Monthly Notices of the ROYAL ASTRONOMICAL SOCIETY

MNRAS **522,** 982–1008 (2023) https://doi.org/10.1093/mnras/stad1043
Advance Access publication 2023 April 8

High-order discontinuous Galerkin hydrodynamics with sub-cell shock capturing on GPUs

Miha Cernetic [1]★ Volker Springel [1] Thomas Guillet [2] and Rüdiger Pakmor [1]

[1] *Max Planck Institut für Astrophysik, Karl-Schwarzschild-Straße 1, D-85748 Garching bei München, Germany*
[2] *Physics and Astronomy, University of Exeter, Exeter EX4 4QL, UK*

Accepted 2023 April 4. Received 2023 April 1; in original form 2022 August 23

ABSTRACT

Hydrodynamical numerical methods that converge with high-order hold particular promise for astrophysical studies, as they can in principle reach prescribed accuracy goals with higher computational efficiency than standard second- or third-order approaches. Here we consider the performance and accuracy benefits of Discontinuous Galerkin (DG) methods, which offer a particularly straightforward approach to reach extremely high order. Also, their computational stencil maps well to modern GPU devices, further raising the attractiveness of this approach. However, a traditional weakness of this method lies in the treatment of physical discontinuities such as shocks. We address this by invoking an artificial viscosity field to supply required dissipation where needed, and which can be augmented, if desired, with physical viscosity and thermal conductivity, yielding a high-order treatment of the Navier–Stokes equations for compressible fluids. We show that our approach results in sub-cell shock capturing ability, unlike traditional limiting schemes that tend to defeat the benefits of going to high order in DG in problems featuring many shocks. We demonstrate exponential convergence of our solver as a function of order when applied to smooth flows, such as the Kelvin–Helmholtz reference problem of Lecoanet et al. We also demonstrate excellent scalability of our GPU implementation up to hundreds of GPUs distributed on different compute nodes. In a first application to driven, subsonic turbulence, we highlight the accuracy advantages of high-order DG compared to traditional second-order accurate methods, and we stress the importance of physical viscosity for obtaining accurate velocity power spectra.

Key words: hydrodynamics – shock waves – turbulence – methods: numerical.

1 INTRODUCTION

Computational fluid dynamics has become a central technique in modern astrophysical research (for reviews see e.g. Trac & Pen 2003; Vogelsberger et al. 2020; Andersson & Comer 2021). It is used in numerical simulations to advance the understanding of countless systems, ranging from planet formation (e.g. Nelson et al. 2000) over the evolution of stars (e.g. Edelmann et al. 2019), and the interplay of gas, black holes, and stars in galaxy formation (e.g. Weinberger et al. 2017), up to extremely large scales involving clusters of galaxies (e.g. Dolag et al. 2009) or the filaments in the cosmic web (e.g. Mandelker et al. 2019).

This wide breadth of scientific applications is also mirrored in a bewildering diversity of numerical discretization schemes. Even so the underlying equations for thin, non-viscous gases – the Euler equations – are the same in a broad class of astrophysical studies, the commonly applied numerical methods come in many different flavours, and are sometimes based on radically different principles. At a basic level, one often distinguishes between Lagrangian and Eulerian discretization schemes. The former partition the gas into elements of (nearly) constant mass, as done for example in the popular smoothed particle hydrodynamics (SPH) approach (e.g. Monaghan 1992) and its many derivatives. In contrast, the latter discretize the volume using a stationary (often Cartesian) mesh (e.g. Stone & Norman 1992), such that the fluid is represented as a field. Hybrid approaches, which for example use an unstructured moving-mesh (Springel 2010) are also possible.

For mesh-based codes, finite-volume and finite-element methods are particularly popular. In the finite-volume approach, one records the averaged state in a cell, which is updated in time by the numerical scheme. This approach combines particularly nicely with the conservative character of the Euler equations, because the updates of the conserved quantities in each cell can be expressed as pair-wise fluxes through cell boundaries, yielding not only a manifestly conservative approach but also a physically intuitive formulation of the numerical method. In finite-element approaches one instead expands the fluid state in terms of basis functions. In spectral methods, the support of the basis functions can be the full simulation domain, for example if Fourier series are used to represent the system.

Discontinuous Galerkin (DG) approaches (first introduced for non-linear problems by Cockburn & Shu 1989), which are the topic of this paper, are a particular kind of finite-element approaches in which a series expansion for the solution is carried out separately within each computational cell (which can have a fairly general shape). Inside a cell, it is thus simply a truncated spectral method. The solutions for each of the cells are coupled with each other, however, at the surfaces of the cells. Interestingly, high-order accuracy of

★ E-mail: cernetic@mpa-garching.mpg.de

global solutions can be obtained simply through the high order of the spectral method applied inside a cell, while it does not require continuity of the solutions at the cell interfaces. This makes it particularly straightforward to extend DG schemes to essentially arbitrarily high order, because this does not make the coupling at cell interfaces any more complicated. This is quite different from high-order finite volume schemes, where the reconstruction step requires progressively deeper stencils at high order (Janett et al. 2019).

Another advantage of the DG approach is that it allows in principle cells of different convergence order to be directly next to each other (Schaal et al. 2015). This makes a spatially varying mesh resolution, or a spatially varying expansion order, more straightforward to implement than in high-order extensions of finite volume methods, where typically the high-order convergence property is compromised at resolution changes unless preserved with special treatments.

Despite these advantages, DG methods have only recently begun to be considered in astrophysics. First implementations and applications include Mocz et al. (2014); Schaal et al. (2015); Kidder et al. (2017); Velasco Romero et al. (2018); Guillet et al. (2019), as well as more recently Lombart & Laibe (2021); Deppe et al. (2022); Markert, Walch & Gassner (2022). We here focus on exploring a new implementation of DG that we developed from the ground up for use with graphical processing units (GPUs). The recent advent of exascale supercomputers has been enabled through the use of graphical processing units (GPUs) or various other types of accelerator units. The common feature of these accelerators is the capability to execute a large number of floating point operations at the expense of lower memory bandwidth and total memory per computing unit (few MBs compared to few GBs on an ordinary compute node) compared to the CPU. Another peculiarity of accelerators is that they have hundreds of computing units (roughly equivalent to CPU cores) which execute operations in a single instruction, multiple data (SIMD) mode. Since many of the newest and largest supercomputers use such accelerators, it becomes imperative to either modify existing simulation codes for their efficient use, or to write new codes optimized for this hardware from scratch.

While there are already many successes in the literature for both approaches (e.g. Schneider & Robertson 2015; Ocvirk et al. 2016; Wibking & Krumholz 2022), most current simulation work in the astrophysical literature is still being carried out with CPU codes. Certainly one reason is that large existing code bases are not easily migrated to GPUs. Another is that not all numerical solvers easily map to GPUs, making it hard or potentially impossible to port certain simulation applications to GPUs.

However, there are also numerous central numerical problems where GPU computing should be applicable and yield sizable speed-ups. One is the study of hydrodynamics with uniform grid resolutions, as needed for turbulence. In this work, we thus focus on developing a new implementation of DG that is designed to run on GPUs. We base our implementation of DG on Schaal et al. (2015) and Guillet et al. (2019), with one critical difference. We do not apply the limiting schemes described in these studies as they defeat the benefits of high-order approaches when strong shocks are present. Rather, we will revert to the idea of deliberately introducing a small amount of artificial viscosity to capture shocks, i.e. to add required numerical viscosity just where it is needed, and ideally with the smallest amount necessary to suppress unphysical oscillatory solutions. As we will show, with this approach the high-order approach can still be applied well to problems involving shocks, without having to sacrifice all high-order information on the stake of a slope limiter.

This paper is structured as follows. In Section 2, we detail the mathematical basis of the Discontinuous Galerkin discretization of hydrodynamics as used by us. In Section 3, we generalize the treatment to include source terms which involve derivatives of the fluid states, such as needed for the Navier–Stokes equations, or for our artificial viscosity treatment for that matter. We then turn to a discussion of shock capturing and oscillation control in Section 4. The following Section 5 is devoted to elementary tests, such as shock tubes and convergence tests for smooth problems. In Section 6 we then show results for 'resolved' Kelvin–Helmholtz instabilities, and in Section 7, we give results for driven isothermal turbulence and discuss to what extent DG methods improve the numerical accuracy and efficiency of such simulations. Implementation and parallelization issues of our code, in particular with respect to using GPUs, are described in Section 8, while in Section 9, we discuss the performance and scalability of our new GPU-based hydrodynamical code. Finally, we give a summary and our conclusions in Section 10.

2 DISCONTINUOUS GALERKIN DISCRETIZATION OF THE EULER EQUATIONS

The Euler equations are a system of hyperbolic partial differential equations. They encapsulate the conservation laws for mass, momentum, and total energy of a fluid, and can be expressed as

$$\frac{\partial \boldsymbol{u}}{\partial t} + \sum_{\alpha=1}^{d} \frac{\partial \boldsymbol{f}_\alpha(\boldsymbol{u})}{\partial x_\alpha} = 0, \tag{1}$$

where the sum runs over the d dimensions of the considered problem. The state vector $\mathbf{u}$ holds the conserved variables: density, momentum density, and total energy density:

$$\boldsymbol{u} = \begin{bmatrix} \rho \\ \rho \boldsymbol{v} \\ e \end{bmatrix}, \quad e = \rho u + \frac{1}{2}\rho \boldsymbol{v}^2. \tag{2}$$

To make our system complete we need an equation of state which connects the hydrodynamics pressure p with the specific internal energy u. If γ is the adiabatic index, i.e. the ratio of the specific heat of the gas at a constant pressure C_p to its specific heat at a constant volume C_v, the ideal gas equation of state is

$$p = \rho u\,(\gamma - 1)\,. \tag{3}$$

We also need to specify the second term of equation (1). The fluxes $\boldsymbol{f_\alpha}(\boldsymbol{u})$ in three dimensions are:

$$\boldsymbol{f}_1 = \begin{pmatrix} \rho v_x \\ \rho v_x v_x + p \\ \rho v_x v_y \\ \rho v_x v_z \\ (\rho e + p) v_x \end{pmatrix}, \quad \boldsymbol{f}_2 = \begin{pmatrix} \rho v_y \\ \rho v_x v_y \\ \rho v_y v_y + p \\ \rho v_y v_z \\ (\rho e + p) v_y \end{pmatrix},$$

$$\boldsymbol{f}_3 = \begin{pmatrix} \rho v_z \\ \rho v_x v_z \\ \rho v_y v_z \\ \rho v_z v_z + p \\ (\rho e + p) v_z \end{pmatrix}. \tag{4}$$

By summarizing the flux vectors into $\boldsymbol{F} = (\boldsymbol{f}_1, \boldsymbol{f}_2, \boldsymbol{f}_3)$, we can also write the Euler equations in the compact form

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot \boldsymbol{F} = 0, \tag{5}$$

which highlights their conservative character. Numerically solving this set of non-linear, hyperbolic partial differential equations is at the heart of computational fluid dynamics. Here we shall consider

the specific choice of a high-order Discontinuous Galerkin (DG) method.

2.1 Representation of conserved variables in DG

In the Discontinuous Galerkin approach, the state vector $\boldsymbol{u}^K(\boldsymbol{x}, t)$ in each cell K is expressed as a linear combination of time-independent, differentiable basis functions $\phi_l^K(\boldsymbol{x})$,

$$\boldsymbol{u}^K(\boldsymbol{x}, t) = \sum_{l=1}^{N} \boldsymbol{w}_l^K(t)\, \phi_l^K(\boldsymbol{x}), \tag{6}$$

where the $\boldsymbol{w}_l^K(t)$ are N time dependent weights. Since the expansion is carried out for each component of our state vector separately, the weights $\boldsymbol{w}_l^K$ are really vector-valued quantities with five different values in 3D for each basis l. Each of these components is a single scalar function with support in the cell K.

The union of cells forms a non-overlapping tessellation of the simulated domain, and the global numerical solution is fully specified by the set of all weights. Importantly, no requirement is made that the piece-wise smooth solutions within cells are continuous across cell boundaries.

We shall use a set of orthonormal basis functions that is equal in all cells (apart from a translation to the cell's location), and we specialize our treatment in this paper to Cartesian cells of constant size. The DG approach can however be readily generalized to other mesh geometries, and to meshes with variable cell sizes. Also, we will here use a constant number N of basis functions that is equal for all cells, and determined only by the global order p of the employed scheme. In principle, however, DG schemes allow this be varied from cell to cell (the so-called p-refinement).

2.2 Time evolution

To derive the equations governing the time evolution of the DG weights w_l^K, we start with the original Euler equation from equation (5), multiply it with one of the basis functions and integrate over the corresponding cell K:

$$\int_K \phi_l^K \frac{\partial \boldsymbol{u}}{\partial t} \mathrm{d}\boldsymbol{x} + \int_K \phi_l^K\, \nabla \boldsymbol{F}\, \mathrm{d}\boldsymbol{x} = 0. \tag{7}$$

Integration by parts of the second term and applying the divergence theorem leads to the so-called weak formulation of the conservation law:

$$\int_K \phi_l^K \frac{\partial \boldsymbol{u}}{\partial t} \mathrm{d}\boldsymbol{x} + \int_{\partial K} \phi_l^K\, \boldsymbol{F}\, \mathrm{d}\boldsymbol{n} - \int_K \nabla \phi_l^K\, \boldsymbol{F}\, \mathrm{d}\boldsymbol{x} = 0, \tag{8}$$

where $|K|$ stands for the volume of the cell (or area in 2D).

If we now insert the basis function expansion of $\boldsymbol{u}$ and make use of the orthonormal property of our set of basis functions,

$$\int_K \phi_l^K(\boldsymbol{x}) \phi_m^K(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} = \delta_{l,k} |K|, \tag{9}$$

we obtain a differential equation for the time evolution of the weights:

$$|K| \frac{\mathrm{d}\boldsymbol{w}_l^K}{\mathrm{d}t} = \int_K \nabla \phi_l^K\, \boldsymbol{F}\, \mathrm{d}\boldsymbol{x} - \int_{\partial K} \phi_l^K\, \boldsymbol{F}^\star(\boldsymbol{u}^+, \boldsymbol{u}^-)\, \mathrm{d}\boldsymbol{n}. \tag{10}$$

Here we also considered that the flux function at the surface of cells is not uniquely defined if the states that meet at cell interfaces are discontinuous. We address this by replacing $\boldsymbol{F}(\boldsymbol{u})$ on cell surfaces with a flux function $\boldsymbol{F}^\star(\boldsymbol{u}^+, \boldsymbol{u}^-)$ that depends on both states at the interface, where $\boldsymbol{u}^+$ is the outwards facing state relative to $\boldsymbol{n}$ (from the neighbouring cell), and $\boldsymbol{u}^-$ is the state just inside the cell. We will typically use a Riemann solver for determining $\boldsymbol{F}^\star$, making this akin to Godunov's approach in finite volume methods. In fact, the same type of exact or approximate Riemann solvers can be used here as well. We use for ordinary gas dynamics a simplified version of the Riemann HLLC solver by Toro (2009) as implemented in the AREPO code (Springel 2010; Weinberger, Springel & Pakmor 2020). We have also included an exact Riemann solver in case an isothermal equation of state is specified.

What remains to be done to make an evaluation of equation (10) practical is to approximate both the volume and surface integrals numerically, and to choose a specific realization for the basis functions. We shall briefly discuss both aspects below. Another ingredient is the definition of the weights for the initial conditions. Thanks to the completeness of the basis, they can be computed by projecting the state vector $\boldsymbol{u}(\boldsymbol{x})$ of the initial conditions onto the basis functions ϕ_l^K of each cell:

$$\boldsymbol{w}_l^K = \frac{1}{|K|} \int_K \boldsymbol{u}\, \phi_l^K\, \mathrm{d}V. \tag{11}$$

If a finite number N of basis functions is used to approximate the numerical solution, the total approximation error is then

$$L1 = \frac{1}{|K|} \int_K \left| \boldsymbol{u}(\boldsymbol{x}) - \sum_{l=1}^{N} \boldsymbol{w}_l^K\, \phi_l^K(\boldsymbol{x}) \right| \mathrm{d}V. \tag{12}$$

We shall use this L1 norm to examine the accuracy of our code when analytic solutions are known.

2.3 Legendre basis function

Following Schaal et al. (2015), we select Legendre polynomials $P_l(\xi)$ to construct our set of basis functions. They are defined on a canonical interval $[-1, 1]$ and can be scaled such that they form an orthogonal basis with normalization chosen as:

$$\int_{-1}^{1} P_l(\xi) P_m(\xi) \mathrm{d}\xi = 2\, \delta_{l,m}. \tag{13}$$

Note that the 0-th order Legendre polynomial is just a constant term, while the 1-st order features a simple pure linear dependence. In general, $P_l(\xi)$ is a polynomial of degree l.

Within each cell, we define local coordinates $\boldsymbol{\xi} \in [-1, 1]^d$. The translation between global coordinates $\boldsymbol{x}$ to local cell coordinates $\boldsymbol{\xi}$ is:

$$\boldsymbol{\xi}^K = \frac{2}{h} \left(\boldsymbol{x} - \boldsymbol{x}_c^K \right), \tag{14}$$

with h being the cell size in one dimension, and $\boldsymbol{x}_c^K$ is the cell centre in world coordinates. Multidimensional basis functions are simply defined as Cartesian products of Legendre polynomials, for example in three dimensions as follows:

$$\phi_l^K(\boldsymbol{x}) = P_l^{\mathrm{3D}}[\boldsymbol{\xi}^K(\boldsymbol{x})], \tag{15}$$

with

$$P_l^{\mathrm{3D}}[\boldsymbol{\xi}^K] \equiv P_{l_x}\left(\xi_x^K\right) \cdot P_{l_y}\left(\xi_y^K\right) \cdot P_{l_z}\left(\xi_z^K\right), \tag{16}$$

where the generalized index l enumerates different combinations of Legendre polynomials $l_x(l)$, $l_y(l)$, and $l_z(l)$ in the different directions. In practice, we truncate the expansion at a predefined order n, and discard all tensor products in which the degree of the resulting polynomial exceeds n. This means that we end up in 3D with

$$N^{\mathrm{3D}}(n) = \frac{1}{6}(n+1)(n+2)(n+3) \tag{17}$$

basis functions, each a product of three Legendre polynomials of orders $l_{z,y,z} \in \{0, \ldots, n\}$. In 2D, we have

$$N^{2D}(n) = \frac{1}{2}(n+1)(n+2), \tag{18}$$

and in 1D the number is $N^{1D}(n) = n + 1$. The expected spatial convergence order due to the leading truncation error is in each case $p = n + 1$. From now on we will refer to p as the order of our DG scheme, with $n = p - 1$ being the highest degree among the involved Legendre polynomials.

In Fig. 1, we show an example of approximating a smooth function with Legendre polynomials of different order and with a different number of cells, but keeping the number of degrees constant. In this case, the approximation error tends to be reduced by going to higher order, even when this implies using fewer cells.

2.4 Gaussian quadrature

An integration of a general function $f(x)$ over the interval $[-1, 1]$ can be approximated by Gaussian quadrature rules, as

$$\int_{-1}^{1} f(x)\,\mathrm{d}x \simeq \sum_{j=1}^{n_g} g_j\, f(x_j) \tag{19}$$

for a set of evaluation points x_j and suitably chosen quadrature weights g_j. We use ordinary Gaussian quadrature with internal points only. The corresponding integration rule with n_g evaluation points is exact for polynomials up to degree $2n_g - 1$. If we use Legendre polynomials up to order n, we therefore should use at least $n_g \geq (n+1)/2$ integration points. Note, however, that the non-linear dependence of the flux function on the state vector $\boldsymbol{u}$ means that we actually encounter rational functions as integrands and not just simple polynomials. As a result, we need unfortunately a more conservative number of integration points for sufficient accuracy and stability in practice. A good heuristic is to take the number of basis functions used for the 1D case as a guide, so that one effectively employs at least one function evaluation per basis function. This means we pick $n_g = n + 1$ in what follows.

Multidimensional integrations, as needed for the surface and volume integrals in our Cartesian setup, can be carried out through tensor products of Gaussian integrations. We denote the corresponding function evaluation points as $\boldsymbol{\xi}_{\boldsymbol{j}}^{\mathrm{vol}} = (x_{j_1}, x_{j_2}, x_{j_3})$ and Gaussian weights as $g_{\boldsymbol{j}}^{\mathrm{vol}} = g_{j_1} \cdot g_{j_2} \cdot g_{j_3}$ for the combination $\boldsymbol{j} = (j_1, j_2, j_3)$ of Gaussian quadrature points needed for integrations over the cell volume in 3D. For surface integrations over our cubical cells, we correspondingly define $\boldsymbol{\xi}_{\boldsymbol{k},x+}^{\mathrm{sur}} = (+1, x_{k_1}, x_{k_2})$, and $\boldsymbol{\xi}_{\boldsymbol{k},x-}^{\mathrm{sur}} = (-1, x_{k_1}, x_{k_2})$ for evaluation points on the right and left surface in the x-direction of one of our cubical cells, with $\boldsymbol{k} = (k_1, k_2)$ and likewise for the y- and z-directions. The corresponding Gaussian quadrature weights are given by $g_{\boldsymbol{k}}^{\mathrm{sur}} = g_{k_1} \cdot g_{k_2}$.

