Numerical solution of the Boltzmann equation with S-model collision integral using tensor decompositions

A.V. Chikitkin\textsuperscript{a,}\textsuperscript{*}, E.K. Kornev\textsuperscript{a}, V.A. Titarev\textsuperscript{a,1}

\textsuperscript{a}Moscow Institute of Physics and Technology, 9 Institutskiy per., Dolgoprudny, Moscow Region, 141701, Russian Federation

\textsuperscript{b}Federal Research Center Computer Science and Control, Russian Academy of Sciences, Vavilov str. 40, Moscow, 119333, Russian Federation

Abstract

Paper presents a new solver for numerical solution of the Boltzmann kinetic equation with Shakhov model collision integral (S-model) for arbitrary spatial domains. Numerical method utilizes Tensor-Train decomposition, which allows to reduce required computer memory for up to 30 times even on a moderate velocity mesh. This improvement is achieved by representing values of distribution function on the structured velocity mesh as a 3D tensor in Tensor-Train format. The resulting numerical method makes it possible to solve complex 3D problems on modern desktop computers.

Our implementation may serve as a prototype code for researchers concerned with numerical solution of the kinetic equations in 3D domains by the discrete velocity method.

\textit{Keywords:} rarefied gas dynamics, Shakhov model, Discrete velocity method, unstructured mesh, Tensor-Train, Tensor decomposition

\textbf{PROGRAM SUMMARY}

\textsuperscript{*}Corresponding author.

\textit{E-mail address:} chikitkin.av@mipt.ru

\textit{Preprint submitted to Computer Physics Communications} December 11, 2019
Program Title: Boltzmann-T

Licensing provisions: MIT

Programming language: Python 2.7

External libraries: Solver is based on the ttpy [1] library

Nature of problem: Numerical solution of the Boltzmann kinetic equation with the S-model collision integral in arbitrary spatial domain

Solution method: Discrete velocity method utilizing tensor decomposition for memory reduction

Restrictions: At present, 1st order space advection scheme is used, solver supports unstructured hexagonal meshes written in StarCD ANSCII format

References

[1] ttpy (https://github.com/oseledets/ttpy) library contains Python implementation of several important procedures for working with tensors in TT-format.

1. Introduction

The Boltzmann kinetic equation (BKE) is the main mathematical model of the theory of rarefied gases. Due to the high dimensionality of the phase space and the complexity of the collision integral, the numerical solution of the BKE is much more complicated and computationally expensive than the numerical solution of macroscopic equations, such as the Navier-Stokes equations of the compressible gas [II].
There are several simplified collision models, which allow to simplify the Boltzmann equation, while preserving a number of its important properties. The simplest is the BGK model [2]. A more accurate approximation is given by the Shakhov model (S-model) [3] and its extension to the diatomic gases by Rykov [4]. Comparisons with calculations using the exact Boltzmann equation, the direct simulation Monte Carlo method, and with experimental data have confirmed good accuracy of the S-model, see e.g. [5, 6, 7, 8, 9] and references therein.

In model kinetic equations the calculation of the collision integral requires only the knowledge of a certain number of macroparameters, or moments of the distribution function, i.e. 3-dimensional integrals over the velocity space. Despite this simplification, their numerical solution is still quite computationally demanding task, especially for three-dimensional applications. One of the approaches to reduce the computational cost and memory requirements of numerical methods for model kinetic equations is to use adaptive mesh in the velocity space [10, 11, 12, 13, 8]. It should be noted that the use of an adaptive unstructured meshes significantly complicates the algorithm of the numerical method and often requires some a-priori information about the problem being solved. The simplest algorithm is constructed with the use of structured Cartesian meshes in the velocity space. In this case, values of the distribution function at all nodes of the mesh form a multidimensional array, which will be hereafter called “tensor”. Therefore, the natural way to speed up the method and reduce the amount of required memory is to use low-rank tensor approximations, which are well-known in the field of linear algebra. This is justified by theoretical estimates showing
that for tensors, generated by the values of smooth functions, such approximations always exist \[14\], \[15\].

There are many studies on this subject. In \[16\] a special tensor format is proposed for approximation of tensors that arise from calculation of the exact collision integral on a tensor mesh. In \[17\] tensor approximations were successfully applied to the numerical method for the Vlasov equation with the BGK model for the collision integral. The memory consumption was reduced 17 times as compared with the standard numerical method on the same meshes. Another version of the numerical method for the Vlasov equation is described in \[18\]. It is noted that the use of tensor decompositions reduces the memory by more than 100 times. A recent paper \[19\] describes the general idea of using tensor decomposition in the numerical method for partial differential equations of a certain type and presents the results of test calculations of simple problems for the Boltzmann equation with the BGK model collision integral.

In the cited papers tensor decompositions are applied to tensors formed by values of distribution function on structured tensor mesh in both physical and velocity space. Such tensors have dimension 4 or 6 depending on the dimensionality of problem. For such dimensions low-rank approximations are especially effective. However, this approach is applicable only to problems with simple boundary conditions and simple geometry so that one can use a structured mesh in physical space, while in many applications computational domain has complex shape. For such problems with complex shape one has to use an unstructured mesh in physical space (for example, tetrahedral, or multi-block structured). In this regard, it is more convenient to approximate
tensor formed by the distribution function values only on the velocity mesh at each point of physical space.

In this paper an analogue of the discrete velocity method is proposed, in which the tensors formed by the values of the distribution function on the velocity mesh at various spatial points are approximated using Tensor-Train format. Examples of test calculations are presented, which show that the proposed approach allows to reduce the computer memory consumption 30-50 times while maintaining satisfactory accuracy; CPU time increases only mildly.

