian schemes, we solve, instead of (2), the PDE

$$\frac{D\boldsymbol{u}}{Dt} = f(\boldsymbol{u}) \tag{3}$$

where $D/Dt = \partial/\partial t - N_A(\cdot)$ is the material derivative, N_A contains advective terms, and f may be of the form $f(\boldsymbol{u}) = L\boldsymbol{u} + \tilde{N}(\boldsymbol{u})$, where $\tilde{N} = N - N_A$.

4.3.1 Explicit Runge-Kutta (ERK) and Implicit Runge-Kutta (IRK)

Explicit Runge-Kutta methods of order 1 to 4 for the integration of (2) with a general f are available in SWEET. In the case of the second-order scheme, different variants are available (midpoint, Heun, Ralston) through options to the method. For more information, we like to refer to Durran (2010).

With respect to implicit Runge-Kutta, assuming that f is linear, the backward Euler (first-order) and Crank-Nicolson (second-order) methods are available. For PDEs, the linear system of equations is solved directly in spectral space with solver methods provided as operators by SWEET. For more information, we also refer to Durran (2010).

We emphasize that implicit time integration is only available for cases where spectral solvers are also provided for the 1st order backward time step. Nevertheless, this does not exclude higher-order implicit solvers per se, see upcoming Section 4.3.7.

4.3.2 Exponential (EXP)

The exact integration of (2), in the case where f is linear, is given by the exponential method

$$\boldsymbol{u}_{n+1} = e^{\Delta t L}\boldsymbol{u}_n \tag{4}$$

In the case where (2) is a vector equation, it is assumed that the operator L is diagonalizable, such that the matrix exponential $e^{\Delta t L}$ can be indeed computed exactly, down to machine precision, by evaluating the exponential of the eigenvalues of L (Moler and Loan, 2003). Therefore, this time stepping scheme is available for $L_G + L_C$ and L_ν in the case of the SWE on the plane, but only for L_G and L_ν in the case of the SWE on the rotating sphere, since L_C is spatially-dependent (through the Coriolis parameter) and is not diagonalizable.

4.3.3 Rational exponential integrator (REXI)

An approach for computing matrix exponentials, without diagonalizability assumptions, is the Rational Exponential Integrations (REXI), proposed by Damle et al. (2013), which approximates exponential functions by a sum of rational terms:

$$e^{\Delta t L} \approx \gamma + \sum_{n=1}^{N} \beta_n (\Delta t L + \alpha_n)^{-1} \tag{5}$$

The coefficients γ, α_n and β_n can be computed with different methods, with an overview given in Schreiber and Brown (2025). Also note that the terms in the summation (5) are independent, such that REXI leads to a parallel time-integration scheme, where we like to refer the reader to (Schreiber et al., 2019) for more details.

With respect to computing the REXI coefficients, SWEET switched from computing them within SWEET to computing them as part of the preprocessing in Python and reading them into SWEET from a file (see Section 5). This is due to the increased simplicity in precomputing and analyzing them in Python.

4.3.4 Exponential Time-Differencing Runge-Kutta (ETDRK)

Proposed by Cox and Matthews (2002), this class of methods integrates (2) with f given by $f(\boldsymbol{u}) = L\boldsymbol{u} + N(\boldsymbol{u})$. Starting from the variation-of-constants formula,

$$\boldsymbol{u}(t_{n+1}) = e^{\Delta t L}\boldsymbol{u}(t_n) + \int_{t_n}^{t_{n+1}} e^{-(s-t_n)L} N(\boldsymbol{u}(s))ds \tag{6}$$

the linear term is computed exactly (or at least very accurately), whereas Runge-Kutta-type approximations are proposed to compute the integral. ETDRK methods of orders 1, 2, and 4 are available in SWEET. The first-order method, for instance, reads

$$\boldsymbol{u}_{n+1} = \varphi_0(\Delta t L)\boldsymbol{u}_n + \Delta t \varphi_1(\Delta t L) N(\boldsymbol{u}_n), \qquad \varphi_0(z) = e^z, \qquad \varphi_1(z) = z^{-1}(\varphi_0(z) - \varphi_0(0)). \tag{7}$$

One of the arguments received by ETDRK is the method for computing the matrix exponential (EXP or REXI), which should be chosen depending on L (as explained in Section 4.3.2, if L is diagonalizable, then the exact matrix exponentials can be computed using EXP).

4.3.5 Semi-Lagrangian Semi-Implicit Stable Extrapolation Two-Time Level Scheme (SETTLS)

Semi-Lagrangian schemes solve the PDE (3), with $f(\boldsymbol{u}) = L\boldsymbol{u} + \tilde{N}(\boldsymbol{u})$ by considering the Lagrangian framework at each time step. Instead of relying on fixed spatial grids as in Eulerian schemes or tracking the particles' trajectories along the temporal domain as in Lagrangian ones, semi-Lagrangian methods estimate the trajectories $(t, \boldsymbol{x}(t))$ arriving at each point $\boldsymbol{x}_j$ of the fixed Eulerian grid at each time t_{n+1}; then, a spatial interpolation provides an approximate solution at the departure points $\boldsymbol{x}_d$ of the trajectories at time t_n. In the Semi-Lagrangian Semi-Implicit Stable Extrapolation Two-Time Level Scheme (SETTLS), the departure points are estimated iteratively using the SETTLS proposed by Hortal (2002), which estimates the velocity field at $t_n + \Delta t/2$ through linear extrapolation and interpolation:

$$\boldsymbol{x}_d^k = \boldsymbol{x}_j - \frac{\Delta t}{2}\left[2\boldsymbol{v}(t_n, \boldsymbol{x}_d^k) - \boldsymbol{v}(t_{n-1}, \boldsymbol{x}_d^k) + \boldsymbol{v}(t_n, \boldsymbol{x}_j)\right] \tag{8}$$

where $\boldsymbol{v}$ is the velocity field, and the solution is updated with a similar extrapolation-interpolation of the nonlinear term and an implicit (Crank-Nicolson) integration of the linear one:

$$\frac{\boldsymbol{u}_{n+1} - \boldsymbol{u}_n^*}{\Delta t} = \frac{1}{2}\left[L\boldsymbol{u}_{n+1} + (L\boldsymbol{u}_n)_n^*\right] + \frac{1}{2}\left[\left(2\tilde{N}(\boldsymbol{u}_n) - \tilde{N}(\boldsymbol{u}_{n-1})\right)_n^* + \tilde{N}(\boldsymbol{u}_n)\right]. \tag{9}$$

Here, the asterisks indicate quantities interpolated to $\boldsymbol{x}_d$. The resulting scheme, which is used *e.g.*, in the Integrated Forecast System of the European Centre for Medium-Range Weather Forecast (IFS-ECMWF), results in a first- or second-order scheme when the interpolation to $\boldsymbol{x}_d$ is linear or cubic, respectively. In SWEET, the building block SETTLS receives the argument `sl_order` to choose the order of accuracy. The trajectory estimation and spatial interpolation are supported by the vector data structure and interpolation functions provided by SWEET, as described in Section 3.

This time integration method can be implemented in different ways, where the default one follows the implementation in the operational code of ECMWF. Other implementations are made available via arguments to this method.

4.3.6 Semi-Lagrangian Exponential Time-Differencing Runge-Kutta (SLETDRK)

Through a derivation in the Lagrangian framework, Peixoto and Schreiber (2019) proposed a semi-Lagrangian variant of the ETDRK schemes, which were further improved to second-order by Caldas Steinstraesser et al. (2025). The first-order scheme, for instance, reads

$$\boldsymbol{u}_{n+1} = \varphi_0(\Delta t L)\left[\boldsymbol{u}_n + \Delta t \psi_1(\Delta t L)\tilde{N}(\boldsymbol{u}_n)\right]_n^*, \qquad \psi_1(z) = \varphi_1(-z) \tag{10}$$

with φ_i defined as in ETDRK. In SWEET, the first- and second-order SL-ETDRK schemes are available, and the user must define the building block to evaluate the matrix exponentials and the accuracy order of the semi-Lagrangian step.

4.3.7 Spectral Deferred Correction (SDC)

The Spectral Deferred Correction (SDC) method, introduced by Dutt et al. (2000), is a numerical integrator of differential equations that provides higher-order accuracy by iteratively computing corrections based on first-order integrators. It decomposes the time interval of a time step $[t_n, t_{n+1}]$ into M quadrature nodes $t_n \le \tau_1 \le \dots \le \tau_M \le t_{n+1}$ on which the solution $\boldsymbol{u}$ of (2) is approximated first using a low-order method (**pre-sweep**). Then, an iterative process (**sweep**) corrects $\boldsymbol{u}$ on each node to a k-th order accurate solution, up to a given number of iterations K, after which a closure method (**post-sweep**) is used to get the solution of (2) at t_{n+1} from the quadrature node solutions.

