

## A LOW-RANK ALGORITHM FOR WEAKLY COMPRESSIBLE FLOW\*

LUKAS EINKEMMER†

**Abstract.** In this paper, we propose a numerical method for solving weakly compressible fluid flow based on a dynamical low-rank projector splitting. The low-rank splitting scheme is applied to the Boltzmann equation with BGK collision term, which results in a set of constant-coefficient advection equations. This procedure is numerically efficient as a small rank is sufficient to obtain the relevant dynamics (described by the Navier–Stokes equations). The resulting method can be combined with a range of different discretization strategies; in particular, it is possible to implement spectral and semi-Lagrangian methods, which allows us to design numerical schemes that are not encumbered by the sonic CFL condition.

**Key words.** dynamical low-rank approximation, projector splitting, Boltzmann equation, fluid dynamics, weakly compressible flow

**AMS subject classifications.** 15A69, 76M25, 65M99

**DOI.** 10.1137/18M1185417

**1. Introduction.** Fluids play a pivotal role in virtually all fields of science and engineering. Consequently, computational fluid dynamics is used from modeling pipe flows on a single workstation to simulating airplanes or turbulent combustion on state of the art supercomputers. The governing partial differential equations (PDEs) are the Navier–Stokes equations. More specifically, in the present work we will consider the compressible isothermal Navier–Stokes equations

$$(1.1) \quad \begin{aligned} \partial_t \rho + \nabla_x \cdot (\rho u) &= 0, \\ \partial_t (\rho u) + \nabla \cdot (\rho u \otimes u) + \nabla p &= \nabla \cdot [\mu (\nabla u + (\nabla u)^T) + \lambda (\nabla \cdot u) I], \\ p &= \rho \theta, \end{aligned}$$

where the density  $\rho$  and the momentum  $\rho u$  are the sought-after quantities. Since we consider the isothermal case the (thermodynamic) temperature  $\theta$  is fixed. The pressure is determined by the ideal gas law  $p = \rho \theta$ . Two material parameters, the dynamic viscosity  $\mu$  and the volume viscosity  $\lambda$ , have to be specified. In the case of vanishing viscosity (i.e.,  $\mu = 0$  and  $\lambda = 0$ ) equations (1.1) are usually referred to as the Euler equations.

The most common approach to solving these equations numerically is to discretize them on an appropriate grid. Historically finite difference and finite volume methods have been used extensively, while in recent years discontinuous Galerkin schemes have become more common. However, especially in the study of turbulence by direct numerical simulation (DNS), spectral methods are often preferred (see, for example, [17, 51]).

This approach (which we will refer to as direct discretization in the following) is very mature and sophisticated numerical methods have been developed in recent decades. Further advantages of this approach are that (at least the basic) numerical

---

\*Submitted to the journal's Methods and Algorithms for Scientific Computing section May 3, 2018; accepted for publication (in revised form) July 9, 2019; published electronically September 10, 2019.

<https://doi.org/10.1137/18M1185417>

†Department of Mathematics, University of Innsbruck, Innsbruck, 6020, Austria (lukas.einkemmer@uibk.ac.at).

algorithms are often easy to understand and implement. Disadvantages include that explicit methods usually need to satisfy the CFL condition for sound waves (which in the weakly compressible setting can be multiple orders of magnitude faster than the flow speed), that the equations are relatively complicated (which puts significant constraints on the design of numerical methods), and that generating an appropriate mesh can be difficult. One approach to overcome the CFL condition is by using so-called semi-Lagrangian methods. There the characteristic curves of the Euler equations are traced backward in time. This is often done using an explicit Runge–Kutta integrator for the corresponding nonlinear ordinary differential equation. Since the endpoint of a given characteristic curve, in general, does not fall on a grid point, an interpolation procedure has to be employed (e.g., polynomial interpolation of a certain degree). For more details we refer to [41, 48]. In this formulation the numerical method is not mass conservative. However, more recently, forward semi-Lagrangian methods have been developed that are able to conserve mass; see, e.g., [38, 24]. Semi-Lagrangian methods can be applied to complicated geometry; however, care has to be taken to obtain sufficiently accurate interpolation schemes and to handle the boundaries appropriately. For more details we refer the reader to [40, 43].

It should be duly noted that a direct discretization of the Navier–Stokes equations is not the only way to perform fluid simulations. It is possible to exploit the fact that the Boltzmann equation (a kinetic model), for an appropriately modeled collision term and initial value, recovers the dynamics of the Navier–Stokes equations (see, for example, [4, 3]). Thus, in principle, we can solve fluid flow problems by integrating the Boltzmann equation in time. In this setting the sought-after quantity, a distribution function or particle-density, is usually denoted by  $f$ . However, these kinetic problems are posed in a  $2d$ -dimensional phase space ( $d$  dimensions of space, as for the Navier–Stokes equations, and  $d$  dimensions of velocity). Thus, a direct discretization is prohibitively expensive from a computational point of view.

However, in the fluid regime (i.e., for thermalized gases or liquids) we know that the distribution in velocity space stays close to a Maxwell–Boltzmann distribution. That is,

$$f(t, x, v) \approx \frac{\rho(t, x)}{(2\pi)^{d/2}} \exp\left(-\frac{1}{2}(v - u(t, x))^2\right).$$

What we actually want to approximate are the moments of  $f$ , which correspond to the macroscopic quantities of density  $\rho$  and momentum  $\rho u$ . These are quantities of interest in fluid simulations, as opposed to the distribution function  $f$ .

Historically, this approach was first used by the lattice Boltzmann method. This method has been considered extensively in the literature (see, for example, [7, 22, 21]). The idea of the lattice Boltzmann method is to discretize the velocity space with only a small number of discrete velocities  $e_j \in \mathbb{R}^d$ . Then, the moments can be computed using a Gaussian-type quadrature

$$(1.2) \quad \rho(t, x) = \int f(t, x, v) dv \approx \sum_j W_j f_j(t, x),$$

$$(1.3) \quad \rho u(t, x) = \int v f(t, x, v) dv \approx \sum_j W_j e_j f_j(t, x),$$

where  $f_j(t, x) \approx f_j(t, x, e_j)$  and  $W_j$  are quadrature weights. We then only have to solve an evolution equation for the (relatively) small number of  $f_j$ s (which are  $d$ -dimensional

functions of  $x$ ). For the classic lattice Boltzmann method in two dimensions, the  $e_j$  are chosen as the corners of a square and the zero vector. Usually, we have 9  $f_j$  in two dimensions (this is referred to as D2Q9). In three dimensions a variety of schemes have been considered (for example, D3Q19 and D3Q27 with 19 and 27  $f_j$ s, respectively). If the length of the square/cube is  $2h/\tau$ , where  $h$  is the grid spacing and  $\tau$  is the time step size, a numerical method (if operated at unit CFL number) can be implemented without discretizing any differential operators. This is a consequence of the fact that the Boltzmann equation is much simpler compared to the Navier–Stokes equations. Further advantages of the lattice Boltzmann method are that it can be parallelized very efficiently [42] and that it can relatively easily handle complicated geometries. Disadvantages include that the amount of memory needed is increased (compared to a direct discretization of the Navier–Stokes equations on a uniform grid) and that the method is most effective if simulations are conducted using a unit CFL number.

The latter, namely, that the method is operated at unit CFL number, has long been considered a distinctive feature of the lattice Boltzmann approach. In this setting the advection is computed exactly and thus no numerical diffusion is introduced. Attempts to operate the lattice Boltzmann method with larger CFL numbers have resulted in the developed of so-called off-lattice Boltzmann methods (see, for example, [36, 12, 21]). It has been argued [21] that these methods can be computationally expensive due to the increased numerical effort. However, due to the increasing gap between CPU and memory performance on modern computer architectures, this limitation is likely to disappear (both classic and off-lattice Boltzmann methods will be memory bound).

A downside of lattice Boltzmann methods is that a splitting error between the advection and the collision term is incurred. To overcome this limitation DUGKS (discrete unified gas kinetic scheme) methods [13] have been introduced. While the lattice Boltzmann method can be interpreted as a finite difference discretization, DUGKS methods constitute a finite volume approach to discretizing the Boltzmann equation. Since they have to approximate the flux between two cell interfaces, numerical diffusion is introduced and those schemes still have to satisfy a CFL condition. However, no splitting error is incurred. Some literature is available that, generally favorably, compares DUGKS methods to the classic lattice Boltzmann approach; see, e.g., [45, 46, 6].

In the present paper we propose an alternative approach both to a direct discretization of the Navier–Stokes equations and to the variants of the lattice Boltzmann method outlined above. Similarly to the latter, our scheme is applied to the Boltzmann equation. However, to reduce the dimensionality of the problem (from  $2d$  to  $d$ ) we perform a low-rank approximation. We then obtain evolution equations that describe the dynamics of the Boltzmann equation constraint to the corresponding low-rank manifold. To accomplish this the dynamical low-rank splitting algorithm introduced in [28] is used. This allows us to represent the evolution in velocity space in more detail. In fact, we obtain evolution equations for functions that depend on  $x$  but not on  $v$  (as in the lattice Boltzmann method). We also obtain similar evolution equations for functions that depend only on  $v$  but not on  $x$ .