2. Mathematical model

In general, the Boltzmann equation of a monatomic gas with a model collision integral has the following form:

$$\frac{\partial f}{\partial t} + \sum_{\alpha} \xi_{\alpha} \frac{\partial f}{\partial x_{\alpha}} = J(f, \xi, a(x, t))$$

(1)

where \( f(t, x, \xi) \) – value of distribution function, \( \xi \) – velocity vector, \( a \) – vector of macroparameters, which are expressed through the moments of the distribution function:

$$n = \int f d\xi, \quad nu = \int \xi f d\xi, \quad \frac{3}{2} mnR_gT + \frac{1}{2} mnu^2 = \frac{1}{2} m \int \xi^2 f d\xi,$$

$$q = \frac{1}{2} m \int \nu v^2 f d\xi, \quad v = \xi - u, \quad \rho = mn, \quad p = \rho R_gT,$$

$$u^2 = \sum_{\alpha=1}^{3} u_{\alpha} u_{\alpha}, \quad v^2 = \sum_{\alpha=1}^{3} v_{\alpha} v_{\alpha}, \quad v^2 = \sum_{\alpha=1}^{3} v_{\alpha} v_{\alpha}, \quad d\xi = d\xi_x d\xi_y d\xi_z.$$
In the Shakhov model [3] the collision integral is given by

\[ J = \nu (f_S - f), \quad \nu = \frac{p}{\mu} \]

\[ f_S = f_M [1 + \frac{4}{5} (1 - Pr) S_\alpha c_\alpha (c^2 - \frac{5}{2})], \quad S_\alpha = \frac{1}{n} \int c_i c_\alpha f d\xi, \]

(3)

\[ f_M = \frac{n}{(2\pi R_g T_w)^{3/2}} \exp(-c^2), \quad c = v/\sqrt{2R_g T}, \quad c^2 = \sum_{\beta=1}^{3} c_\beta c_\beta \]

Here \( \mu = \mu(T) \) – dynamic viscosity, \( Pr = 2/3 \) – Prandtl number, \( f_M \) – locally Maxwell (equilibrium) distribution function, \( R_g \) – gas constant.

At the boundaries of the computational domain in physical space it is necessary to specify distribution function values for molecules whose velocity vector is directed inside the domain. On the surface of the body, the boundary condition of diffuse reflection with full thermal accommodation to the surface temperature \( T_w \) is used. The distribution function of reflected molecules is written as:

\[ f_w = \frac{n_w}{(2\pi R_g T_w)^{3/2}} \exp\left(-\frac{\xi^2}{2R_g T_w}\right) \]

(4)

The density \( n_w \) of reflected molecules is found from the impermeability condition:

\[ \int_{\xi_n > 0} \xi_n f d\xi + \int_{\xi_n < 0} \xi_n f_w d\xi \]

(5)

where \( \xi_n \) is the projection of the velocity onto the normal to the surface, directed outside the computational domain, \( f \) is distribution function of molecules coming to the wall.

For symmetry planes the following boundary condition is set:

\[ f(t, x, \xi) = f(t, x, \xi_1), \quad \xi_1 = \xi - 2\xi_n n. \]

(6)
where $n$ – outward looking unit normal vector for plane.

For the free stream condition the distribution function is equal to the Maxwell distribution function for prescribed values of free-stream macroparameters.

3. Discrete velocity method

In this paper, we use a variant of the discrete velocity method described in [20], [13], [21]. For brevity, we explain the main idea using an explicit first-order method, although implicit scheme of arbitrary approximation order can be used.

We introduce a uniform Cartesian mesh in the velocity space:

$$\xi_{\alpha,i} = \xi_{\text{min}} + (i - 1)\Delta\xi, \ i = 1, \ldots, N, \ \alpha = 1, 2, 3$$

The integrals in the velocity space are replaced by the 2nd order quadrature formula:

$$\int g(\xi) \, d\xi \approx \Delta\xi^3 \sum_{i,j,k=1}^{N} g(\xi_{1i}, \xi_{2j}, \xi_{3k})$$

(7)

The values of the distribution function at the nodes of the velocity mesh form a three-dimensional tensor, which is denoted by $\hat{f}$

$$\hat{f}(t, \mathbf{x})(i, j, k) = f(t, \mathbf{x}, \xi_{1i}, \xi_{2j}, \xi_{3k}), \ i, j, k = 1, \ldots, N$$

(8)

Writing the kinetic equation at each node of the velocity mesh, we obtain a system of $N^3$ linear constant-coefficient equations with a source term. This system can be written in the tensor form:

$$\hat{f}_t + (\hat{\xi}_1 \circ \hat{f})_{x_1} + (\hat{\xi}_2 \circ \hat{f})_{x_2} + (\hat{\xi}_3 \circ \hat{f})_{x_3} = \nu(\hat{f}_S - \hat{f})$$

(9)
where "\(\circ\)" denotes the component-wise (Hadamard) product of tensors, \(\hat{\xi}_\alpha\) are tensors formed by the values of the \(\alpha\) velocity component at each node of the velocity mesh, \(\alpha = 1, 2, 3\).