Integrator-based SDC: Rather than discretizing first with explicit and/or implicit methods, we start with the SDC in its most generic "integrator" form [1], which can be written as

$$\forall m \in \{0 \dots M-1\}, \quad \boldsymbol{u}_{m+1}^{k+1} = \boldsymbol{u}_m^{k+1} + I_m^{m+1}\left[f(\boldsymbol{u}^{k+1})\right] - I_m^{m+1}\left[f(\boldsymbol{u}^k)\right] + \sum_{i=1}^{M} s_{m,i} f(\boldsymbol{u}_i^k), \quad \boldsymbol{u}_0^k = \boldsymbol{u}(t_n),\ \tau_0 = t_n, \tag{11}$$

where $I_m^{m+1}\left[f(\boldsymbol{u}^k)\right]$ can be any time integrator (interior node) that integrates the solution within $[\tau_m, \tau_{m+1}]$ using $\boldsymbol{u}_m^k$ as initial solution, and $s_{m,i}$ are node-to-node quadrature weights computed by integrating the Lagrange polynomial $(l_i)_{1\le i\le M}$ defined by $(\tau_m)_{1\le m\le M}$:

$$s_{m,i} = \int_{\tau_m}^{\tau_{m+1}} l_i(s)ds. \tag{12}$$

[1] https://www.martin-schreiber.info/pub/papers_permalinks/spectral_deferred_corrections_with_less_pain_ver_2024_05_21.pdf

For instance, using respectively a Backward/Forward Euler time integrator leads to the following sweep update formula:

$$\boldsymbol{u}_{m+1}^{k+1} = \boldsymbol{u}_m^{k+1} + (\tau_{m+1} - \tau_m)\left[f(\boldsymbol{u}_{m+1}^{k+1}) - f(\boldsymbol{u}_{m+1}^{k})\right] + \sum_{i=1}^{M} s_{m,i} f(\boldsymbol{u}_i^k) \quad \text{(implicit)} \tag{13}$$

$$\boldsymbol{u}_{m+1}^{k+1} = \boldsymbol{u}_m^{k+1} + (\tau_{m+1} - \tau_m)\left[f(\boldsymbol{u}_{m}^{k+1}) - f(\boldsymbol{u}_{m}^{k})\right] + \sum_{i=1}^{M} s_{m,i} f(\boldsymbol{u}_i^k) \quad \text{(explicit)} \tag{14}$$

If we decompose the right hand side of (2) into an implicit and explicit part $f = f_I + f_E$, then we can combine the two previous formula, as suggested by Minion (2003), to form an IMEX sweep update:

$$\boldsymbol{u}_{m+1}^{k+1} = \boldsymbol{u}_m^{k+1} + (\tau_{m+1} - \tau_m)\left[f_I(\boldsymbol{u}_{m+1}^{k+1}) - f_I(\boldsymbol{u}_{m+1}^{k})\right] + (\tau_{m+1} - \tau_m)\left[f_E(\boldsymbol{u}_{m}^{k+1}) - f_E(\boldsymbol{u}_{m}^{k})\right] + \sum_{i=1}^{M} s_{m,i} f(\boldsymbol{u}_i^k) \tag{15}$$

Fixed-Point-based SDC : as suggested by Huang et al. (2006), we can interpret SDC in a different way by looking at the collocation problem related to (2):

$$\boldsymbol{u}(t_{n+1}) = \boldsymbol{u}(t_n) + \int_{t_n}^{t_{n+1}} f(\boldsymbol{u}), \tag{16}$$

and writing it on each quadrature node:

$$\forall m \in \{1 \dots M\}, \quad \boldsymbol{u}_m = \boldsymbol{u}_0 + \int_{t_n}^{t_{n+1}} f(\boldsymbol{u}), \quad \boldsymbol{u}_0 := \boldsymbol{u}(t_n). \tag{17}$$

Using the quadrature weights $q_{m,i} := \int_{t_n}^{\tau_m} l_i(s)ds$, we can write this as an all-at-once system:

$$U = U_0 + \boldsymbol{Q} f(U) \Leftrightarrow U - \boldsymbol{Q} f(U) = U_0, \quad \boldsymbol{Q} = (q_{m,i}), \tag{18}$$

where $U = [\boldsymbol{u}_1, \dots, \boldsymbol{u}_M]$ and $U_0 = [\boldsymbol{u}_0, \dots, \boldsymbol{u}_0]$. Then, SDC is simply a preconditioned fixed-point iteration solving (18):

$$[I - \boldsymbol{Q}_\Delta f](U^{k+1} - U^k) = U_0 - U^k + \boldsymbol{Q} f(U^k), \tag{19}$$

where I is the identity operator and $\boldsymbol{Q}_\Delta$ is a lower triangular approximation of $\boldsymbol{Q}$. This can be rearranged into a zero-to-node sweep update:

$$U^{k+1} = U_0 + \boldsymbol{Q}_\Delta\left[f(U^{k+1}) - f(U^k)\right] + \boldsymbol{Q} f(U^k). \tag{20}$$

Since $\boldsymbol{Q}_\Delta$ is lower triangular, the all-at-once sweep update can be written for each node as:

$$\forall m \in \{1 \dots M\}, \quad \boldsymbol{u}_m^{k+1} = \boldsymbol{u}_0 + \sum_{i=1}^{m} q_{i,m}^{\Delta}\left[f(\boldsymbol{u}_i^{k+1}) - f(\boldsymbol{u}_i^k)\right] + \sum_{i=1}^{M} q_{m,i} f(\boldsymbol{u}_i^k). \tag{21}$$

Finally, if we define a strictly lower triangular $\boldsymbol{Q}_{\Delta,E}$ for an explicit part and a lower triangular $\boldsymbol{Q}_{\Delta,I}$ for an implicit part, then we can write the IMEX SDC zero-to-node sweep update:

$$\forall m \in \{1 \dots M\}, \quad \boldsymbol{u}_m^{k+1} = \boldsymbol{u}_0 + \sum_{i=1}^{m} q_{i,m}^{\Delta,I}\left[f_I(\boldsymbol{u}_i^{k+1}) - f_I(\boldsymbol{u}_i^k)\right] + \sum_{i=1}^{m-1} q_{i,m}^{\Delta,E}\left[f_E(\boldsymbol{u}_i^{k+1}) - f_E(\boldsymbol{u}_i^k)\right] + \sum_{i=1}^{M} q_{m,i} f(\boldsymbol{u}_i^k). \tag{22}$$

In particular, we can build the $\boldsymbol{Q}_\Delta$ matrices to follow a Backward/Forward Euler scheme to retrieve the same update as (15). All matrices used in SDC are precomputed in the Python-based preprocessing script, making use of advanced Python libraries. The matrices are read into SWEET from files generated in the preprocessing step.

4.3.8 Strang-Splitting (SS)

This building block allows integrating the governing equations through an operator splitting. In SWEET, first- and second-order splitting are available, corresponding respectively to

$$A^{\Delta t} \circ B^{\Delta t}, \qquad A^{\Delta t/2} \circ B^{\Delta t} \circ A^{\Delta t/2} \tag{23}$$

where A^h and B^h are the time stepping schemes applied in each step with time step size h. Therefore, the building block receives as arguments two building blocks `METHOD_i(arguments_i)`, $i = 1, 2$, each with its own arguments, where any building block can be chosen as `METHOD_i`, including `SS` itself, thus allowing to introduce further splitting.

4.3.9 Subcycling (SUBC)

The subcycling time-integration node allows integrating specific terms of the governing equations with a smaller time step size relative to the other terms. More specifically, if a time step size Δt is set for the simulation, then the user defines an integer $n \geq 1$ such that the leaf nodes handled by `SUBC` are integrated with $\Delta t/n$. Although subcycling is a trivial method, its potential is highly underestimated because of its flexibility to overcome time-stepping stiffness by subcycling only particular terms.

4.4 Examples of composed time stepping schemes

We present below a non-exhaustive list of examples of combined time integration methods that can be provided for a simulation in SWEET through the command-line option `--timestepping-method=`. These examples refer to the SWE on the rotating sphere; time integration schemes for the other equations available in SWEET can be defined analogously. A visual representation of two composed time integration schemes is illustrated in Figure 2.

- **ERK(ln,order=4)**: solve the full inviscid equation without topography (both the linear and nonlinear terms) using a 4-th order explicit Runge-Kutta scheme of the Heun type;
- **ERK(ADDT(l,na),order=4)**: the same as above, but neglecting the nonlinear term N_R;
- **ERK(ADDT(ln,b),order=4)**: the same as the first example, but including the bathymetry term, if available in the chosen benchmark test;
- **ERK(ADDT(lg,lc,na,nr,NEGT(ln)),order=4)**: solve each term separately and subtract the full equation, resulting in a zero tendency, which may be useful for debugging purposes;
- **SS(IRK(l,order=2),ERK(n,order=2,method=heun),order=2)**: solve the linear and nonlinear terms using respectively a 2nd order implicit scheme (Crank-Nicolson) and a 2nd order explicit scheme of Heun type, which are combined using a 2nd order strang-splitting scheme;
- **SS(SS(IRK(l,order=2),ERK(n,order=2,method=heun),order=2),IRK(visc(order=4,visc=1e14),order=1), order = 1)**: the same as above, but with an a-posteriori application (*i.e.*, corresponding to a 1st order splitting) of a 4th order viscosity with coefficient $\nu = 10^{14}$ solved implicitly (using backward Euler).

- **SS(SUBC(ERK(l,order=4),n=5),ERK(n,order=2,method=heun),order=2)**: solve the linear and nonlinear terms using respectively a 4th-order explicit Runge-Kutta scheme and a 2nd order explicit scheme of Heun type, which are combined using a 2nd order Strang-splitting scheme; the integration of the linear term is performed with subcycling, using a time step size $\Delta t/5$;
- **SETTLS(l,na(sl_order=2),nr)**: use the semi-Lagrangian semi-implicit SETTLS, with a second-order semi-Lagrangian approach for the advection; in the SETTLS building block, the first, second, and third arguments correspond respectively to the terms to be treated by the scheme as linear, nonlinear advection, and remainder nonlinear;
- **SETTLS(lg,na(sl_order=2),null)**: the same as above, but neglecting both L_C and N_R;
- **ETDRK(EXP(lg),ADDT(lc,n),order=2)**: second-order ETDRK scheme with only the gravity linear term being computed exactly; the linear Coriolis term is treated together with the nonlinear ones;
- **ETDRK(REXI(l),n,order=2)**: the same as above, but the exponential of the full linear term is approximated using REXI.