Putting everything together, we arrive at a full set of discretized evolutionary equations for the weights. For definiteness, we specify this here for the 3D case:

$$\begin{aligned}
\frac{\mathrm{d}\boldsymbol{w}_l^K}{\mathrm{d}t} = {} & \frac{1}{4}\sum_{\alpha=1}^{3}\sum_{\substack{\boldsymbol{j}\in\\ [1,n_g]^3}}\left\{ f_\alpha\left[\boldsymbol{u}^K\left(\boldsymbol{\xi}_{\boldsymbol{j}}^{\mathrm{vol}}\right)\right]\cdot\frac{\partial P_l^{3\mathrm{D}}\left(\boldsymbol{\xi}_{\boldsymbol{j}}^{\mathrm{vol}}\right)}{\partial\xi_\alpha}\right\} g_{\boldsymbol{j}}^{\mathrm{vol}} \\
& - \frac{1}{8}\sum_{\alpha=1}^{3}\sum_{\substack{\boldsymbol{k}\in\\ [1,n_g]^2}}\left\{ P_l^{3\mathrm{D}}\left(\boldsymbol{\xi}_{\boldsymbol{k},\alpha+}^{\mathrm{sur}}\right) f_\alpha^\star\left[\boldsymbol{u}^{K,\alpha+}\left(\xi_{\boldsymbol{k},\alpha-}^{\mathrm{sur}}\right), \boldsymbol{u}^K\left(\boldsymbol{\xi}_{\boldsymbol{k},\alpha+}^{\mathrm{sur}}\right)\right]\right. \\
& \left. - P_l^{3\mathrm{D}}\left(\boldsymbol{\xi}_{\boldsymbol{k},\alpha+}^{\mathrm{sur}}\right) f_\alpha^\star\left[\boldsymbol{u}^K\left(\boldsymbol{\xi}_{\boldsymbol{k},\alpha-}^{\mathrm{sur}}\right), \boldsymbol{u}^{K,\alpha-}\left(\boldsymbol{\xi}_{\boldsymbol{k},\alpha+}^{\mathrm{sur}}\right)\right]\right\} g_{\boldsymbol{k}}^{\mathrm{sur}}.
\end{aligned} \tag{20}$$

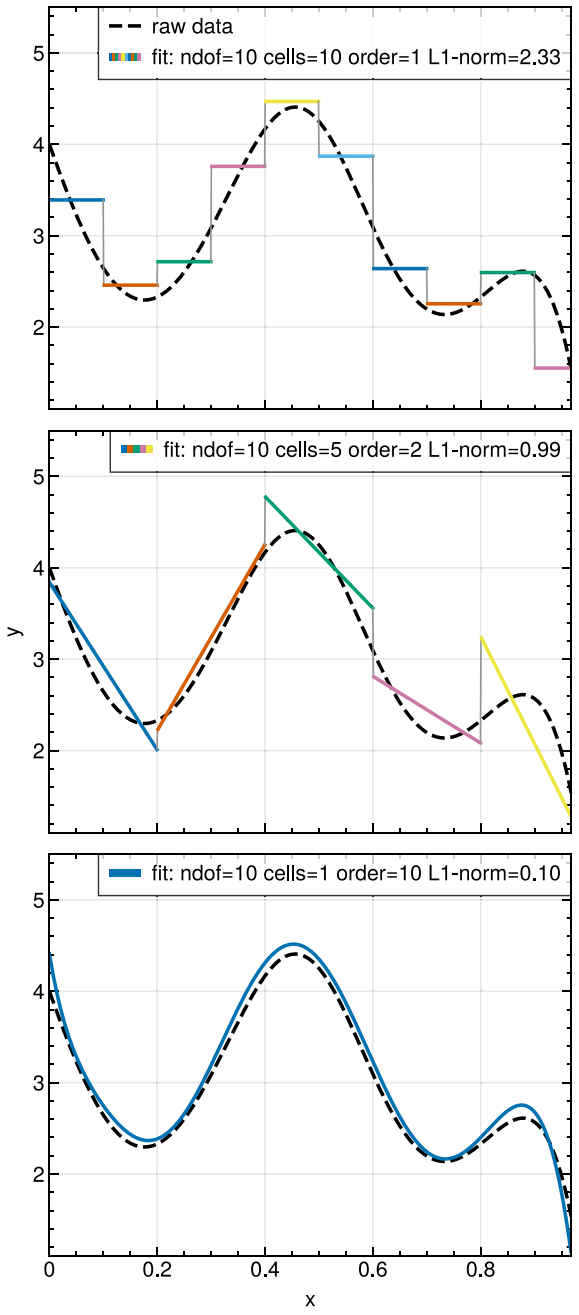

Figure 1. An example of fitting an arbitrary, smooth function $y = f(x)$ with 10 degrees of freedom, but varying number of cells and polynomial orders used for these cells, as labelled in the different panels. The L1 error norm for approximating the function is highest in case piece-wise constant approximations are used, while it drops when fewer, but piece-wise linear cells are used, and finally reaches its lowest value when a single cell with a single 10th order polynomial is used.

Here the notation $\boldsymbol{u}^{K,\alpha+}$ and $\boldsymbol{u}^{K,\alpha-}$ refer to the state vectors evaluated for the right and left neighbouring cells of cell K in the direction of axis α, respectively. The state vector evaluations themselves are given by

$$\boldsymbol{u}^K(\boldsymbol{\xi}) = \sum_{l=1}^{N} \boldsymbol{w}_l^K\, P_l^{3\mathrm{D}}(\boldsymbol{\xi}). \tag{21}$$

Note that the pre-factor $1/|K|$ in front of the surface integral terms in equation (20) turns into 1/8 as a result of the change of integration variables mediated by equation (15). The volume integral acquires a factor of $2/h$ from the coordinate transformation, thus the final pre-factor becomes 1/4. The numerical computation of the time derivative of the weights based on a current set of weights is in principle straightforward using equation (20), but evidently becomes more elaborate at high-order, involving numerous sums per cell.

In passing we note that instead of just counting the number of cells per dimensions, both the storage effort and the numerical work needed is better measured in terms of the number of degrees of freedom per dimension. A fixed number of degrees of freedom (and thus storage space) can be achieved with different combinations of cell size and expansion order. The hope in using high-order methods is that they deliver better accuracy for a fixed number of degrees of freedom, or arguably even more importantly, better accuracy at fixed computational expense.

2.5 Time integration

With

$$\dot{\boldsymbol{w}} \equiv \frac{\mathrm{d}\boldsymbol{w}_l^K}{\mathrm{d}t} \tag{22}$$

in hand, standard ODE integration methods such as the broad class of Runge–Kutta integrations can be used to advance the solution forward in time. We follow standard procedure and employ strongly positivity preserving (SPP) Runge–Kutta integration rules as defined in Schaal et al. (2015, Appendix D). Note that when higher spatial order is used, we correspondingly use a higher order time integration method, such that the time integration errors do not start dominating over spatial discretization errors. The highest time integration method we use is a 5 stage 4-th order SSP RK method.

The time-step size Δt is set conservatively as

$$\Delta t_{\max} = f_{\mathrm{CFL}} \frac{h}{2p\left(c_{s,\max} + v_{\max}\right)}, \tag{23}$$

where h is the cell size, f_{CFL} is the Courant–Friedrichs–Lewy factor, $c_{s,\max}$ denotes the global maximum sound speed, and $v_{\max}$ is the global maximum kinematic velocity, respectively. We use a f_{CFL} of 0.5 for all problems except the shock tube, Sedov blast wave, and double blast wave where a more conservative 0.3 was used instead.

For high order runs ($p > 4$) we did not see time integration errors to start dominating over the spatial discretization errors, despite employing only a 4-th order RK scheme. We attribute this to our use of a low Courant factor and to including global maximum velocities in the time-step criterion. Once the errors from time integration would start to dominate at high order, we could recover sufficient accuracy of our time integration scheme by appropriately scaling the time-step size as $h^{r/4}$.

3 TREATMENT OF VISCOUS SOURCE TERMS

As we will discuss later on, our approach for capturing physical discontinuities (i.e. shocks and contact discontinuities) in gas flows deviates from the classical slope-limiting approach and instead relies on a localized enabling of artificial viscosity. Furthermore, we will generalize our method to also account for physical dissipative terms, so that we arrive at a treatment of the full compressive Navier–Stokes equations.

To introduce these methods, we start with a generalized set of Euler equations in 3D that are augmented with a diffusion term in all fluid variables,

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot \boldsymbol{F} = \nabla \cdot (\varepsilon \nabla \boldsymbol{u}), \tag{24}$$

where $\boldsymbol{u}$ and $\boldsymbol{F}$ are the state vector (6) and the flux matrix (4), respectively.

The crucial difference between the normal Euler equations (1) and this dissipative form is the introduction of a second derivative on the right-hand side, which modifies the character of the problem from being purely hyperbolic to an elliptic type, while retaining manifest conversation of mass, momentum, and energy. This second derivative can however not be readily accommodated in our weight update equation obtained thus far. Recall, the reason we applied integration by parts and the Gauss' theorem going from equation (5) to equation (8) was to eliminate the spatial derivative of the fluxes. If we apply the same approach to $\nabla \cdot (\varepsilon \nabla \boldsymbol{u})$ we are still left with one ∇-operator acting on the fluid state.

3.1 The uplifting approach

In a seminal paper, Bassi & Rebay (1997) suggested a particular treatment of this second derivative inspired by how one typically reduces second (or higher) order ordinary differential equations (ODEs) to first order ODEs. Bassi & Rebay (1997) reduce the order of equation (24) by introducing the gradient of the state vector, $\boldsymbol{S} \equiv \nabla \boldsymbol{u}$, as an auxiliary set of unknowns. This yields a system of two partial differential equations:

$$\boldsymbol{S} - \nabla \boldsymbol{u} = 0, \tag{25}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot (\boldsymbol{F} - \epsilon \boldsymbol{S}) = 0. \tag{26}$$

Interestingly, if we consider a basis function expansion for $\boldsymbol{S}$ for each cell in the same way as done for the state vector, then the weak formulation of the first equation can be solved with the DG formalism using as input only the series expansion of the current state $\boldsymbol{u}$. This entails again an integration by parts that yields volume and surface integrations for each cell. To compute the latter, one needs to adopt a surface state $\boldsymbol{u}^\star$ for potentially discontinuous jumps $\boldsymbol{u}^+$ and $\boldsymbol{u}^-$ across the cell boundaries. Bassi & Rebay (1997) suggest to use the arithmetic mean $\boldsymbol{u}^\star = [\boldsymbol{u}^- + \boldsymbol{u}^+]/2$ for this, so that obtaining the series expansion coefficients for $\boldsymbol{S}$ is straightforward. One can then proceed to solve equation (26), with a largely identical procedure than for the Euler equation, except that the ordinary flux $\boldsymbol{F}$ is modified by subtracting the viscous flux $\boldsymbol{F}_{\mathrm{visc}} = \epsilon \boldsymbol{S}$. At cell interfaces one furthermore needs to define the viscous flux uniquely somehow, because $\boldsymbol{S}$ can still be discontinuous in general at cell interfaces. Here Bassi & Rebay (1997) suggest to use the arithmetic mean again.

A clear disadvantage of this procedure, which we initially implemented in our code, is that it significantly increases the computational cost, memory requirements, and code complexity, because the computation of $\boldsymbol{S}$ involves the same set of volume and surface integrals that are characteristic of the DG approach, except that it actually has to be done *three times* as often than for $\boldsymbol{u}$ in 3D, once for each spatial dimension. But more importantly, we have found that this method is prone to robustness problems, in particular if the initial conditions already contain large discontinuities across cells. In this case, the estimated derivatives inside a cell can reach unphysically large values by the jumps seen on the outer sides of a cell.

In hindsight, this is perhaps not too surprising. For a continuous solution, there is arguably little if anything to be gained by solving equation (25) with the DG algorithm if a polynomial basis is in use. Because this must then return a solution identical to simply taking

the derivatives of the basis functions (which are analytically known) and retaining the coefficients of the expansion. On the other hand, if there are discontinuities in $\boldsymbol{u}$ at the boundaries, the solution for $\boldsymbol{S}$ sensitively depends on the (to a certain degree arbitrary) choice made for resolving the jumps in the computation of the surface integrals for $\boldsymbol{S}$. In particular, there is no guarantee that using the arithmetic mean does not induce large oscillations or unphysical values for $\boldsymbol{S}$ in the interior of cells in certain cases.

For all these reasons we have ultimately abandoned the Bassi & Rebay (1997) method, because it does not yield a robust solution for the diffusion part or the equations in all situations, and does not converge rapidly at high order either. Instead, we conjecture that the key to high order convergence of the diffusive part of the PDE system is the availability of a consistently defined continuous solution across cell boundaries.

3.2 Surface derivatives

For internal evaluations of the viscous flux (which in general may depend on $\boldsymbol{u}$ and $\nabla\boldsymbol{u}$) within a cell, we use the current basis function expansion of the solution in the cell and simply obtain the derivative by analytically differentiating the basis functions. We argue that this is the most natural choice as the same interior solution $\boldsymbol{u}$ is used for computing the ordinary hydrodynamical flux.

The problem, however, lies with the surface terms of the viscous flux, as here neither the value of the state vector nor the gradient are uniquely defined, and unlike for the hyperbolic part of the equation, there is no suitable 'Riemann solver' to define a robust flux for the diffusion part of the equation. Simply taking arithmetic averages of the two values that meet at the interface for the purpose of evaluating the surface viscous flux is not accurate and robust in practice.

We address this problem by constructing a new *continuous* solution across a cell interface by considering the current solutions in the two adjacent cells of the interface, and projecting them onto a new joint polynomial expansion in a rectangular domain that covers part (or all) of the two adjacent cells. This approach is similar to the recovery method proposed by van Leer & Nomura (2005) in their work on solving the diffusion equation in DG. This interpolated solution minimizes the L_2 difference to the original (in general discontinuous) solutions in the two cells, but it is continuous and differentiable at the cell interface by construction. The quantities $\boldsymbol{u}$ and $\nabla\boldsymbol{u}$ needed for the evaluation of the viscous surface flux are then computed by evaluating the new basis function expansion at the interface itself.

A sketch of the adopted procedure is shown in Fig. 2. The two solutions in the two adjacent cells are given by

$$\boldsymbol{u}^{K^-}(\boldsymbol{x}) = \sum_{l=1}^{N} \boldsymbol{w}_l^{K^-}\, \phi^{K^-}(\boldsymbol{x}). \tag{27}$$

and

$$\boldsymbol{u}^{K^+}(\boldsymbol{x}) = \sum_{l=1}^{N} \boldsymbol{w}_l^{K^+}\, \phi^{K^+}(\boldsymbol{x}). \tag{28}$$

We now seek an interpolated solution in terms of a set of new basis functions $\psi^{K^\star}$ defined on the domain $K^\star$, i.e.

$$\tilde{\boldsymbol{u}}^{K^\star}(\boldsymbol{x}) = \sum_{l=1}^{N^\star} \boldsymbol{q}_l^{K^\star}\, \psi^{K^\star}(\boldsymbol{x}). \tag{29}$$

In order to avoid a degradation of accuracy if the solution is smooth, and to provide sufficient accuracy for the gradient, we adopt order $n+1$ for the polynomial basis of $\tilde{\boldsymbol{u}}^{K^\star}$. As for ordinary cells, the

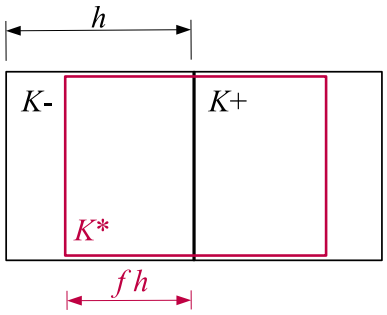

Figure 2. Two cells K^- and K^+ that meet at a joint face. The corresponding polynomial solutions $u-$ and $u+$ are in general discontinuous at the interface. To unambiguously define a joint solution and its gradient on the interface, we construct an interpolant solution on a domain $K*$ placed symmetrically around the interface. In the normal direction, a fraction f of both cells is covered (we pick either $f = 3/4$ or $f = 1$ in practice), in the transverse direction(s), the cells are covered in full.

generalized index l enumerates different combinations $[l_x(l), l_y(l), l_z(l)]$ of Legendre polynomials and their Cartesian products in the multidimensional case. If, for example, the two cells are oriented along the x-axis, we define

$$\psi_l^{K^\star}(\boldsymbol{x}) = P_{l_x}\left(\xi_x^{K^\star}\right) \cdot P_{l_y}\left(\xi_y^{K}\right) \cdot P_{l_z}\left(\xi_z^{K}\right), \tag{30}$$

where now the mapping of the x-extension of the domain $K^\star$ into the standard interval $[-1, 1]$ is correspondingly modified as

$$\xi_x^{K^\star} = \frac{1}{f\,h}\left(x - \frac{x_c^{K^-} + x_c^{K^+}}{2}\right), \tag{31}$$

where f is the fraction of overlap of each of the two cells (see Fig. 2). The coefficients $\boldsymbol{q}_l^{K^\star}$ can then be readily obtained by carrying out the projection integrals

$$\begin{aligned}\boldsymbol{q}_l^{K^\star} &= \frac{1}{|K^\star|}\int_{K^\star} \boldsymbol{u}(\boldsymbol{x})\, \psi_l^{K^\star}(\boldsymbol{x})\,\mathrm{d}V \\ &= \frac{1}{|K^\star|}\sum_{m=1}^{N}\left[\boldsymbol{w}_m^{K^-}\int_{K^-}\phi_m^{K^-}\psi_l^{K^\star}\,\mathrm{d}V + \boldsymbol{w}_m^{K^+}\int_{K^+}\phi_m^{K^+}\psi_l^{K^\star}\,\mathrm{d}V\right].\end{aligned} \tag{32}$$

The projection is a linear operation, and the overlap integrals of the Legendre basis functions can be pre-computed ahead of time. In fact, many evaluate to zero due to the orthogonality of our Legendre basis. In particular, this is the case for the transverse basis functions if their order is not equal, so that the projection effectively becomes a sparse matrix operation that expresses the new expansion coefficients in the normal direction as a sum of one or several old expansion coefficients in the normal direction. This can be more explicitly seen by defining Legendre overlap integrals as

$$A_{m,l}^{-} = \int_{-1}^{0} P_m(2fx+1)P_l(x)\,\mathrm{d}x, \tag{33}$$

$$A_{m,l}^{+} = \int_{0}^{1} P_m(2fx+1)P_l(x)\,\mathrm{d}x. \tag{34}$$

Then the new coefficients can be computed as follows

$$\boldsymbol{q}_{(l_x,l_y,l_z)}^{K^\star} = \frac{1}{2f}\sum_{m_x=0}^{l_x}\left[A_{m_x,l_x}^{-}\boldsymbol{w}_{(m_x,l_y,l_z)}^{K^-} + A_{m_x,l_x}^{+}\boldsymbol{w}_{(m_x,l_y,l_z)}^{K^+}\right]. \tag{35}$$

Note that for transverse dimensions, only the original Legendre polynomials contribute, hence the new coefficients are simply linear

combinations of coefficients that differ only in the order of the Legendre polynomial in the x-direction. Also note that for the transverse dimensions, the highest Legendre orders l_y and l_z that are non-zero are the same as for the original coefficients, i.e. the fact that we extend the order to $n+1$ becomes only relevant for the direction connecting the two cells.

Another point to note is that the basis function projection can be carried out independently for the left-hand and right-hand side of an interface (corresponding to the first and second part of the sum in equation 35), each yielding a partial result that can be used in turn to evaluate partial results for $\tilde{\boldsymbol{u}}$ $\nabla\tilde{\boldsymbol{u}}$ at the interface. Adding up these partial results then yields the final interface state and interface gradient. This means that this scheme does not require to send the coefficients $\boldsymbol{w}^{K^\pm}$ to other processors in case K^- and K^+ happen to be stored on different CPUs or GPUs, only 'left' and 'right' states for $\tilde{\boldsymbol{u}}$ and $\nabla\tilde{\boldsymbol{u}}$ need to be exchanged (which are the partial results that are then summed instead of taking their average), implying the same communication costs as, for example, methods that would rely on taking arithmetic averages of the values obtained separately for the K^- and K^+ sides.

Finally, we choose $f = 3/4$ for the size of the overlap region for $n \leq 2$, but $f = 1$ for higher order $n > 2$. For the choice of $f = 3/4$, the estimate for the first derivative of the interpolated solution ends up being

$$\nabla\tilde{\boldsymbol{u}} = \frac{\boldsymbol{u}^+ - \boldsymbol{u}^-}{h}\boldsymbol{n}, \tag{36}$$

for piece-wise constant states, where h is the cell spacing, $\boldsymbol{n}$ is the normal vector of the interface, and $\boldsymbol{u}^\pm$ are the average states in the two cells. This intuitively makes sense for low order. In particular, this will pick up a reasonable gradient even if one starts with a piece-wise constant initial conditions, and even if $n = 0$ (corresponding to DG order $p = 1$) is used. We also obtain the expected convergence orders for diffusion problems (see below) with this choice when $n \leq 2$ is used. On the other hand, we have found that it is necessary to include the full available information of the two adjacent cells by adopting $f = 1$ for still higher order in order to obtain the expected high-order convergence rates for diffusion problems also for $n > 2$.