A standard finite-volume method is used to discretize the left-hand side of the resulting system. The computational domain in physical space is divided into finite volumes (polyhedrons) \(V_i, i = 1, \ldots, N_C\). System (9) is integrated over \(V_i\), the volume integral is replaced by the sum of surface integrals over the cell faces from the fluxes projected onto the normal to the face. Thus we obtain a semi-discrete scheme of the following form:

\[
\frac{d\hat{f}_i}{dt} = -\frac{1}{|V_i|} \sum_l \hat{\Phi}_{li} + \hat{J}_i, \ i = 1, \ldots, N_C
\]

\(\hat{\Phi}_{li} = \int_{A_{li}} \hat{\xi}_{n,li} \circ \hat{f} dS, \ \hat{\xi}_{n,li} = n_{1,li} \hat{\xi}_1 + n_{2,li} \hat{\xi}_2 + n_{3,li} \hat{\xi}_3\) (10)

Here \(\hat{f}_i\) is the tensor formed by the values of the integral averages of the distribution function over cell \(i\), \(n_{li}\) is the outer normal vector of the \(l\)-th face of the cell with index \(i\), \(A_{li}\) is the face of the cell. The final form of the method depends on the flux approximation and time-marching scheme.

For brevity we consider first order method: distribution function is assumed to be piece-wise constant, the numerical flux is given by an exact or approximate solution of the one-dimensional Riemann problem along normal vector at each face center. In the case of the exact solution (CIR scheme), the expression for flux is the following:

\[
\hat{\Phi}_{li} = |A_{li}| \left( \frac{1}{2} \hat{\xi}_{n,li} \circ (\hat{f}_L + \hat{f}_R) - \frac{1}{2} |\hat{\xi}_{n,li}| (\hat{f}_L - \hat{f}_R) \right)
\]

(11)

Here \(\hat{f}_L, \hat{f}_R\) are the values to the left and right of the face with respect to the normal. If the cell is adjacent to the boundary, one of these values is set
based on the boundary condition.

It should be noted, that in (11) instead of $|\hat{\xi}_{n,li}|$ some estimate can be used. This may be interpreted as Riemann solver of the Rusanov type.

Using the explicit Euler method to solve the ODE system, we obtain the fully discrete one-step method:

$$\frac{\hat{f}_i^{n+1} - \hat{f}_i^n}{\Delta t} = R_i^n = -\frac{1}{|V_i|} \sum_l \hat{\Phi}_{li} + \hat{J}_i(\hat{f}_i^n), \quad i = 1, \ldots, N_C$$

(12)

Let us introduce additional notation for a brief description of the computational algorithm. We denote $N_F$ – the number of all faces in the mesh. We assume that the normal to the face is given on each face. Let $\text{sign}[i,j]$ be the number equal to +1 if the normal to the face $j$ is external with respect to the cell $i$, and −1 otherwise. The procedure for calculating distribution function in each cell on the next time layer $n + 1$ is listed in algorithm [1].
Algorithm 1 Algorithm: time step

1: \ldots \quad \triangledown \text{set boundary conditions}
2: \text{for } j = 1, N_F \text{ do} \quad \triangledown \text{fluxes on faces}
3: \quad \hat{F}[j] = \frac{1}{2} \hat{\xi}_n[j] \odot \left( \hat{f}_L[j] + \hat{f}_R[j] \right) - \frac{1}{2} |\hat{\xi}_n[j]| \odot \left( \hat{f}_R[j] - \hat{f}_L[j] \right)
4: \text{end for}
5: \text{for } i = 1, N_C \text{ do} \quad \triangledown \text{compute right-hand side } R
6: \quad \hat{R}[i] = \text{computeJ}(\hat{f}[i]) \quad \triangledown \text{compute collision integral}
7: \quad \text{for } j \in \{ \text{faces of cell } i \} \text{ do} \quad \triangledown \text{loop over faces of cell } i
8: \quad \quad \hat{R}[i] = \hat{R}[i] + \text{sign}[i, j] \frac{A[j]}{V[i]} \hat{F}[j] \quad \triangledown \text{add flux with sign}
9: \quad \text{end for}
10: \text{end for}
11: \text{for } i = 1, N_C \text{ do} \quad \triangledown \text{Compute values on the next time layer}
12: \quad \hat{f}[i] = \hat{f}[i] + \Delta t \hat{R}[i]
13: \text{end for}

Pseudo-code of the function for computing the model collision integral is given in algorithm 3. The function \textit{sum} calculates the sum of all elements of the tensor, the symbol \( \hat{1} \) denotes the tensor consisting of ones: \( \hat{1}(i_1, i_2, i_3) = 1 \), the function \textit{maxwell} returns the tensor formed by values of Maxwell function for given macroparameters on the velocity mesh.
Algorithm 2 Calculation of collision integral