The evolution equations obtained are still significantly simpler compared to the Navier–Stokes equations (essentially we obtain a constant-coefficient advection with an inhomogeneity). Thus, a range of space discretization strategies can be employed relatively easily. In particular, we can use (true) spectral methods (as opposed to the pseudospectral approach which is common for the direct discretization of the Navier–Stokes equations). Furthermore, it is possible within this approach to construct a numerical method that can overcome the CFL condition imposed by the

speed of sound. This is particularly relevant for weakly compressible simulations. Another interesting property of the projector-splitting integrator is that it mimics the properties of the (continuous) Boltzmann equation as we approach the limit of vanishing viscosity (i.e., as we consider the limit that yields the Euler equations from the Navier–Stokes equations).

Let us note that low-rank approximations have been extensively used in quantum mechanics. See, in particular, [35, 34] for the MCTDH approach to molecular quantum dynamics in the chemical physics literature and [26, 27, 8] for a computational mathematics point of view. Some uses of dynamical low-rank approximation in areas outside quantum mechanics are described in [39, 15, 33, 37]. In a general mathematical setting, dynamical low-rank approximation has been studied in [18, 19, 30]. A major algorithmic advance for the time integration was achieved with the projector-splitting methods first proposed in [28] for matrix differential equations and then developed further for various tensor formats in [27, 29, 14, 16, 31]. Low-rank approximations for computational plasma physics (i.e., the collisionless but magnetized Boltzmann equation) have been considered in [20, 9]. Note, however, that these schemes try to capture kinetic effects that occur far away from thermodynamic equilibrium. This means that the Navier–Stokes equations (or any other model that considers only the moments of  $f$ ) are not applicable in this setting.

The outline of this paper is as follows. We introduce the proposed numerical algorithm in section 2. Then, in section 3, we investigate the behavior of the low-rank projector splitting as the viscosity vanishes. Numerical results are presented in section 4. Finally, we conclude in section 5.

## 2. Numerical method.

We start from the Boltzmann equation

$$(2.1) \quad \partial_t f(t, x, v) + v \cdot \nabla_x f(t, x, v) = \frac{1}{\epsilon} C(f)(x, v),$$

with the BGK collision operator

$$C(f) = f^{\text{eq}} - f,$$

where  $\epsilon > 0$  is a (usually small) parameter and

$$(2.2) \quad f^{\text{eq}}(t, x, v) = \frac{\rho(t, x)}{(2\pi)^{d/2}} \exp\left(-\frac{1}{2}(v - u(t, x))^2\right),$$

where  $d \in \{1, 2, 3\}$  is the dimension of the problem. The sought-after quantity is  $f$  (which depends on the parameter  $\epsilon$ ). The density  $\rho$  and the velocity  $u$  (strictly speaking, the momentum  $\rho u$ ) can then be computed as follows:

$$\rho = \int f \, dv, \quad \rho u = \int v f \, dv.$$

It is well known that the dynamics of (2.1) still fully captures the (very complicated) dynamics of the compressible isothermal Navier–Stokes equations (for  $\epsilon > 0$ ) and the Euler equations in the limit  $\epsilon \rightarrow 0$ . More precisely, we obtain (1.1) with  $\mu = \epsilon$  and  $\lambda = -2\epsilon/d$ . In the latter case the right-hand side of (2.1), i.e., the BGK collision term, constrains the solution to precisely the form given in (2.2), where  $\rho$  and  $u$  are, as of yet, undetermined quantities. For more details on the derivation, and the Chapman–Enskog expansion used therein, we refer the reader to [4, 3, 49].

The question that remains to be answered is why a low-rank representation makes sense in the present setting. We will discuss this in the case of the Euler equations (i.e., for  $\epsilon \rightarrow 0$ ). In this setting, we know that the solution satisfies the form specified by (2.2) at all times. Note that this is not a low-rank representation due to the presence of both velocity ( $v$ ) and position ( $x$ ) dependent functions in the exponential. However, if the flow velocity is small compared to the speed of sound (i.e., in the weakly compressible case) we can use

$$(2.3) \quad f^{\text{eq}} = \frac{\rho}{(2\pi)^{d/2}} \exp\left(-\frac{v^2}{2}\right) \left(1 + v \cdot u + \frac{(v \cdot u)^2}{2} - \frac{u^2}{2}\right) + \mathcal{O}(u^3).$$

This is a low-rank approximation with rank 6 and 10 for two- and three-dimensional problems, respectively. For comparison, a lattice Boltzmann method usually requires 9 directions in two dimensions and 19 to 27 directions in three dimensions (see the discussion in the introduction). Thus, at least in principle, representing the solution by a low-rank representation is a viable approach.

It should be noted that for a small  $\epsilon > 0$ , the usual setting of the Navier–Stokes equations, there is a priori no guarantee that we obtain a low-rank solution. However, the collision operator forces  $f_\epsilon$  to stay close to a low-rank function. This further motivates the proposed approach and we will see in section 4 that usually quite low ranks are sufficient in order to obtain excellent agreement with the dynamics of interest.

As the initial value we choose a function of the form

$$f(0, x, v) = \frac{\rho^0(x)}{(2\pi)^{d/2}} \exp\left(-\frac{1}{2} (v - u^0(x))^2\right).$$

This is not yet a low-rank representation. However, in an actual implementation we can either use the expansion given in (2.3) or perform a singular value decomposition once the problem is discretized. What remains to be determined here is the density  $\rho^0$  and the velocity  $u^0$  (or alternatively, the momentum  $\rho u^0$ ). These are specified according to the fluid problem for which a numerical solution is sought.

Since (2.1) is posed in a  $2d$ -dimensional phase space, its direct solution is prohibitively expensive. This is particularly true in the present setting as the dynamics stays close to a low-rank manifold (see the previous discussion). Thus, the goal of this section is to derive an algorithm that approximates the Boltzmann equation (2.1) by a low-rank representation. To that end, the function  $f(t, x, v)$  is constrained to the following form:

$$(2.4) \quad f(t, x, v) = \sum_{ij} X_i(t, x) S_{ij}(t) V_j(t, v),$$

where  $S \in \mathbb{R}^{r \times r}$  and we call  $r$  the rank of the representation. Note that the dependence of  $f$  on the phase space  $(x, v) \in \Omega \subset \mathbb{R}^{2d}$  is now approximated by the functions  $\{X_i: i = 1, \dots, r\}$  and  $\{V_j: j = 1, \dots, r\}$  which depend only on  $x \in \Omega_x \subset \mathbb{R}^d$  and  $v \in \Omega_v \subset \mathbb{R}^d$ , respectively. It might be tempting to introduce a representation in which the number of functions  $X_i$ , denoted by  $r_x$ , and the number of functions  $V_j$ , denoted by  $r_v$ , are different. However, this situation can, by taking linear combinations, always be reduced to a representation of the above form with  $r = \min(r_x, r_v)$ . Thus, nothing is gained by being more general. In (2.4) and the following discussion we always assume that summation indices run from 1 to  $r$  and we thus do not specify these bounds.

Now, for a fixed time  $t$ , we seek an approximation to the exact particle-density function that lies in

$$\overline{\mathcal{M}} = \left\{ f \in L^2(\Omega) : f(x, v) = \sum_{ij} X_i(x) S_{ij} V_j(v) \text{ with } S \in \mathbb{R}^{r \times r}, X_i \in L^2(\Omega_x), V_j \in L^2(\Omega_v) \right\}.$$

This implies that the solution can be written as a linear combination of tensor products formed by functions that depend only on  $x$  and  $v$ , respectively. Let us duly note, however, that the functions  $X_i$  and  $V_j$  used in the representation are completely arbitrary. Thus, no a priori choice of a basis has to be made. In fact, determining these functions is an integral part of the algorithm that is proposed in this work. We further note that the functions  $X_i$  and  $V_j$  are time dependent and are chosen by the algorithm in such a fashion as to reduce the approximation error.