3.3 The Navier–Stokes equations

While we will use the above form of the dissipative terms for our treatment of artificial viscosity (see below), we also consider the full Navier–Stokes equations. They are given by:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla\cdot\boldsymbol{F} = \nabla\cdot\boldsymbol{F}_{\rm NS}, \tag{37}$$

where now the Navier–Stokes flux vector $\boldsymbol{F}_{\rm NS}$ is a non-linear function both of the state vector $\boldsymbol{u}$ and its gradient $\nabla\boldsymbol{u}$. We pick the canonical form

$$\boldsymbol{F}_{\rm NS} = \begin{pmatrix} 0 \\ \boldsymbol{\Pi} \\ \boldsymbol{v}\cdot\boldsymbol{\Pi} + \chi(\gamma-1)\rho\nabla u \end{pmatrix}, \tag{38}$$

with a viscous tensor

$$\boldsymbol{\Pi} = \nu\rho\left(\nabla\boldsymbol{v} + \nabla\boldsymbol{v}^{\rm T} + \frac{2}{3}\nabla\cdot\boldsymbol{v}\right) \tag{39}$$

that dissipates shear motions with viscosity ν. We also include optional heat conduction with thermal diffusivity χ. Note that the derivatives of the primitive variables can be easily obtained from the derivatives of the conservative variables when needed, for example $\nabla\boldsymbol{v} = [\nabla(\rho\boldsymbol{v}) - \boldsymbol{v}\nabla\rho]/\rho$, and one can thus express the velocity gradient $\nabla\boldsymbol{v}$ in terms of $\nabla\boldsymbol{u}$ and $\boldsymbol{u}$.

3.4 Passive tracer

Finally, for later application to the Kelvin–Helmholtz problem, we follow Lecoanet et al. (2016) and add a passive, conserved tracer variable to the fluid equations. The density of the tracer is $c\rho$, with c being its dimension-less relative concentration. It can be added as a further row to the state vector $\boldsymbol{u}$. Since the tracer is conserved and simply advected with the local velocity, the corresponding entry in the flux vector is $c\rho\boldsymbol{v}$. Further, we can also allow for a diffusion of the tracer with diffusivity η, by adding $\eta\rho\nabla c$ in the corresponding row of the Navier–Stokes flux vector. The governing equation for the passive tracer dye is hence

$$\frac{\partial(c\rho)}{\partial t} + \nabla\cdot(c\rho\boldsymbol{v}) = \nabla(\eta\rho\nabla c). \tag{40}$$

4 SHOCK CAPTURING AND OSCILLATION CONTROL

4.1 Artificial viscosity

High-order numerical methods are prone to oscillatory behaviour around sharp jumps of density or pressure. Such physical discontinuities arise naturally at shocks in supersonic fluid motion, and they are an ubiquitous phenomenon in astrophysical gas dynamics. In fact, the Euler equations have the interesting property that perfectly initial conditions can evolve with time into states that feature real discontinuities. The physical dissipation that must happen in these jumps is implicitly dictated by the conservation laws, but discrete numerical methods may not always produce the required level of dissipation, such that post-shock oscillations are produced that are reminiscent of the Gibbs phenomenon in Fourier series expansion around jump discontinuities.

Our DG code produces these kinds of oscillations with increasing prominence at higher and higher order when discontinuities are present. And once the oscillations appear, they do not necessarily get quickly damped because of the very low numerical dissipation of high-order DG. Shocks, in particular, seed new oscillations with time, because inside cells the smooth *inviscid* Euler equations are evolved – in which there is no dissipation at all. Thus the entropy production required by shocks is simply not possible. Note that the oscillations are not only physically wrong, they can even cause negative density or pressure fluctuations in some cells, crashing the code.

One approach to prevent this are the so-called slope limiters. In particular, the family of min-mod slope limiters is highly successfully used in second-order finite volume methods. While use of them in DG methods is possible, applying them in high order settings by discarding the high-order expansion coefficients whenever the slope limiter kicks in (see Schaal et al. 2015; Guillet et al. 2019) is defeating much of the effort to going to high order in the first place. Somehow constructing less aggressive high-order limiters that can avoid this is a topic that has seen much effort in the literature, but arguably only with still limited success. In fact, the problem of coping with shocks in high-order DG is fundamentally an issue that still awaits a compelling and reasonably simple solution. Recent advanced treatments had to resort to replacing troubled cells with finite volume solutions computed on small grid patches that are then blended with the DG solution (e.g. Zanotti et al. 2015; Markert, Gassner & Walch 2021).

We here return to the idea that this problem may actually be best addressed by resurrecting the old idea of artificial viscosity (Persson & Peraire 2006). In other inviscid hydrodynamical methods, in particular in the Lagrangian technique of smoothed particle hydrodynamics, it is evident and long accepted that artificial viscosity must be added to capture shocks. Because the conservation laws ultimately dictate the amount of entropy that needs to be created in shocks, the exact procedure for adding artificial viscosity is not overly critical. What is critical, however, is that the there is a channel for dissipation and entropy production. It is also clear that shocks in DG can be captured in a sub-cell fashion only if the required dissipation is provided somehow, either through artificial viscosity that is ideally present only at the place of the shock front itself where it is really needed, or by literally capturing the shock by subjecting the 'troubled cell' to a special procedure in which it is, for example, remapped to grid of finite volume cells.

Persson & Peraire (2006) suggested to use a discontinuity (or rather oscillation) sensor to detect the need for artificial viscosity in a given cell. For this, they proposed to measure the relative contribution of the highest order Legendre basis functions in representing the state of the conserved fields in a cell. A solution of a smooth problem is expected to be dominated by the lower order weight coefficients, and statistically the low order weights should be much larger than their high order counterparts. In contrast, for highly oscillatory solutions in a cell (which often are created as pathological side-effects of discontinuities), the high order coefficients are more strongly expressed.

We adopt the same discontinuity sensor as Persson & Peraire (2006). For every cell K, we can calculate the conserved variables $\boldsymbol{u}(\boldsymbol{x})$ using either the full basis in the normal way,

$$\boldsymbol{u}(\boldsymbol{x}) = \sum_{i=1}^{N(p)} \boldsymbol{w}_l^K \phi_l$$

or by omitting the highest order basis functions that are not present at the next lower expansion order, as

$$\hat{\boldsymbol{u}}(\boldsymbol{x}) = \sum_{i=1}^{N(p-1)} \boldsymbol{w}_l^K \phi_l$$

The discontinuity/oscillatory sensor S^K in cell K can now be defined as

$$S^K = \frac{\int_K (u-\hat{u})(u-\hat{u})\mathrm{d}V}{\int_K u(x)u(x)\,\mathrm{d}V}, \tag{41}$$

where we restrict ourselves to one component of the state vector, the density field. Note that due to the orthogonality of our basis functions, this can be readily evaluated as

$$S^K = \frac{\sum_{l=N_{p-1}}^{N_p} \left[w_l^K\right]^2}{\sum_{l=1}^{N_p} \left[w_l^K\right]^2} \tag{42}$$

in terms of sums over the squared expansion coefficients. While we have $0 \leq S^K \leq 1$, we expect S^K to generally assume relatively small values even if significant oscillatory behaviour is already present in K, simply because the natural magnitude of the expansion coefficients declines with their order rapidly. Persson & Peraire (2006) argue that the coefficients should scale as $1/p^2$ in analogy with the scaling of Fourier coefficients in 1D, so that typical values for S^K in case oscillatory solutions are present may scale as $1/p^4$. Our tests indicate a somewhat weaker scaling dependence, however, for oscillatory solutions developing for identical ICs, where the troubled cells scale approximately as $S^K \sim 1/p^2$ as a function of order.

In the approach of Persson & Peraire (2006), artificial viscosity is invoked in cells once their S^K value exceeds a threshold value, above which it is ramped up smoothly as a function of S^K to a predefined maximum value. While this approach shows some success in controlling shocks in DG, it is problematic that strong oscillations need to be present in the first place *before* the artificial viscosity is injected to damp them. In a sense, some damage must have already happened before the fix is applied.

For capturing shocks we therefore argue it makes more sense to resort to a physical shock sensor which detects rapid, non-adiabatic compressions in which dissipation should occur. We therefore propose here to adapt ideas widely used in the SPH literature (Morris & Monaghan 1997; Cullen & Dehnen 2010), namely to consider a time-dependent artificial viscosity field that is integrated in time using suitable source and sink functions. Adopting a dimension-less viscosity strength $\alpha(\boldsymbol{x}, t)$, we propose the evolutionary equation

$$\frac{\partial \alpha}{\partial t} = \dot{\alpha}_{\mathrm{shock}} + \dot{\alpha}_{\mathrm{wiggles}} - \frac{\alpha}{\tau} \tag{43}$$

for steering the spatially and temporarily variable viscosity. For the moment we use a simple shock sensor $\dot{\alpha}_{\mathrm{shock}} = f_v \max(0, -\boldsymbol{\nabla}\cdot\boldsymbol{v})$ based on detecting compression, where $f_v \sim 1.0$ can be modified to influence how rapidly the viscosity should increase upon strong compression. In the absence of sources, the viscosity decays exponentially on a time-scale

$$\tau = f_\tau \frac{h}{p\, c_s}, \tag{44}$$

where h/p is the expected effective spatial resolution at order p, c_s is the local sound speed, and $f_\tau \sim 0.5$ is a user-controlled parameter for setting how rapidly the viscosity decays again after a shock transition.

Finally, the term $\dot{\alpha}_{\mathrm{wiggles}}$ in equation (43) is a further source term added to address the occurrence of oscillatory behaviour away from shocks. In fact, this typically is seeded directly ahead of strong shocks, for example when the high-order polynomials in a cell with a shock trigger oscillations in the DG cell directly ahead of the shock through coupling at the interface. Another typical situation where oscillations can occur are sharp, moving contact discontinuities. Here the shock sensor would not be effective in supplying the needed viscosity as there is no shock in the first place. We address this problem by considering the *rate of change* of the oscillatory senor S^K as a source for viscosity, in the form

$$\dot{\alpha}_{\mathrm{wiggles}} = f_w \max\left(0, \frac{\mathrm{d}\log S^K}{\mathrm{d}t}\right), \tag{45}$$

for $S^K > S_{\mathrm{onset}}$, otherwise $\dot{\alpha}_{\mathrm{wiggles}} = 0$. When $\mathrm{d}\log S^K/\mathrm{d}t$ is positive and large, oscillatory behaviour is about to grow and the cell is on its way to become a troubled cell, indicating that this should better be prevented with local viscosity. In this way, oscillatory solutions can be much more effectively controlled than waiting until they already reached a substantial size. It is nevertheless prudent to restrict the action of this viscosity trigger to cells that have S_K above a minimum value S_{onset}, otherwise the code would try to suppress even tiny wiggles, which would invariably lead to very viscous behaviour. In practice, we set $S_{\mathrm{onset}} = 10^{-4}/p^2$, and we compute $\mathrm{d}\log S^K/\mathrm{d}t$ based on the time derivatives of the weights of the previous time-step.

We add α as a further field component to our state vector $\boldsymbol{u}$, meaning that it is spatially variable and is expanded in our set of basis functions. We do not advect the α field with the local flow velocity as to allow it to fall behind moving discontinuities and to fully suppress any excited oscillations there. Also, advecting the α field at high order would require a limiting scheme for this field itself.

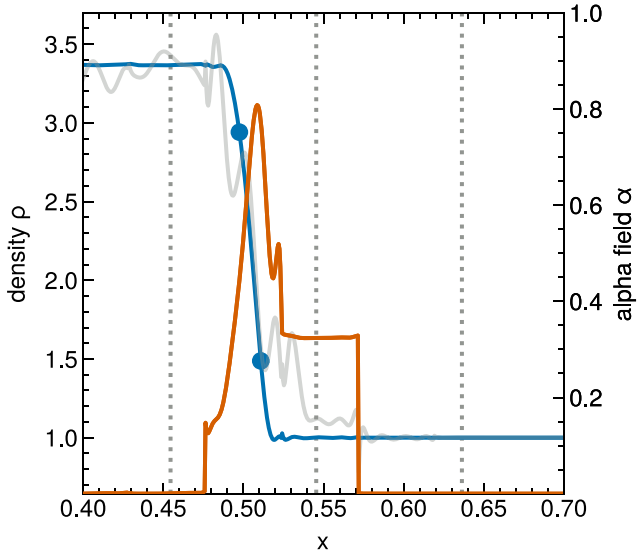

Figure 3. Zoom into a Mach number $\mathcal{M} = 4$ shock that is simulated with order $p = 9$. The upstream gas has unit density and unit pressure. Individual mesh cell boundaries are indicated with dotted lines. The density field obtained with artificial viscosity included is shown as a solid blue, while the result without artificial viscosity is shown as a grey line in the background. The artificial viscosity field itself is shown as orange line (scale on the right). The analytic shock position at the displayed time is at $x = 0.5$, in the middle of one of the mesh cells. The circles mark the locations where the density has reached 20 and 80 per cent, respectively, of the shock's density jump. We use the distance $\Delta x_{\rm shock}$ of the corresponding points as a measure of the shock width.

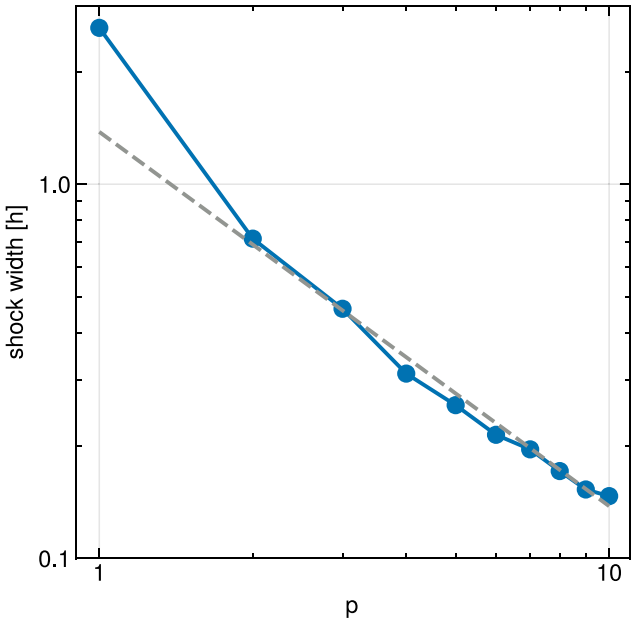

Figure 4. Shock width in units of the cell size as a function of the order p of our DG code, for a Mach number $\mathcal{M} = 4$ shock that runs into gas at rest. The dashed line marks a $\Delta x_{\rm shock} \propto 1/p$ power law, which accurately describes our measurements, except for the lowest order result with piecewise constant states, which is so highly diffusive that it does not require any artificial viscosity.

Note that in the post-/pre-shock region we can assume the first term of equation (43) to be unimportant. Once the wiggles are suppressed the second term disappears as well, so that then the default choice of parameters suppresses any existing α field to percent level in a handful of time-steps. Only the shock sensor source function is actually variable in a cell, whereas our oscillatory sensor affects the viscosity throughout a cell.

Finally, the actual viscosity applied in the viscous flux of equation (37) is parametrized as

$$\epsilon = \alpha c_s \frac{h}{p}, \tag{46}$$

and we impose a maximum allowed value of $\alpha_{\rm max} = 1$, primarily as a means to prevent overstepping and making the scheme violate the the von Neumann stability requirement for explicit integration of the diffusion equation, which would cause immediate numerical instability. Since our time-step obeys the Courant condition, this is fortunately not implying a significant restriction for effectively applying the artificial viscosity scheme, but it imposes an upper bound that can be used safely without making the time-integration unstable.

We have found that the above parametrization works quite reliably, injecting viscosity only at discontinuities and when spurious oscillations need to be suppressed, while at the same time not smoothing out solutions excessively. Fig. 3 shows an example for a Mach number $\mathcal{M} = 3$ shock that is incident from the left on gas with unit density and unity pressure, and adiabatic index $\gamma = 5/3$. The simulation has been computed at order $p = 9$, and at the displayed time, the shock position should be at $x = 0.5$, for a mesh resolution of $h = 1.0/21$. We show our DG result as a thick blue line, and also give the viscosity field $\alpha(x)$ as a red line. Clearly, the shock is captured at a fraction of the cell size, with negligible ringing in the pre- and post-shock regions. This is achieved thanks to the artificial viscosity, which peaks close to the shock centre, augmented by additional weaker viscosity in the cell ahead of the shock, which would otherwise show significant oscillations as well. This becomes clear when looking at the solution without artificial viscosity, which is included as a grey line in the background.

The blue circles in Fig. 3 mark the places in which the solution has reached 20 and 80 per cent of the height of the shock's density jump. We can operationally define the difference in the corresponding x-coordinates as the width $\Delta x_{\rm shock}$ with which the shock is numerically resolved. In Fig. 4 we show measurements of the shock width for the same set-up, except for varying the employed order p. We see that the shock width declines with higher order, accurately following the desired relationship $\Delta x_{\rm shock} \propto 1/p$, except for the lowest order $p = 1$, which deviates towards broader width compared to the general trend. The importance of this result for the DG approach can hardly be overstated, given that it has been a nagging problem for decades to reliably capture shocks at sub-cell resolution in DG without having to throw away much of the higher resolution information. The result of Fig. 4 essentially implies that shocks are resolved with the same width for a fixed number of degrees of freedom, independent of the employed order p. Whereas using higher order at a fixed number of degrees of freedom is thus not providing much of an advantage for making shocks thinner compared to using more cells, it at least does not degrade the solution. But smooth parts of a solution can then still benefit from the use of higher order.

In total our artificial viscosity method uses five parameters, one for each of the three terms of equation (43), a further general scaling factor α which is applied to the total viscous flux as defined in equation (46), and an onset threshold $S_{\rm onset}$. In this way we are able to individually control the suppression of shocks, wiggles, and the decay time of the viscous field as well as the total magnitude of

viscous flux. The default values we adopted for these parameters throughout this work are $\alpha = 1.0, f_v = 2.5, f_\tau = 0.5, f_w = 0.2$, and $S_{\rm onset} = 10^{-4}$.

4.2 Positivity limiter

With our artificial viscosity approach described above we intend to introduce the necessary numerical viscosity where needed, such that slope limiting becomes obsolete. However, for further increasing robustness of our code, it is desirable that it also runs stably if a too weak or no artificial viscosity is specified, or if its strength is perhaps locally not sufficient for some reason in a particularly challenging flow situation. To prevent a breakdown of the time evolution in this case, we consider an optional positivity limiter following Zhang & Shu (2010) and Schaal et al. (2015). This can be viewed as a kind of last line of defense against the occurrence of oscillations in a solution that ventures into the regime of unphysical values, such as negative density or pressure. The latter can happen even for arbitrarily small time-steps, especially when higher order methods are used where such robustness problems tend to be more acute.

Finite-element and finite-volume hydrodynamical codes typically employ procedures such as slope limiters to cope with these situations, this means they locally reduce the order of the scheme (effectively making it more diffusive) by discarding high-order information. A similar approach is followed by the positivity limiter described here, which is based on Schaal et al. (2015), with an important difference in how we select the evaluation points. We stress however that the positivity limiter is not designed to prevent oscillations, only to reduce them to a point that still allows the calculation to proceed.

For a given cell, we first determine the average density $\overline{\rho}$ in the cell, which is simply given by the 0-th order expansion coefficient for the density field of the given cell, and we likewise determine the average pressure $\overline{p}$ of the cell. If either $\overline{\rho}$ or $\overline{p}$ is negative, a code crash is unavoidable.

Otherwise, we define a lowest permissible density $\rho_{\rm bottom} = 10^{-6}\bar{\rho}$. Next, we consider the full set of quadrature evaluation points $\{\boldsymbol{x}_i\}$ relevant for the cell, which is the union of the points used for internal volume integrations and the points used for surface integrals on the outer boundaries of the cell. We then determine the minimum density $\rho_{\rm min}$ occurring for the field expansions among these points. In case $\rho_{\rm min} < \rho_{\rm bottom}$, which includes the possibility that $\rho_{\rm min}$ is negative, we calculate a reduction factor $f = (\bar{\rho} - \rho_{\rm bottom}) / (\bar{\rho} - \rho_{\rm min})$ and replace all higher order weights of the cell with

$$\boldsymbol{w}_l'^K = f\, \boldsymbol{w}_l^K \quad \text{for } l > 1. \tag{47}$$

This limits the minimum density appearing in any of the discrete calculations to $\rho_{\rm bottom}$. By applying the correction factor f to all fields and not just the density, we avoid to potentially amplify relative fluctuations in the velocity and pressure fields.

We proceed similarly for limiting pressure oscillations, except that here no simple reduction factor can be computed to ensure that $p_{\rm min}$ stays above $p_{\rm bottom}$, due to the non-linear dependence of the pressure on the energy, momentum, and density fields. Instead, we simply adopt $f = 0.5$ and repeatedly apply the pressure limiter until $p_{\rm min} \geq p_{\rm bottom}$.

In our test simulations the positivity limiter, as expected, does not trigger for inherently smooth problems and thus is in principle not needed. However, when starting simulations with significant discontinuities in the initial conditions, the positivity limiter usually kicks in at the start for a couple of time-steps, especially for high order simulations, until the artificial viscosity is able to tame the spurious oscillations, making the positivity limiter superfluous in the subsequent evolution.

5 BASIC TESTS

In this section, we consider a set of basic tests problems that establish the accuracy of our new code both for smooth problems, as well as for problems containing strong discontinuities such as shocks or contact discontinuities. We shall begin with a smooth hydrodynamic problem that is suitable for verifying code accuracy for the inviscid Euler equations. We then turn to testing the diffusion solver of the code, as an indirect means to test the ability of our approach to stably and accurately solve the viscous diffusion inherent in the Navier–Stokes equations. We then consider shocks and the supersonic advection of a discontinuous top-hat profile to verify the stability of our high-order approach when dealing with such flow features. Applications to Kelvin–Helmholtz instabilities and driven turbulence are treated in separate sections.