1: procedure computeJ($\hat{f}, \hat{\xi}_x, \hat{\xi}_y, \hat{\xi}_z$)
2:     $n = \Delta \xi^3 \text{sum}(\hat{f})$ \hspace{1cm} \triangleright \text{numerical density}$
3:     $u_x = \Delta \xi^3 \text{sum}(\hat{\xi}_x \circ \hat{f})/n, \; u_y = \ldots$
4:     $\hat{\xi}_2 = \hat{\xi}_x \circ \hat{\xi}_x + \hat{\xi}_y \circ \hat{\xi}_y + \hat{\xi}_z \circ \hat{\xi}_z$
5:     $u^2 = u_x^2 + u_y^2 + u_z^2$
6:     $T = \frac{1}{3nR_g} \left( \Delta \xi^3 \text{sum}(\hat{\xi}_2 \circ \hat{f}) - nu^2 \right)$
7:     $\rho = mn, p = \rho R_g T$ \hspace{1cm} \triangleright m - mass of one molecule
8:     $\hat{c}_x = \frac{\hat{\xi}_x - u_x \hat{1}}{\sqrt{2R_g T}}, \; \hat{c}_y = \ldots$
9:     $\hat{c}_2 = \hat{c}_x \circ \hat{c}_x + \hat{c}_y \circ \hat{c}_y + \hat{c}_z \circ \hat{c}_z$
10:    $S_x = \Delta \xi^3 \text{sum}(\hat{c}_x \circ \hat{c}_2 \circ \hat{f}), \; S_y = \ldots$
11:    $\hat{f}_M = \text{maxwell}(n, T, u_x, u_y, u_z, \hat{\xi}_x, \hat{\xi}_y, \hat{\xi}_z)$
12:    $\hat{f}_S = \hat{f}_M \circ \left( 1 + \frac{4}{5} (1 - Pr) (S_x \hat{c}_x + S_y \hat{c}_y + S_z \hat{c}_z) \circ \left( \hat{c}_2 - \frac{5}{2} \hat{1} \right) \right)$
13:    $J = \frac{p}{\mu(T)} (\hat{f}_S - \hat{f})$
14:    return $J$
15: end procedure

The main observation that can be made from listed algorithms is that one step of the numerical method requires only a few simple operations with tensors, namely:

1. component-wise sum of two tensors
2. component-wise product of two tensors
3. sum of all elements in a tensor, or, in the case of nonuniform Cartesian
mesh in velocity space, convolution of the following form:

\[ S = \sum_{i,j,k} \hat{f}(i,j,k)u(i)v(j)w(k) \]  

(13)

where \( u, v, w \) are 1D vectors consisting of weights of a quadrature rule.

It follows from this observation that if there is some parametric representation of tensors, storage of all tensor elements can be avoided.

The same applies for many implicit methods. In our code we implemented a version of LU-SGS method. This method is very effective, since it’s computational cost is only about 50% larger then one of the explicit method. For brevity we do not list all formulas, details of the implementation in the context of kinetic solvers can be found in [22, 13, 23].

In the next section we briefly formulate general idea of tensor decompositions and describe Tensor-Train decomposition used in our work.

4. Tensor decompositions

Tensor decompositions extend the idea of separation of variables to multidimensional arrays. In the two-dimensional case, for any matrix of rank \( r \) the singular value decomposition (SVD) exists:

\[ A = U\Sigma V^T, \quad A(i_1, i_2) = \sum_{k=1}^{r} \sigma_k u_k(i_1)v_k(i_2) \]  

(14)

The Eckart-Young theorem states that the best approximation of the rank \( r' \) < \( r \) to the matrix \( A \) in the 2-norm and the Frobenius norm is obtained by dropping the \( r - r' \) terms in SVD of \( A \), which correspond to the smallest singular numbers. The low-rank approximation allows one to reduce the
required storage memory to $n \cdot r$, where $n$ is the size of the matrix (for the case of a square matrix) and reduce complexity of matrix-vector operations.

A direct generalization of the form (14) and of the definition of the rank of the tensor in the multidimensional case is the canonical decomposition (CANDDECOMP, PARAFAC) [24].

$$A(i_1, \ldots, i_d) = \sum_{\alpha=1}^{r} U^1_{\alpha}(i_1) \ldots U^d_{\alpha}(i_d)$$ (15)

where $r$ is called tensor rank.

It’s use in numerical methods is limited due to the lack of stable algorithms. Nevertheless, there are theoretical estimates, which show, that tensors formed by values of smooth function on Cartesian meshes can be approximated with high accuracy by low-rank tensor [15].

In the three-dimensional case, the Tucker decomposition is often used [25]:

$$A(i_1, i_2, i_3) = \sum_{k_1, k_2, k_3=1}^{r_1, r_2, r_3} G(k_1, k_2, k_3) u(i_1)v(i_2)w(i_3)$$ (16)

This representation allows to employ robust SVD based procedures for fast linear algebra operation for tensors in this format.

Obviously, Tucker decomposition does not allow to circumvent the “curse of dimensionality”, since $r^d$ elements are needed to store the core $G$ for dimension $d$. However, in many problems the ranks $r_j$ are very small.

There are two formats for tensors of arbitrary dimension $d$, which generalize idea of Tucker format: hierarchical-Tucker (HT) format [26] and Tensor-Train (TT) format [27]. Both formats are based on a dimensionality reduction tree and use the SVD of auxiliary matrices for a low-rank approximation of an arbitrary tensor.
In the current paper we use the TT format, and the ttpy library \(\text{https://github.com/oseledets/ttpy}\), in which all the basic operations with tensors in this format are implemented.

In the TT format a tensor is represented as:

\[
A(i_1, \ldots, i_d) = \sum_{\alpha_1, \ldots, \alpha_{d-1}} G_1(i_1, \alpha_1)G_2(\alpha_1, i_2, \alpha_2) \ldots G_{d}(\alpha_{d-1}, i_d), \alpha_k = 1, r_k
\]

\(G_k\) are called TT-cores. Two cores - the first and the last - are matrices whereas all the rest are 3D tensors. The numbers \(r_k\) are called TT- ranks.

A shorter form is given as a product of matrices:

\[
A(i_1, \ldots, i_d) = G_1(i_1)G_2(i_2) \ldots G_d(i_d)
\]

(18)

Here \(G_1(i_1)\) is a row vector, \(G_d(i_d)\) is a column vector, all the others \(G_k(i_k)\) are matrices.