It is clear that this representation is not unique. In particular, we can make the assumption that  $(X_i, X_k) = \delta_{ik}$  and  $(V_j, V_l) = \delta_{jl}$ , where  $(\cdot, \cdot)$  is the inner product on  $L^2(\Omega_x)$  and  $L^2(\Omega_v)$ , respectively. We consider a path  $f(t)$  on  $\overline{\mathcal{M}}$ . Then, we have

$$(2.5) \quad \partial_t f = \sum_{ij} (X_i(\partial_t S_{ij}) V_j + (\partial_t X_i) S_{ij} V_j + X_i S_{ij} (\partial_t V_j)).$$

Our goal is now to derive evolution equations for the factors of the low-rank approximation; that is, for the quantities  $X_i(t, x)$ ,  $S(t)$ , and  $V_j(t, v)$ . To do that, (2.5) is not yet enough as it alone does not uniquely determine the time evolution of the low-rank factors. However, if we impose the conditions  $(X_i, \dot{X}_j) = (V_i, \dot{V}_j) = 0$ , then  $S_{ij}$  is uniquely determined by  $\partial_t f$ . This follows easily from the fact that

$$(2.6) \quad \partial_t S_{ij} = (X_i V_j, \partial_t f).$$

We then project both sides of (2.5) onto  $X_i$  and  $V_j$ , respectively, and obtain

$$(2.7) \quad \sum_j S_{ij} (\partial_t V_j) = (X_i, \partial_t f) - \sum_j (\partial_t S_{ij}) V_j,$$

$$(2.8) \quad \sum_i S_{ij} (\partial_t X_i) = (V_j, \partial_t f) - \sum_i X_i (\partial_t S_{ij}).$$

From these relation it follows that the  $X_i$  and  $V_j$  are uniquely defined if  $S$  has full rank (this, in particular, implies that  $S$  and  $S^T$  are invertible). Thus, we seek an approximation such that for each time  $t$  the function  $f(t, \cdot, \cdot)$  lies on the manifold

$$\mathcal{M} = \left\{ f \in L^2(\Omega) : f(x, v) = \sum_{ij} X_i(x) S_{ij} V_j(v) \text{ with } S \in \mathbb{R}^{r \times r}, X_i \in L^2(\Omega_x), V_j \in L^2(\Omega_v) \text{ and } (X_i, X_k) = \delta_{ik}, (V_j, V_l) = \delta_{jl}, S \text{ has full rank} \right\}$$

with the corresponding tangent space

$$\begin{aligned} \mathcal{T}_f \mathcal{M} &= \left\{ \dot{f} \in L^2(\Omega) : \dot{f}(x, v) = \sum_{ij} \left( X_i(x) \dot{S}_{ij} V_j(v) + \dot{X}_i(x) S_{ij} V_j(v) + X_i(x) S_{ij} \dot{V}_j(v) \right), \right. \\ &\quad \left. \text{with } \dot{S} \in \mathbb{R}^{r \times r}, \dot{X}_i \in L^2(\Omega_x), \dot{V}_j \in L^2(\Omega_v), \text{ and } (X_i, \dot{X}_j) = (V_i, \dot{V}_j) = 0 \right\}, \end{aligned}$$

where  $f$  is given by (2.4). The solution of the Boltzmann equation does not necessarily lie on the manifold  $\mathcal{M}$ . Thus, we consider the time evolution of the Boltzmann equation projected onto the tangent space. That is,

$$(2.9) \quad \partial_t f = -P(f) \left( v \cdot \nabla_x f - \frac{1}{\epsilon} C(f) \right),$$

where  $P(f)$  is the orthogonal projector onto the tangent space  $\mathcal{T}_f \mathcal{M}$ , as defined above.

We will consider the projection  $P(f)g$  for a moment. We have

$$\begin{aligned} P(f)g &= \sum_{ij} (X_i(\partial_t S_{ij})V_j + (\partial_t X_i)S_{ij}V_j + X_i S_{ij}(\partial_t V_j)) \\ &= \sum_{ij} X_i(\partial_t S_{ij})V_j + \sum_j \left( \sum_i S_{ij}(\partial_t X_i) \right) V_j + \sum_i X_i \left( \sum_j S_{ij}(\partial_t V_j) \right). \end{aligned}$$

Then plugging in (2.6)–(2.8) we get

$$P(f)g = \sum_j (V_j, g) V_j - \sum_{ij} X_i(X_i V_j, g) V_j + \sum_i (X_i, g) X_i.$$

Let us introduce the following two vector spaces:  $\bar{X} = \text{span}\{X_i : i = 1, \dots, r\}$  and  $\bar{V} = \text{span}\{V_j : j = 1, \dots, r\}$ . Then we can write the projector as

$$(2.10) \quad P(f)g = P_{\bar{V}}g - P_{\bar{V}}P_{\bar{X}}g + P_{\bar{X}}g,$$

where  $P_L$  is the orthogonal projector onto the vector space  $L$ . The decomposition of the projector into these three terms forms the basis of our splitting procedure (for matrix equations this was first suggested in [28]).

We proceed by substituting  $g = -v \cdot \nabla_x f + \frac{1}{\epsilon} C(f)$  into (2.10). This at once gives a three-term splitting for (2.9). More precisely, for the first-order Lie splitting we have to solve the equations

$$(2.11) \quad \partial_t f = -P_{\bar{V}} \left( v \cdot \nabla_x f - \frac{1}{\epsilon} C(f) \right),$$

$$(2.12) \quad \partial_t f = P_{\bar{V}}P_{\bar{X}} \left( v \cdot \nabla_x f - \frac{1}{\epsilon} C(f) \right),$$

$$(2.13) \quad \partial_t f = -P_{\bar{X}} \left( v \cdot \nabla_x f - \frac{1}{\epsilon} C(f) \right)$$

one after another. In the following discussion we will consider the first-order Lie splitting algorithm with step size  $\tau$ .

We assume that the initial value for the algorithm is given in the following form:

$$f(0, x, v) = \sum_{ij} X_i^0(x) S_{ij}^0 V_j^0(v).$$

First, let us consider (2.11). Since the set  $\{V_j : j = 1, \dots, r\}$  forms an orthonormal basis of  $\bar{V}$  (for each  $t$ ), we have

$$(2.14) \quad f(t, x, v) = \sum_j K_j(t, x) V_j(t, v), \quad K_j(t, x) = \sum_i X_i(t, x) S_{ij}(t),$$

where  $K_j(t, x)$  is the coefficient of  $V_j$  in the corresponding basis expansion. We duly note that  $K_j$  is a function of  $x$  but not of  $v$ ). We then rewrite (2.11) as follows:

$$\begin{aligned} & \sum_j \partial_t K_j(t, x) V_j(t, v) + \sum_j K_j(t, x) \partial_t V_j(t, v) \\ &= - \sum_j \left( V_j(t, \cdot), v \mapsto v \cdot \nabla_x f(t, x, v) - \frac{1}{\epsilon} C(f)(x, v) \right) V_j(t, v). \end{aligned}$$

The solution of this equation is given by  $V_j(t, v) = V_j(0, v) = V^0(v)$  and

$$(2.15) \quad -\partial_t K_j(t, x) = \sum_l c_{jl}^1 \cdot \nabla_x K_l(t, x) + \frac{1}{\epsilon} (K_j(t, x) - c_j^3(K)(t, x) \rho(K)(t, x))$$

with

$$c_{jl}^1 = \int v V_j^0 V_l^0 dv, \quad c_j^3(K) = \int V_j^0 h^{\text{eq}}(K) dv, \quad \rho(K) = \sum_j K_j \int V_j^0 dv,$$

where we have used the decomposition  $f^{\text{eq}} = \rho h^{\text{eq}}$ . The evolution equation is obtained by equating coefficients in the basis expansion. A very useful property of the present splitting is that we have to only update the  $K_j$  but not the  $V_j$ . We further note that  $c_{jl}^1 = (c_{jl}^{1;x_1}, c_{jl}^{1;x_2})$  (for  $d = 2$ ) is a vector quantity. Also note that we use  $c_j^3$  here (instead of  $c_j^2$ ) to keep the notation in line with [9], where  $c_j^2$  was used for the term originating from the electric field (which is not present for standard fluid flow). However, since, as briefly discussed in the conclusion, the proposed numerical method could conceivably be generalized to magnetohydrodynamic problems, we have chosen this notation.

Equation (2.15) is completely posed in a  $d$ -dimensional (as opposed to  $2d$ -dimensional) space. Thus, we proceed by integrating (2.15) with initial value

$$K_j(0, x) = \sum_i X_i^0(x) S_{ij}^0$$

until time  $\tau$  to obtain  $K_j^1(x) = K_j(\tau, x)$ . However, this is not sufficient as the  $K_j^1$  are not necessarily orthogonal (a requirement of our low-rank representation). Fortunately, this is easily remedied by performing a QR decomposition

$$K_j^1(x) = \sum_i X_i^1(x) S_{ij}^1$$

to obtain orthonormal  $X_i^1$  and the matrix  $S_{ij}^1$ . Once a space discretization has been introduced, this QR decomposition can be simply computed by using an appropriate function from a software package such as LAPACK. However, from a mathematical point of view, the continuous dependence on  $x$  causes no issues. For example, the modified Gram–Schmidt process works just as well in the continuous formulation considered here.

Second, we proceed in a similar way for (2.12). In this case both  $V_j^0$  and  $X_i^1$  are unchanged and only  $S_{ij}$  is updated. The corresponding evolution equation is given by

$$(2.16) \quad \begin{aligned} \partial_t S_{ij}(t) &= \left( X_i^1 V_j^0, (x, v) \mapsto (v \cdot \nabla_x f(t, x, v) - \frac{1}{\epsilon} C(f)(x, v)) \right) \\ &= \sum_{lk} (d_{il}^1 \cdot c_{jk}^1) S_{lk} + \frac{1}{\epsilon} (S_{ij}(t) - e_{ij}(S)) \end{aligned}$$

with

$$d_{il}^1 = \int X_i^1 \nabla_x X_l^1 dx, \quad e_{ij}(S) = \int X_i^1 \rho(S) V_j^0 h^{\text{eq}}(S) d(x, v).$$

Note that in this case the evolution equation depends neither on  $x$  nor on  $v$ . We now integrate (2.16) with initial value  $S_{ij}(0) = S_{ij}^1$  until time  $\tau$  and obtain  $S_{ij}^2 = S_{ij}(\tau)$ . This completes the second step of the algorithm.