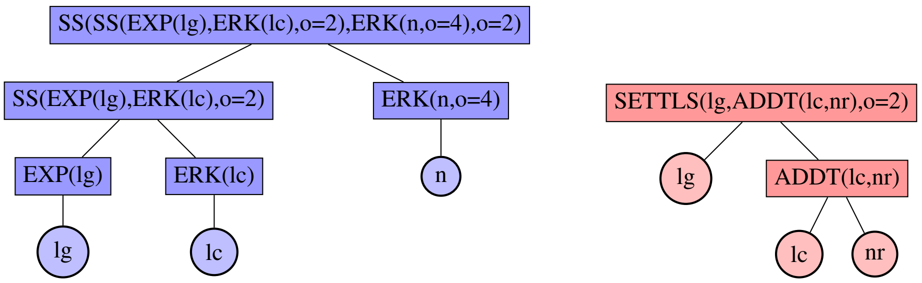


Figure 2. Visual representation of two composed time stepping scheme, respectively in blue and red. The terms inside the boxes are defined in Sections 4.2 and 4.3.

4.5 Limitations of the approach

So far, we have not encountered any non-applicability of our developed approach for single-stepping methods, which require the state at one particular point in time as an input and provide a new state or update as an output. This differs from multi-stepping methods, which require multiple time states as input and special handling of the first time steps. For the SETTLS methods and variants described above, this time, state management is handled directly within the respective SETTLS implementation. Generic support for multi-stepping in the tree-like structure does not appear to be easily feasible with a clear software design as well as an efficient implementation, and is left for future work.

4.6 Parallel-in-time methods

In addition to serial integration of atmospheric models, SWEET allows testing and studying various parallel-in-time (PinT) methods, which have raised an increasing interest in the atmospheric modeling community in the last decade (see *e.g.*, Haut and Wingate (2014); Abel et al. (2020); Hamon et al. (2020); Caldas Steinstraesser et al. (2024)). The most common PinT approaches consist of replacing an expensive, serial temporal integration of the problem by a non-intrusive iterative computation of several time steps in parallel, in a predictor-corrector procedure. Three of these methods, Parareal (Lions et al., 2001), Multi-Grid Reduction In Time (MGRIT, Friedhoff et al. (2013)), and the Parallel Full Approximation Scheme in Space and Time (PFASST, Emmett and Minion (2012)), are currently implemented in SWEET. In all cases, the implementation is performed through existing open-source libraries, namely XBraid (see (xbr)) for Parareal and MGRIT and LibPFASST (see (Minion et al., 2018)) for PFASST, allowing for an efficient MPI-based parallel implementation.

It should be noticed that these PinT methods do not fit in the temporal discretization structure described in the previous paragraphs, *i.e.*, they are not building blocks to compose a user-defined time stepping scheme. In this sense, SWEET can be understood to have two main time-integration variants: a serial one, defined by the composed time stepping schemes as described above; and a parallel one, relying on XBraid or LibPFASST. In the case of Parareal and MGRIT (through XBraid), however, there is an interface between the two variants: as detailed below, these methods need to be combined with two or more arbitrary serial time integration schemes, and our implementation allows using any user-composed scheme, in a very versatile approach.

Before giving more details on the three PinT methods and their implementation, let us highlight their importance considering the objectives of SWEET. These schemes have been seen, in the past two to three decades, as an alternative approach for overcoming the saturation of spatial parallelization and improving the time-to-solution in massively parallel computing systems. It is well known, however, that temporal parallelization is inefficient, in terms of convergence and stability, when applied to simple hyperbolic problems, *e.g.*, the one-dimensional advection equation (see *e.g.*, Gander and Vandewalle (2007); Ruprecht (2018)), which has discouraged further studies and applications to more complex problems dominated by advective processes, such as those describing atmospheric dynamics. E.g., shallow water equations are a hyperbolic model describing the propagation of waves at different speeds and interacting with each other, which impose challenges to the temporal parallelization. Therefore, the implementation of PinT methods in SWEET facilitates investigations around this novel time integration approach, allowing the conduct of parametrization studies and combining them with different serial schemes, also considering meaningful benchmarks for atmospheric modeling. Similarly, PinT methods have attracted growing interest as a promising means of transparently introducing dynamic resource management into time integration codes. This approach allows the resources allocated to a job to be adjusted dynamically at runtime by modifying the number of parallel time steps. However, the impact of such dynamic changes on the stability, convergence, and performance of PinT methods across different problem types remains poorly understood. By integrating dynamic PinT implementations into SWEET, it becomes possible to systematically study the effects of dynamic resource management on PinT methods in the context of atmospheric modeling.

We briefly describe these PinT methods in the following paragraphs and provide references for further details. All of them can be executed in SWEET with an MPI parallelization in time, which, combined with spectral discretization of the space, allows for performing hybrid Open-MPI spatio-temporal parallel simulations.

4.6.1 Parareal and MGRIT

Parareal (Lions et al., 2001) is an iterative predictor-corrector parallel-in-time method based on the simultaneous use of two discretizations of the problem: a fine one, which provides relatively accurate but computationally expensive solutions, and a coarse one, which is less expensive but also less accurate. In each Parareal iteration, the coarse method is integrated sequentially along the temporal domain, providing a low-expensive prediction, and corrections are computed in parallel (*i.e.*, along several time steps simultaneously) using the fine scheme. Along iterations, the Parareal solution converges to the solution obtained via the serial integration of the fine time stepping method, and the goal is to obtain a good approximation within a few iterations, *i.e.*, within a smaller computational time compared to the fine solution. MGRIT (Friedhoff et al., 2013), in turn, is a multi-level, iterative predictor-corrector PinT method based on spatial multigrid schemes. It can be seen as a generalization of Parareal not only due to the possibility of defining more than two levels of discretization, but also to several other parameters that can influence the convergence and cost of the method. Both methods are non-intrusive, such that the governing equations and methods considered on each discretization level can be arbitrarily chosen by the user.

Our implementation of Parareal and MGRIT in SWEET, through XBraid (xbr), aims to provide this non-intrusive parallel-in-time framework, which may be of great importance for the research around the temporal parallelization of atmospheric circulation models. The discretization schemes, as well as the spectral and temporal resolutions, can be chosen independently from one level to another. Therefore, on a given discretization level, any of the composed time stepping schemes and orders described in Section 4 can be used. For instance, if there are L discretization levels, then one can set the command-line argument

```
xbraid-timestepping-method=METHOD_1(arguments 1);METHOD_2(arguments ↙
    ↳ 2);...;METHOD_L(arguments L)
```

where the methods `METHOD_i(arguments_i)` are defined from the finest to the coarsest levels, and do not need to correspond to the same scheme. This implementation allows a very flexible study on the performance of Parareal and MGRIT as a function of the choices of time stepping schemes and discretization parameters. Also note that, with this implementation, even different governing equations can be considered at different levels, which is a very flexible and powerful approach for studying Parareal and MGRIT. For instance, one can define level-dependent viscosity parameters, aiming to find a nice spot between stability and accuracy, or define the cheap, coarse models by considering simplified governing equations.

4.6.2 PFASST

The Parallel Full Approximation Scheme in Space and Time (PFASST) is an implicit-explicit, iterative, multi-level, parallel-in-time integration method originally developed by Emmett and Minion (2012). It is a time-parallel expansion of SDC methods which, for linear problems, can be understood and expressed as a multigrid method in time and space (Bolten et al., 2016). In serial, the PFASST algorithm is reduced to an MLSDC integrator. SWEET supports using PFASST as a time integration method by interfacing with LibPFASST (Minion et al., 2018), a Fortran-based implementation of the algorithm.

SWEET uses a PFASST integrator based on the IMEX splitting described in Hamon et al. (2019): When interfacing with LibPFASST, we split the right-hand side into three terms: the stiff linear gravitational wave terms L_G, the non-stiff linear and nonlinear terms $L_C + N_A + N_R$, and the (hyper-)viscosity terms L_ν. As in the original description of semi-implicit integrators for atmospheric flow (Robert et al., 1972), L_G is evaluated implicitly (due to its potentially non-linear time-step size depending on diffusive stiffness), while the remaining terms are treated explicitly.

LibPFASST allows to adjust (without implementing from scratch inside SWEET): the number of iterations, the number of levels (SDC vs. MLSDC), type of quadrature nodes, and the level of parallelism. The first three are command-line arguments, and the latter depends on the number of MPI processes. As in the interface with XBraid, granular (hyper-)viscosity treatment is possible, allowing us to investigate the minimal amount of (numerical/eddy) viscosity necessary in order to guarantee stability of highly accurate numerical solvers.

We do not go into further details of LibPFASST in the present work, as the complexity of the underlying algorithm is out of scope for this paper. However all details can be found in (Emmett and Minion, 2012), and some application examples in HPC contexts can be found in (Speck et al., 2012, 2014).

4.6.3 Dynamic resource management for parallel-in-time methods

Dynamic Resource Management (DRM) enables runtime adjustments to the resources allocated to a job, increasing the flexibility of HPC schedulers to improve key system metrics such as throughput and energy efficiency. In addition, DRM supports application types with dynamically varying resource requirements, such as adaptive mesh refinement methods.