Storage of the TT tensor requires \(O(dnr^2)\) memory, therefore, for small ranks, significant memory savings are obtained compared to \(n^d\) storage for the full tensor.

The following operations with TT tensors are important for applying TT decomposition in the discrete velocity method:

1. Computation of tensor \(B\) in TT format with minimum TT ranks, which approximates the full tensor \(A\) with a given relative accuracy:

\[
\|A - B\|_F \leq \epsilon \|A\|_F
\]

\(\| \cdot \|_F\) – Frobenius norm. Algorithm requires \(O(dnr^3)\) operation, if \(r_k \leq r\).
2. Component-wise sum: if tensors $A$ and $B$ of the same size are represented in TT format, i.e.

$$A(i_1, \ldots, i_d) = A_1(i_1) \ldots A_d(i_d), \quad B(i_1, \ldots, i_d) = B_1(i_1) \ldots B_d(i_d)$$

then $C = A + B$ has TT-representation with cores:

$$C_k(i_k) = \begin{bmatrix} A_k(i_k) & 0 \\ 0 & B_k(i_k) \end{bmatrix}, \quad k = 2, \ldots, d - 1$$

$$C_1(i_1) = [A_1(i_1) \quad B_1(i_1)], \quad C_d(i_d) = \begin{bmatrix} A_d(i_d) \\ B_d(i_d) \end{bmatrix}$$

Element-wise sum does not require any calculations while the TT ranks of the sum are equal to the sum of the TT ranks of the $A$ and $B$.

3. The element-wise (Hadamard) product $C = A \circ B$ of two tensors is represented in TT format with cores:

$$C_k(i_k) = A_k(i_k) \otimes B_k(i_k)$$

where $\otimes$ – Kronecker product of matrices.

Element-wise multiplication requires $O(dnr^4)$ operations; the ranks of the product are equal to the product of the ranks of the factors.

4. Algorithm for tensor rounding in TT-format, i.e. for tensor $A$ in the TT format with ranks $r_k$ one can find tensor $B$ with lower ranks such that

$$\|A - B\|_F \leq \epsilon \|A\|_F$$

The algorithm consists of a sequence of SVD and QR decompositions of auxiliary unfolding matrices and has complexity $O(dnr^3)$
5. Computation of convolution ($O(dnr^2)$):

$$S = \sum_{i_1\ldots i_d} A(i_1, \ldots, i_d)u_1(i_1) \ldots u_d(i_d) \quad (19)$$

All the listed basic procedures allow to rewrite the algorithm of discrete velocity method as a sequence of operations with tensors in TT-format. Element-wise operations are replaced by their TT-analogues, besides, intermediate rounding is added to prevent the growth of TT-ranks.

It should be noted that TT format is redundant for $d = 3$, because TT-cores are still 3D tensors. For the case $d = 3$ the Tucker expansion may be more efficient. Nevertheless, the modification of the algorithm will be essentially the same for any tensor format, but with the TT format one can easily switch to higher dimension, for example, for problems of state-to-state kinetics, where the distribution function depends also on the energy level numbers. Besides, in low-dimensional problems it is possible to apply artificial increase of dimension, which often gives an additional gain [28]. For these reasons, the TT format was chosen in this work.

The next section describes the details of the adaptation of the algorithm.

5. **Tensorized discrete velocity method**

In the tensorized version of the method all low-rank arrays are constructed immediately in the TT form. Since the Maxwell distribution function is the product of 3 1D functions, we can construct the TT tensor with ranks 1 with corresponding TT-cores (projections of 1D functions onto 1D mesh).
Since the tensors $\hat{\xi}_1, \hat{\xi}_2, \hat{\xi}_3$ also have rank 1, most of the tensors arising in the calculation of the collision integral have small ranks. For example, the tensor $\hat{f}^S$ has TT ranks $\leq 10$, regardless of the size of the velocity mesh.

The tensor $\hat{\xi}_n$ on each face has a TT rank of $\leq 3$ (actually, ranks are at most 2), because it is the sum of three rank-1 tensors:

$$\hat{\xi}_n = n_1\hat{\xi}_1 + n_2\hat{\xi}_2 + n_3\hat{\xi}_z$$

The only bottleneck is the tensor $|\hat{\xi}_n[j]|$ or the tensor $\hat{\xi}^+_n$, in which the negative values in the tensor $\hat{\xi}_n$ is replaced by 0. These tensors can not be approximated with high accuracy by tensors with small ranks since in general case the normal vector $n$ does not coincide with one of the coordinate axes and tensor is a projection of non-smooth function on the mesh.

Nevertheless, in the formula for the face flux (11) one can replace the tensor $|\hat{\xi}_n[j]|$ with some estimate. This can be interpreted as replacing the exact numerical flux with a Rusanov-type flux. In our numerical experiments we used TT approximations of $|\hat{\xi}_n[j]|$ with ranks 4 for all faces.

The figure 1 shows a comparison between the cross sections at $i_3 = const$ of the exact tensor $|\hat{\xi}_n[j]|$ and its low-rank approximation. It can be seen that the estimate mimics well the exact function. Tests have shown that for first- and second-order schemes this approximation does not significantly affect the accuracy of the computed solution, but for higher-order schemes, a better approximation may be needed.