Finally, we consider (2.13). Similar to the first step we have

$$f(t, x, v) = \sum_i X_i(t, x) L_i(t, v), \quad L_i(t, v) = \sum_j S_{ij}(t) V_j(t, v).$$

As before, it is easy to show that the  $X_i$  remain constant during that step. Thus, the  $L_j$  satisfy the evolution equation

$$(2.17) \quad \begin{aligned} -\partial_t L_i(t, v) &= \left( X_j^1, x \mapsto \left( v \cdot \nabla_x f(t, x, v) - \frac{1}{\epsilon} C(f)(x, v) \right) \right) \\ &= \sum_l (d_{il}^1 \cdot v) L_l + \frac{1}{\epsilon} (L_i - d_i^3(L)(v)) \end{aligned}$$

with

$$d_i^3(L)(v) = \int X_i^1 \rho(L) h^{\text{eq}}(L) dx.$$

We then integrate (2.17) with initial value

$$L_i(0, v) = \sum_j S_{ij}^2 V_j^0(v)$$

up to time  $\tau$  to obtain  $L_i^1(v) = L_i(\tau, v)$ . Since, in general, the  $L_i^1$  are not orthogonal we have to perform a QR decomposition

$$L_i^1(v) = \sum_j S_{ij}^3 V_j^1(v)$$

to obtain  $S_{ij}^3$  and  $V_j^1$ . Finally, the output of our Lie splitting algorithm is

$$f(\tau, x, v) \approx \sum_{ij} X_i^1(x) S_{ij}^3 V_j^1(v).$$

For simplicity, we have introduced the low-rank algorithm in the context of the first-order Lie splitting here. If we denote by  $\varphi_\tau^K(g)$ ,  $\varphi_\tau^S(g)$ , and  $\varphi_\tau^L(g)$  the partial flows given by (2.15), (2.16), and (2.17), respectively, with initial value  $g$  integrated from  $t = 0$  to  $t = \tau$ , then a Lie splitting step of size  $\tau$  is given by

$$f(t, \cdot, \cdot) \approx \varphi_\tau^L \circ \varphi_\tau^S \circ \varphi_\tau^K(f(0, \cdot, \cdot)).$$

---

**Algorithm 1.** A second-order accurate low-rank Strang splitting algorithm for the Boltzmann equation.

---

**Input:**  $X_i^0, S_{ij}^0, V_j^0$  (such that  $f(0, x, v) \approx \sum_{i,j} X_i^0(x) S_{ij}^0 V_j^0(v)$ )

**Output:**  $X_i^2, S_{ij}^5, V_j^1$  (such that  $f(\tau, x, v) \approx \sum_{i,j} X_i^2(x) S_{ij}^5 V_j^1(v)$ )

- 1: Compute the coefficient  $c_{jl}^1$  from  $V_j^0$ .
  - 2: Solve (2.15) with initial value  $\sum_i X_i^0 S_{ij}^0$  up to time  $\tau/2$  to obtain  $K_j^1$ .
  - 3: Perform a QR decomposition of  $K_j^1$  to obtain  $X_i^1$  and  $S_{ij}^1$ .
  - 4: Compute the coefficient  $d_{il}^1$  from  $X_i^1$ .
  - 5: Solve (2.16) with initial value  $S_{ij}^1$  up to time  $\tau/2$  to obtain  $S_{ij}^2$ .
  - 6: Solve (2.17) with initial value  $\sum_j S_{ij}^2 V_j^0$  up to time  $\tau$  to obtain  $L_i^1$ .
  - 7: Perform a QR decomposition of  $L_i^1$  to obtain  $V_j^1$  and  $S_{ij}^3$ .
  - 8: Compute the coefficient  $c_{jl}^1$  from  $V_j^1$ .
  - 9: Solve (2.16) with initial value  $S_{ij}^3$  up to time  $\tau/2$  to obtain  $S_{ij}^4$ .
  - 10: Solve (2.15) with initial value  $\sum_i X_i^1 S_{ij}^4$  up to time  $\tau/2$  to obtain  $K_j^2$ .
  - 11: Perform a QR decomposition of  $K_j^2$  to obtain  $X_i^2$  and  $S_{ij}^5$ .
- 

The extension to second-order Strang splitting, which we use in the numerical simulations conducted in section 4, is straightforward. In this case we have

$$f(t, \cdot, \cdot) \approx \varphi_{\tau/2}^K \circ \varphi_{\tau/2}^S \circ \varphi_\tau^L \circ \varphi_{\tau/2}^S \circ \varphi_{\tau/2}^K(f(0, \cdot, \cdot)).$$

The corresponding algorithm is outlined in detail in Algorithm 1. Since steps 10–11 of the previous time step can be combined with steps 1–2 of the current time step, the Strang splitting scheme is computationally no more expensive than the Lie splitting scheme.

Note that, to some extent, the algorithm introduced here has certain similarities with a lattice Boltzmann method. In particular, the  $X_i(t, x)$  roughly correspond to the  $f_i(t, x)$  in the introduction. However, there are important differences. In a lattice Boltzmann method the distribution function  $f$  would be represented as

$$f(t, x, v) = \sum_i W_i f_i(t, x) \delta(v - e_i).$$

This yields the correct moments according to (1.2) and (1.3). For the proposed algorithm, however, we consider the functions  $V_j(t, v)$  which are propagated in time. Thus, we consider not only a single velocity per  $X_i$  but rather a distribution of velocities.

**2.1. Discretization.** The evolution equations for  $S_{ij}$  and  $L_i$  do not involve any spatial derivatives and thus require no further discretization (with the exception of the coefficients, which are constant during the corresponding substep).

However, the evolution equation that describe the dynamics of the  $K_j$  are given by (for simplicity we only consider the two-dimensional case here; however, the extension to three dimensions is immediate)

$$(2.18) \quad \partial_t K_j = - \sum_l \left( c_{jl}^{1;x_1} \partial_{x_1} K_l + c_{jl}^{1;x_2} \partial_{x_2} K_l \right) - \frac{1}{\epsilon} (K_j - c_j^3(K)(x) \rho(K)(x)).$$

Since this is a constant-coefficient advection, we can choose virtually any space discretization scheme (finite differences, finite volumes, etc.) to obtain

$$(2.19) \quad \partial_t K = AK - \frac{1}{\epsilon} (K - c^3(K)\rho(K)),$$

where  $K = [K_1, \dots, K_r]$  and  $A$  is a matrix that represents the discretized differential operator.

Let us pause here for a moment. In the literature a number of different techniques have been developed to solve the Euler equations (or, more generally, fluid flow where sharp gradients occur). Often such techniques are based on upwind schemes. While implementing upwind schemes for a scalar constant-coefficient advection equation is a rather simple task, the (nonscalar and nonlinear) nature of the Euler equations makes this significantly more challenging in practice. For a good review we refer the reader to [44]. One way to generalize upwind schemes is to solve a Riemann problem at the cell interface, which can incur a significant computational cost. Now, note that since equations (2.18) are constant-coefficient advects, most of these difficulties are avoided for the numerical scheme proposed here. Thus, upwind schemes can be implemented relatively easily as part of the proposed numerical algorithm. For flows with high Reynolds number the diffusion introduced by such schemes is a concern. In this case so-called high-resolution schemes, which usually limit the numerical flux, are more appropriate (we once again refer to [44]). However, even in this setting the simplicity of the equations considered here aids in the choice of a good algorithm. We will not explore this topic further in the present paper, but we consider this as future work.

In principle, (2.19) can be solved by an appropriate time integrator. Note, however, that using an explicit method would introduce a CFL condition. In this case we would use substepping. That is, a smaller time step is used to solve (2.19) compared to the splitting scheme. However, this can be avoided by employing a semi-Lagrangian approach (as discussed in the following) or a spectral approach (as discussed in the next section). To do that we first apply a further splitting procedure to (2.18). For Lie splitting this yields

$$K(\tau, \cdot) \approx \varphi_\tau^\epsilon \left( e^{-\tau c^{1;x_2} \partial_{x_2}} e^{-\tau c^{1;x_1} \partial_{x_1}} K(0, \cdot) \right),$$

where  $\varphi_\tau^\epsilon$  is the partial flow generated by the collision term. The crucial part is the computation of

$$M(t, x) = e^{-\tau c^{1;x_1} \partial_{x_1}} M(0, x),$$

which is equivalent to solving the PDE

$$\partial_t M(t, x) = -c^{1;x_1} \partial_{x_1} M(t, x).$$

Now, since  $c^{1;x_1}$  is symmetric, there exists an orthogonal matrix  $T$  such that  $T c^{1;x_1} T^T = D$ , where  $D$  is a diagonal matrix. All the ingredients can be computed efficiently as  $c^{1;x_1} \in \mathbb{R}^{r \times r}$  (i.e., these are small matrices). We now change variables to  $\bar{M} = TM$  and obtain

$$\partial_t \bar{M}_j(t, x) = -D_{jj} \partial_{x_1} \bar{M}(t, x).$$

This is now a set of scalar one-dimensional advection equations with constant coefficients and can thus be treated by a semi-Lagrangian approach.