5.1 Isentropic vortex

The isentropic vortex problem of Yee, Sandham & Djomehri (1999), Yee, Vinokur & Djomehri (2000) is a time-independent smooth vortex flow, making it a particularly useful test for the accuracy of higher order methods, because they should reach their theoretically optimal spatial convergence order if everything is working well (e.g. Schaal et al. 2015; Pakmor et al. 2016). We follow here the original setup used in Yee et al. (1999) by employing a domain with extension $[-5, 5]^2$ in 2D and an initial state given by:

$$v_x(\boldsymbol{r}) = -\frac{\beta y}{2\pi} \exp\left(\frac{1-r^2}{2}\right) \tag{48}$$

$$v_y(\boldsymbol{r}) = \frac{\beta x}{2\pi} \exp\left(\frac{1-r^2}{2}\right) \tag{49}$$

$$u(\boldsymbol{r}) = 1 - \frac{\beta^2}{8\gamma\pi^2} \exp\left(1-r^2\right) \tag{50}$$

$$\rho(\boldsymbol{r}) = [(\gamma-1)u(\boldsymbol{r})]^{\frac{1}{\gamma-1}}, \tag{51}$$

where we choose $\gamma = 1.4$, and $\beta = 5$. We evolve the vortex with different DG expansion order n and different mesh resolutions $N_{\rm grid}^2$ until time $t = 10$, and then measure the resulting L1 approximation error of the numerical result for the density field relative to the analytic solution (which is identical to the initial conditions). In order to make the actual measurement of $L1$ independent of discretization effects, we use $n + 2$ Gaussian quadrature for evaluating the volume integral appearing in equation (12). Likewise, we use this elevated order when projecting the initial conditions onto the discrete realization of DG weights of our mesh.

In Fig. 5 we show measurements of the L1 error as a function of grid resolution $N_{\rm grid}$, for different expansion order from $p = 1$ to $p = 8$. The left-hand panel shows that the errors decrease as power laws with spatial resolution for fixed n, closely following the expected convergence order $L1 \propto N_{\rm grid}^{-p}$ in all cases (except for the $p = 1$ resolution, which exhibits slightly worse behaviour – but this order is never used in practice because of its dismal convergence properties).

Interestingly, the data also shows that for a given grid resolution, the L1 error goes down *exponentially* with the order of the scheme. This is shown in the right-hand panel of Fig. 5, which shows the L1 error in a log-linear plot as a function of order p, so that exponential convergence manifests in a straight decline. This particularly rapid decline of the error with p for smooth problems makes it intuitively

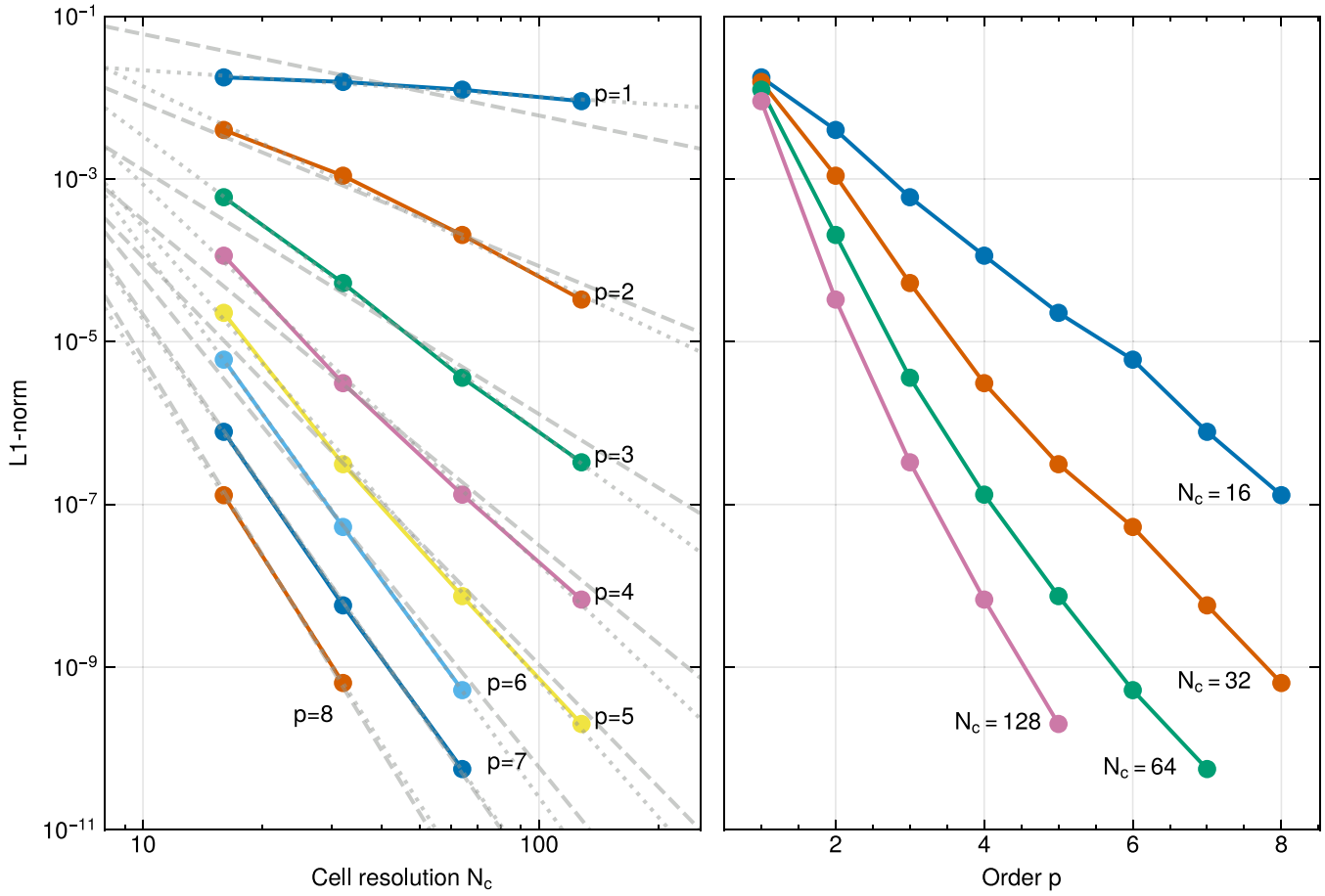

Figure 5. Convergence of the Yee et al. (1999, 2000) vortex when evolved for $t = 10.0$ time units. The left-hand panel shows the error norm in the density fields as a function of spatial grid resolution, for eight different orders p of our DG scheme. The measured convergence orders for $L1$ (coloured lines) are close to the expected $L1 \propto N_c^{-p}$ power laws (dashed grey lines). The actually achieved convergence orders (fitted power laws, shown as dotted lines) are typically even slightly better than expected, except for the lowest order $p = 1$. The panel on the right-hand side shows the same data, but as a function of DG order p, using a log-linear plot. For fixed grid resolution, the error declines *exponentially* with the order p of the scheme, highlighting the very fast improvement of accuracy when the DG order is increased. We note that the imposed periodic boundaries for the chosen box size of 10 lead to an edge effect which puts the lower boundary of the L1-norm to $\sim 10^{-11}$.

clear that it can be advantageous to go to higher resolution if the problem at hand is free of true physical discontinuities.

5.2 Diffusion of a Gaussian pulse

To test our procedures for simulating the diffusion part of our equations, in particular our treatment for estimating surface gradients at interfaces of cells, we first consider the diffusion of a Gaussian pulse, with otherwise stationary gas properties. For simplicity, we consider gas at rest and with uniform density and pressure, and we consider the evolution of a small Gaussian concentration of a passive tracer dye under the action of a constant diffusivity.

For definiteness, we consider a tracer concentration $c(\mathbf{x}, t)$ given by

$$c(\mathbf{x}, t) = c_b + \sum_{\mathbf{j}} \frac{c_g}{2\pi\sigma^2} \exp\left(-\frac{(\mathbf{x}-\mathbf{j})^2}{2\sigma^2}\right), \quad \text{with } \sigma^2 = 2\eta t, \tag{52}$$

placed in a unit domain $[-0.5, 0.5]^2$ in 2D with periodic boundary conditions. Here the sum over $\mathbf{j}$ effectively accounts for a Cartesian grid of Gaussian pulses spaced one box size apart in all dimensions to properly take care of the periodic boundary conditions. If we adopt a fixed diffusivity η and initialize $c(\mathbf{x}, t)$ at some time t_0, then the analytic solution of equation (40) tells us that equation (52) will also describe the dye concentration at all subsequent times $t > t_0$.

For definiteness, we choose $\eta = 1/128$, $c_b = 1/10$, $c_g = 1$, and $t_0 = 1$, and examine the numerically obtained results at time $t = t_0 + 3 = 4$ by computing their L1 error norm with respect to the analytic solution. In the top panel of Fig. 6, we show the convergence of this diffusion process as a function of the number of grid cells used, for the first five DG expansion orders. Reassuringly, the L1 error norm decays as a power law with the cell size, in each case with the expected theoretical optimum $L1 \propto N_{\rm cells}^{-p}$. This shows that our treatment of the surface derivatives is not only stable and robust, but is also able to deliver high-order convergence.

The bottom panel of Fig. 6 shows that this also manifests itself in an exponential convergence as function of DG expansion order when the mesh resolution is kept fixed. For this result, we adopted $N_{\rm cells} = 8$ and went all the way to 10-th order.

While these results do not directly prove that our implementation is able to solve the full Navier–Stokes equations at high-order, they represent an encouraging pre-requisite. Also, we note that both the version without viscous source terms (i.e. the Euler equations), as well as the viscous term itself when treated in isolation converges

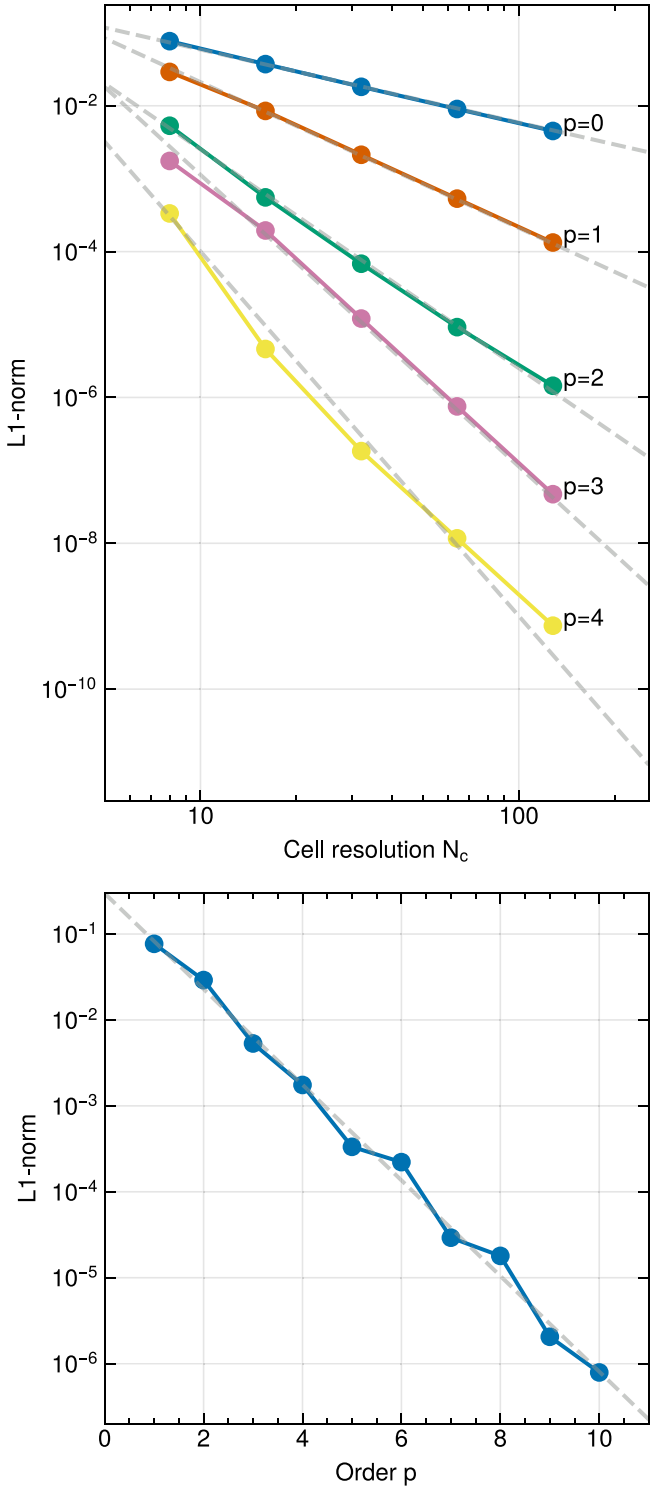

Figure 6. Convergence of the diffusion process of a Gaussian profile when started from a smooth state. The top panel shows results for runs carried out at different mesh resolution N_c and DG expansion order p, as labelled. For fixed expansion order, the L1 error declines as a power law as a function of the spatial grid resolution, with the slope of the expected convergence rate. In the bottom panel, we show the error as a function of order at a fixed grid resolution of $N_c = 8$. In this case, the error declines exponentially as a function of the expansion order.

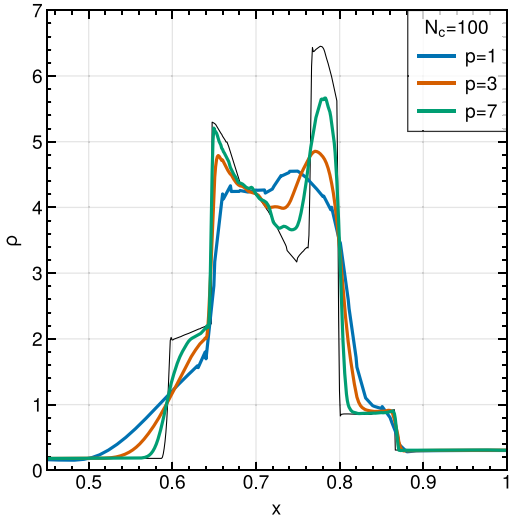

Figure 7. Double blast wave problem at fixed spatial resolution, but for increasing DG order. This shows clearly that our new artificial viscosity method can cope with strong shocks, and that adding higher order information is still worthwhile in treating problems with strongly interacting shocks. For reference, a high resolution result with $N_c = 10\,000$, $p = 1$ is shown as thin black line.

at high order. We will later on compare to a literature result for the Kelvin–Helmholtz instability in a fully viscous simulation to back up this further and to test a situation where the full Navier–Stokes equations are used.

5.3 Double blast wave

To test the ability of our DG approach to cope with strong shocks, particularly at high order, we look at the classic double blast wave problem of Woodward & Colella (1984). The initial conditions are defined in the 1D domain [0,1] for a gas of unit density and adiabatic index $\gamma = 7/5$, which is initially at rest. By prescribing two regions of very high pressure, $P = 1000$ for $x < 0.1$, and $P = 100$ for $x > 0.9$, in an otherwise low-pressure $P = 0.1$ background, the time evolution is characterized by the launching of very strong shock and rarefaction waves that collide and interact in complicated ways. Because of the difficulty of this test for shock-capturing approaches, it has often been studied in previous work to examine code accuracy and robustness (e.g. Stone et al. 2008; Springel 2010).

In order to highlight differences due to different DG orders, we have run deliberately low-resolution realizations of the problem, using 100 cells of equal size within the region [0,1]. We have then evolved the initial conditions with order $p = 2$, $p = 4$, or $p = 8$. Furthermore, we examine a run done with four times as many cells carried out at order $p = 2$. This latter simulation has the same number of degrees of freedom as the $p = 8$ simulation, and thus should have a similar effective spatial resolution. For comparison purposes, we use a simulation carried out with 10 000 cells at order $p = 2$, which can be taken as a result close to a converged solution. All simulations were run with our artificial viscosity implementation using our default settings for the method (which do not depend on order p).

In Fig. 7, we show the density profile at the time $t = 0.038$, as done in many previous works, based on our 100 cell runs. Clearly, the shock fronts and contact discontinuities of the problem are quite heavily smoothed out for the $p = 2$ run with 100 cells, due to the low resolution of this setup. However, the quality of the result can be progressively improved by going to higher order while keeping the

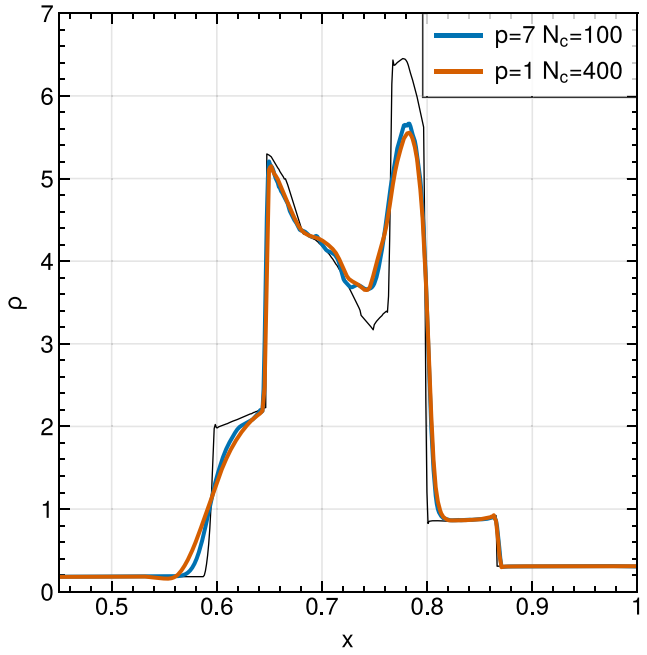

Figure 8. Double blast wave problem at fixed number of degrees of freedom for two different combinations of order and spatial resolution. This shows that for strong shocks the total number of degrees of freedom determines accuracy of our solution. For reference, a high resolution result with $N_c = 10\,000$, $p = 1$ is shown as thin black line.

number of cells fixed, as seen by the results for $p = 4$ and $p = 8$. This is in itself important. It shows that even problems dominated by very strong physical discontinuities are better treated by our code when higher order is used. The additional information this brings is not eliminated by slope-limiting in our approach, thanks to the sub-cell shock capturing allowed by our artificial viscosity technique.

Finally, in Fig. 8 we compare the $p = 7$, 100 cell result to the $p = 1$ result using 400 cells. Recall that the order of the method is $p + 1$ and the total number of degrees of freedom of the two simulations is therefore the same. We find essentially the same quality of the results, which is another important finding. This demonstrates that to first order only the number of degrees of freedom per dimension is important for determining the ability of our DG code to resolve shocks. Putting degrees of freedom into higher expansion order instead of into a larger number of cells is thus not problematic for representing shocks. At the same, it also does not bring a clear advantage for such flow structures. This is because shocks are ultimately always broadened to at least the spatial resolution limit. Real discontinuities therefore only converge with first order in spatial resolution, and high-order DG schemes do not provide a magic solution for this limitation as their effective resolution is set by the degrees of freedom. Still, as our results show, DG can be straightforwardly applied to problems with strong shocks using our artificial viscosity formulation. When there is a mixture of smooth regions and shocks in a flow, the smooth parts can still benefit from the higher order accuracy while the shocks are rendered with approximately the same accuracy as done with a second-order method with the same number of degrees of freedom.

It is interesting to compare our results to other results in the literature. First we compare to an older DG implementation with a moment limiter by Krivodonova (2007). At low orders their mode by mode limiter performs marginally better than our implementation, but as it was pointed by Vilar (2019) the mode by mode limiting does not work well for higher orders. In contrast, our method remains stable and offers steadily improving results as the order is increased. Vilar (2019) uses a DG implementation with a posteriori limiting where troubled cells are detected at the end of the time-step and then recomputed using a finite-volume method and flux reconstruction. Their approach is also able to resolve the complicated interactions of shocks and rarefaction waves and yields a steadily improving result with higher resolution as well. To demonstrate that our DG implementation is competitive with state-of-art weighted essentially non-oscillatory (WENO) schemes we compare our results with those reported by Zhao et al. (2017). They simulated this problem using an 8-th order WENO scheme with 400 grid cells. The WENO implementation performs here somewhat better at a given number of degrees of freedom compared to our DG method. Note that the number of degrees of freedom in a WENO scheme is order independent and therefore our $p = 7$ run at at 100 cells has twice the number of degrees of freedom as their 8-th order run with 400 cells, yet their result is closer to the ground truth than ours.

5.4 Advection of a top-hat pulse

Next we consider the problem of supersonically advecting a strong contact discontinuity in the form of an overdense square that is in pressure equilibrium with the background. This tests the ability of our code to cope with a physical discontinuity that is not self-steeping, unlike a shock, i.e. once the contact discontinuity is (excessively) broadened by numerical viscosity, it will invariably retain the acquired thickness. In fact, the advection errors inherently present in any Eulerian mesh-based numerical method will continue to slowly broaden a moving contact discontinuity with time, in contrast to Lagrangian methods, which can cope with this situation in principle free of any error.

A problem that starts with a perfectly sharp initial discontinuity furthermore tests the ability of our DG approach to cope with a situation where strong oscillatory behaviour is sourced in the higher order terms, an effect that is especially strong if the motion is supersonic and the system's state is characterized by large discontinuities. Here any naive implementation that does not include any type of limiter or artificial viscosity terms will invariably crash due to the occurrence of unphysical values for density and/or pressure. The square advection problem is thus also a sensitive stability test for our high-order Discontinuous Galerkin method.

In our test we follow the setup-up of Schaal et al. (2015), but see also Hopkins (2015) for a discussion of results obtained with particle-based Lagrangian codes. In 2D, we consider a domain $[0, 1]^2$ with pressure $P = 2.5$ and $\gamma = 7/5$. The density inside the central square of side length 0.5 is set to $\rho = 4$, and outside of it to $\rho = 1$. A velocity of $v_x = 100$ is added to all the gas, and in the y-direction we add $v_y = 50$. We simulate until $t = 1.0$, at which point the pulse has been advected 100 times through the periodic box in the x-direction and half that in the y-direction, and it should have returned exactly to where it started. We have also run the same test problem generalized to 3D, with an additional velocity of $v_z = -25$ in the z-direction, and as well in 1D, where only the motion in the x-direction is present. In general, the multidimensional tests behave very similarly to the 1D tests, with the size of the overall error being determined by the largest velocity. For simplicity, we therefore here restrict ourselves in the following to report explicit results for the 1D case only.