After all operations which may lead to a large increase in TT ranks, TT rounding was added with prescribed relative error $\epsilon$. It should be noted that when applying a specific tensor format, it is necessary to take into account the computational complexity of each element-wise operation and
rounding, not only the asymptotic growth rate, but also the constants included in the estimates. For example, in the method under consideration, it makes no sense to insert rounding after each operation, which leads to an increase in ranks: it is more optimal to do rounding after several operations. In addition, it makes sense to reorder some operations, since it is more preferable to avoid Hadamard multiplication of two tensors with large ranks while element-wise sums for the same ranks are relatively cheap.

The key modification in the tensorized version of the implicit LU-SGS method is the simplified element-wise division by the following tensor in each cell:

\[
\hat{D}_i = \left( \frac{1}{\Delta t} + \nu_i \right) I + \frac{1}{2 |V_i|} \sum_{j \in \text{neib}(i)} S_{ji} |\hat{\xi}_{n,ji}| \tag{20}
\]

where \(S_{ji}\) is face area. There is no algorithm for exact component-wise di-
vision in TT-format. One way to compute the result in TT-format is to use some cross-approximation technique [29], which computes small number of elements of the resulting tensor and constructs low-rank approximation based on these values. We adopt a simpler approach: since tensor $D_i$ is used in a preconditioner we can use any estimate $\hat{D}_{i}^{\text{est}}$ providing that $\hat{D}_{i}^{\text{est}} > \hat{D}_i$. Therefore it is convenient to use 1-rank approximation of the form $\hat{D}_{i}^{\text{est}}(i_1, i_2, i_3) = u_1(i_1)u_2(i_2)u_3(i_3)$ ($u_1, u_2, u_3$ - 1d vectors), since exact element-wise division can be computed in $O(nr^2)$ operations for any TT-tensor:

$$\hat{B}(i_1, i_2, i_3) = \frac{A(i_1, i_2, i_3)}{\hat{D}_{i}^{\text{est}}(i_1, i_2, i_3)} = \frac{G_1^A(i_1) G_2^A(i_2) G_3^A(i_3)}{u_1(i_1) u_2(i_2) u_3(i_3)} \frac{G_1^A(i_1) G_2^A(i_2) G_1^A(i_2)}{u_1(i_1) u_2(i_2) u_1(i_2)}$$

Such operation can be easily implemented using NumPy package broadcasting ability.

6. Implementation

For comparison between two methods both standard discrete velocity method and it’s tensorized version have been implemented in Python language. We use Python 2.7 since ttpy library is based on this Python version.

Program consists of three main Python modules:

1. read_starc.py – an auxiliary module for reading an unstructured mesh in StarCD format. It contains class Mesh, constructor of this class takes path to the folder with mesh files and creates an object where all information needed in numerical method is stored (cell volumes, face
normals, etc.) This object is then serialized using pickle module. After that in run script mesh object is read from serialization file.

2. solver.py – this module contains function implementing standard first order discrete velocity method and additional routines and structures.

3. solver_tt.py – contains implementation of tensorized version of the discrete velocity method.

Besides, there are four scripts for two test problems: the first is 1D shock wave structure problem, and the second – flow past planar circular cylinder (see section 7).

The shock wave test can be used for the first validation and experiments, since the spatial mesh is very small and so is the computational time.

The second test demonstrates that the tensorized version of algorithm provides a significant memory reduction in real-life problems.

The spatial mesh for additional tests can be created using any appropriate software. StarCD is a widespread format so one can convert mesh from almost any format to StarCD format. We used Ansys ICEM to create mesh for our tests.

In order to solve a new problem one need to create an object of the “Problem” class (see listing 1) and pass it to solver together with object of “Mesh” class.

Listing 1: “Problem” class

```python
class Problem:
    def __init__(self, bc_type_list = None,
                 bc_data = None, f_init = None):
        # list of boundary conditions’ types
```
# according to order in starcd '.bnd' file
# list of strings
self.bc_type_list = bc_type_list
# data for b.c.
# list of lists
self.bc_data = bc_data
# Function to set initial condition
self.f_init = f_init

For example, in the listing boundary and initial condition for flow past cylinder is defined. For now, a basic set of boundary conditions is implemented, including in-out conditions (which are actually the same), wall b.c. and symmetry in each coordinate direction.

Listing 2: Setting initial and boundary condition for flow past cylinder

```python
f_init = lambda x, y, z, vx, vy, vz: tt.tensor(
solver_tt.f_maxwell(
    vx, vy, vz, T_l, n_l, u_l, 0., 0., gp.Rg))

f_bound = tt.tensor(
solver_tt.f_maxwell(
    vx, vy, vz, T_l, n_l, u_l, 0., 0., gp.Rg))

fmax = tt.tensor(
solver_tt.f_maxwell(
    vx, vy, vz, T_w, 1., 0., 0., 0., gp.Rg))

problem = solver_tt.Problem
```
7. Test problem

The problem of a high-speed rarefied gas flow past a circular cylinder is considered. The setup of the problem is taken from [30]. The kinetic solution by the S-model equation and the exact Boltzmann equation was compared against the DSMC solution in a number of recent papers [8, 7, 31] for large free-stream Mach numbers (up to 25) and good agreement was observed. The geometry of the computational domain along with the spatial mesh is shown in the figure 2. The problem is essentially two-dimensional, but we solve it as 3D on the 3D mesh with one cell in $z$ direction. Mesh is treated as unstructured by the solver. The flow is directed along $x$-axis, boundary condition (4) is set on the wall. At the remaining boundaries, the symmetry boundary condition is used.