**2.2. Spectral discretization.** Pseudospectral methods are widely used in some fluid problems (for example, for turbulent DNS simulations [17, 51]). Here we will

show that (true) spectral methods can be very naturally incorporated into the proposed low-rank scheme. To do that we perform the Fourier transformation with respect to  $x$  of (2.18). This yields

$$(2.20) \quad \partial_t \hat{K}(t, k) = A(k) \hat{K}(t, k) - \frac{1}{\epsilon} \left( \hat{K} - c^3(K) \rho(K) \right),$$

where  $\hat{K}_j$  denotes the Fourier transform of  $K_j$  and we have defined  $\hat{K} = [\hat{K}_1, \dots, \hat{K}_r]$  and  $K = [K_1, \dots, K_r]$ . This would be sufficient for a pseudospectral approach. However, we can turn this into a spectral method by further splitting (2.20). This is possible since we only have to treat constant-coefficient advection equations and the nonlinear term (i.e., the collision operator) is free of spatial derivatives. In particular, this is in contrast to the Navier–Stokes equations, where the nonlinear terms involve spatial differentiation. For Lie splitting this yields

$$\hat{K}(\tau, k) \approx \varphi_\tau^\epsilon \left( e^{\tau A(k)} \hat{K}(0, k) \right),$$

where  $\varphi_\tau^\epsilon$  is the partial flow generated by the collision term. The exponential can be readily computed in Fourier space as  $A(k) \in \mathbb{R}^{r \times r}$  (and thus we only have to compute the exponential of a small matrix). We also note that this approach is, obviously, not encumbered by a CFL condition.

**2.3. Computational efficiency.** In this section, we will discuss the computational characteristics of the proposed algorithm. Solving the evolution equations is at most  $\mathcal{O}(rn^d)$  (both in terms of cost as well as in terms of storage), where  $n$  is the number of grid points per direction. In addition, we have to compute various coefficients. To compute the coefficients

$$c_{jl}^1 = \int v V_j^0 V_l^0 dv, \quad d_{il}^1 = \int X_i^1 \nabla_x X_l^1 dx$$

requires a computational cost of  $\mathcal{O}(r^2 n^d)$  and  $\mathcal{O}(r^2)$  storage. Now, naively computing  $c_j^3$  and  $d_i^3$  would be quite expensive and could easily dominate the run time of our algorithm. However, we can also accomplish this with a computational cost of  $\mathcal{O}(r^2 n^d)$ . To do that we proceed as follows. First, we write

$$h^{\text{eq}}(x, v) = \frac{1}{(2\pi)^{d/2}} \exp\left(-\frac{v^2}{2}\right) \sum_k h_k^X(x) h_k^V(v),$$

where the sum is over  $10/6$  ( $d = 3/2$ ) entries and each  $h^X$  and  $h^V$  is a monomial (see the expansion in (2.3)). Thus, we exploit the low-rank expansion of  $h^{\text{eq}}$ . Then we rewrite  $c_j^3$  as

$$c_j^3(x) = \sum_k h_k^X(x) I_{jk}^1, \quad I_{jk}^1 = \frac{1}{(2\pi)^{d/2}} \int V_j(v) \exp\left(-\frac{v^2}{2}\right) h_k^V(v) dv$$

and  $d_j^3$  as

$$d_i^3(v) = \frac{1}{(2\pi)^{d/2}} \exp\left(-\frac{v^2}{2}\right) \sum_k h_k^V(v) I_{ik}^2, \quad I_{ik}^2 = \int X_i \rho h_k^X dx.$$

Both computing the integrals and summing the results to obtain  $c_j^3$  and  $d_i^3$  require a computational cost of  $\mathcal{O}(r^2n^d)$ . Finally, we can use  $c_j^3$  to compute  $e_{ij}$  as follows:

$$e_{ij} = \int X_i \rho c_j^3 dx.$$

This has a computational cost of  $\mathcal{O}(r^2n^d)$ . Thus, the entire algorithm can be implemented with a computational cost of  $\mathcal{O}(r^2n^d)$  and a storage cost of  $\mathcal{O}(rn^d)$ . One might be worried that the proposed algorithm requires  $\mathcal{O}(r^2n^d)$  arithmetic operations. However, we only require  $\mathcal{O}(rn^d)$  memory operations. The latter, in a reasonable implementation, dominates the performance of the algorithm on all present and, most likely, all future computer systems. A hope is that (especially in three dimensions) the rank  $r$  can be chosen smaller than the number of PDEs in an (off-grid) lattice Boltzmann method or in a DUGKS scheme. Then, from that perspective, the amount of memory required and the number of memory operations we have to perform is reduced. On the other hand, the number of arithmetic operations is increased. This is precisely the kind of numerical algorithm that is expected to perform very well on the next generation of supercomputers (i.e., exascale systems). Also, such algorithms are desperately needed to fully exploit accelerators, such as graphic processing units and the Intel Xeon Phi. For more information we refer the reader to the ASCAC report on exascale computing [2].

As we will see in section 4, it is often sufficient to use significantly fewer grid points in the velocity (i.e.,  $v$ ) directions than in the spatial (i.e.,  $x$ ) directions. Thus, for the following efficiency analysis we neglect the computational cost of integrals in  $v$ . This together with the observation made above, i.e., that the problem is memory bound, gives us the opportunity to compare the computational efficiency of the proposed low-rank approximation to lattice Boltzmann and DUGKS schemes. The evolution equation for  $K$ , equation (2.15), solves an advection problem and thus requires  $rn^d$  memory loads and stores, where  $n$  is the number of grid points in the  $x$ -direction. This corresponds to the advection step in the lattice Boltzmann approach and to computing the numerical flux in DUGKS schemes, respectively. In the latter two cases we require  $pn^d$  memory loads and stores, where  $p$  is the number of particles per grid point.

In the next step of the low-rank algorithm we have to compute  $d_{il}^1$  and  $e_{ij}$ . To do that we require the updated value of  $X_i$  and  $\rho$ . However, since the  $X_i$  enter only in a local fashion in this computation and  $\rho$  is a linear combination of  $X_i$ , this can be done while the updated values are still in cache (from the QR decomposition). Consequently, in an optimized implementation no expensive memory operations are required to compute  $d_{il}^1$  and  $e_{ij}$ .

The last step of the algorithm requires the computation of  $d_i^3$  and thus of  $I_{ik}^2$ . Now,  $I_{ik}^2$  depends on  $\rho$  and  $\rho$  in turn depends on  $S$ , which is modified in the previous step. Thus, it is not possible to compute  $I_{ik}^2$  alongside  $d_{il}^1$  and  $e_{ij}$ , i.e., as soon as the updated value of  $X_i$  becomes available. Fortunately, we can write

$$I_{ik}^2 = \sum_l \beta_l \alpha_{ikl}, \quad \alpha_{ikl} = \int X_i X_l h_k^X dx, \quad \beta_l = \int L_l dv.$$

Now, the  $\alpha_{ikl}$  can be computed while  $X_i$  is still in cache, i.e., immediately after the QR decomposition, and thus incur no additional cost in terms of memory transactions. We have here once again traded slow memory access in favor of arithmetic computations.

Thus, we have shown that the necessary coefficients can be computed with little additional cost. In addition, the low-rank algorithm requires evaluations of the

equilibrium function, but so do lattice Boltzmann and DUGKS schemes. Thus, in this respect there is little difference from a computational point of view. The only additional cost incurred by the proposed algorithm is due to the QR decomposition that is used to compute  $X_i$  and  $S_{ij}$  from  $K_j$ . Consequently, if we operate the low-rank algorithm with a CFL number that is smaller than or equal to unity, it is somewhat more expensive per degree of freedom. However, the rank  $r$ , particularly in three dimensions, can be chosen smaller than the number of particles  $p$ , negating this performance deficit. Notwithstanding the previous discussion, as outlined in sections 2.1 and 2.2, the point here is that the low-rank approach can use discretizations that overcome the CFL condition, rendering the comparative cost of the QR decomposition negligible. Moreover, we can construct schemes that use a spectral discretization, which is difficult to incorporate into a DUGKS or an off-lattice Boltzmann framework. To perform a fair comparison between all the different approaches would require a number of well-optimized codes. This is beyond the scope of the present paper. However, we consider it as future work.

**3. The numerical algorithm in the inviscid limit.** An important consideration for the present algorithm is the limit  $\epsilon \rightarrow 0$ . As outlined in section 2, the continuous problem (i.e., the Boltzmann equation) converges to the Euler equations in this case. To put this statement in the present framework, in the limit  $\epsilon \rightarrow 0$  the solution of the Boltzmann equation yields a Maxwell–Boltzmann distribution in phase space.

In general, however, there is no guarantee that a numerical approximation conserves this behavior. However, in the present section we show that each part of the projector splitting satisfies a very similar constraint.