A recent approach to DRM, known as Dynamic Processes with PSets (DPP) (Huber et al., 2024), introduced generic design principles to enable dynamic resource management in parallel programming models such as MPI. Due to its generality, DPP has been successfully applied to a variety of application types, including iterative solvers (Huber et al., 2025), in situ workloads (Ju et al., 2025), and parallel-in-time (PinT) methods. DPP offers an abstraction for dynamic process management based on the concepts of Process Sets (PSets) and Process Set Operations (PSetOps). PSets are unique labels that identify arbitrary groups of processes within an application. PSetOps express dynamic process changes as relationships between existing PSets and new PSets created by the resource management system.

These DPP design principles have been used to extend the MPI Sessions interface in Open MPI. The Open MPI extensions have been used to introduce dynamic resource support in PinT libraries such as LibPFASST and XBraid, where we provide additional information in the following paragraphs.

PFASST demonstrates robust scalability in terms of wall-clock time (Hamon et al., 2020), benefiting significantly from increased parallelism by increasing the number of parallel time steps. While a greater number of parallel time steps improves wall-clock performance, it has been shown to negatively impact accuracy. Due to this impact on accuracy, increasing parallelism beyond a certain threshold becomes counterproductive. Therefore, the previously mentioned OpenMPI extensions were leveraged to incorporate DRM support into PFASST, thereby enabling dynamic adjustment of the number of parallel time steps. For demonstration purposes, the number of parallel time steps is currently varied randomly at fixed intervals during the simulation. Work is currently underway to evaluate PFASST parameters and implement convergence-informed adaptivity, aiming to exploit PFASST's scalability while maintaining accuracy.

The XBraid library has been extended with DRM support using DPP via the DMR library (Iserte et al., 2018). Currently, only temporal parallelization is supported when enabling resource adaptivity. Similar to LibPFASST, research on convergence-informed adaptivity is considered future work.

For both LibPFASST and XBraid, SWEET provides a versatile testbed of highly accurate numerical solvers for exploring dynamic resource strategies in PinT methods.

5 Compilation and execution pipeline

The pipeline implemented in SWEET, from the setup of benchmark tests and executables to the processing of results outputted by the numerical simulations, aims to combine computational efficiency in high-intensive computing parts with straightforwardness and readability in less critical ones. This is achieved through a combination of C++ and Python, relying on user-friendly configuration scripts where "MULE" refers to the overall scripts supporting this.

Preprocessing: As part of the preparation of a benchmark test, the user configures the benchmarks entirely in Python specifying, e.g., compilation options (*e.g.*, enabling of spherical harmonics transformations, if MPI parallelization is used or not), runtime options/program parameters (*e.g.*, the governing equations to be tested, time integration, time step size, spectral resolution, the frequency for outputting results), configuration specifics to the benchmark to be tested (*e.g.*, the benchmark name itself, physical parameters such as the mean fluid height), and parallelization parameters (*e.g.*, number of cores, number of threads per core) which are validated based on the configurations of the system which is being used. This Python-based script is then used for generating bash scripts, containing also the compilation and execution commands and options. Examples of preprocessing scripts are provided along with the source code.

Moreover, non-critical computing tasks (in terms of computational efficiency) are also performed in a Python-based preprocessing step, instead of performing them during runtime. For example, the REXI and SDC coefficients (see Section 4.3) are precomputed and stored in a file, which will be read during execution.

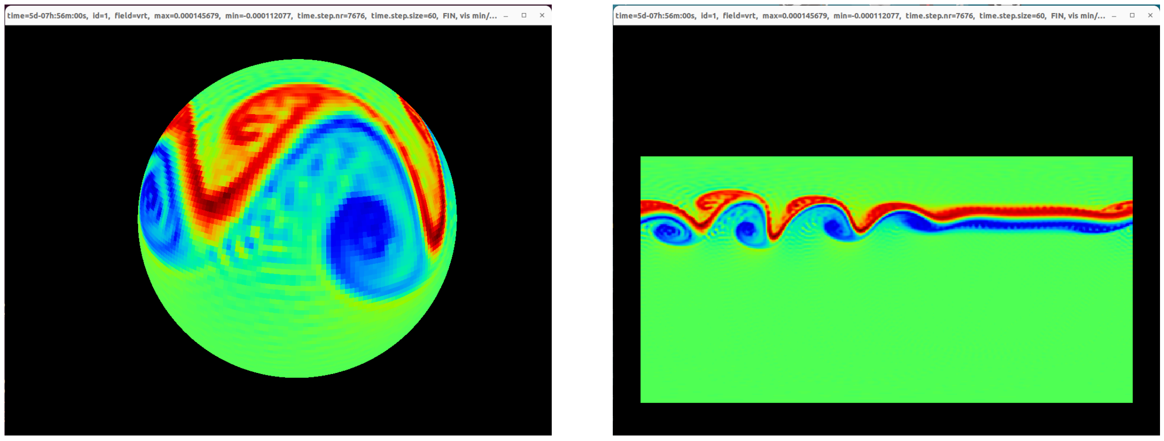
Compilation: SWEET is compiled using the build tool SCons, using commands generated from the preprocessing script described above. SWEET supports different programs, where each program typically relates to an implementation of a particular ODE or PDE. Each program has its own main .cpp file where default compilation options are encoded in the header of this main .cpp file for convenience. E.g., a PDE on the sphere always requires spherical harmonics transformations to be enabled.

Execution: The main core of SWEET (execution of the simulation and output generation) is written in C++ for performance reasons, aiming at the execution of simulations in HPC systems. The simulation is executed from command-line arguments, which are generated by the preprocessing script described above and include all user-defined options. Depending on the user's preferences, results are stored in binary files at a given frequency in time. Moreover, in order to easily allow HPC performance studies, SWEET outputs wall-clock times, which can be easily extracted in the postprocessing step.

Postprocessing: A variety of postprocessing scripts, written in Python, are provided by SWEET, offering a straightforward toolbox for convergence and performance studies. They allow, for example, to plot the solution at specific times, to compute and plot the difference between two simulations, to compute the errors between several simulations and a given reference and plot them as a function of a given parameter (*e.g.*, the time step size), and to plot the errors as a function of the wall clock time. Examples on how to use the scripts are also provided in the source code.

Execution on compute clusters: The way to create bash scripts as an output of the preprocessing step is intentionally similar to the way the batch submission system works on compute clusters. SWEET includes a variety of scripts that allow users to load an environment specific to a compute cluster. The aforementioned job generation then automatically adds the respective headers and other content to the batch job submission script to prepare them for submission. Additional batch scripts are provided, which are named in the same way across different clusters but are tailored to each compute cluster, allowing job submission, status checks, etc. With such an abstraction layer, the cluster-specific way to run benchmarks is hidden from the user. Although this sounds like a huge advantage, we should also mention that additional effort has to be put into this for each new compute cluster.

Graphical User Interface: SWEET also provides an optional graphical user interface (GUI) on the plane and the sphere, which is especially useful for debugging purposes. The solution is displayed during runtime, and the simulation can be stopped and continued through keyboard shortcuts. Shortcuts are also available for moving, zooming, advancing single-time steps of the simulation, changing the displayed data field, and changing between the visualization on the sphere and on the plane. Such an interactive investigation allows, e.g., to directly observe numerical instabilities and their corresponding region, step-by-step time integrate the solution, zoom in/out at relevant areas, and rapidly explore different combinations of time integration methods. Consequently, this leads to a strong reduction in time for developing and debugging new numerical methods. Illustrative screenshots are presented in Figure 3.



```
time=5d-07h:56m:00s, id=1, field=vrt, max=0.000145679, min=-0.000112077, time.step.nr=7676, time.step.size=60, FIN, vis min/...
```

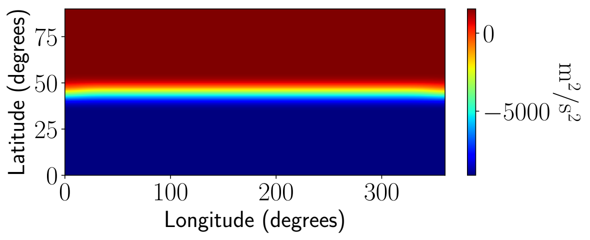
```
time=5d-07h:56m:00s, id=1, field=vrt, max=0.000145679, min=-0.000112077, time.step.nr=7676, time.step.size=60, FIN, vis min/...
```

Figure 3. Screenshot of the GUI provided by SWEET, displaying the vorticity field in the Galewsky unstable jet test case for the SWE on the rotating sphere, after approximately 5 days of simulation. Left: visualization on the sphere; right: visualization on the plane (projection from the sphere).

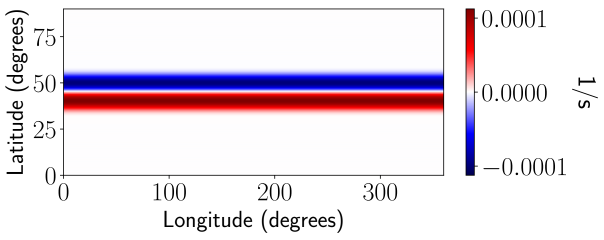
6 Numerical examples

In this section, we present some numerical examples executed with SWEET, in order to illustrate some of the functionalities of the software. We present the command-line arguments and highlight the versatility of the software for executing simulations with different time stepping schemes or even governing equations without further compilation. We also present some postprocessing results produced with scripts provided along with the software, allowing for easy comparison of the different runs. We present three sets of simulations considering the Galewsky barotropic instability test case (Galewsky et al., 2004), which is a standard benchmark in atmospheric modeling studies: in the first one, we integrate these benchmark considering the full inviscid SWE on the rotating sphere using various time integration schemes, defined using the composability approach presented above; in the second-one, we integrate variants of the governing equations, which are easily defined using the same approach; finally, we perform PinT simulations relying on the composed schemes.