In Fig. 9, we show the density profile of the pulse at $t = 1.0$ when 10 cells per dimension are used, for different DG expansion orders p. A second-order accurate method, $p = 1$, which is equivalent or slightly better than common second-order accurate finite volume methods

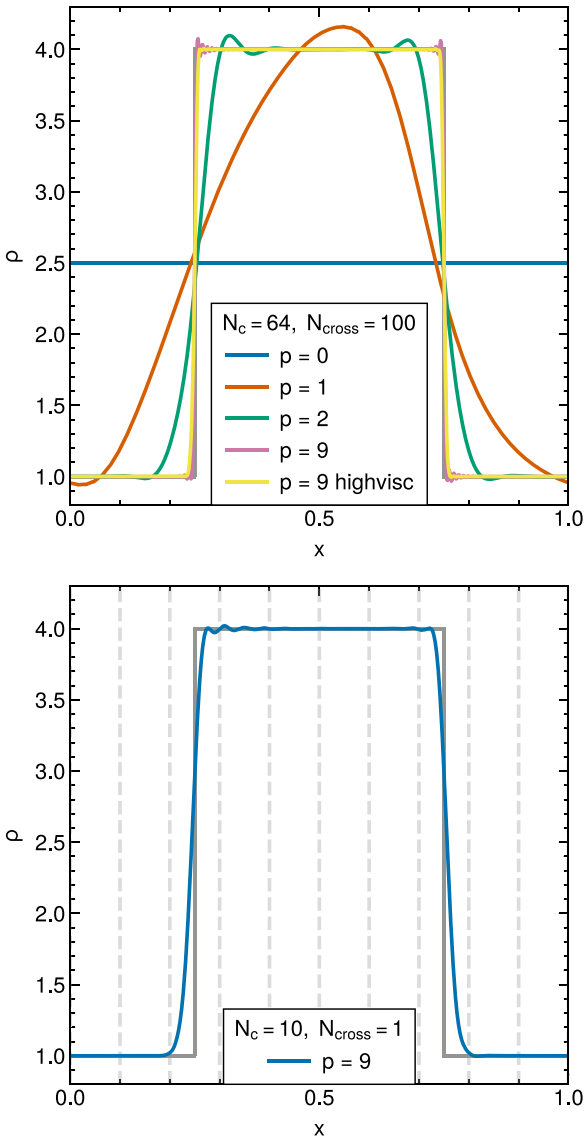

Figure 9. Top panel: Square advection problem at $t = 1.0$ for different expansion orders p using 64 grid cells in each case. At this time, the top hat profile has been advected 100 times through the box. The initial profile, which is the analytic solution in this case, is shown as a solid grey line in the background. Different numerical results are given for polynomial orders $p = 0$, 1, 2, and $p = 9$, as well as for $p = 9$ with a higher artificial viscosity setting for stronger wiggle suppression. Bottom panel: Square advection problem at $t = 0.01$ for $p = 9$ using 10 grid cells. The profile has been advected through the box once. The dotted vertical lines indicate grid cell borders. Sub-cell shock capturing can be observed.

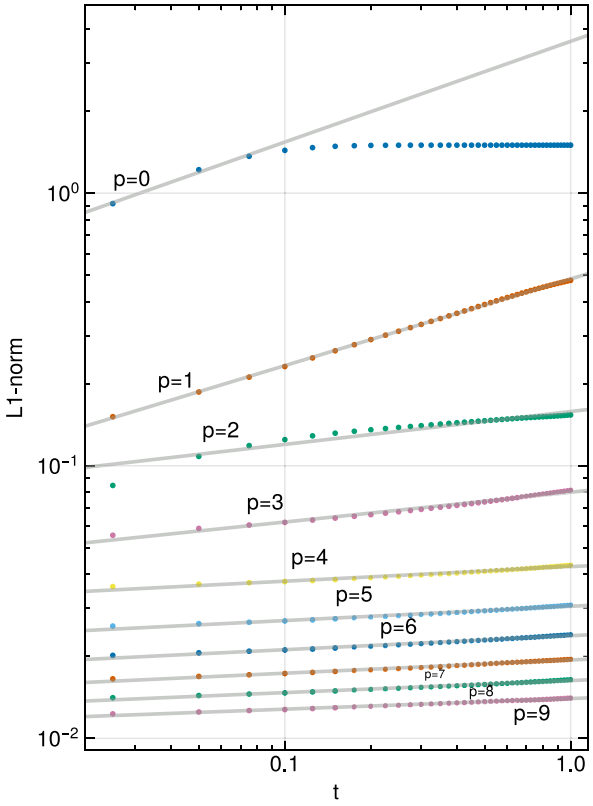

Figure 10. Time evolution of the L1 error norm for the density in the square advection problem, calculated for polynomial orders $p = 0$ to $p = 9$ (from top to bottom) using 64 grid cells in each case. The individual measurements for numerical outputs are shown with filled circles, the lines are power-law fits $L1 \propto t^n$. Note that not only the absolute error at any given time declines with increasing order p, also the slopes n become progressively shallower. This means that the numerical diffusivity of the code becomes smaller for higher order, reducing advection errors substantially. The measured slopes n for the $p = 0$ to $p = 9$ cases are in that sequence: 0.427, 0.335, 0.172, 0.056, 0.054, 0.049, 0.048, 0.046, 0.039, and 0.028. In the $p = 0$ case, only the first three points were used to measure the slope.

(see also Schaal et al. 2015) has already washed out the profile substantially at this time. Already order $p = 2$ does substantially better, with $p = 3$ results starting to resemble a top hat profile. The 10th order run with $p = 9$ is able to retain the profile very sharply, albeit with a small amount of ringing right at the discontinuities. Similarly to our results for shocks, we thus find that our code is able to make good use of higher order terms if they are available in the expansion basis. Applying simple limiting schemes such as min-mod in the high-order case is in contrast prone to lose much of the benefit of high-order when string discontinuities are present in the simulation, simply because these schemes tend to discard sub-cell information beyond linear slopes. We also show a $p = 9$ order run with higher artificial viscosity injection in regions with wiggles by using a lower $S_{onset} = 10^{-6}$. Spurious oscillations get dampened at the cost of a slightly wider shock front.

Comparing our results in the bottom part of Fig. 9 to the DG implementation with posteriori correction by Vilar (2019, their figs 5 and 7a), we can see that both methods successfully capture the sharp transition in a sub-cell fashion. For this comparison it is worth noting that in our case the shock is resolved within one cell, while in the setup of Vilar (2019) the discontinuity occurs at the border of two cells.

To look more quantitatively at the errors, we show in Fig. 10 the L1 error norm of the density field as a function of time, for all orders from $p = 0$ to $p = 9$. We see that the lowest order does very poorly on this problem, due to its large advection errors. In fact, $p = 0$ loses the profile completely after about 10 box transitions, yielding a uniform average density throughout the whole box. When one uses higher order, both the absolute error at any given time but also the rate of residual growth of the error with time drop progressively. The latter can be described by a power law $L1 \propto t^n$, with a slope n that we measure to be just 0.028 for $p = 9$, while it is still 0.335 for a second-order, $p = 1$ method. The longer a simulation runs, the

larger the accuracy advantage of a high-order method over lower order methods thus becomes.

6 KELVIN–HELMHOLTZ INSTABILITIES

Simulations of the Kelvin–Helmholtz (KH) instability have become a particularly popular test of hydrodynamical codes (e.g. Price 2008; Cha, Inutsuka & Nayakshin 2010; Junk et al. 2010; Springel 2010; Valcke et al. 2010; Berlok & Pfrommer 2019; Tricco 2019; Borrow et al. 2022), arguably initiated by an influential comparison of SPH and Eulerian codes by Agertz et al. (2007), where significant discrepancies in the growth of the perturbations in different numerical methods had been identified. One principal complication, however, is that for initial conditions with an arbitrarily sharp discontinuity the non-linear outcome is fundamentally ill-posed at the discretized level (e.g. Robertson et al. 2010; McNally, Lyra & Passy 2012), because for an ideal gas the shortest wavelengths have the fastest growth rates, so that one can easily end up with KH billows that are seeded by numerical noise at the resolution limit, rendering a comparison of the non-linear evolution between different methods unreliable. Furthermore, in the inviscid case, the non-linear outcome is fundamentally dependent on the numerical resolution so a converged solution does not even exist.

Lecoanet et al. (2016) have therefore argued that using smooth initial conditions across the whole domain combined with an explicit physical viscosity is a better choice, because this allows in principle converged numerical solutions to be reached. We follow their approach here, and in particular compare to the reference solution determined by Lecoanet et al. (2016) using the spectral code DEDALUS (Burns et al. 2020) at high resolution.

Specifically, following Lecoanet et al. (2016) we adopt as initial conditions a shear flow with a smoothed density and velocity transition:

$$\rho(x,\,y) = 1 + \frac{\Delta\rho}{\rho_0}\frac{1}{2}\left[\tanh\left(\frac{y-y_1}{a}\right) - \tanh\left(\frac{y-y_2}{a}\right)\right],$$
$$v_x(x,\,y) = u_{\rm flow}\left[\tanh\left(\frac{y-y_1}{a}\right) - \tanh\left(\frac{y-y_2}{a}\right) - 1\right], \quad (53)$$

with $u_{\rm flow} = 1$, $a = 0.05$, $y_1 = 0.5$, and $y_2 = 1.5$ in a periodic domain with side length $L = 2$. This is perturbed with a small velocity component in the y-direction to seed a KH billow on large scales:

$$v_y(x,\,y) = A\sin(2\pi x)\left[\exp\left(-\frac{(y-y_1)^2}{\sigma^2}\right) + \exp\left(-\frac{(y-y_2)^2}{\sigma^2}\right)\right], \quad (54)$$

where $A = 0.01$ and $\sigma = 0.2$ is chosen. The pressure is initialized everywhere to a constant value, $P(x,\,y) = P_0$, with $P_0 = 10$. With these choices, the flow stays in the subsonic regime with a Mach number $\mathcal{M} \sim 0.25$.

The free parameter $\Delta\rho/\rho_0$ describes the presence or absence of a density 'jump' across the two fluid phases that stream past each other. By adding a passive tracer field

$$c(x,\,y) = \frac{1}{2}\left[\tanh\left(\frac{y-y_2}{a}\right) - \tanh\left(\frac{y-y_1}{a}\right) + 2\right] \quad (55)$$

to the initial conditions, we can study the KH instability also easily for the case of a vanishing density jump. In fact, we shall focus on the case $\Delta\rho/\rho_0 = 0$ here as it is free of the particularly subtle inner vortex instability in the late non-linear evolution of the KH problem (Lecoanet et al. 2016), which further complicates the comparison of different codes.

To realize the above initial conditions we evaluate them within each cell of our chosen mesh at multiple quadrature points in order to project them onto our DG basis. We perform this initial projection using a Gauss integration that is 2 orders higher than that employed in the run itself. This ensures that integration errors from the projection of the initial conditions onto our DG basis are subdominant compared to the errors incurred by the time evolution, and are thus unimportant.

We choose identical values for shear viscosity ν, thermal diffusivity χ, and dye diffusivity η. Below, we mostly focus on discussing results for a Reynolds number $\mathrm{Re} = 10^5$ for which we set $\nu = \chi = \eta = 2u_{\rm flow}/\mathrm{Re} = 2 \times 10^{-5}$. We have also carried out simulations with a higher Reynolds number $\mathrm{Re} = 10^6$, obtaining qualitatively similar results, although these simulations require higher resolution for convergence and thus tend to be more expensive.

6.1 Visual comparison

A visual illustration of the time evolution of the dye concentration for a simulation with $\mathrm{Re} = 10^5$ and $\Delta\rho/\rho_0 = 0$ is shown in Fig. 11. In this calculation, 64 DG cells were used to cover the x-range [0,1], which is the relevant number to compare to the resolution information in Lecoanet et al. (2016). Expansion order $p = 6$ has been used in this particular run. It is nicely seen that the KH billow seeded in the initial conditions is amplified in linear evolution until a time $t \sim 1 - 2$, then it rolls up multiple times in a highly non-linear evolution, before finally strong mixing sets in that progressively washes out the dye concentration throughout the vortex.

Upon visual inspection, this time evolution compares very closely to that reported by Lecoanet et al. (2016). In Fig. 12 we make this comparison more explicit by showing results obtained for different order p at a number of times 'face-to-face' with their reference simulation. In each of the panels, the left half of the picture contains the DEDALUS result at resolution 3096×6144, while the right half gives our results at 64×128 resolution, but with different orders p. We have deliberately chosen this modest resolution for this comparison in order to allow some differences to be seen after all. They show up clearly only at second-order in the top row, while at $p = 4$ they are only discernible at times $t = 4$ and $t = 6$ as faint discontinuities at the middle of the images, where the result from DEDALUS meets that from our DG code. Already by $p = 5$, visual inspection is insufficient to identify clear differences. We note that for higher DG grid resolutions, this becomes rapidly extremely difficult already for lower orders.

6.2 Dye entropy

An interesting more quantitative comparison of our simulations to those of Lecoanet et al. (2016) can be carried out by considering the evolution of the passive scalar 'dye' in some detail. The technical implementation of this passive tracer is described in Section 3.4.

A dye entropy per unit mass can be defined as $s = -c\ln c$, and its volume integral is the dye entropy

$$S = \int \rho s\,\mathrm{d}V, \quad (56)$$

which can only monotonically increase with time. The dye entropy can be viewed as a useful quantitative measure for the degree of mixing that occurs as a result of the non-linear KH instability. To guarantee an accurate measurement of the dye entropy, we perform the integral above at two times higher spatial order than employed in the actual simulation run. We also note than when computing the dye

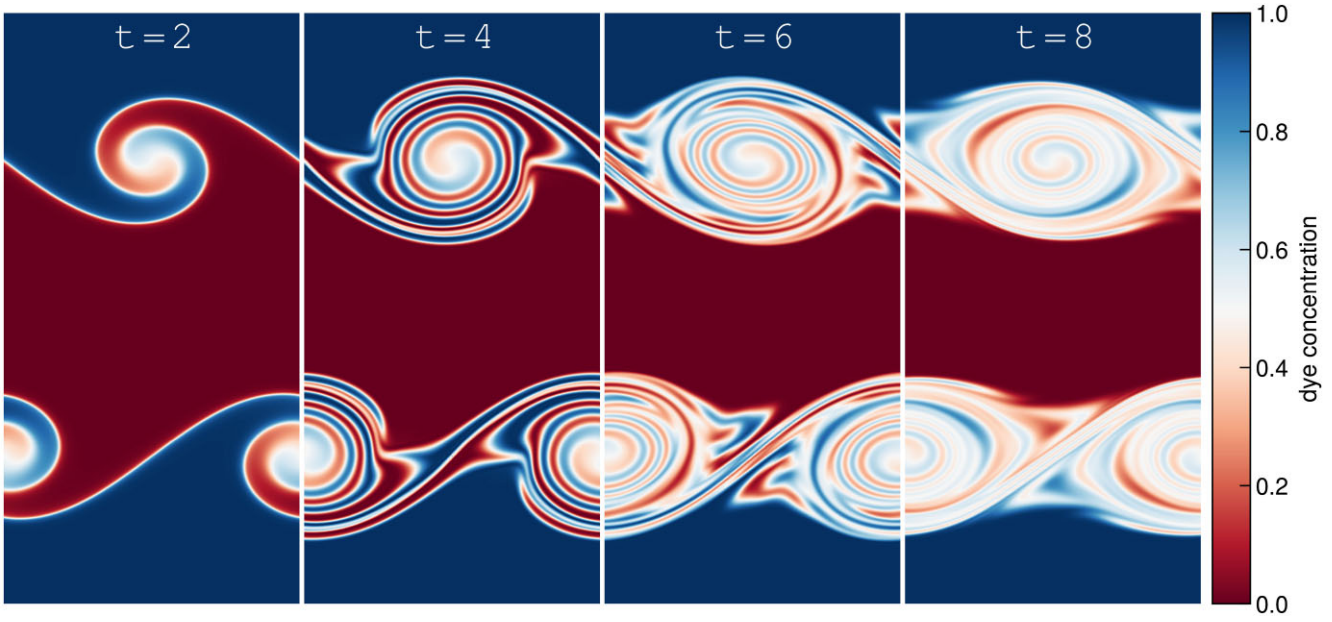

Figure 11. Time evolution of the dye concentration in a Kelvin–Helmholtz simulation using 64 DG-cells along the x-range [0,1], at order $p = 5$, using a viscosity setting of $\mathrm{Re} = 10^5$ and $\Delta\rho/\rho_0 = 0$.

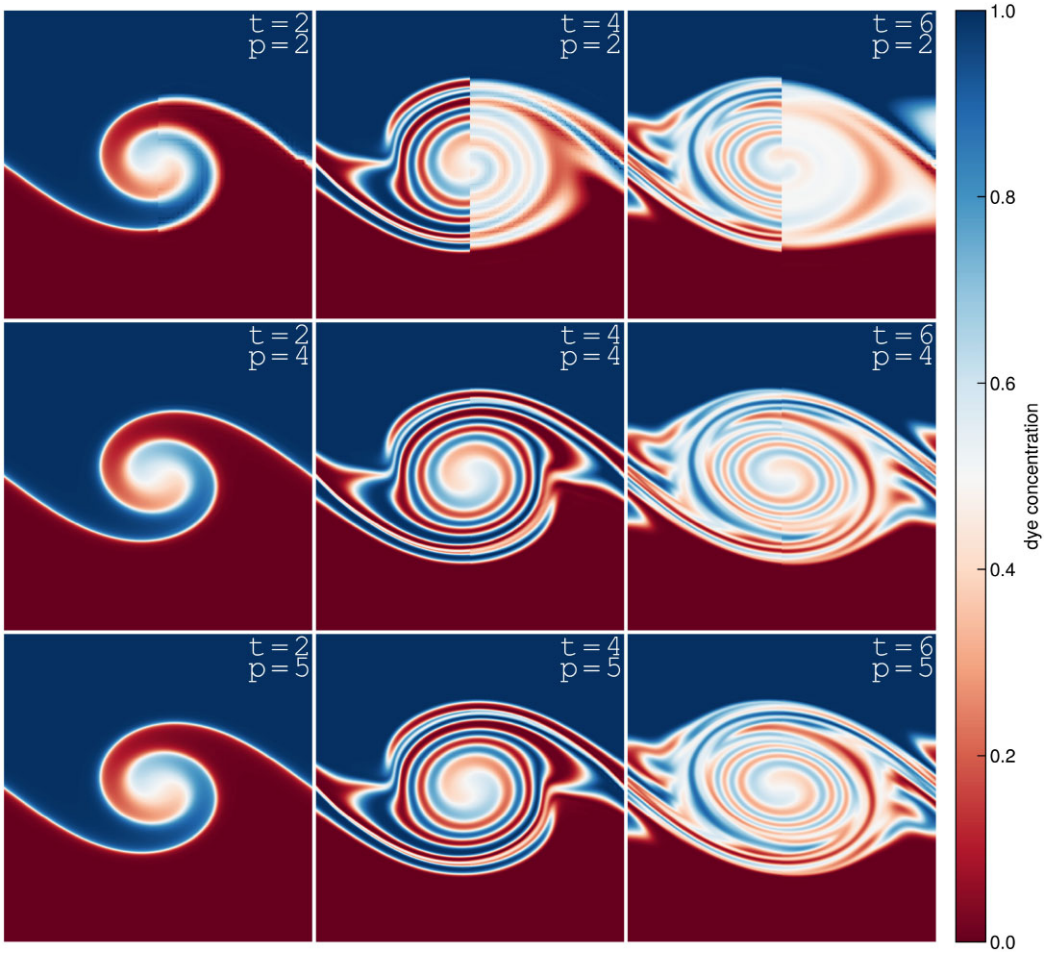

Figure 12. Dye concentration in Kelvin–Helmholtz simulations, using $\mathrm{Re} = 10^5$ and $\Delta\rho/\rho_0 = 0$, compared at fixed grid resolution but different times t and order p. Each of the nine panels shows the high-resolution DEDALUS reference result (Lecoanet et al. 2016) in the left half, and our DG result (at different order p as labelled) in the right half. All DG simulations were done with $N_c = 64$ grid cells.

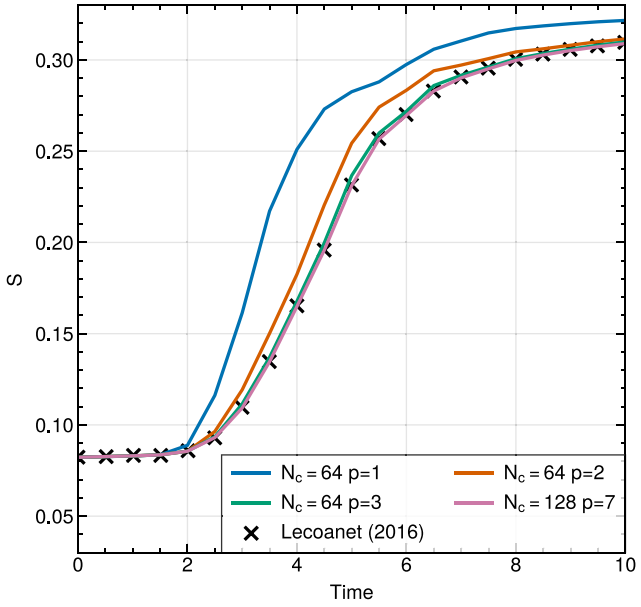

Figure 13. Volume integrated dye entropy as a function of time. We show our DG simulation results with 64 cells using orders $p = 1$ to $p = 3$, and a calculation with 128 cells and order $p = 7$. All simulations were ran with Re $= 10^5$ and a density jump $\Delta\rho/\rho_0 = 0$. Already the run with 64 cells and $p = 3$ shows an essentially converged result; still better resolutions yield perfect agreement with the very high resolution results obtained by Lecoanet et al. (2016) with the DEDALUS and ATHENA codes.