First, we consider the evolution equations for  $K_j$  (i.e., (2.15)). If we take  $\epsilon \rightarrow 0$  we obtain the constraint

$$K_j - c_j^3(K)(x)\rho(K)(x) = 0,$$

which can be written as

$$K_j = \frac{\rho(K)}{(2\pi)^{d/2}} \int V_j(v) \exp\left(-\frac{1}{2}(v - u(K))^2\right) dv.$$

This is just the projection of the Maxwell–Boltzmann distribution onto the space spanned by the  $V_j$ . Thus, as long as  $\exp\left(-\frac{1}{2}(v - u(K))^2\right)$  can be represented accurately in the low-rank manifold (which, as we have discussed in section 2, is indeed the case for weakly compressible flows) this subflow of the splitting algorithm respects the constraints imposed by the continuous problem.

Now, let us consider the evolution equations for  $L_i$  (i.e., (2.17)). For  $\epsilon \rightarrow 0$  we obtain

$$L_i = \int X_i \rho(L) \exp\left(-\frac{1}{2}(v - u(L))^2\right) dx.$$

This takes the Maxwell–Boltzmann distribution and projects it onto the space spanned by the  $X_i$ . Thus, we again conclude that if the corresponding low-rank manifold can accurately represent the Boltzmann–Maxwell distribution our numerical algorithm will naturally enforce the corresponding constraint.

Having considered both the evolution equations for  $K_j$  and  $L_i$ , it should come as no surprise that we obtain a very similar result for the evolution equations for  $S_{ij}$  (i.e., (2.16)). In this setting we obtain

$$S_{ij} = \int X_i V_j \rho(S) \exp\left(-\frac{1}{2}(v - u(S))^2\right) d(x, y),$$

which once again is just the projection onto the space spanned by the  $X_i V_j$ . Thus, if we can assume that our low-rank approximation is able to exactly represent the Maxwell–Boltzmann distribution, then the dynamical low-rank splitting would yield exactly the correct form of the distribution function  $f$ .

**4. Numerical results.** In this section we will perform numerical simulations with the proposed algorithm. As a comparison we consider a classic fluid solver that uses the second-order MacCormack method.

**4.1. Propagation of sound waves.** As the first test case we consider a simple plane wave propagating in the  $y$ -direction. It can be easily shown that if we can neglect the nonlinear term in the Navier–Stokes equations (i.e., for small velocities), the damped wave equation

$$\partial_{tt}\rho + \mu\Delta(\partial_t\rho) = \Delta\rho$$

is obtained. Due to the ideal gas law  $p = \rho$  this can also be written as a pressure wave. For small damping (i.e., small viscosity  $\mu$ ) we obtain the plane wave solutions

$$(4.1) \quad \rho(t, x, y) = 1 + \delta \sin(k_x x - \omega t) + \delta \sin(k_y y - \omega t)$$

with  $\omega$  the frequency,  $(k_x, k_y)$  the wave vector, and  $\delta$  the amplitude of the wave. Since the speed of sound is equal to unity, frequency and wave vector are coupled by the dispersion relation  $\omega^2 = k_x^2 + k_y^2$ .

For the numerical example we consider the initial value

$$\begin{aligned} \rho(0, x, y) &= 1 + \delta \sin(2\pi y), \\ u_1(0, x, y) &= 0, \\ u_2(0, x, y) &= \delta \sin(2\pi y) \end{aligned}$$

on the domain  $\Omega = [0, 1] \times [0, 1]$ . As described above, for small viscosity and  $\delta \ll 1$  this results in a plane wave solution traveling in the  $y$ -direction with unit speed. An interesting point here is that the step size of any explicit numerical method would be dictated by the CFL condition imposed by the speed of sound. That is, it would have to satisfy  $\tau \leq h$ , where  $\tau$  is the time step size and  $h$  is the grid spacing. On the other hand, the dynamic low-rank splitting proposed here should be able to take time steps dictated by accuracy (i.e., time steps that are significantly larger).

The results presented in Figure 4.1 are meant to check this reasoning and to verify the code in this simple setting. We observe that with the dynamical low-rank Strang splitting we can take almost 30 times larger time steps compared to the MacCormack method. The low-rank approximation does not conserve mass exactly. However, in this setting the conservation of mass is still acceptable (on the order of  $10^{-6}$ ), especially considering that we take quite large time steps. If conserving mass up to machine precision is desired, the method developed in [10] can be used. It ensures mass conservation by adding a constraint that is solved alongside the evolution equations for the low-rank factors. It should be emphasized that this algorithm can be implemented with very little computational overhead.

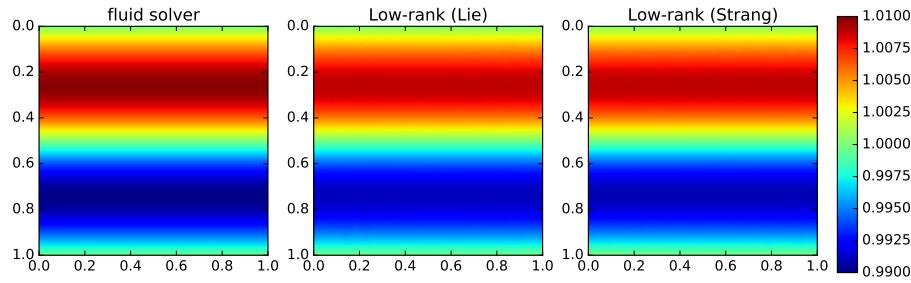


FIG. 4.1. The density  $\rho$  at time  $t = 1$  is shown for the classic fluid solver, Lie splitting, and Strang splitting (with  $\epsilon = 10^{-3}$ ). For all simulations centered differences with 128 grid points per direction are used and the rank is set to 10. The time step size for the classic fluid solver is set to  $\tau = 7 \cdot 10^{-3}$  (a CFL number of 0.9). For Lie splitting  $\tau = 0.1$  and for Strang splitting  $\tau = 0.2$  has been used.

**4.2. Shear flow.** Here we consider a shear flow that is given by

$$(4.2) \quad \begin{aligned} \rho(0, x, y) &= 1, \\ u_1(0, x, y) &= v_0 \begin{cases} \tanh\left(\frac{y-\frac{1}{4}}{\Delta}\right), & y \leq \frac{1}{2}, \\ \tanh\left(\frac{\frac{3}{4}-y}{\Delta}\right), & y > \frac{1}{2}, \end{cases} \\ u_2(0, x, y) &= \delta \sin(2\pi x). \end{aligned}$$

That is, we have a velocity profile in the  $y$ -direction that changes relatively abruptly from  $v_0 = 0.1$  to  $-v_0$  (as we have chosen  $\Delta = 1/30$ ). A small perturbation ( $\delta = 5 \cdot 10^{-3} = 0.05 \cdot v_0$ ) is then added to the velocity in the  $y$ -direction. This problem has been used as a test problem for (mostly incompressible) fluid flow in a number of publications [5, 25, 11].

First, we consider a modest Reynolds number ( $\text{Re} = 300$ ). The corresponding results are shown in Figure 4.2. As is common for such studies we have plotted the vorticity. We observe excellent agreement between the proposed low-rank algorithm and the classic fluid solver. Let us also note that the low-rank algorithm is not encumbered by a CFL condition. In fact, we can take a time step that is almost 15 times as large compared to the fluid solver.

Second, we increase the Reynolds number to  $\text{Re} = 1000$ . This is a more challenging problem in the sense that finer structures appear in the solution. The numerical results are shown in Figure 4.3. We once again observe excellent agreement between our low-rank algorithm and the classic fluid solver. In fact, all of the conclusions drawn for the case  $\text{Re} = 300$  can be applied to the present case as well.

The last point we want to make here is that it is usually not necessary to use a large number of grid points in the velocity direction. To demonstrate this, we have repeated our numerical experiment with only 16 grid points in the  $v$ -directions, while still using 128 grid points in the space directions. In that setting the computational performance is completely dictated by solving equation (2.15). Nevertheless, as Figure 4.4 demonstrates, the numerical results show excellent agreement compared to Figure 4.3, where 128 grid points were used in the velocity directions.