The following dimensional parameters was chosen: Free stream velocity $v_0 = 2630$ m/s, $n_0 = 2 \cdot 10^{23}$ m$^{-3}$, $T_0 = 200\, K$, wall temperature $T_w = 5T_0$, the cylinder radius $r = 1.35 \cdot 10^{-5}$ m. Knudsen number calculated by the parameters of the free stream and the radius of the cylinder $Kn \approx 0.56$, free
stream Mach number equals 10. The power law was used for viscosity:

\[ \mu(T) = \mu_0 \left( \frac{T}{T_0} \right)^{0.734} \]  

\hspace{1cm} (21)

Figure 2: Computational domain and mesh for test problem

In the tensorized method the relative accuracy \( \epsilon = 10^{-7} \) was used. Uniform velocity mesh contains \( N = 64 \) nodes in each direction, number of cells in the spatial mesh equals 1600. For this test case we choose a relatively coarse space mesh so that the standard method can be run on a desktop computer. Therefore, near the surface the mesh resolution is poor (the height of the first cell is too large), but here we concentrate on comparison between two methods rather than accurate computation of the heat transfer.

The figure 3 shows the temperature distribution obtained by the standard and tensorized methods. The figure 4 shows graphs of the temperature versus the normal coordinate for the stagnation line and at an angle of 45 degrees. Temperature was chosen for comparison since it is more sensible quantity, differences for density and velocity are much smaller. The plots
show that tensorized method provides good accuracy even at the shock wave front.

In the figure 5, distribution of the ratio \( (r_1 r_2 + r_1 + r_2)/n_V^2 \) is shown, where \( r_k \) are the TT-ranks of the distribution function in each cell, and \( N \) - number of nodes in velocity mesh in one dimension. I can be seen that TT-rounding works like adaptive mesh refinement: near the inflow, where the distribution function is almost equilibrium, ranks are very small. Near the shock wave and surface ranks automatically increases in order to provide prescribed accuracy.

It is clear from presented figures that tensorized method yields almost the same accuracy as standard discrete velocity method. Memory size for storage of distribution function in tensorized method is more than 30 smaller than in the standard method.

Another advantage of tensorized method is that it still allows to study behaviour of distribution function itself. Figure 6 shows z-slice of distribution function tensor in cell with \( x = 2.46 \times 10^{-5}, y = 10^{-6} \) (near stagnation line on shock front). In this area flow is strongly non-equilibrium and distribution function has two peaks. It can be seen that difference is negligible, i.e. tensorized method successfully captures the main properties of distribution function. Despite the significant memory reduction, for this test case computational time of both methods is approximately equal. The reason is the high cost of element-wise multiplication and TT-rounding. The same situation is reported in other studies, for instance [17]. However, for this test we use very small velocity mesh (64\(^3\) nodes). For larger meshes tensorized algorithm would be faster than the standard one.
8. Concluding remarks and perspectives

The Boltzmann-T solver for numerical solution of kinetic Boltzmann equation is described. The solver provides a working example of implementation of a tensorized discrete velocity method. This implementation demonstrates prospects of using tensor decompositions for significant mem-
From our experience we draw the following conclusions, which may be useful for other researchers dealing with tensorized versions of discrete velocity method:

1. For the present case Tucker format seems to be more efficient than Tensor Train format, since storage reduces to $O(r^3 + nr)$ instead of $O(nr^2)$ in case of TT format.

2. Problem with tensors generated by non-smooth function (such as $|\hat{\xi}_n|$) can be overcome if the spherical coordinate system is used in the veloc-
Figure 5: Ratio \( \left( r_1 r_2 + r_1 + r_2 \right) / N^2 \) for relative accuracy \( \epsilon = 10^{-7} \)

Figure 6: Slice of distribution function tensor. Left - standard method, right - tensorized method

ity space. In this case tensors like \( |\hat{\xi}_n| \) are low-rank. One possible drawback is that spherical coordinates lead to more complicated quadrature formulas.
3. In this paper we consider the most straightforward approach for algorithm modification: all basic operations are replaced to tensorized analogues. The more elegant approach is to use cross-approximation techniques like [29]. Nevertheless, in our opinion, the straightforward approach is more robust and does not require deep understanding of underlying tensor algorithms.

4. In all tensor formats storage and operations cost are proportional to $n$ - length of original tensor in one direction. For large $n$ artificial increase of dimensionality or so-called quantized tensor formats [28] can be used in order to decrease memory consumption even further.

In future we plan to implement a parallel version of our solver using mpi4py package and space mesh decomposition. Besides, we plan to add model collision integrals for diatomic gas with internal degrees of freedom. The numerical method will be extended to higher orders, tetrahedral space meshes, and unsteady problems.

Acknowledgements

The authors would like to thank Sergey Dolgov, Maxim Rakhuba and Ivan Oseledets for their helpful recommendations regarding tensor formats.

A. Chikitkin is supported by President Grant MK-2855.2019.1.

References

[1] M. Petrov, A. Tambova, V. Titarev, S. Utyuzhnikov, A. Chikitkin, Flowmodellium software package for calculating high-speed flows of com-
[2] P. Betahatnagar, E. Gross, M. Krook, A model for collision processes in gases, Phys. Rev 94 (1954) 511–525 (1954).

[3] E. Shakhov, Generalization of the Krook kinetic relaxation equation, Fluid Dynamics 3 (5) (1968) 95–96 (1968).

[4] V. Rykov, A model kinetic equation for a gas with rotational degrees of freedom, Fluid Dynamics 10 (6) (1975) 959–966 (1975).

[5] F. Sharipov, V. Seleznev, Data on internal rarefied gas flows, J. Phys. Chem. Ref. Data 27 (3) (1998) 657–706 (1998).