In the Galewsky test case, a Gaussian bump perturbation in the geopotential field leads to the formation of vortices in an initially stationary zonal jet profile. Despite its relatively simple initial solution (Figure 4), the complex dynamics after a few days of simulation (Figure 5) make it especially challenging to develop numerical methods able to provide accurate and stable solutions. Guidelines for evaluating this test case quantitatively and qualitatively have been proposed by Scott et al. (2015).

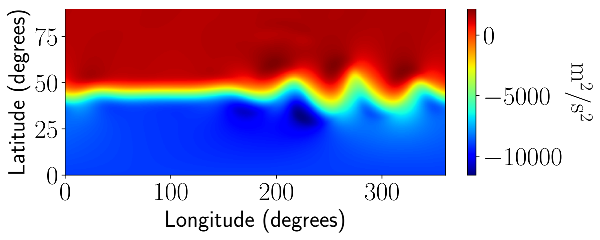


(a) Geopotential perturbation Φ'

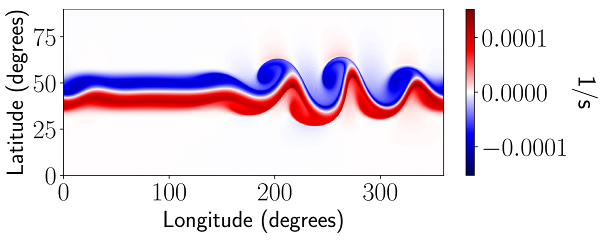


(b) Vorticity field ξ

Figure 4. Initial solution of the Galewsky barotropic instability test case.



(a) Geopotential perturbation Φ'



(b) Vorticity field ξ

Figure 5. Solution of the Galewsky barotropic instability test case after 5 days

A basic bash command for executing the unstable jet test case in SWEET, using a serial time integration scheme, reads

```
$sweet_executable -M 64 --benchmark-name=galewsky ↙
↳ --timestepping-method=METHOD --dt=480 -t 432000.0 -o 86400
```

which is automatically generated using Python job generation scripts (see Section 5) and placed in a bash script named `run.sh`. This script also contains the SCons command required for compiling the executable `$sweet_executable`. In the above command, `-M`, `--dt`, `-t`, and `-o` specify, respectively, the spectral resolution, the time step size, the final time of simulation, and the time step size for producing output files. `--benchmark-name` specifies the test case to execute among those coded in SWEET. Finally, the timestepping method is specified by setting `--timestepping-method=METHOD` as described in Section 4.

To further illustrate SWEET's usefulness as a tool for atmospheric modeling research, relying on benchmark test cases that are standard in the scientific community and represent complex and challenging dynamics, we also present qualitative results of the test case proposed by Polvani et al. (1994) for the study of two-dimensional shallow water turbulence. In the test, the authors studied the generation of coherent vortex structures from initially balanced solutions as a function of the Rossby and Froude numbers. This benchmark test case is also implemented in SWEET and can be executed very similarly to the Galewsky one:

```
$sweet_executable -N 200 --benchmark-name=polvani  --polvani-rossby=0.01 ↙
↳ --polvani-froude=0.04 --timestepping-method=METHOD --dt=0.0025 -t 1000 -o 1000
```

where the parameters were chosen following Polvani et al. (1994). Contrary to the Galewsky test case, we set the spatial resolution *-N* (from which the spectral resolution is set following the anti-aliasing 3/2-rule). It is also necessary to provide the Rossby and Froude numbers, which determine the initial conditions, through the parameters *–polvani-rossby* and *–polvani-froude*, respectively.

All simulations were performed on Intel Xeon Gold 6130 @2.10GHz nodes of the GRICAD cluster from the University of Grenoble Alpes. Each node contains 32 physical cores, distributed between two sockets. Serial time integrations were performed using 16 OpenMP threads for spatial parallelization, bound to cores in the same socket. Parallel-in-time simulations were conducted using a hybrid MPI-OpenMP parallelization: by allocating N_{proc} nodes, N_{proc} MPI tasks are defined for the temporal parallelization, with each task being attributed to a socket within a node, and OpenMP-based spatial parallelization with 16 threads is set within each socket.

6.1 Integration using various time stepping schemes

In this first set of simulations, we integrate the Galewsky benchmark test case considering the full inviscid SWE on the rotating sphere (*i.e.*, considering the terms L_G, L_C, N_A and N_R in eq. (1)), using the following schemes, which can be easily identified following Section 4:

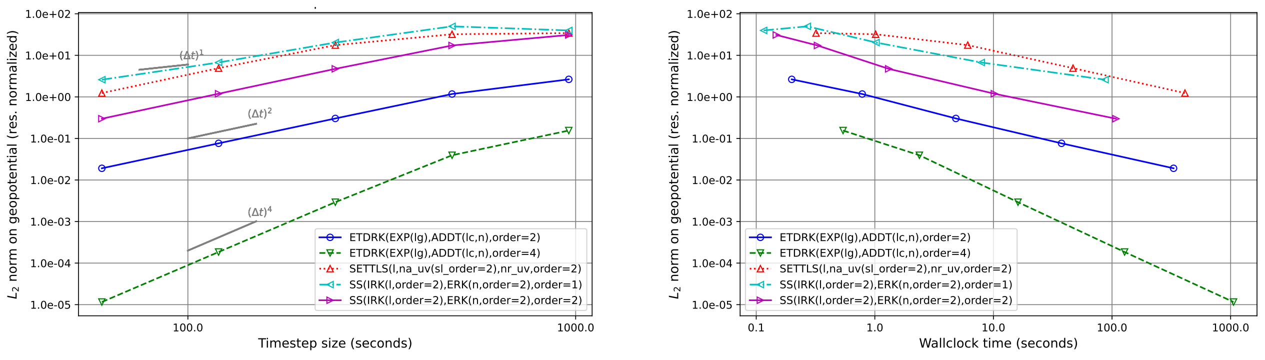
- `SS(IRK(l,order=2),ERK(n,order=2),order=1)`
- `SS(IRK(l,order=2),ERK(n,order=2),order=2)`
- `ETDRK(EXP(lg),ADDT(lc,n),order=2)`

- `ETDRK(EXP(lg),ADDT(lc,n),order=4)`
- `SETTLS(l,na(sl_order=2),nr,order=2)`

Output files for the prognostic variables (perturbation of the geopotential field (Φ'), vorticity field (ξ), and divergence field (δ)) are produced at a frequency determined by the command-line argument `-o`. By default, these output files are in binary format and contain the solution in the spectral space (*i.e.*, the spectral coefficients of the solution are stored); if necessary, the postprocessing scripts compute the solution in the latitude-longitude grid. It is also possible to directly output text files containing the solution in the grid space.

For each time stepping scheme, the spectral resolution M and the time step size Δt are chosen respectively in $\{32, 64, 128, 256, 512\}$ and $\{60, 120, 240, 480, 960\}$ (s), with $M\Delta t$ being kept constant. For each simulation, errors are computed w.r.t. a reference solution obtained from a simulation using a 4th-order explicit Runge-Kutta scheme (`--timestepping-method=ERK(ln,order=4)`), with same spectral resolution and time step size four times smaller, in order to measure exclusively the errors related to the temporal discretization; it is also possible, however, to compare solutions with different spectral resolutions using interpolation procedures implemented in the postprocessing scripts.

The simulations are executed for 5 days. Since quantitative measurements of the error in long runs are not recommended in this test case due to the complex nature of the solution (Peixoto and Schreiber, 2019), we present the relative discrete L_2 errors on the geopotential field after 1 day (before a clear formation of the vortices) in Figure 6a, in which the theoretical orders of convergence of the time stepping schemes can be verified.



(a) Time step size vs error

(b) Wall-clock time vs errors

Figure 6. L_2 error on the perturbation of geopotential field after 1 day of simulation in the Galewsky test case, as a function of the time step size (left) and the wall-clock time, for various time stepping schemes.

Figure 6b presents the errors on the geopotential as a function of the wall-clock time. This graph allows for easy comparison of the time-stepping schemes in terms of the trade-off between accuracy and computational cost, with curves closest to the bottom left corner of the graph representing better compromises.

6.2 Simulation of variants of the SWE

We now illustrate how the composable time stepping construction in SWEET can be easily used to perform simulations of variants of the governing equations, by neglecting or including specific terms. This can be very useful for studying the influence of individual physical processes, understanding how they are influenced by discretization choices, including artificial viscosity to improve the stability, debugging and validating time stepping schemes (*e.g.*, if it is known how a given method should behave if the equations contain only linear terms).