**5. Conclusion and outlook.** We have introduced a numerical algorithm for solving the weakly compressible Navier–Stokes equations that is based on a dynamical low-rank splitting algorithm. The behavior of this algorithm has been investigated

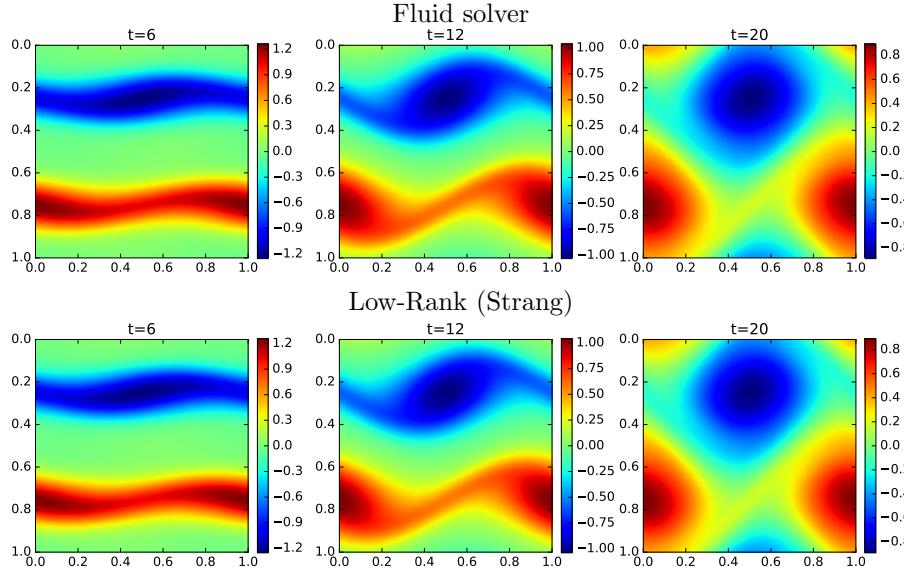


FIG. 4.2. The time evolution of the vorticity  $\omega = \partial_x u_2 - \partial_y u_1$  for the shear flow given by initial values (4.2) is shown (with  $Re = 300$ ). The results from a classic fluid solver are shown on the top and the results from the dynamical low-rank splitting are shown on the bottom. For all simulations centered differences with 128 grid points per direction are used and the rank is set to 10. The time step size for the classic fluid solver is set to  $\tau = 7 \cdot 10^{-3}$  (a CFL number of 0.9), while for the low-rank solver the time step size is set to  $\tau = 0.1$ . For the low-rank implementation Strang splitting is used.

and numerical simulations have been conducted that show excellent agreement with a classic fluid solver.

The algorithm has been considered in the context of weakly compressible isothermal flow with periodic boundary conditions. However, these restrictions are not fundamental. For example, the extension to temperature dependent flows is immediate. Only a time and space dependent temperature has to be introduced in (2.2). This (slightly) changes the collision operator, but the numerical method remains virtually unaffected. Moreover, the expansion (2.3) is valid only for small velocities (i.e., weakly compressible flow). However, this does not mean that we can not efficiently represent the solution by a low-rank function even in the case of fully compressible flow. In fact, it is not even clear that (2.3) is the best low-rank approximation (i.e., the approximation with the smallest rank) one can obtain. We have only considered periodic boundary conditions here. However, for the lattice Boltzmann method or DUGKS schemes it is very well understood how to impose the no-slip boundary conditions that are ubiquitous in computational fluid dynamics. This is usually realized by imposing a bounce-back scheme [47, 6]. However, other approaches are possible as well [23, 32, 50]. Although these boundary conditions are usually formulated in the discrete setting, they represent boundary conditions that connect the outflow boundary with the inflow boundary of the Boltzmann equation (see, e.g., [1], where the authors formulate boundary conditions for the Boltzmann equation and then discretize them to obtain appropriate boundary conditions for the lattice Boltzmann method). Such boundary conditions can be employed in the low-rank algorithm. A further extension of the present algorithm that could be envisaged is the modeling of multi-phase flow. The lattice Boltzmann method, in particular, has been used for simulating

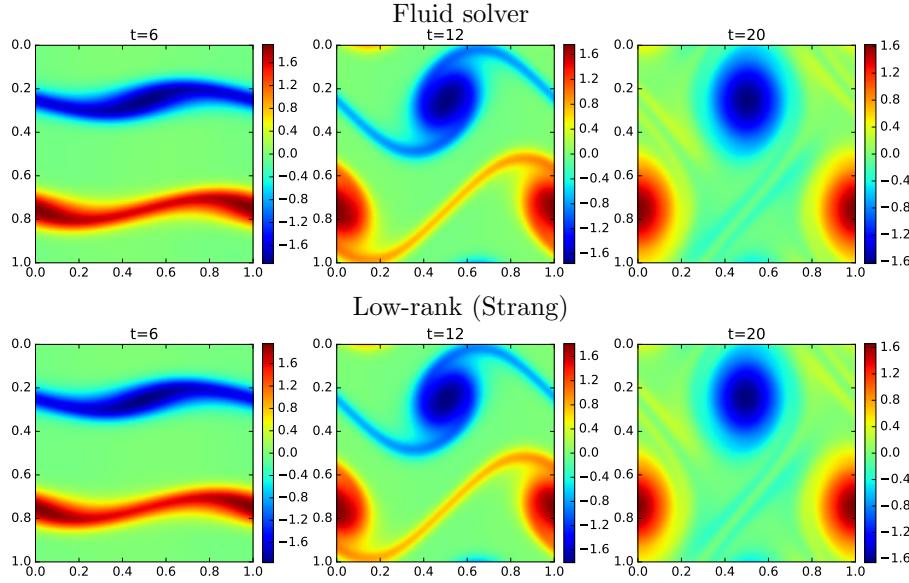


FIG. 4.3. The time evolution of the vorticity  $\omega = \partial_x u_2 - \partial_y u_1$  for the shear flow given by initial values (4.2) is shown (with  $Re = 1000$ ). The results from a classic fluid solver are shown on the top and the results from the dynamical low-rank splitting are shown on the bottom. For all simulations centered differences with 128 grid points per direction are used and the rank is set to 10. The time step size for the classic fluid solver is set to  $\tau = 7 \cdot 10^{-3}$  (a CFL number of 0.9), while for the low-rank solver it is set to  $\tau = 0.1$ . For the low-rank implementation Strang splitting is used.

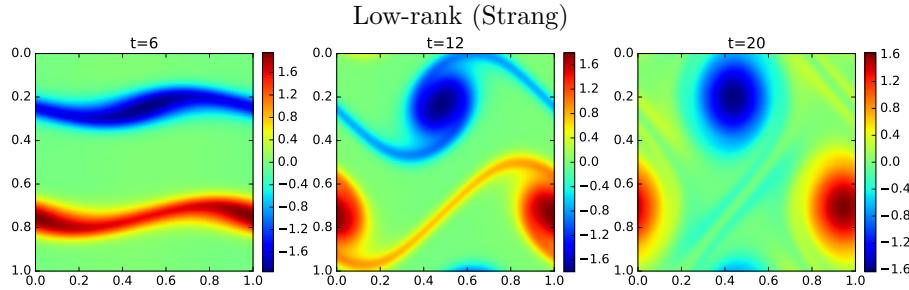


FIG. 4.4. The time evolution of the vorticity  $\omega = \partial_x u_2 - \partial_y u_1$  for the shear flow given by initial values (4.2) is shown (with  $Re = 1000$ ). The dynamical low-rank splitting is used as the integrator. For all simulations centered differences with 128 grid points in the space directions and 16 grid points in the velocity directions are used. The time step size is set to  $\tau = 0.1$  and Strang splitting has been employed.

complicated multiphase flows; see, e.g., [7]. The advantage of this approach, compared to a direct discretization, is that only the collision operator needs to be modified. This fact should make an extension of our approach to multiphase flows relatively straightforward. The distribution function for each phase would be restricted to a low-rank manifold. The resulting evolution equations for the different phases of the fluid are then only coupled by the collision operator. All of this is the subject of future research.

Furthermore, the method proposed here offers a path forward for simulations that need to resolve some kinetic effects. Such problems are common in various fields

of plasma physics. Full-scale simulations with the Boltzmann (collisional Vlasov) equation are often prohibitive from a computational point of view. However, as has been shown in [9] low-rank approximations are still able to resolve a range of kinetic effects quite well. The method proposed here would thus conceivably allow us to extend fluid models (say, magnetohydrodynamics) to a regime in which kinetic effects are needed.