[6] V. Titarev, E. Shakhov, Computational study of a rarefied gas flow through a long circular pipe into vacuum, Vacuum, Special Issue “Vacuum Gas Dynamics: Theory, experiments and practical applications” 86 (11) (2012) 1709–1716 (2012).

[7] A. Frolova, V. Titarev, Recent progress on supercomputer modelling of high-speed rarefied gas flows using kinetic equations, Supercomputing frontiers and innovations 5 (3) (2018) 117–121 (2018).

[8] V. Titarev, Application of model kinetic equations to hypersonic rarefied gas flows, Computers & Fluids 169 (2018) 62–70 (2018).

[9] V. Titarev, A. Frolova, V. Rykov, P. Vashchenkov, A. Shevyrin, Y. Bondar, Comparison of the shakhov kinetic equation and dsmc method as
applied to space vehicle aerothermodynamics, Journal of Computational and Applied Mathematics 364 (2020).

[10] R. Arslanbekov, V. Kolobov, A. Frolova, Kinetic solvers with adaptive mesh in phase space, Physical Review E 88 (2013) 063301 (2013).

[11] C. Baranger, J. Claudel, N. Hérouard, L. Mieussens, Locally refined discrete velocity grids for stationary rarefied flow simulations, J. Comput. Phys. 257 (PA) (2014) 572–593 (Jan. 2014).

[12] W. Guo, Y. Cheng, A sparse grid discontinuous galerkin method for high-dimensional transport equations and its application to kinetic simulations, SIAM Journal on Scientific Computing 38 (6) (2016) A3381–A3409 (2016).

[13] V. Titarev, S. Utyuzhnikov, A. Chikitkin, Openmp + mpi parallel implementation of a numerical method for solving a kinetic equation, Computational Mathematics and Mathematical Physics 56 (11) (2016) 1919–1928 (2016).

[14] E. Tyrtyshnikov, Kronecker-product approximations for some function-related matrices, Linear Algebra and Its Applications 379 (1-3 SPEC. ISS) (2004) 423–437 (2004).

[15] E. Tyrtyshnikov, Tensor approximations of matrices generated by asymptotically smooth functions, Sbornik Mathematics 194 (5-6) (2003) 941–954 (2003).
[16] B. Khoromskij, Structured data-sparse approximation to high order tensors arising from the deterministic boltzmann equation, Mathematics of Computation 76 (259) (2007) 1291–1315 (2007).

[17] S. Dolgov, A. Smirnov, E. Tyrtyshnikov, Low-rank approximation in the numerical modeling of the farley-buneman instability in ionospheric plasma, Journal of Computational Physics 263 (2014) 268–282 (2014).

[18] K. Kormann, A semi-lagrangian vlasov solver in tensor train format, SIAM Journal on Scientific Computing 37 (4) (2015) B613–B632 (2015).

[19] A. Boelens, D. Venturi, D. Tartakovsky, Parallel tensor methods for high-dimensional linear pdes, Journal of Computational Physics 375 (2018) 519–539 (2018).

[20] V. Titarev, Implicit numerical method for computing three-dimensional rarefied gas flows on unstructured meshes, Computational Mathematics and Mathematical Physics 50 (10) (2010) 1719–1733, cited By 23 (2010).

[21] V. Titarev, M. Dumbser, S. Utyuzhnikov, Construction and comparison of parallel implicit kinetic solvers in three spatial dimensions, Journal of Computational Physics 256 (2014) 17–33 (2014).

[22] V. Titarev, Efficient deterministic modelling of three-dimensional rarefied gas flows, Commun. Comput. Phys. 12 (1) (2012) 161–192 (2012).

[23] A. Chikitkin, M. Petrov, V. Titarev, S. Utyuzhnikov, Parallel versions of implicit lu-sgs method, Lobachevskii Journal of Mathematics 39 (4) (2018) 503–512 (2018).
[24] L. De Lathauwer, B. De Moor, J. Vandewalle, On the best rank-1 and rank-(r1, r2, . . . , rn) approximation of higher-order tensors, SIAM Journal on Matrix Analysis and Applications 21 (4) (2000) 1324–1342 (2000).

[25] L. R. Tucker, Implications of factor analysis of three-way matrices for measurement of change, Problems in measuring change 15 (1963) 122–137 (1963).

[26] L. Grasedyck, Hierarchical singular value decomposition of tensors, SIAM Journal on Matrix Analysis and Applications 31 (4) (2009) 2029–2054 (2009).

[27] I. Oseledets, Tensor-train decomposition, SIAM Journal on Scientific Computing 33 (5) (2011) 2295–2317 (2011).

[28] S. Dolgov, B. Khoromskij, Two-levelqtt-tucker format for optimized tensor calculus, SIAM Journal on Matrix Analysis and Applications 34 (2) (2013) 593–623 (2013).

[29] I. Oseledets, E. Tyrtyshnikov, Tt-cross approximation for multidimensional arrays, Linear Algebra and Its Applications 432 (1) (2010) 70–88 (2010).

[30] A. Lofthouse, L. Scalabrin, I. Boyd, Velocity slip and temperature jump in hypersonic aero thermodynamics, Journal of Thermophysics and Heat Transfer 22 (1) (2008) 38–49 (2008).
[31] V. Titarev, A. Frolova, Application of model kinetic equations to computing of super- and hypersonic flows of molecular gas, Fluid Dynamics 53 (4) (2018) 536–551 (2018). doi:10.1134/S0015462818040110