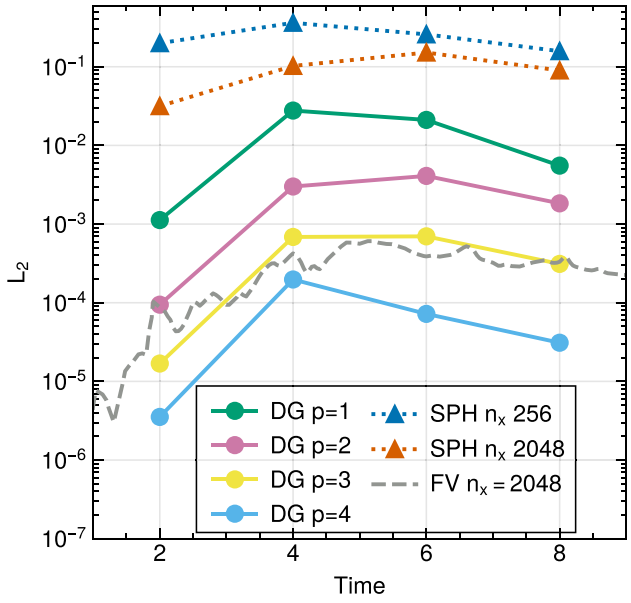

Figure 14. Volume-averaged L_2-error norm of the difference in the dye concentration relative to a high-resolution spectral result as a function of time, for a set of DG simulations carried out with 64 cells and different expansion order $p = 1$ to $p = 4$ (as labelled), for Re $= 10^5$ and a density jump $\Delta\rho/\rho_0 = 0$. The DG results are presented with filled circles at the four available output times of the spectral simulation, the connecting lines are there simply to guide the eye. Similarly, we include SPH results by Tricco (2019) as triangles at two different resolutions. Finally, the dashed line refers to the result obtained by Lecoanet et al. (2016) using the finite-volume code ATHENA with 2048 cells.

entropy we use our entire simulation domain (although left and right halves give identical values), and we then normalize to half of the volume to make our values directly comparable to those of Lecoanet et al. (2016).

In Fig. 13, we show measurements of the dye entropy evolution for several of our runs, compared to the converged results obtained by Lecoanet et al. (2016) consistently with the DEDALUS and ATHENA codes. We obtain excellent agreement already for 64 cells and order $p = 4$, corresponding to 256 degrees of freedom per dimension. Our underresolved simulations with fewer cells and/or degrees of freedom show an excess of mixing and higher dye entropy, as expected.

We note that Tricco (2019) have also studied this same reference problem using SPH. Interestingly, they find that simulations that are carried out at lower resolution than required for (approximate) convergence show an underestimate of dye mixing, marking an important qualitative difference to the mesh-based computations. The SPH simulations also require a substantially higher number of resolution elements to obtain an approximately converged result. Tricco (2019) get close to achieving this for the dye concentration by using 2048 particles per dimension, but even then the dye entropy of their result falls slightly below the converged result at $t = 8$.

6.3 Error norm

Finally, we consider a direct comparison of the dye entropy fields obtained in our simulations to the DEDALUS reference solution made publicly[1] available by Lecoanet et al. (2016) at a grid resolution of 3096 × 6144 points. To perform a quantitative comparison, we consider the L_2-norm of the difference in the dye fields, defined as

$$L_2 = \left[\frac{1}{V}\int (c_{\mathrm{DG}} - c_{\mathrm{Lecoanet}})^2 \,\mathrm{d}V\right]^{1/2}. \tag{57}$$

In Fig. 14 we show first the time evolution of the L_2-norm, for DG simulations carried out with 64 cells and orders $p = 2$ to $p = 5$. We also include results reported by Lecoanet et al. (2016) for the ATHENA mesh code at a resolution of 1024 cells, as well as SPH results by Tricco (2019) at particle resolutions of 256 and 2048, respectively. Our $p = 4$ results with 64 cells are already as good as ATHENA with 2048 cells, demonstrating that far fewer degrees of freedom are sufficient when a high order method is used for this smooth problem. In contrast, a relatively noisy method such as SPH really struggles to obtain truly accurate results. Even at the 2048 resolution, the errors are orders of magnitude larger than for the mesh-based methods, and the sluggish convergence rate of SPH will make it incredibly costly, if possible at all, to push the error down to the level of what our DG code, or ATHENA, comparably easily achieve.

In Fig. 15, we examine the error as a function of the employed DG expansion order. For a fixed number $N_c = 64$ of cells, we show the L_2 error at time $t = 4$, for orders $p = 1$ up to $p = 9$. It is reassuring that we again find exponential convergence for this problem, where the error drops approximately linearly with p on this log-linear plot. This demonstrates that we can fully retain the ability to converge at high-order for our compressible Navier–Stokes solver, which is additionally augmented with thermal and dye diffusion processes. We consider this to be a very important validation of our numerical methodology and actual code implementation.

[1] https://doi.org/10.5281/zenodo.5803759

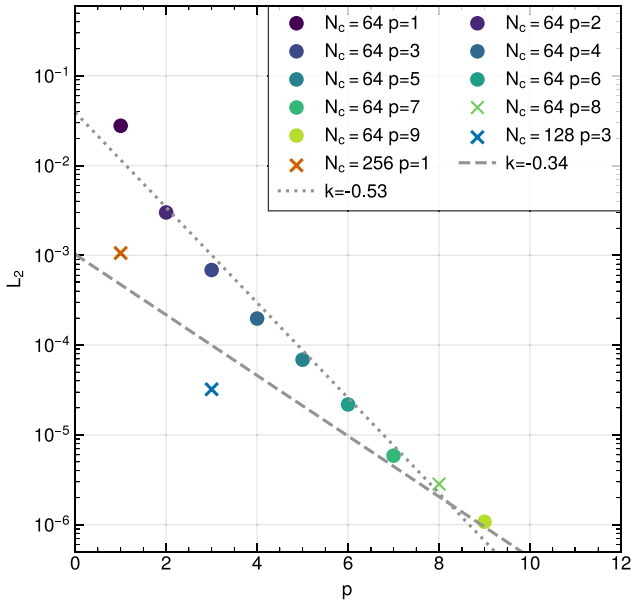

Figure 15. Volume-averaged L_2 error norm of the dye concentration field as a function of DG order p for a set of simulations with Re $= 10^5$ and a density jump $\Delta\rho/\rho_0 = 0$ at $t = 4$. The circles show simulations with $N_c = 64$ cells with progressively increasing order p (the run with $p = 8$ is shown with a cross symbol while still being a member of the sequence of simulations with circles). The crosses highlight three simulations with the same number of degrees of freedom, reached with different combinations of N_c and p. The dotted line is a fit showing the rapid convergence we achieve with increasing order p at $N_c = 64$. The dashed line indicates the convergence rate for three simulations with equal number of degrees of freedom, as we increase the order. Among the three runs with an equal number of degrees of freedom, the one with the highest order p achieves the lowest L_2-norm.

Another interesting comparison is to consider simulations that have an equal number of degrees of freedom, but different cell numbers and expansion orders. In the figure (marked with crosses), we also include results for the three cases $N_c = 64/p = 8$, $N_c = 128/p = 4$, and $N_c = 256/p = 2$, which all have the same number of degrees of freedom per dimension. Strictly speaking, the higher order ones have actually slightly less, given that the number $N^{2D}(p) = p(p + 1)/2$ of degrees of freedom per cell is slightly less than p^2 for $p > 1$, see equation (18). Regardless, the run with $N_c = 64$ clearly has the lowest error. This confirms once more that for a smooth problem it is typically worthwhile in terms of yielding the biggest gain in accuracy to invest additional degrees of freedom into higher order rather than additional cells.

7 DRIVEN SUB-SONIC TURBULENCE

The phenomenon of turbulence describes the notion of an unsteady, random flow that is characterized by the overlap of swirling motions on a variety of scales (e.g. Pope 2000). In 3D, one finds that if fluid motion is excited on a certain scale (the injection scale) it tends to decay into complex flow features on ever smaller scales, helped by fluid instabilities such as the Kelvin–Helmholtz instability. Eventually, the vortical motions become so small that they are eliminated by viscosity on the so-called dissipation scale. If the injection of kinetic energy on large scales persists and is quasi-stationary, a fully turbulent state develops which effectively exhibits a transport of energy from the injection to the dissipation scale. For incompressible isotropic, subsonic turbulence, the statistics of velocity fluctuations in such a turbulent flow are described by the Kolmogorov velocity power spectrum, which has a power-law shape in the inertial range, and a universal shape in the dissipative regime.

For astrophysics, turbulence plays a critical role in many environments, including the intracluster medium, the interstellar medium, or the buoyantly unstable regions in stars. Numerical simulations need to be able to accurately follow turbulent flows, for example in order to correctly describe the mixing of different fluid components, or the amplification of magnetic fields. However, this is often a significant challenge as the scale separation between injection and dissipation scales in astrophysical settings can be extremely large, while for 3D simulation codes it is already difficult to resolve even a moderate difference between injection and dissipation scales. In addition, most astrophysical simulations to date rely on numerical viscosity exclusively instead of including an explicit physical viscosity, something that can in principal modify the shape of the dissipative part of the turbulent power spectrum, thereby creating turbulent velocity statistics that differ from the expected universal form because they are directly affected by aspects of the numerical method.

Of course, the general accuracy of a numerical method is also important for how well turbulence can be represented. For example, Bauer & Springel (2012) have pointed out that the comparatively large noise in SPH makes it difficult for this technique to accurately represent subsonic turbulence. While this can in principle be overcome with sufficiently high numerical effort, it is clear that methods that have a low degree of numerical viscosity combined with the ability to accurately account for physical viscosity should be ideal for turbulence simulations. Our DG approach has these features, and especially in the regime of subsonic turbulence, where shocks are expected to play only a negligible role, the DG method should be particularly powerful.

This motivates us to test this idea in this section by considering isothermal, subsonic, driven turbulence in periodic boxes of unit density. The subsonic state refers to the average kinetic energy of the flow in units of the soundspeed, as measured through the Mach number. Instead of directly imposing an isothermal equation of state, we simulate gas with a normal ideal gas equation of state and reset the temperature every time-step such that a prescribed sound speed is retained. We have checked that this does not make a difference for any of our results, but this approach allows us to use our approximate, fast HLLC Riemann solver instead of having to employ our exact, but slower isothermal Riemann solver.

7.1 Driving

To create the turbulence, we drive fluid motions on large scales. To do this consistently at high order, we add a source function $\boldsymbol{s}(\boldsymbol{x}, t)$ to the right-hand side of the Euler equations, both in the momentum equation and as work function $\boldsymbol{s} \cdot \boldsymbol{v}$ in the energy equation. These source terms have to be integrated with Gaussian quadrature over cell volumes to retain the high-order discretization.

For setting up the driving field $\boldsymbol{s}(\boldsymbol{x}, t)$, we follow standard techniques as implemented in Bauer & Springel (2012), which in turn are directly based on Schmidt, Niemeyer & Hillebrandt (2006b) and Federrath, Klessen & Schmidt (2008, 2009). The acceleration field is constructed in Fourier space by populating modes in the narrow range $2\pi/L \leq k \leq 2 \times 2\pi/L$, with amplitudes scaled to follow $\propto k^{-5/3}$ over this range. The phases of the forcing modes are drawn from an Ornstein–Uhlenbeck process. They are periodically updated whenever a time interval Δt has elapsed, while keeping a temporal correlation over a time-scale t_s with the previous phases. This effectively yields a smoothly varying, random driving field.

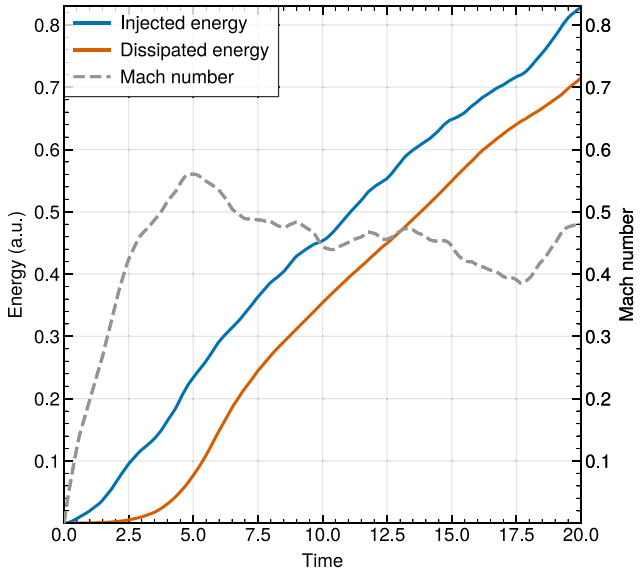

Figure 16. Cumulative injected and dissipated energy, as well as global Mach number, as a function of time in one of our driven turbulence simulations. The gas is initially at rest, and put into motion by the driving. Eventually, energy injection is balanced by dissipation in a time-averaged fashion, and the difference between the cumulative injected and dissipated energy is reflected in the kinetic energy as measured by the Mach number.

Our specific settings for update frequency, coherence time-scale, and distribution function for drawing the driving phases are as in Bauer & Springel (2012, their table 1, left-hand column).

We here also restrict ourselves to include only solenoidal driving, i.e. we project out all compressive modes in Fourier space by a Helmholtz decomposition. Specifically, if $\boldsymbol{s}$ is the principal acceleration field constructed in the above fashion, we project it as

$$\hat{\boldsymbol{s}}(\boldsymbol{k}) = \left(\delta_{ij} - \frac{k_i k_j}{\boldsymbol{k}^2}\right) \boldsymbol{s}(\boldsymbol{k}) \tag{58}$$

in Fourier space to end up with an acceleration field $\hat{\boldsymbol{s}}$ that is free of compressive modes, which would only produce a spectrum of additional sound waves in our subsonic case.

7.2 Results for subsonic turbulence

All our turbulence simulations are started with gas of uniform density at rest. We monitor the average kinetic energy, as well as the total cumulative injected kinetic energy and the total cumulative dissipated energy, allowing us to verify the establishment of a quasi-stationary state. An example for this is shown in Fig. 16, where we illustrate the build-up of the turbulent state in terms of the total energies. There is an initial ramp up phase of the turbulence until $t \sim 5$, during which the Mach number grows nearly linearly to its final quasi-stationary time-averaged value of $\mathcal{M} \simeq 0.47$. The cumulative injected energy grows approximately linearly with time, whereas the dissipated energy tracks it with a time lag, because the initial evolution until $t \sim 2.5$ does not yet show any significant dissipation. The difference between the injected and dissipated energies is the current kinetic energy of the gas, and thus is effectively given by the Mach number.

In Fig. 17, we show a visual example of the quasi-stationary turbulent state established after some time, here simulated with $N_c = 128$ cells and order $p = 4$. The slice through the magnitude of the velocity field illustrates the chaotic structures characteristic of

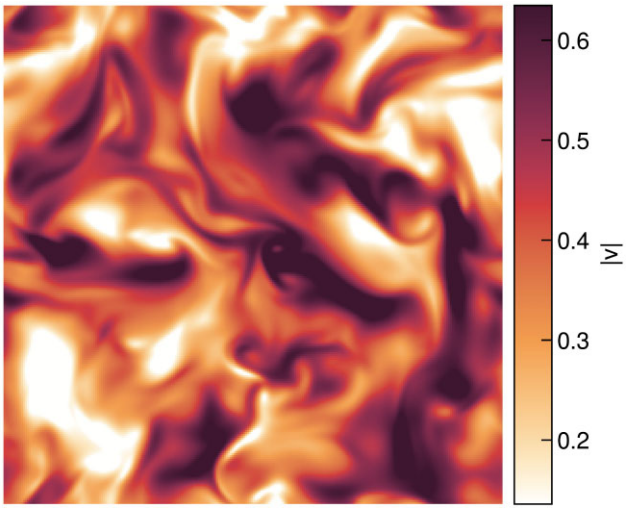

Figure 17. 2D slice through a driven, isothermal, subsonic 3D turbulence simulation depicting the velocity amplitude $|v| = (v_x^2 + v_y^2 + v_z^2)^{1/2}$ at $t = 20.48$, for a simulation with $N_c = 128$, $p = 4$, and $\mathrm{Re} = 10^5$.

turbulence. Even though there are some steep gradients in the velocity field, the velocity varies smoothly overall, reflecting the absence of strong shock waves in this subsonic case.

To statistically analyse the turbulent state, we turn to measuring power spectra of the velocity field at multiple output times, and then consider a time-average spectrum to reduce the influence of intermittency. To calculate the final power spectrum of a simulation, we average over 64 velocity power spectrum measurements over the time interval $5.12 < t < 20.48$.

7.2.1 *Inviscid treatment of gas*

The behaviour of inviscid gas is described by the Euler equations of equation (2). Because of the simplicity of this model and the desire to run simulations with as little viscosity as possible to maximize the inertial range of the turbulence, it is a popular choice for the study of turbulence. For example, the largest driven turbulence simulation to date by Federrath et al. (2016) were performed using inviscid gas, as well as many other studies in the field (e.g. Schmidt, Hillebrandt & Niemeyer 2006a; Federrath et al. 2008; Federrath et al. 2010; Price & Federrath 2010; Bauer & Springel 2012; Bauer et al. 2016).

In the top two panels of Fig. 18, we show such simulations carried out with our DG code. In all such simulation, the energy injected at large scales follows the Kolmogorov spectrum and cascades from large to small scales. This part of the spectra is called the inertial range and it follows the $k^{-5/3}$ Kolmogorov spectrum closely, even though our gas is compressible and the density fluctuations for our Mach number are not negligible any more. Note that all our plots are compensated with a $k^{5/3}$ factor, such that the Kolmogorov spectrum corresponds to a horizontal line. The extent of the inertial range is primarily determined by the total number of degrees of freedom in an inviscid simulation. However, as we transition from the inertial range to the dissipation portion of the spectra, a noticeable bump can be seen in which the spectrum significantly exceeds the power-law extrapolation from larger scales. As energy is being transferred from larger to smaller scales, creating ever smaller eddies, it eventually reaches scales at which the code cannot resolve smaller eddies any more. This leads to a build-up of an energy excess at this characteristic scale, until the implicit numerical viscosity terms become strong

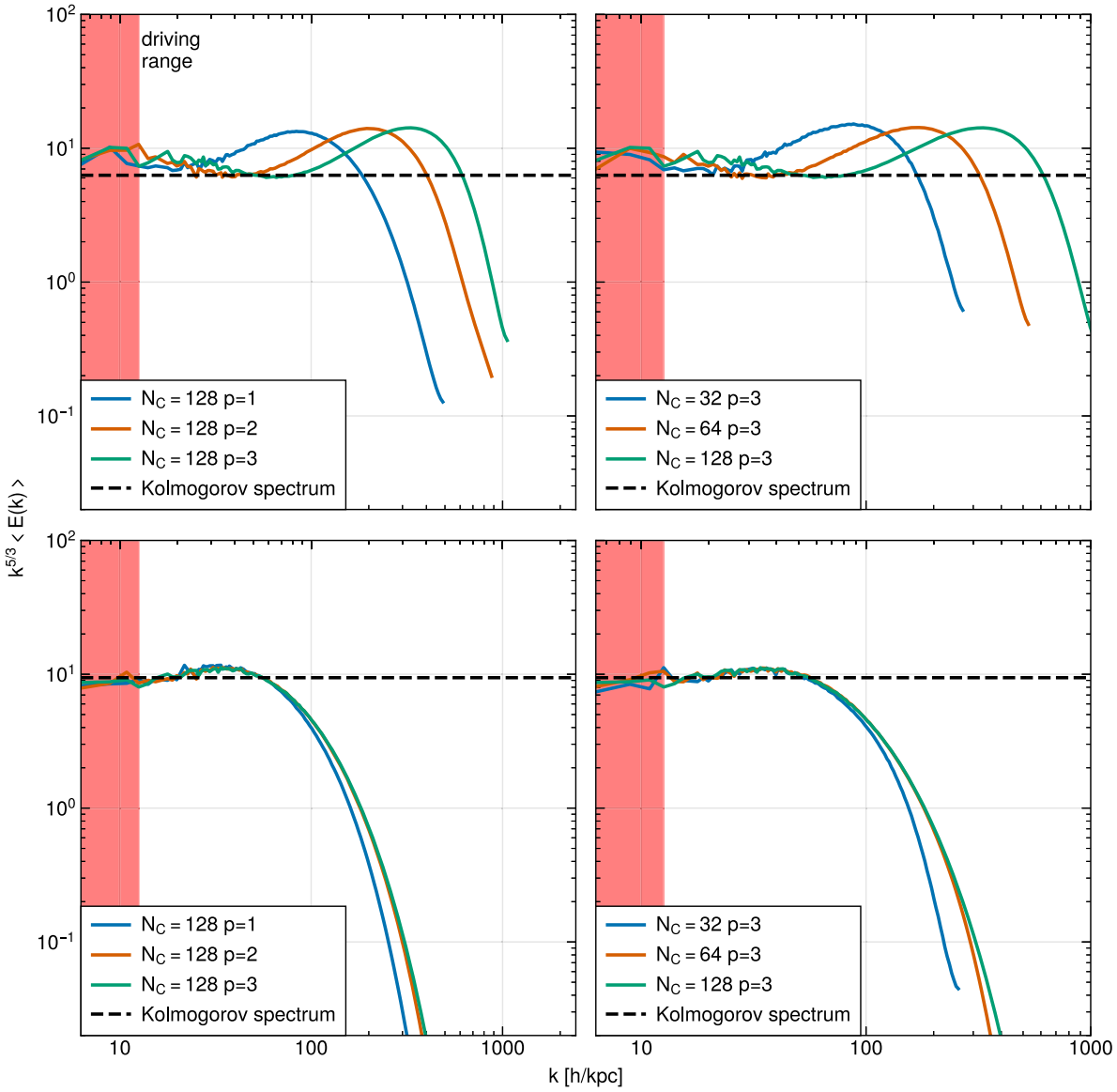

Figure 18. Compensated velocity power spectra of driven turbulence simulations as a function of wavenumber for varying numbers of cells, and varying spatial order. The panels in the top row show simulations where the Euler equations were solved, whereas the bottom two panels give results where the full compressible Navier–Stokes equations with a prescribed physical viscosity were used. The region marked with a red shade is the driving range.

enough to dissipate away the arriving energy flux. This effect is commonly known in numerical studies of turbulence and referred to as the 'bottleneck' effect. It should be pointed out that experimental determinations of turbulent velocity spectra also show a weak form of this effect (see Verma & Donzis 2007, and references therein). Küchler, Bewley & Bodenschatz (2019) later even measured the relation between the amplitude of the bump and R_λ of the flow. The problem of numerical simulations of inviscid gas is however that the shape of the bump is determined by numerical details of the hydrodynamic code and that it is usually excessively pronounced.