We consider the Galewsky barotropic instability test case, as in the results presented above. The numerical integration is performed up to $t = T = 5$ days, using the second-order Eulerian ETDRK scheme, with $\Delta t = 60$s and a spectral resolution $M = 512$. Simulations are performed considering the following arguments for `--timestepping-method`:

1. `ETDRK(EXP(lg),ADDT(lc,n),order=2)`
2. `ETDRK(EXP(lg),lc,order=2)`
3. `SS(ETDRK(EXP(lg),ADDT(lc,n),order=2),IRK(visc(order=2,visc=1e+6),order=1),o=1)`
4. `SS(ETDRK(EXP(lg),ADDT(lc,n),order=2),IRK(visc(order=6,visc=1e+26),order=1),o=1)`

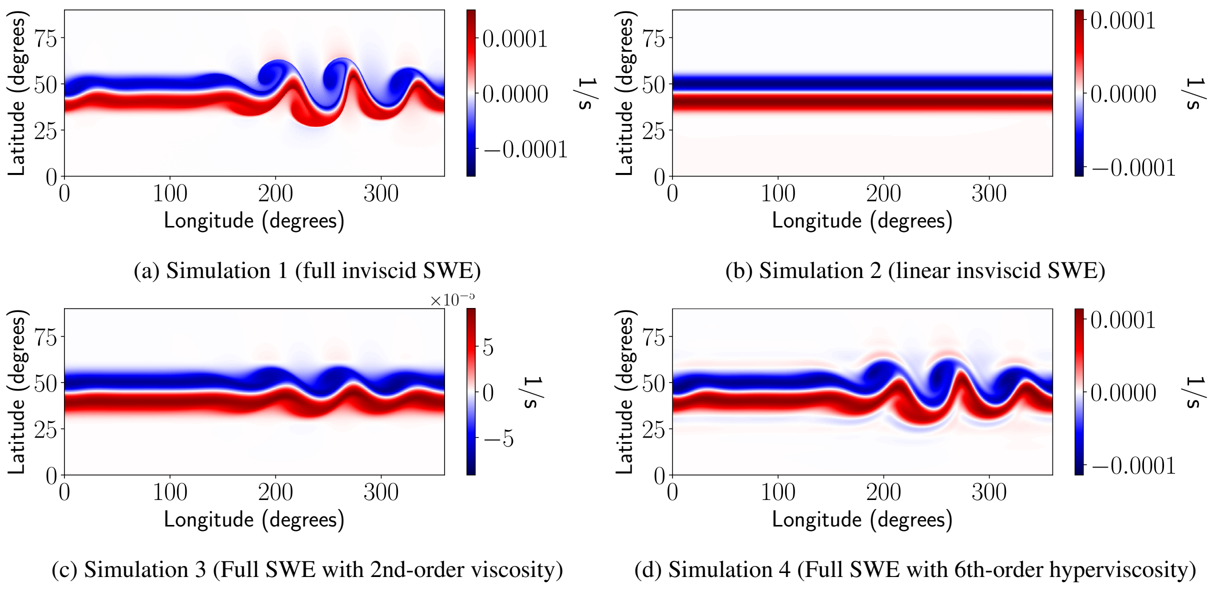
Simulation 1 considers the full inviscid SWE; in Simulation 2, all nonlinear terms are neglected; and in Simulations 3 and 4, we add an artificial (hyper-)viscosity term to the full equations, respectively with orders $q = 2$ and $q = 6$, and coefficients $\nu = 10^6 \mathrm{m}^2\mathrm{s}^{-1}$ and $\nu = 10^{26} \mathrm{m}^6\mathrm{s}^{-1}$. In these last two simulations, the viscosity term is solved using the backward Euler method after the integration of the other terms (*i.e.*, as the second step of a first-order splitting). We also support short-hand notations, *e.g.*, `o=` for `order=`

Figure 7 presents the vorticity field obtained after 5 days in each simulation. As expected, the complex dynamics characterizing the Galewsky test case and induced by nonlinear interactions are no longer present in Simulation 2, where only linear terms are considered, which can be useful in the validation step of the development of a given time stepping scheme. In Simulations 3 and 4, we observe the influence of different choices of artificial viscosity parameters: in both cases, a clear smoothing of the vorticity field is observed, which is more pronounced in the case of a low-order viscosity, since it damps a larger region of the wavenumber spectrum.

We remark that all simulations are run using the same executable, *i.e.*, no further compilation is required in order to activate or deactivate specific terms of the governing equations. Indeed, adding or including terms means only that the respective blocks are included or not in the composed time stepping scheme.

6.3 Parallel-in-time simulations

We now present a parallel-in-time simulation in order to illustrate the versatile implementation of PinT methods in SWEET. We consider the Galewsky barotropic instability test cases for the full inviscid SWE on the rotating sphere, which we integrate



(a) Simulation 1 (full inviscid SWE)
(b) Simulation 2 (linear insviscid SWE)
(c) Simulation 3 (Full SWE with 2nd-order viscosity)
(d) Simulation 4 (Full SWE with 6th-order hyperviscosity)

Figure 7. Vorticity field ξ of the Galewsky barotropic instability test case after 5 days, considering different variants of the SWE.

using the Parareal method, through the interface with the XBraid library. As mentioned in Section 4.6, XBraid implements the more general MGRIT algorithm, so we set up a Parareal simulation by choosing specific parameters.

A bash command for executing a MGRIT simulation in SWEET, also generated using Python job generation scripts, reads

```
$sweet_xbraid_executable -M 64 --benchmark-name=galewsky  ↙
   ↪ --xbraid-timestepping-method=METHOD --xbraid-cfactor=8 ↙
   ↪ --xbraid-max-levels-2 [...] --dt=60 -t 432000.0 -o 86400
```

Some of the options are common to the serial case, as presented in the beginning of this section. The option `--dt` refers to the time step size on the finest level; `--xbraid-timestepping-method` defines the time integration schemes used on each level; `--xbraid-cfactor` sets the temporal coarsening between consecutive levels; and `--xbraid-max-levels` sets the maximum number of discretization levels, if allowed by the other discretization choices. `[...]` represents several other options starting with `--xbraid` allowing to set other parameters defined in XBraid.

As discussed above, an important feature of Parareal and MGRIT is their non-intrusiveness and versatile character, allowing the user to choose any time stepping schemes to integrate the equations on each discretization level, which is possible in our implementation in SWEET. We perform simulations by considering a second-order implicit-explicit and a second-order Eulerian ETDRK scheme on the fine and coarse levels, respectively. On the coarse level, we also include an artificial fourth-order viscosity term, and we test three values for the viscosity coefficient ($\nu = 10^{15}\text{m}^4/\text{s}$, $\nu = 10^{16}\text{m}^4/\text{s}$, and $\nu = 10^{17}\text{m}^4/\text{s}$). These configurations are set using the following command-line options:

```
--xbraid-timestepping-method="SS(IRK(l,order=2), ERK(ADDT(na_uv,nr_uv), order=2), ↙
    ↳ order=2);SS(ETDRK(EXP(lg),ADDT(lc,na_uv,nr_uv), order=2), ↙
    ↳ IRK(visc(order=4,visc=1e+16), order=1), order=1)".
```

where the only difference between simulations is the value of `visc`. The composed time stepping schemes for the fine and coarse levels are separated by a semicolon.

Figure 8 presents the evolution of the errors, along iteration, between the parallel-in-time solution and the solution provided by the serial integration of the fine time stepping scheme (IMEX with time step size $\Delta t = 60$s). Simulations with smaller artificial viscosity on the coarse level present a fast error decrease at the initial iterations, but it soon stagnates or leads to larger errors, which may be linked to the stronger hyperbolic character of the equations. On the other hand, a larger viscosity coefficient leads to an initially slower convergence, but it leads to smaller errors; however, stagnation and error increase are also observed after several iterations.

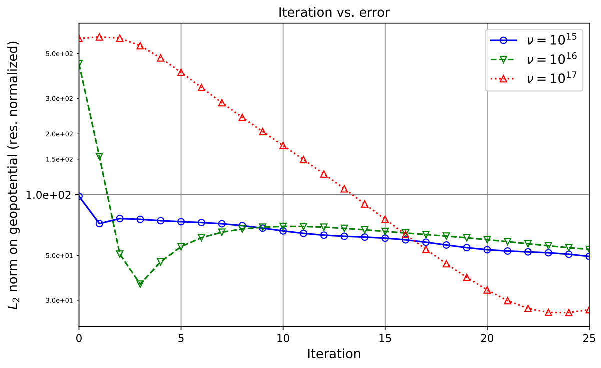


Figure 8. L_2 error on the perturbation field after six days of simulation along iterations, between the parallel-in-time solution and the solution provided by serial integration of the fine time stepping scheme.

These results illustrate the challenges in developing and applying temporal parallelization in complex fluid dynamics problems. Therefore, it is of great interest to have a versatile and straightforward tool for supporting this research. The implementation proposed in SWEET allows several interesting possibilities for studies around PinT methods. For instance, we defined different viscosity configurations on each level, such that we are able to improve the stability on the coarse one (where the time step size is larger), while preserving the accuracy of the fine discretization. We could also easily define different governing equations for different discretization levels, *e.g.*, on the coarse one we can consider a simplified model, by neglecting specific terms.