**Acknowledgment.** We would like to thank Christian Lubich (University of Tübingen) for the many helpful discussions.

## REFERENCES

- [1] S. ANSUMALI AND I. V. KARLIN, *Kinetic boundary conditions in the lattice Boltzmann method*, Phys. Rev. E, 66 (2002), 026311.
- [2] S. ASHBY ET AL., *The Opportunities and Challenges of Exascale Computing*, Report of the ASCAC Subcommittee on Exascale Computing, 2010.
- [3] C. BARDOS, F. GOLSE, AND C. D. LEVERMORE, *Fluid dynamic limits of kinetic equations II. Convergence proofs for the Boltzmann equation*, Commun. Pure Appl. Math., 46 (1993), pp. 667–753.
- [4] C. BARDOS, F. GOLSE, AND D. LEVERMORE, *Fluid dynamic limits of kinetic equations. I. Formal derivations*, J. Stat. Phys., 63 (1991), pp. 323–344.
- [5] J. B. BELL, P. COLELLA, AND H. M. GLAZ, *A second-order projection method for the incompressible Navier-Stokes equations*, J. Comput. Phys., 85 (1989), pp. 257–283.
- [6] Y. BO, P. WANG, Z. GUO, AND L. WANG, *DUGKS simulations of three-dimensional Taylor-Green vortex flow and turbulent channel flow*, Comput. Fluids, 155 (2017), pp. 9–21.
- [7] S. CHEN AND G. D. DOOLEN, *Lattice Boltzmann method for fluid flows*, Annu. Rev. Fluid Mech., 30 (1998), pp. 329–364.
- [8] D. CONTE AND C. LUBICH, *An error analysis of the multi-configuration time-dependent Hartree method of quantum dynamics*, ESAIM Math. Model. Numer. Anal., 44 (2010), pp. 759–780.
- [9] L. EINKEMMER AND C. LUBICH, *A low-rank projector-splitting integrator for the Vlasov–Poisson equation*, SIAM J. Sci. Comput., 40 (2018), pp. B1330–B136, <https://doi.org/10.1137/18M116383X>.
- [10] L. EINKEMMER AND C. LUBICH, *A quasi-conservative dynamical low-rank algorithm for the Vlasov equation*, SIAM J. Sci. Comput., to appear.
- [11] L. EINKEMMER AND M. WIESENBERGER, *A conservative discontinuous Galerkin scheme for the 2D incompressible Navier–Stokes equations*, Comput. Phys. Commun., 185 (2014), pp. 2865–2873.
- [12] A. FAKHARI AND T. LEE, *Numerics of the lattice Boltzmann method on nonuniform grids: Standard LBM and finite-difference LBM*, Comput. Fluids, 107 (2015), pp. 205–213.
- [13] Z. GUO, K. XU, AND R. WANG, *Discrete unified gas kinetic scheme for all Knudsen number flows: Low-speed isothermal case*, Phys. Rev. E, 88 (2013), 033305.
- [14] J. HAEGEMAN, C. LUBICH, I. OSELEDETS, B. VANDEREYCKEN, AND F. VERSTRAETE, *Unifying time evolution and optimization with matrix product states*, Phys. Rev. B, 94 (2016), 165116.
- [15] T. JAHNKE AND W. HUISINGA, *A dynamical low-rank approach to the chemical master equation*, J. Math. Biol., 70 (2008), pp. 2283–2302.
- [16] E. KIERI, C. LUBICH, AND H. WALACH, *Discretized dynamical low-rank approximation in the presence of small singular values*, SIAM J. Numer. Anal., 54 (2016), pp. 1020–1038.
- [17] J. KIM, P. MOIN, AND R. MOSER, *Turbulence statistics in fully developed channel flow at low Reynolds number*, J. Fluid Mech., 177 (1987), pp. 133–166.
- [18] O. KOCH AND C. LUBICH, *Dynamical low-rank approximation*, SIAM J. Matrix Anal. Appl., 29 (2007), pp. 434–454.
- [19] O. KOCH AND C. LUBICH, *Dynamical tensor approximation*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2360–2375.
- [20] K. KORMANN, *A semi-Lagrangian Vlasov solver in tensor train format*, SIAM J. Sci. Comput., 37 (2015), pp. 613–632.
- [21] A. KRÄMER, K. KÜLLMER, D. REITH, W. JOPPICH, AND H. FOYSI, *Semi-Lagrangian off-lattice Boltzmann method for weakly compressible flows*, Phys. Rev. E, 95 (2017), 023305.
- [22] P. LALLEMAND AND L. LUO, *Theory of the lattice Boltzmann method: Dispersion, dissipation, isotropy, Galilean invariance, and stability*, Phys. Rev. E, 61 (2000), pp. 6546–6562.

- [23] P. LAVALLEE, J. P. BOON, AND A. NOULLEZ, *Boundaries in lattice gas flows*, Phys. D, 47 (1991), pp. 233–240.
- [24] M. LENTINE, J. T. GRÉTARSSON, AND R. FEDKIW, *An unconditionally stable fully conservative semi-Lagrangian method*, J. Comput. Phys., 230 (2011), pp. 2857–2879.
- [25] J. LIU AND C. SHU, *A high-order discontinuous Galerkin method for 2D incompressible flows*, J. Comput. Phys., 160 (2000), pp. 577–596.
- [26] C. LUBICH, *From Quantum to Classical Molecular Dynamics: Reduced Models and Numerical Analysis*, European Mathematical Society, 2008.
- [27] C. LUBICH, *Time integration in the multiconfiguration time-dependent Hartree method of molecular quantum dynamics*, Appl. Math. Res. Express. AMRX, 2015 (2015), pp. 311–328.
- [28] C. LUBICH AND I. V. OSELEDETS, *A projector-splitting integrator for dynamical low-rank approximation*, BIT, 54 (2014), pp. 171–188.
- [29] C. LUBICH, I. V. OSELEDETS, AND B. VANDEREYCKEN, *Time integration of tensor trains*, SIAM J. Numer. Anal., 53 (2015), pp. 917–941.
- [30] C. LUBICH, T. ROHWEDDER, R. SCHNEIDER, AND B. VANDEREYCKEN, *Dynamical approximation by hierarchical Tucker and tensor-train tensors*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 470–494.
- [31] C. LUBICH, B. VANDEREYCKEN, AND H. WALACH, *Time Integration of Rank-Constrained Tucker Tensors*, preprint, arXiv:1709.02594, 2017.
- [32] R. S. MAIER, R. S. BERNARD, AND D. W. GRUNAU, *Boundary conditions for the lattice Boltzmann method*, Phys. Fluids, 8 (1996), pp. 1788–1801.
- [33] H. MENA, A. OSTERMANN, L. PFURTSCHELLER, AND C. PIAZZOLA, *Numerical Low-Rank Approximation of Matrix Differential Equations*, arXiv:1705.10175, 2017.
- [34] H.-D. MEYER, F. GATTI, AND G. A. WORTH, *Multidimensional Quantum Dynamics*, John Wiley & Sons, New York, 2009.
- [35] H. D. MEYER, U. MANTHE, AND L. S. CEDERBAUM, *The multi-configurational time-dependent Hartree approach*, Chem. Phys. Lett., 165 (1990), pp. 73–78.
- [36] M. MIN AND T. LEE, *A spectral-element discontinuous Galerkin lattice Boltzmann method for nearly incompressible flows*, J. Comput. Phys., 230 (2011), pp. 245–259.
- [37] E. MUSHARBASH, F. NOBILE, AND T. ZHOU, *Error analysis of the dynamically orthogonal approximation of time dependent random PDEs*, SIAM J. Sci. Comput., 37 (2015), pp. A776–A810.
- [38] R. D. NAIR, J. S. SCROGGS, AND F. H. M. SEMAZZI, *A forward-trajectory global semi-Lagrangian transport scheme*, J. Comput. Phys., 190 (2003), pp. 275–294.
- [39] A. NONNENMACHER AND C. LUBICH, *Dynamical low-rank approximation: applications and numerical experiments*, Math. Comput. Simul., 79 (2008), pp. 1346–1357.
- [40] J. M. QIU, *High-order mass-conservative semi-Lagrangian methods for transport problems*, in Handbook of Numerical Methods for Hyperbolic Problems: Basic and Fundamental Issues, Handb. Numer. Anal. 17, Elsevier/North-Holland, Amsterdam, 2016, pp. 353–382.
- [41] A. ROBERT, *A stable numerical integration scheme for the primitive meteorological equations*, Atmos. Ocean, 19 (1981), pp. 35–46.
- [42] F. SCHORNBAUM AND U. RÜDE, *Massively parallel algorithms for the lattice Boltzmann method on nonuniform grids*, SIAM J. Sci. Comput., 38 (2016), pp. C96–C126.
- [43] A. STANIFORTH AND J. CÔTÉ, *Semi-Lagrangian integration schemes for atmospheric models – A review*, Mon. Weather Rev., 119 (1991), pp. 2206–2223.
- [44] B. VAN LEER, *Upwind and high-resolution methods for compressible flow: From donor cell to residual-distribution schemes*, in Proceedings of the 16th AIAA Computational Fluid Dynamics Conference, 2006.
- [45] P. WANG, L.-P. WANG, AND Z. GUO, *Comparison of the LBE and DUGKS Methods for DNS of Decaying Homogeneous Isotropic Turbulence*, preprint, arXiv:1601.03815, 2016.
- [46] P. WANG, L. ZHU, Z. GUO, AND K. XU, *A comparative study of LBE and DUGKS methods for nearly incompressible flows*, Commun. Comput. Phys., 17 (2015), pp. 657–681.
- [47] S. WOLFRAM, *Cellular automaton fluids 1: Basic theory*, J. Stat. Phys., 45 (1986), pp. 471–526.
- [48] D. XIU AND G. E. KARNIADAKIS, *A semi-Lagrangian high-order method for Navier–Stokes equations*, J. Comput. Phys., 172 (2001), pp. 658–684.
- [49] K. XU AND X. HE, *Lattice Boltzmann method and gas-kinetic BGK scheme in the low-Mach number viscous flow simulations*, J. Comput. Phys., 190 (2003), pp. 100–117.
- [50] X. YIN AND J. ZHANG, *An improved bounce-back scheme for complex boundary conditions in lattice Boltzmann method*, J. Comput. Phys., 231 (2012), pp. 4295–4303.
- [51] M. YOKOKAWA, K. ITAKURA, A. UNO, T. ISHIHARA, AND Y. KANEDA, *16.4-Tflops direct numerical simulation of turbulence by a Fourier spectral method on the Earth Simulator*, in Proceedings of the 2002 ACM/IEEE Conference on Supercomputing, 2002, pp. 1–17.