The bottleneck effect cannot be fixed by using higher resolution, or higher order for that matter. Indeed, in the top two panels of Fig. 18 we can see that the bottleneck moves to ever smaller scales with increasing cell number at a fixed spatial order, and similarly it moves towards smaller scales if we increase the spatial order of our method at a fixed number of cells. While both avenues of adding further degrees of freedom successfully widen the inertial range and push the dissipative regime to smaller scales, they unfortunately cannot eliminate the 'bump' in the bottleneck, or address the equally incorrect detailed shape of the dissipation regime itself. This detailed shape changes slightly as we vary the order p because the precise way of how numerical dissipation interacts with the flow is modified by this, while in contrast increasing the number of cells leaves the shape unchanged, because this just moves the dissipation regime to smaller scales in a scale-invariant fashion.

The only way around this and to get closer to velocity spectra seen in experimental studies of turbulence is to solve the full compressible Navier–Stokes equation, where the dissipative regime is set not by numerics, but by the physical viscosity of the gas itself. If this viscosity is large enough, it will effectively dissipate energy at scales larger than our numerical viscosity. We consider this case in the following subsection.

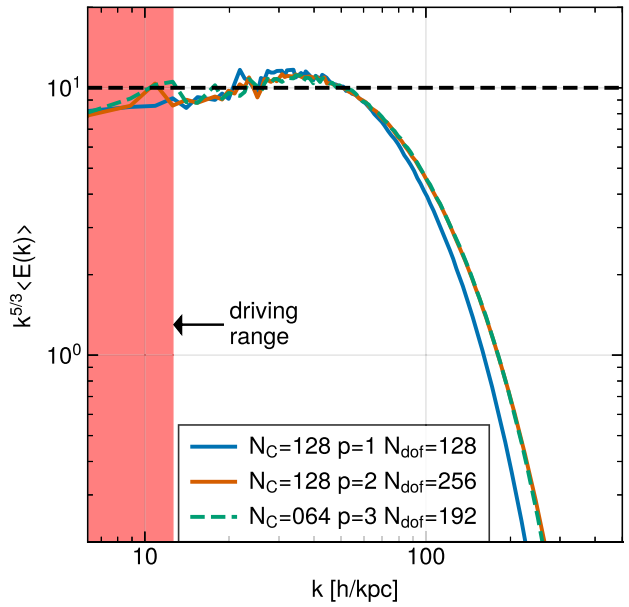

Figure 19. Compensated velocity power spectra as a function of wavenumber for a similar number of degrees of freedom, but varying the spatial order and the number of cells. The total wall-clock time for the simulation runs $128^3|p=2$, $128^2|p=3$, and $64^3|p=4$ on 16 A100 GPUs were 0.9, 3.9, and 1.8 h, respectively. We note that one can keep the converged result obtained with $N_c = 128$ and $p = 3$ by going to fewer cells and higher order (the $N_c = 64$ and $p = 4$ run), while still achieving a speed-up.

7.2.2 Viscous treatment of gas

We now consider driven turbulence results akin to the simulations just discussed, with the only difference being that we are now solving the full compressible Navier–Stokes equations as described in Section 3.3. In the bottom two panels of Fig. 18, we display compensated velocity power spectra with physical viscosity added. Such full Navier–Stokes simulations exhibit the proper behaviour of the 'bottleneck' effect, as the location and shape of the bump become resolution-independent and do not depend on numerical code details any more. Such simulations are in the literature referred to as 'direct numerical simulations' or DNS. Our code can achieve DNS for turbulence by either increasing the resolution or the spatial order, as is evident in the bottom two panels of Fig. 18.

To determine if increasing the order of our method or its resolution is more beneficial, we compare three simulations with approximately the same number of degrees of freedom, but different resolutions and orders in Fig. 19. The orange line shows a run we can consider a converged DNS result with $N_c = 128$ and $p = 3$. A simulation with identical N_c but lower p in blue fails to fully converge. On the other hand, the green dashed line shows a simulation with eight times fewer total number of cells, but at a higher spatial order. It has as many degrees of freedom as the simulation shown in blue, and yet its power spectra matches that of the simulation shown in orange. We can therefore conclude that running driven turbulence at higher order is preferable to increasing the cell resolution. Or to put it another way, if there is a limited number of degrees of freedom that can be represented due to memory constraints, it is better to 'spend' the memory on higher p than N_c. In the present case, a comparison of the wall-clock time between the high cell resolution and high order runs shows an about 2× faster calculation time at high order versus using a higher cell resolution. For even high order, this CPU-time advantage may not persist, but the memory advantage will. Given that turbulence simulations tend to be memory-bound, this in itself can already be a significant advantage.

8 CODE DETAILS

8.1 Parallelization strategy

Modern supercomputers consist by now of thousands to millions of computing cores, a trend which is bound to continue. Recently, however, the most significant gains in computational performance (measured in floating point operations per second – FLOPS) have came from dedicated accelerator cards. These are most commonly, but not always, graphics processing units (GPUs) that have been repurposed to do general computational work. Accelerators achieve a large number of FLOPS by foregoing large, per-core caches and advanced control circuitry for single compute units, while at the same time they are able to execute large sets of threads concurrently in a data-parallel fashion.

Current GPU-accelerated computers typically consist of normal, CPU-equipped compute nodes that are outfitted with attached GPU cards. Utilizing the power of both, CPUs and GPUs, efficiently with such heterogeneous machines is challenging. It requires not only a suitable subdivision of the work, but often also an algorithmic restructuring of the computations such that they can be mapped efficiently onto the massively parallel execution model of GPUs, as well as prescriptions for data placement and movement between the separate memory of CPU and GPUs. The problem becomes even harder when multiple compute nodes with distributed memory, each with their own GPUs, are supposed to work together on a tightly coupled problem. Efficient and scalable massively parallel codes for such machines must decompose the problem into multiple parts, distribute the parts among the available compute units, and only exchange data between various parts when really needed.

In the present version of our code TENETGPU,[2] we address this by an implementation that can execute a given hydrodynamical problem flexibly either on one or several GPUs, on one or multiple CPU cores, or a mixture thereof. Independent on how GPUs and CPU-cores are distributed onto different compute nodes, TENETGPU can in this way make use of whatever is available, up to extremely powerful systems such as the first exascale supercomputers that are presently put into service (which are GPU-accelerated, such as 'Frontier', ranked the fastest in the world according to the Top500 list released 2022 May 30).

To achieve this flexibility, we split the mesh along the x-axis into different slabs, which can have different thickness, if desired. Each slab is either computed by a different GPU, or by one CPU core. The communication between slabs, which is realized with the Message Passing Interface (MPI), thus needs to happen along the x-dimension between neighbouring slabs only, as all the needed data along the other two axes is locally available for the corresponding slab. The data that is communicated consists of surface states or surface fluxes at Gauss points needed for integrations over cell areas. For driving the GPU computations, each GPU requires a separate CPU core as well. For example, if one has a compute node with 32 cores and 2 GPUs as accelerator cards, a simulation with 256^3 mesh cells could be run by assigning slabs with a thickness of 98 cells to each of the GPUs, while letting the remaining 30 compute cores each work on

[2]While our code is written from scratch for GPUs, its first version has been heavily inspired by the code TENET of Schaal et al. (2015), hence we named ours TENETGPU. Source code: https://bitbucket.org/Migelo/gpu_testbed

slabs with a thickness of 2 cells each. Of course, this particular mixed execution example would only make sense if each of the GPUs would be around ∼50 times faster than a single CPU core. In practice, the speed difference is typically considerably larger, so most of the work should typically be assigned to GPUs if those are available.

We also note that for the moment our code supports only meshes with uniform and fixed resolution. However, a more general domain decomposition than just a slab-based decomposition is planned for the future and in principle straightforward. This can, in particular, remove the obvious scaling limitation of our current approach, where the number of cells per dimension sets the maximum number of GPUs or CPU cores that could be employed.

8.2 GPU computing implementation

The above parallelization strategy makes it clear that our code is neither a plain CPU code nor a pure GPU code. Rather, it implements its core compute functionality where needed twice, in a CPU-only version and in a GPU-only version. Both versions can be interchangeably used for any given slab taken from the global computational mesh, and they produce the same results. While this approach evidently requires some extra coding, we have found that this is actually quite helpful for code validation, as well as for quantifying the relative performance of CPU and GPU versions. Further, the extra coding effort can be greatly alleviated by using wherever possible functions that can be compiled and executed both by GPUs and CPUs based on a single implementation.

For the GPU code, we have used the CUDA programming model available for Nvidia GPU devices. All our code is written in low level C++, and we presently do not make use of programming models such as OpenACC, special GPU-accelerated libraries, or new C++ language features that allow GPU-based execution of standard libraries via execution policies. Our programming model is thus best described as MPI-parallel C++, accelerated with CUDA[3] when GPUs are available. If no GPUs are available, the code can still be compiled into a CPU-only version.

For storing static data such as coefficients of Legendre polynomials or Gaussian quadrature weights, we try to make use of the special constant memory on GPUs, which offers particularly high performance, also in comparison to the ordinary general memory. Likewise, for computing parallel reductions across individual cells, we make use of the special shared memory. However, the size of the corresponding memory spaces is quite limited, and varies between different GPU hardware models. This can necessitate adjustments of the used algorithms at compile time, depending on code settings such as the expansion order and on which execution platform is used. We address this by defining appropriate compile-time switches, such that these adjustments are largely automatic.

We note that the data of slabs that are computed with GPUs need to fit completely on GPUs as we refrain from transmitting the data from the front end host computer to the GPU on every time-step. Instead, the data remains on the GPU for maximum performance, and only when a simulation is finished or a temporary result should be output to disc it is copied back from the GPU to the front end host. Wherever such transfers are needed, we use pinned memory on the front end to achieve maximum bandwidth between the host and GPUs. GPUs can access such pinned memory directly, without going through the host CPU first. The problem sizes we are able to efficiently tackle with GPUs are therefore limited by the total combined GPU memory available to a run. Modern GPUs typically have some 10 GBs of main memory, but the detailed amount can vary greatly depending on the model, and is course a matter of price as well. The communication between adjacent slabs is organized such that communication and computation can in principle overlap. This is done such that first the surface states are computed and a corresponding MPI exchange with the neighbouring slabs is initiated. While this proceeds, the volume integrals for slabs are carried out by the GPU, and only once this is completed, the work continues with the received surface data.

Because slabs that are computed on GPUs need to be executed in a massively thread-parallel fashion with shared-memory algorithms, some changes in the execution logic compared to the effectively serial CPU code are required. For example, to avoid race-conditions in our GPU code without needing to introduce explicit locks, we process the mesh in a red-black checkerboard fashion. Finally, we note that we also implemented a scheme that makes our results binary identical when the number of mesh slabs is changed. This ultimately relates to the question about how the wrap-around between the leftmost and rightmost planes of the mesh in our periodic domain is implemented. Here the order in which fluxes from the left and right neighbouring cells is added to cells needs to be unique and independent of the location and number of slabs in the box in order to avoid that different floating point rounding errors can be introduced when the number of slabs is changed.

8.3 Memory usage

Before closing this section, it is perhaps worthwhile to discuss the memory need of our DG simulations, as this is ultimately determining the maximum size of simulations that can be done for a given number of GPUs. To represent a scalar field such as the density ρ at order p, we need for every cell a certain number of basis function weights $N^{d\mathrm{D}}(p)$, were d is the number of spatial dimensions, see equations (17) and (18). When multiplied with the number of cells, we obtain the number of degrees of freedom, which is identical to the number of floating point variables needed to stored the full density field. If we write the total number of cells as $(N_c)^d$, then the total number of variables that need to be stored for the DG weight vector is

$$N_w = (2+d)(N_c)^d N^{d\mathrm{D}}(p). \tag{59}$$

Here we assumed that we simulate the plain Euler equations without viscosity, where we need $(2+d)$ conserved fields to describe the flow. If we account for our artificial viscosity field, which will always be required for problems involving shocks, this number goes up by one further unit, yielding

$$N_w = (3+d)(N_c)^d N^{d\mathrm{D}}(p). \tag{60}$$

A passive tracer field, if activated, would add a further unit in the pre-factor. In 2D and 3D, a conservative upper bound for $N^{d\mathrm{D}}(p)$ is p^d, but this is not particularly tight. Already for $p = 2$, $N^{3\mathrm{D}}$ is lower than p^3 by a factor of 2, for $p = 4$ this grows to a factor 3.2, and for $p = 10$ the difference is more than a factor 4.5.

Another significant source of memory need lies in our time-stepping algorithm. At present we use stability preserving Runge–Kutta schemes that require a temporary storage of the time derivatives of the weights, evaluated at several different points in time, depending on the order of the Runge–Kutta scheme, which we adjust according to the chosen p. The required temporary storage space $N_{\dot{w}}$ is thus a multiple of N_w, with a pre-factor that depends on the chosen order p,

[3] We presently use the CUDA toolkit version 11.4, the GNU G++ 11 compiler and the C++17 standard. For message passing, we prefer the OpenMPI-4 library, for Fourier transforms we use FFTW-3 and for random number generation we rely on GSL 2.4.

Table 1. Minimum memory need for our DG code when a 3D simulation is assumed with $(N_c)^3$ cells and expansion order p, including allowing for an artificial viscosity field. Here double precision with 8 bytes per floating point number has been assumed.

N_c	p	min. memory need
128	1	512 MB
128	2	1440 MB
128	3	3520 MB
128	5	9856 MB
128	9	37.81 GB
2048	1	2048 GB
2048	2	5760 GB
2048	3	13.75 TB
2048	5	38.5 TB
2048	9	151.3 TB

i.e.

$$N_{\dot{w}} = f_t(p) N_w. \tag{61}$$

Here $f_t(p)$ depends on the number of stages in the Runge–Kutta scheme. Presently, we use a setup where $f_t(p) = p$ for $p \leq 3$, and $f_t(p) = 5$ otherwise. The minimum amount of total storage (in terms of needed floating point numbers) required by the code is thus

$$N_w = [3 + d + f_t(p)](N_c)^d N^{dD}(p). \tag{62}$$

During execution of our code using multiple GPUs or CPU cores, some temporary buffer space is furthermore required to hold, in particular, send and receive buffers for fluid states or fluxes along slab surfaces orthogonal to the x-direction. These tend to be sub-dominant, however, compared to the memory requirements to store the weights and their time derivatives themselves. The latter thus represent the quantities that need to be primarily examined to decide about the feasibility of a simulation in terms of its memory needs. When we use the oscillatory sensor for controlling artificial viscosity, some further temporary storage is needed as well, but since this is again small compared to N_w since only two scalar quantities per cell are needed, this conclusion is not changed. Note that our DG approach does not need to store gradient fields for any of the fields, which is different from many finite volume methods such as, for example, AREPO. Also, use of the Navier–Stokes solver instead of simulating just the Euler equations does not increase the primary memory needs in any significant way.

In Table 1, we give a few examples of the memory need for a small set of simulation sizes and simulation orders, which illustrates the memory needs of the code, and which can be easily scaled to other problem sizes of interest. A single Nvidia A100 GPU with 40 GB of RAM could thus still run a $N_c = 128$ problem at order $p = 9$, or a 512^3 problem at quadratic order $p = 1$. For carrying out a 2048^3 simulation at $p = 1$, a cluster offering at least 52 such devices would already be necessary.

9 CODE PERFORMANCE

In order to fully utilize large parallel supercomputers, a code has to be able to run efficiently not only on a single core on one CPU, but also on hundreds to thousands of cores on many CPUs. The degree to which this can accelerate the total runtime of a computation is encapsulated by the concept of parallel scalability. Similarly, for a GPU-accelerated code it is of interest to what extent the use of a GPU can speed up a computation compared to using an ordinary CPU. If more than a single GPU is used, one is furthermore interested in whether a code can efficiently make simultaneous use of several, perhaps hundreds of GPUs. In this section, we examine these aspects and present results of weak- and strong scaling tests of our new code.

9.1 Weak scaling

Weak scaling performance describes a situation where a set of simulations of increasing size is run and compared, but where the load per computational unit, be it a CPU core or a GPU in our case, is kept constant. The time to perform a single time-step should remain constant in this case, increasing only due to communication-related overhead, through work-load imbalances, or through other types of parallelization losses, for example if a code contains residual serial work that scales with the problem size.

Weak scaling results of our code are shown in Fig. 20. We run a 3D box with constant density and pressure using the Navier–Stokes equations, the positivity limiter, and artificial viscosity. This setup is computationally very close to problems we are running in production. We consider problem sizes of 128^3, 160^3, 200^3, 256^3, 320^3, and 512^3 cells, forming a sequence that approximately doubles in size, with a factor of 64 enlargement from the smallest to the largest runs. To compensate for the fact that the problem size does not exactly double every time, we increase the number of cells, we apply a correction factor to the timing results at each resolution.[4] Correspondingly, we execute these problems with one Nvidia A100 GPU for the smallest mesh size, and 64 GPUs for the largest mesh size, keeping the load per GPU roughly constant. The results are shown in the left-hand panel of Fig. 20. For comparison, we also measure the execution speed if instead every GPU is replaced by four CPU cores of Intel Xeon-6138 processors. The corresponding results are shown in the right-hand panel of Fig. 20. Finally, we repeat these measurements for different DG expansion orders $p = 0 - 5$.

The results in the figure show generally good weak scalability, but also highlight some performance losses for large problem sizes. These arise in part because our domain is split into slabs and not cubes. Larger problems lead to ever thinner slabs with a larger surface-to-volume ratio and thus more communication between different slabs. We also see the influence of enhanced communication on weak scalability when data needs to be transferred across node boundaries. At higher orders the weak scaling is generally better, as the compute-to-communicate time ratio shifts strongly to the compute side.

9.2 Strong scaling

Strong scaling is a test where one runs a problem of given size on an ever increasing number of compute units. Contrary to weak scaling, the load per compute unit decreases in this test, and the time to perform a single time-step should decrease in inverse proportion to the increasing computational power applied to solve the problem.

We show a strong scaling result in Fig. 21, again carried out for a 3D box with constant density and pressure using the Navier–Stokes equations, the positivity limiter and artificial viscosity. For definiteness, we use a simulation with 256^3 cells, and consider orders

[4]The current version of the GPU part of the code can only run if N_c and the number of slabs in the x-direction per rank are even. This and the fact that N_c has to be an integer in any case prevents ideal doubling of problem size. The correction factors we apply are: 128^3: 1.0, 160^3: 0.977, 200^3: 0.954, 256^3: 1.0, 320^3: 0.977, and 512^3: 1.0.