6.4 Polvani test cases

Finally, we illustrate numerical simulations of the test case by Polvani et al. (1994). Determining initial conditions for these benchmarks is considered to be rather challenging. Given the support of various operators in spectral space in SWEET, it allows for easy integration. Among the several simulations presented by the authors, we consider the initial conditions G, with Rossby and Froude numbers equal, respectively, to $R = 1.00$ and $F = 0.30$. Figure 9 presents the initial condition and the numerical solution at $t = 100$s and $t = 500$s, using a 4th-order explicit Runge-Kutta scheme. An eighth-order viscosity is applied. The composed time-stepping scheme is

```
SS(ERK(ln,order=4), IRK(visc(order=8,visc=1e-5), order=1), order=1)
```

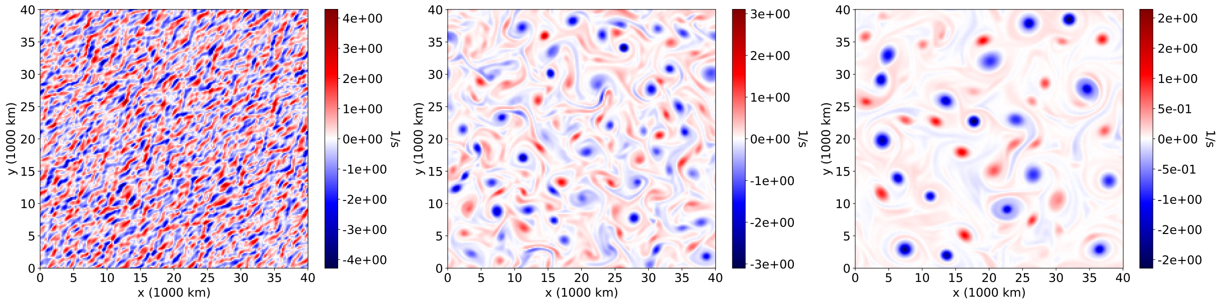


Figure 9. Vorticity field in the test case G by (Polvani et al., 1994), with Rossby number $R = 1.00$ and Froude number $F = 0.30$, integrated with a 4th-order explicit Runge Kutta scheme. (a) initial condition; (b) $t = 100$; and (c) $t = 500$s.

7 Conclusions

In this work, we presented the software SWEET, an open-source software for the numerical integration of global spectral methods on the bi-periodic plane and the sphere, focusing on the shallow water equations for atmospheric circulation modeling. The main goal of this paper was to present the versatility of SWEET as a tool for developing and analyzing time integration schemes.

The diversity and complexity of numerical methods for numerically integrating time-dependent differential equations usually require considerable code development efforts not only to implement these methods, but also to compare them, investigate modifications motivated, *e.g.*, by the physical processes described by the governing equations, and execute them in high-performance computing systems. SWEET allows a significant reduction of such efforts. Through a tree-based composability approach, the user can construct different time integration schemes by composing elementary time integration building blocks. Various methods are provided by SWEET, from more classical ones such as Eulerian Runge-Kutta schemes to more recent ones such as Spectral Deferred Correction and semi-Lagrangian exponential methods. The parallel-in-time time integration schemes Parareal, MGRIT, and PFASST are available as well, also following a very versatile approach. Through this same composability structure, the user can also easily modify the governing equations to be integrated, *e.g.*, by neglecting specific terms. Both the construction of the time integration scheme and the choice of the terms to be integrated are done exclusively via a command-line option, not requiring any further compilation.

SWEET also provides a good balance between computational efficiency and a straightforward, user-friendly interface. Whereas SWEET's core is developed in C++, with support for execution on different platforms, including HPC ones, pre- and postprocessing are performed through Python scripts, allowing easy setup and analysis of the numerical simulations.

To illustrate these features, we presented numerical simulations of benchmark test cases which are commonly used in atmospheric modeling research.

Data availability. The data used to generate the plots presented in this manuscript are publicly available in Zenodo at https://doi.org/10.5281/zenodo.17624458 (Schreiber et al., 2025b). This dataset can also be generated using the code and scripts provided, as described in the Code availability section.

Code availability. SWEET source code is publicly available in https://gitlab.inria.fr/sweet/sweet and stored in Zenodo at https://doi.org/10.5281/zenodo.17424607 (Schreiber et al., 2025a). An informative webpage is available in https://sweet.gitlabpages.inria.fr/sweet-www/. The pre- and postprocessing scripts for the numerical simulations presented in this article are available in https://gitlab.inria.fr/sweet/sweet/-/tree/main/benchmarks_sphere/paper_GMD and also stored in https://doi.org/10.5281/zenodo.17424607 (Schreiber et al., 2025a).

Author contributions. KG: Software, Investigation, Writing (original draft preparation + review and editing). FH: Software, Investigation, Methodology DH: Software, Investigation, Methodology, Writing (original draft preparation + review and editing). TL: Formal analysis, Software, Investigation, Methodology, Validation, Writing (original draft preparation + review and editing). PSP: Conceptualization, Funding acquisition, Formal analysis, Software, Supervision, Methodology, Writing (original draft preparation + review and editing) JGCS: Conceptualization, Formal analysis, Software, Methodology, Validation, Writing (original draft preparation + review and editing) MS: Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Writing (original draft preparation + review and editing). VS: Software, Investigation, Methodology, Validation, Writing (original draft preparation + review and editing).

Competing interests. The authors declare that they have no conflict of interest.

Acknowledgements. We gratefully acknowledge Andreas Schmitt, who once contributed to SWEET's development with Parareal Semi-Lagrangian solvers for the Burgers' equation in SWEET. We also thank Nathanaël Schaeffer, who modified the SHTns library to better fit SWEET's requirements.

This work was supported by the São Paulo Research Foundation (FAPESP) grants 2021/03777-2, 2021/06176-0, as well as the Brazilian National Council for Scientific and Technological Development (CNPq), Grant 303436/2022-0. This project also received funding from the Federal Ministry of Education and Research and the European High-Performance Computing Joint Undertaking (JU) under grant agreement

No 955701, Time-X. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Belgium, France, Germany, Switzerland.

Most of the computations presented in this paper were performed using the GRICAD infrastructure (https://gricad.univ-grenoble-alpes.fr), which is supported by Grenoble research communities.

References

XBraid: Parallel multigrid in time, http://llnl.gov/casc/xbraid.

Abel, N., Chaudhry, J., Falgout, R., and Schroder, J.: Multigrid-Reduction-in-Time for the Rotating Shallow Water Equations, https://doi.org/10.2172/1647444, 2020.

Bolten, M., Moser, D., and Speck, R.: A Multigrid Perspective on the Parallel Full Approximation Scheme in Space and Time, arXiv:1603.03586 [math], 2016.

Burns, K. J., Vasil, G. M., Oishi, J. S., Lecoanet, D., and Brown, B. P.: Dedalus: A flexible framework for numerical simulations with spectral methods, Physical Review Research, 2, 023 068, 2020.

Caldas Steinstraesser, J. G., Peixoto, P. d. S., and Schreiber, M.: Parallel-in-time integration of the shallow water equations on the rotating sphere using Parareal and MGRIT, Journal of Computational Physics, 496, 112 591, https://doi.org/10.1016/j.jcp.2023.112591, 2024.

Caldas Steinstraesser, J. G., Schreiber, M., and Peixoto, P. S.: Analysis and improvement of a semi-Lagrangian exponential scheme for the shallow-water equations on the rotating sphere, ESAIM: Mathematical Modelling and Numerical Analysis, 59, 1531–1564, https://doi.org/10.1051/m2an/2025034, 2025.

Cox, S. M. and Matthews, P. C.: Exponential Time Differencing for Stiff Systems, Journal of Computational Physics, pp. 430–455, https://doi.org/https://doi.org/10.1006/jcph.2002.6995, 2002.

Damle, A., Beylkin, G., Haut, T., and Monzón, L.: Near optimal rational approximations of large data sets, Applied and Computational Harmonic Analysis, 35, 251–263, https://doi.org/10.1016/j.acha.2012.08.011, 2013.

Durran, D.: Numerical methods for fluid dynamics : with applications to geophysics, Springer, New York, 2010.

Dutt, A., Greengard, L., and Rokhlin, V.: Bit Numerical Mathematics, 40, 241–266, https://doi.org/10.1023/a:1022338906936, 2000.

Emmett, M. and Minion, M. L.: Toward an Efficient Parallel in Time Method for Partial Differential Equations, Communications in Applied Mathematics and Computational Science, 7, 105–132, https://doi.org/10.2140/camcos.2012.7.105, 2012.

Friedhoff, S., Falgout, R. D., Kolev, T. V., MacLachlan, S. P., and Schroder, J. B.: A Multigrid-in-Time Algorithm for Solving Evolution Equations in Parallel, in: Presented at: Sixteenth Copper Mountain Conference on Multigrid Methods, Copper Mountain, CO, United States, Mar 17 - Mar 22, 2013, http://www.osti.gov/scitech/servlets/purl/1073108, 2013.

Frigo, M. and Johnson, S. G.: FFTW: An adaptive software architecture for the FFT, in: Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181), vol. 3, pp. 1381–1384, IEEE, 1998.

Galewsky, J., Scott, R. K., and Polvani, L. M.: An initial-value problem for testing numerical models of the global shallow-water equations, Tellus A: Dynamic Meteorology and Oceanography, 56, 429–440, https://doi.org/10.3402/tellusa.v56i5.14436, 2004.

Gander, M. J. and Vandewalle, S.: Analysis of the Parareal Time-Parallel Time-Integration Method, SIAM J. Scientific Computing, 29, 556–578, https://doi.org/10.1137/05064607X, 2007.

Hamon, F. P., Schreiber, M., and Minion, M. L.: Multi-level spectral deferred corrections scheme for the shallow water equations on the rotating sphere, Journal of Computational Physics, 376, 435–454, https://doi.org/10.1016/j.jcp.2018.09.042, 2019.

Hamon, F. P., Schreiber, M., and Minion, M. L.: Parallel-in-time multi-level integration of the shallow-water equations on the rotating sphere, Journal of Computational Physics, 407, 109 210, https://doi.org/10.1016/j.jcp.2019.109210, 2020.