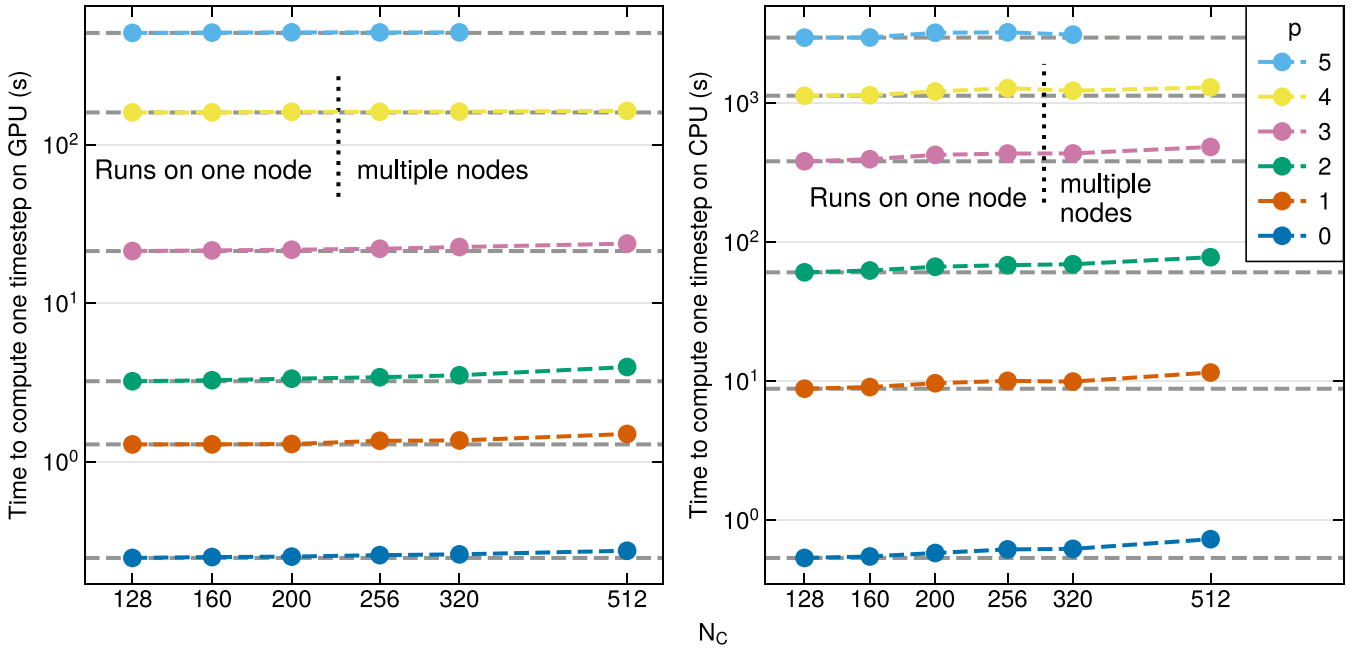

Figure 20. Weak scaling of TENETGPU for a 3D test problem. The y-axis shows the time taken to compute one time-step averaged over a small number of time-steps. The left-hand panel shows results for GPU execution when the problem size N_c^3, measured in terms of the number of cells N_c per dimension, increases in several steps by close to a factor of two from $N_c = 128$ to $N_c = 512$ cells, and when between 1 and 64 GPUs are applied to the problem. In contrast, the right-hand panel gives results when the problems are executed on CPUs instead, using from 4 to 256 cores, again keeping in each case the load per computational element constant. We carry out the measurements for different expansion order, from $p = 0$ to $p = 5$. Ideal weak scaling corresponds to horizontal lines (dashed). The dotted vertical line marks the transition between using CPU cores or GPUs associated with a single compute node of our cluster, and the use of multiple nodes in which MPI data exchange via the Intel Omni-Path takes place. The missing measurement at $p = 5$ is due to the large memory required to store communication buffers, which make the $N_c = 512$ problem not fit onto 64 GPUs. The missing data points at $N_c = 400$ are due to 400 not being divisible by 32, as this would lead to uneven distribution of work across the GPUs we did not consider these runs.

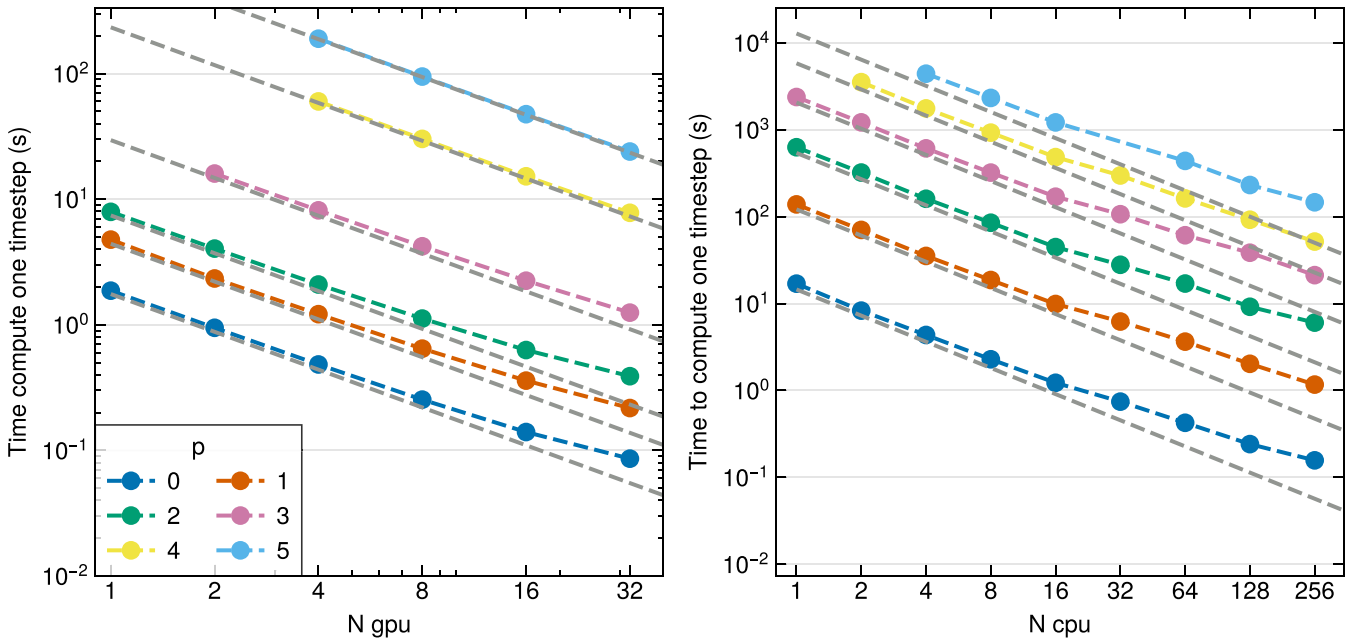

Figure 21. Strong scaling of TENETGPU for a 3D test problem of size 256^3 cells. The y-axis shows the average time taken to carry out one time-step. The left-hand panel shows timing results when between 1 and 16 Nvidia A100 GPUs are used, while the right-hand panel gives results when between 1 and 256 ordinary Intel Xeon-6138 cores are used. Ideal strong scalability corresponds to the dashed lines indicated in the panels. Missing data points at high orders and low number of compute devices are due to the fact that such large problems do not fit on a single GPU/node.

$p = 0$ to $p = 5$. The left-hand panel of Fig. 21 shows the average execution time for a single step when 1, 2, 4, 8, or 16 Nvidia A100 GPUs are used. In contrast, the right-hand panel of Fig. 21 shows the average execution time when CPU cores on a cluster with 2 Intel Xeon-6138 CPUs are used, with 40 cores per node. We show results from 1 core to 256 cores. Especially in the latter case, one sees clear limits of strong scalability, as communication costs become quite large if the problem is decomposed into slabs that are just a single cell wide. This stresses that there is always a limit for strong scalability, something that is known as Ahmdahl's law. By enlarging the problem size, this limit can however usually be pushed to larger parallel partition sizes. Another major contributor to the degradation that happens when going from 16 to 32 cores is the saturation of available memory bandwidth of a single 20-core socket. We verified this using the STREAM benchmark.[5]

9.3 CPU versus GPU benchmark

Another interesting question is how the absolute speed of GPU execution of our code compares to running it only on ordinary CPU cores. To estimate this speedup, we take the average execution times to compute a time-step from our weak scaling results for both the GPU and CPU runs and consider their ratio. We do this for the three considered DG orders $p = 2$ to $p = 4$, and for the varying problem sizes and number of compute units used. Since we had used 4 CPU cores to pair up with 1 GPU, we rescale the results in two different ways, to either compare the execution performance of four Nvidia A100 GPUs with 40 Intel Xeon-6138 cores – which is how one of our compute nodes is equipped – or to the performance of a single GPU compared to one CPU core (which thus gives 10 times higher values).

The corresponding results are illustrated in Fig. 22. The speedup of GPU execution at the node-level is modest for order $p = 2$, as there are not enough floating point operations to fill up the GPUs. At $p = 2, 3$ we reach the highest node-level speedup observed among this set of runs, it peaks at just over $8\times$ the CPU speed for large problems. This runs show better performance because there are a lot of floating points operations to perform at the same time, and all intermediate results still fit into the GPU's limited shared memory. Such shared memory is 'on chip' and therefore about $\sim 100\times$ faster than global memory. Once the intermediate results become too large to fit into shared memory, the code determines the maximum number of quadrature points it can process at once and proceeds forward in batches of n quadrature points. At this point, a single GPU is about 80 times as fast as a CPU core, but when comparing a fully equipped GPU node to a fully equipped CPU node, more realistic numbers are in the ballpark of ~ 8. Note that this speedup metric is based on the specific hardware configuration of the cluster the authors had access to throughout this project. While the configuration of four Nvidia A100 GPUs paired with about 40 Intel Xeon cores quite typically reflects the general HPC situation in 2021 and 2022, the corresponding hardware characteristics are not universal and can be expected to evolve substantially in future generations of CPU-GPU systems. In any case, the performances we find are not far away from the ratio of the nominal peak performances of the involved compute devices for double precision arithmetic (which we have used here throughout), but this comparison also suggests that there is still some modest room for improvement in the performance of our GPU implementation.

[5] https://github.com/intel/memory-bandwidth-benchmarks

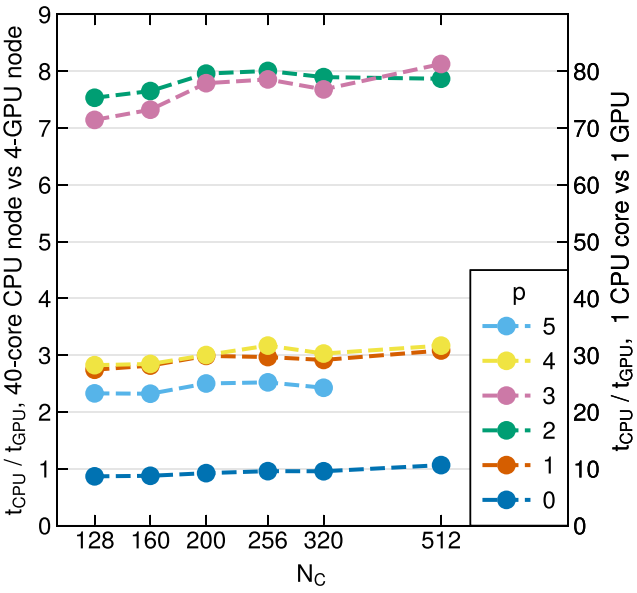

Figure 22. Ratio of time taken to calculate one time-step of test simulations with the Navier–Stokes solver on GPUs or CPUs, based on our weak scaling test runs. The left vertical scale shows results when we normalize them to the speed ratio for using 4 Nvidia A100 GPUs versus 40 Intel Xeon 6138 CPU cores, while the right scale normalizes the speed results to a comparison of 1 GPU versus 1 CPU core.

10 SUMMARY AND CONCLUSIONS

In this study, we have described a novel hydrodynamical simulation code which is based on the mathematical Discontinuous Galerkin approach. The fluid state is expanded in this method into a set of spatially varying basis functions with time-variable weights, yielding a separation of the temporal and spatial dependencies. The time evolution of the weights is obtained in a weak formulation of the underlying partial differential equations of fluid dynamics.

Our work builds up on the earlier development of a DG code by Schaal et al. (2015) and Guillet et al. (2019), but extends it into several crucial directions. First of all, we have developed a novel GPU implementation from scratch, thereby demonstrating the substantial potential of these acceleration devices for achieving higher computational performance in astrophysical applications. This potential has already been identified in a few first finite-volume hydrodynamical GPU codes in astrophysics, but ours is the first one that can carry out DG calculations of the full Navier–Stokes equations at very high order of $p = 10$ and beyond.

Secondly, we have introduced a novel approach to shock-capturing at high order, solving the long-standing problem that standard slope-limiting techniques do not work well at high order and tend to discard in troubled cells much of the advantage that is supposed to be delivered by a high order approach. The latter can only be rescued if the DG method is able to capture physical discontinuities in a sub-cell fashion. By means of our new source routines for a time-dependent artificial viscosity field, we have demonstrated very good shock-capturing ability of our code, with a shock broadening that closely tracks the effective spatial resolution h/p that we expect from the method based on its number of degrees of freedom per dimension. While this does not necessarily give high-order approaches an advantage for representing a shock compared with a lower order method with the same number of degrees of freedom, at least it also is not worse – using a high-order approach will however in any case still be beneficial for all smooth parts of a flow. If it

performs at the same time as well as a lower order method in places where there is a shock, this can be a significant advantage. For contact discontinuities, similar considerations apply, but here high-order methods have the additional advantage of exhibiting greatly reduced numerical diffusivity. Contact discontinuities that move over substantial time-spans therefore also benefit from the use of higher order.

Thirdly, we have stressed that the use of physical viscosity is often a key to make problems well posed and amenable to direct numerical solutions. Here we have introduced a novel method to define the viscous surface fluxes at cell interfaces. This is based on arriving at unambiguous derivatives at interfaces by projecting the two piece-wise solutions in the adjacent cells onto a continuous basis function expansion covering both cells. The derivatives can then be computed in terms of analytic derivatives of the basis functions. We have shown that this technique is robust, consumes much less memory and computational effort than the uplifting technique, and most importantly, it converges at the expected rapid convergence rate when high order is used.

In fact, in several of our test problems, we could show that our DG code shows for smooth problems exponential convergence as a function of expansion order p, while for fixed order, the L_1 error norm declines as a power law of the spatial resolution, $L_1 \propto h^p$. These favourable properties suggest that it is often worthwhile to invest additional degrees of freedom into the use of higher expansion order rather than employing more cells. However, since every DG cell effectively represents a small spectral problem in which the required solution evaluations and volume integrations are carried out in real space, the computational cost to advance a single cell also increases rapidly with order p. In practice, this can make the optimal order quite problem dependent.

With our present implementation, we could obtain excellent agreement with the reference Kelvin–Helmholtz solution computed by Lecoanet et al. (2016) with the spectral code DEDALUS. Remarkably, we achieved this already with 64 cells and order $p = 4$, for which our results are equally as accurate as those obtained with the finite volume code ATHENA at second order using 2048 cells. This again shows the potential of the DG approach. Given that in this work we could overcome one of its greatest weaknesses in an accurate, simple, and robust way – namely the treatment of shocks at high order – we are confident that the DG method could soon turn into a method of choice in astrophysical applications, rivaling the traditional finite volume techniques. Our next planned steps to make this a reality are to add additional physics such as radiative cooling and self-gravity to our code, and to provide functionality for local refinement and derefinement (h-adaptivity), as well as to allow for varying the expansion order used in a single cell (p-adaptivity). The high performance we could realize with our GPU implementation, which outperforms modern multicore CPUs by a significant factor, furthermore strengthens the case to push into this direction, which seems also a necessity to eventually be able to harness the power of the most powerful supercomputers at the exascale level for unsolved problems in astrophysical research.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous referee whose comments significantly improved the quality of this work. We also thank Philipp Grete for discussions about the bottleneck effect and to Damien Begue for insights about recovery based advanced DG methods. The authors also acknowledge helpful discussions with numerous students at MPA, this pandemic-era work could not be finished without your support.

DATA AVAILABILITY

Data of specific test simulations can be obtained upon reasonable request from the corresponding author.

REFERENCES

Agertz O. et al., 2007, MNRAS, 380, 963
Andersson N., Comer G. L., 2021, Living Rev. Relat., 24, 3
Bassi F., Rebay S., 1997, J. Comput. Phys., 131, 267
Bauer A., Springel V., 2012, MNRAS, 423, 2558
Bauer A., Schaal K., Springel V., Chandrashekar P., Pakmor R., Klingenberg C., 2016, in Bungartz H., Neumann P., Nagel W. E., eds, Lecture Notes in Computational Science and Engineering, Vol. 113, Software for Exascale Computing – SPPEXA 2013–2015. Springer, Berlin, p. 381
Berlok T., Pfrommer C., 2019, MNRAS, 485, 908
Borrow J., Schaller M., Bower R. G., Schaye J., 2022, MNRAS, 511, 2367
Burns K. J., Vasil G. M., Oishi J. S., Lecoanet D., Brown B. P., 2020, Phys. Rev. Res., 2, 023068
Cha S.-H., Inutsuka S.-I., Nayakshin S., 2010, MNRAS, 403, 1165
Cockburn B., Shu C.-W., 1989, Math. Comput., 52, 411
Cullen L., Dehnen W., 2010, MNRAS, 408, 669
Deppe N. et al., 2022, Phys. Rev. D, 105, 123031
Dolag K., Borgani S., Murante G., Springel V., 2009, MNRAS, 399, 497
Edelmann P. V. F., Ratnasingam R. P., Pedersen M. G., Bowman D. M., Prat V., Rogers T. M., 2019, ApJ, 876, 4
Federrath C., Klessen R. S., Schmidt W., 2008, ApJ, 688, L79
Federrath C., Klessen R. S., Schmidt W., 2009, ApJ, 692, 364
Federrath C., Roman-Duval J., Klessen R. S., Schmidt W., Mac Low M. M., 2010, A&A, 512, A81
Federrath C., Klessen R. S., Iapichino L., Hammer N. J., 2016, preprint (arXiv:1607.00630)
Guillet T., Pakmor R., Springel V., Chandrashekar P., Klingenberg C., 2019, MNRAS, 485, 4209
Hopkins P. F., 2015, MNRAS, 450, 53
Janett G., Steiner O., Alsina Ballester E., Belluzzi L., Mishra S., 2019, A&A, 624, A104
Junk V., Walch S., Heitsch F., Burkert A., Wetzstein M., Schartmann M., Price D., 2010, MNRAS, 407, 1933
Kidder L. E. et al., 2017, J. Comput. Phys., 335, 84
Krivodonova L., 2007, J. Comput. Phys., 226, 879
Küchler C., Bewley G., Bodenschatz E., 2019, J. Stat. Phys., 175, 617
Lecoanet D. et al., 2016, MNRAS, 455, 4274
van Leer B., Nomura S., 2005, 17th AIAA Computational Fluid Dynamics Conference. American Institute of Aeronautics and Astronautics, Inc. Toronto, Ontario, Canada
Lombart M., Laibe G., 2021, MNRAS, 501, 4298
Mandelker N., van den Bosch F. C., Springel V., van de Voort F., 2019, ApJ, 881, L20
Markert J., Gassner G., Walch S., 2021, Commun. Appl. Math. Comput., available at: https://doi.org/10.1007/s42967-021-00120-x
Markert J., Walch S., Gassner G., 2022, MNRAS, 511, 4179
McNally C. P., Lyra W., Passy J.-C., 2012, ApJS, 201, 18
Mocz P., Vogelsberger M., Sijacki D., Pakmor R., Hernquist L., 2014, MNRAS, 437, 397
Monaghan J. J., 1992, ARA&A, 30, 543
Morris J. P., Monaghan J. J., 1997, J. Comput. Phys., 136, 41
Nelson R. P., Papaloizou J. C. B., Masset F., Kley W., 2000, MNRAS, 318, 18
Ocvirk P. et al., 2016, MNRAS, 463, 1462
Pakmor R., Springel V., Bauer A., Mocz P., Munoz D. J., Ohlmann S. T., Schaal K., Zhu C., 2016, MNRAS, 455, 1134
Persson P.-O., Peraire J., 2006, Sub-Cell Shock Capturing for Discontinuous Galerkin Methods. AIAA Inc., Reston, VA

Pope S. B., 2000, Turbulent Flows. Cambridge Univ. Press, Cambridge
Price D. J., 2008, J. Comput. Phys., 227, 10040
Price D. J., Federrath C., 2010, MNRAS, 406, 1659
Robertson B. E., Kravtsov A. V., Gnedin N. Y., Abel T., Rudd D. H., 2010, MNRAS, 401, 2463
Schaal K., Bauer A., Chandrashekar P., Pakmor R., Klingenberg C., Springel V., 2015, MNRAS, 453, 4278
Schmidt W., Hillebrandt W., Niemeyer J. C., 2006a, Comput. Fluids, 35, 353
Schmidt W., Niemeyer J. C., Hillebrandt W., 2006b, A&A, 450, 265
Schneider E. E., Robertson B. E., 2015, ApJS, 217, 24
Springel V., 2010, MNRAS, 401, 791
Stone J. M., Norman M. L., 1992, ApJS, 80, 753
Stone J. M., Gardiner T. A., Teuben P., Hawley J. F., Simon J. B., 2008, ApJS, 178, 137
Toro E., 2009, Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction. Springer, Berlin, p. XXIV, 724
Trac H., Pen U.-L., 2003, PASP, 115, 303
Tricco T. S., 2019, MNRAS, 488, 5210
Valcke S., de Rijcke S., Rödiger E., Dejonghe H., 2010, MNRAS, 408, 71
Velasco Romero D. A., Han Veiga M., Teyssier R., Masset F. S., 2018, MNRAS, 478, 1855
Verma M. K., Donzis D., 2007, J. Phys. A Math. Gen., 40, 4401
Vilar F., 2019, J. Comput. Phys., 387, 245
Vogelsberger M., Marinacci F., Torrey P., Puchwein E., 2020, Nat. Rev. Phys., 2, 42
Weinberger R. et al., 2017, MNRAS, 465, 3291
Weinberger R., Springel V., Pakmor R., 2020, ApJS, 248, 32
Wibking B. D., Krumholz M. R., 2022, MNRAS, 512, 1430
Woodward P., Colella P., 1984, J. Comput. Phys., 54, 115
Yee H. C., Sandham N. D., Djomehri M. J., 1999, J. Comput. Phys., 150, 199
Yee H. C., Vinokur M., Djomehri M. J., 2000, J. Comput. Phys., 162, 33
Zanotti O., Fambri F., Dumbser M., Hidalgo A., 2015, Comput. Fluids, 118, 204
Zhang X., Shu C.-W., 2010, J. Comput. Phys., 229, 8918
Zhao F., Pan L., Li Z., Wang S., 2017, Comput. Fluids, 159, 81

This paper has been typeset from a TEX/LATEX file prepared by the author.