Haut, T. and Wingate, B.: An Asymptotic Parallel-in-Time Method for Highly Oscillatory PDEs, SIAM Journal on Scientific Computing, 36, A693–A713, https://doi.org/10.1137/130914577, 2014.

Hortal, M.: The development and testing of a new two-time-level semi-Lagrangian scheme (SETTLS) in the ECMWF forecast model, Quarterly Journal of the Royal Meteorological Society, 128, 1671–1687, https://doi.org/10.1002/qj.200212858314, 2002.

Huang, J., Jia, J., and Minion, M.: Accelerating the convergence of spectral deferred correction methods, Journal of Computational Physics, 214, 633–656, 2006.

Huber, D., Schreiber, M., Schulz, M., Pritchard, H., and Holmes, D.: Design Principles of Dynamic Resource Management for High-Performance Parallel Programming Models, https://doi.org/10.48550/arXiv.2403.17107, 2024.

Huber, D., Iserte, S., Schreiber, M., Peña, A. J., and Schulz, M.: Bridging the Gap Between Genericity and Programmability of Dynamic Resources in HPC, in: ISC High Performance 2025 Research Paper Proceedings (40th International Conference), pp. 1–11, 2025.

Iserte, S., Mayo, R., Quintana-Ortí, E. S., Beltran, V., and Peña, A. J.: DMR API: Improving cluster productivity by turning applications into malleable, Parallel Computing, 78, 54–66, https://doi.org/https://doi.org/10.1016/j.parco.2018.07.006, 2018.

Ju, Y., Huber, D., Perez, A., Ulbl, P., Markidis, S., Schlatter, P., Schulz, M., Schreiber, M., and Laure, E.: Dynamic Resource Management for In-Situ Techniques Using MPI-Sessions, in: Recent Advances in the Message Passing Interface, edited by Blaas-Schenner, C., Niethammer, C., and Haas, T., pp. 105–120, Springer Nature Switzerland, Cham, 2025.

Lions, J.-L., Maday, Y., and Turinici, G.: Résolution d'EDP par un schéma en temps “pararéel”, Comptes Rendus de l'Académie des Sciences - Series I - Mathematics, 332, 661–668, https://doi.org/10.1016/s0764-4442(00)01793-6, 2001.

Maher, P., Gerber, E. P., Medeiros, B., Merlis, T. M., Sherwood, S., Sheshadri, A., Sobel, A. H., Vallis, G. K., Voigt, A., and Zurita-Gotor, P.: Model Hierarchies for Understanding Atmospheric Circulation, Reviews of Geophysics, 57, 250–280, https://doi.org/10.1029/2018RG000607, 2019.

Mengaldo, G., Wyszogrodzki, A., Diamantakis, M., Lock, S.-J., Giraldo, F. X., and Wedi, N. P.: Current and Emerging Time-Integration Strategies in Global Numerical Weather and Climate Prediction, Archives of Computational Methods in Engineering, 26, 663–684, https://doi.org/10.1007/s11831-018-9261-8, 2018.

Minion, M., Krull, B., Emmett, M., and Goetschel, S.: libpfasst v1.0, [Computer Software] https://doi.org/10.11578/dc.20180711.5, https://doi.org/10.11578/dc.20180711.5, 2018.

Minion, M. L.: Semi-implicit spectral deferred correction methods for ordinary differential equations, Commun. Math. Sci., 1, 471–500, 2003.

Moler, C. and Loan, C. V.: Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later, SIAM Review, 45, 3–49, https://doi.org/10.1137/s00361445024180, 2003.

Peixoto, P. S. and Schreiber, M.: Semi-Lagrangian Exponential Integration with Application to the Rotating Shallow Water Equations, SIAM Journal on Scientific Computing, 41, B903–B928, https://doi.org/10.1137/18M1206497, 2019.

Polvani, L. M., McWilliams, J. C., Spall, M. A., and Ford, R.: The coherent structures of shallow-water turbulence: Deformation-radius effects, cyclone/anticyclone asymmetry and gravity-wave generation, Chaos: An Interdisciplinary Journal of Nonlinear Science, 4, 177–186, https://doi.org/10.1063/1.166002, 1994.

Randall, D. A., Bitz, C. M., Danabasoglu, G., Denning, A. S., Gent, P. R., Gettelman, A., Griffies, S. M., Lynch, P., Morrison, H., Pincus, R., and Thuburn, J.: 100 Years of Earth System Model Development, Meteorological Monographs, 59, 12.1 – 12.66, https://doi.org/10.1175/AMSMONOGRAPHS-D-18-0018.1, 2018.

Raphaldini, B., Peixoto, P. S., Teruya, A. S. W., Raupp, C. F. M., and Bustamante, M. D.: Precession resonance of Rossby wave triads and the generation of low-frequency atmospheric oscillations, Physics of Fluids, 34, https://doi.org/10.1063/5.0091383, 2022.

Rathgeber, F., Ham, D. A., Mitchell, L., Lange, M., Luporini, F., McRae, A. T., Bercea, G.-T., Markall, G. R., and Kelly, P. H.: Firedrake: automating the finite element method by composing abstractions, ACM Transactions on Mathematical Software (TOMS), 43, 1–27, 2016.

Robert, A., Henderson, J., and Turnbull, C.: An Implicit Time Integration Scheme for Baroclinic Models of the Atmosphere, 1972.

Ruprecht, D.: Wave propagation characteristics of Parareal, Computing and Visualization in Science, 19, 1–17, https://doi.org/10.1007/s00791-018-0296-z, 2018.

Schaeffer, N.: Efficient spherical harmonic transforms aimed at pseudospectral numerical simulations, Geochemistry, Geophysics, Geosystems, 14, 751–758, https://doi.org/10.1002/ggge.20071, 2013.

Schmitt, A., Schreiber, M., Peixoto, P., and Schäfer, M.: A numerical study of a semi-Lagrangian Parareal method applied to the viscous Burgers equation, Computing and Visualization in Science, 19, 45–57, https://doi.org/10.1007/s00791-018-0294-1, 2018.

Schreiber, M. and Brown, J.: A Unification and Investigation of Rational Approximation of Exponential Integration Methods, https://hal.science/hal-04363335, working paper or preprint, 2025.

Schreiber, M. and Loft, R.: A parallel time integrator for solving the linearized shallow water equations on the rotating sphere, Numerical Linear Algebra with Applications, 26, https://doi.org/10.1002/nla.2220, 2018.

Schreiber, M., Peixoto, P. S., Haut, T., and Wingate, B.: Beyond spatial scalability limitations with a massively parallel method for linear oscillatory problems, The International Journal of High Performance Computing Applications, 32, 913–933, https://doi.org/10.1177/1094342016687625, 2017.

Schreiber, M., Schaeffer, N., and Loft, R.: Exponential integrators with parallel-in-time rational approximations for the shallow-water equations on the rotating sphere, Parallel Computing, 85, 56–65, https://doi.org/10.1016/j.parco.2019.01.005, 2019.

Schreiber, M., Gaddameedi, K., Hamon, F., Huber, D., Lunet, T., Peixoto, P., Caldas Steinstraesser, J. G., and Schüller, V.: SWEET - Shallow Water Equation Environment for Tests, https://doi.org/10.5281/ZENODO.17424607, 2025a.

Schreiber, M., Gaddameedi, K., Hamon, F., Huber, D., Lunet, T., Peixoto, P., Caldas Steinstraesser, J. G., and Schüller, V.: Dataset from the numerical simulations presented in the article "SWEET - Shallow Water Equation Environment for Tests v1.0", submitted to GMD, https://doi.org/10.5281/ZENODO.17624458, 2025b.

Scott, R. K., Harris, L. M., and Polvani, L. M.: A test case for the inviscid shallow-water equations on the sphere, Quarterly Journal of the Royal Meteorological Society, 142, 488–495, https://doi.org/10.1002/qj.2667, 2015.

Shipton, J., Cotter, C., Bendall, T., Gibson, T., Mitchell, L., Ham, D., Wingate, B., Acreman, D., and Schreiber, M.: Compatible finite element methods and parallel-in-time schemes for numerical weather prediction., in: EGU General Assembly Conference Abstracts, p. 22676, 2020.

Speck, R., Ruprecht, D., Krause, R., Emmett, M., Minion, M., Winkel, M., and Gibbon, P.: A massively space-time parallel N-body solver, in: SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, pp. 1–11, IEEE, 2012.

Speck, R., Ruprecht, D., Emmett, M., Bolten, M., and Krause, R.: A space-time parallel solver for the three-dimensional heat equation, in: Parallel Computing: Accelerating Computational Science and Engineering (CSE), pp. 263–272, IOS Press, 2014.

Temperton, C.: Treatment of the Coriolis Terms in Semi-Lagrangian Spectral Models, Atmosphere-Ocean, 35, 293–302, https://doi.org/10.1080/07055900.1997.9687353, 1997.

Williamson, D. L.: The Evolution of Dynamical Cores for Global Atmospheric Models, Journal of the Meteorological Society of Japan. Ser. II, 85B, 241–269, https://doi.org/10.2151/jmsj.85B.241, 2007.

Williamson, D. L., Drake, J. B., Hack, J. J., Jakob, R., and Swarztrauber, P. N.: A standard test set for numerical approximations to the shallow water equations in spherical geometry, Journal of Computational Physics, 102, 211–224, https://doi.org/10.1016/s0021-9991(05)80016-6, 1992.