Development of Fast Deterministic Physically Accurate Solvers for Kinetic Collision Integral for Applications of Near Space Flight and Control Devices

PP-SAS-KY06-001-P3

Deliverable D5

Software Acceptance Report

Submitted by
Alexander Alekseenko
20 July 2015

for User Productivity Enhancement, Technology Transfer, and Training (PETTT) Program

High Performance Computing Modernization Program (HPCMP)

Contract No.: GS04T09DBC0017

Government Users/Sponsors/Partners: Dr. E. Josyula

Alexander Alekseenko

20 July, 2015

TABLE OF CONTENTS

B. APPENDIX B. DGVLIB USER MANUAL

B.1 Summary

The software library DGVlib encompasses a set of tools that can be used to solve kinetic equations of gas dynamics. The name of the library, DGVlib, stands for nodal-Discontinuous Galerkin discretizations in the Velocity variable (nDGV). The library includes subroutines to perform nDGV discretizations in the three dimensional velocity space including capabilities for non-uniform octree discretizations, subroutines for evaluating collision operators in gas-kinetic models, including the BGK, ES-BGK and the new velocity-dependent BGK model with velocity dependent collision frequency. The library also includes subroutines to evaluate the Boltzmann collision integral using Korobov quadratures. The library may be linked to zero, one and two dimensional solvers. In the case of multiple spatial dimensions, in order to use the library, the spatial cells/nodes have to be numbered using a single index.

B.1.1 Outline of the Document

This document is intended to explain the capabilities of DGVlib and to help the user to apply these capabilities to simulation of non-continuum gas flows using kinetic equations. The document provides descriptions of the following components of the library:

1. The collection of subroutines to build elements, nodes, and basis functions of nodal discontinuous Galerkin discretizations of functions in three dimensional velocity space and the affiliated data structures.
2. The subroutines for evaluation of the collision operators in gas kinetic models, including the BGK, the ES-BGK, and the new velocity-dependent BGK model with velocity dependent collision frequency.
3. The subroutines for evaluation of the Boltzmann collision operator using Korobov quadratures.
4. The interfaces between the library and the solvers in zero, one and two spatial dimensional.

B.2 DGVlib: Introduction

The software library DGVlib comprises tools for implementing nodal discontinuous Galerkin (nodal-DG) discretizations of kinetic equations in the velocity variable. Key capabilities of the library are the subroutines for the evaluation of the collision operator using a BGK-type model with velocity-dependent collision frequency (VD-BGK) and for the evaluation of the Boltzmann collision operator using Korobov quadrature nodes. In the following sections, an overview of DGVlib is given, its features are described and directions on how to use the software are given.

B.3 General Description

Subroutines of the DGVlib library can be divided into three groups based on their use and purpose. The first group consists of subroutines that implement nodal-discontinuous Galerkin (nDGV) discretizations in the velocity space. These subroutines are used to construct an nDGV discretization on uniform meshes and also on octrees. As a result, one use of the DGVlib library is to implement nDGV discretizations in solvers. The second group encompasses subroutines for evaluating the kinetic collision operator using various models. The implemented models include the Bhatnagar-Gross-Crook (BGK) model, the ellipsoidal-statistical BGK (ES-BGK) model and the BGK model with velocity dependent collision frequency and enforced relaxation of moments (VD-BGK). In addition, subroutines that implement evaluation of the Boltzmann collision operator using the approach of [3] and using Korobov quadratures are included in the library. The third group of subroutines consists of auxiliary subroutines that initialize and maintain the data structure and serve as building blocks to other subroutines. Although many of them have well-defined functions, most of the time, the user is not expected to work with these subroutines directly.

In Figure 1 the overall functionality of the library is described. The library subroutines are invoked in the driver. In particular, for most uses, the subroutine `InitDGV` needs to be called at the initialization time. This subroutine will set up the variables and data to be used for the evaluation of the collision integral. Parameters of the nDGV discretization and options for evaluating the collision integral are specified in the parameter file `DGVparameters.dat`. Once the necessary data structures have been established, the driver will use the data structures to develop the full, space and velocity, discretization of the solution and will proceed with time integration. During the time integration, library subroutines are called at each step to provide values of the kinetic collision operator. A conglomerate subroutine `UniversalCollisionOperatorDGV` provides a convenient way to evaluate the collision operator. This subroutine automatically chooses the appropriate model based on the l^2 norm of deviation of the solution from the local Maxwellian.

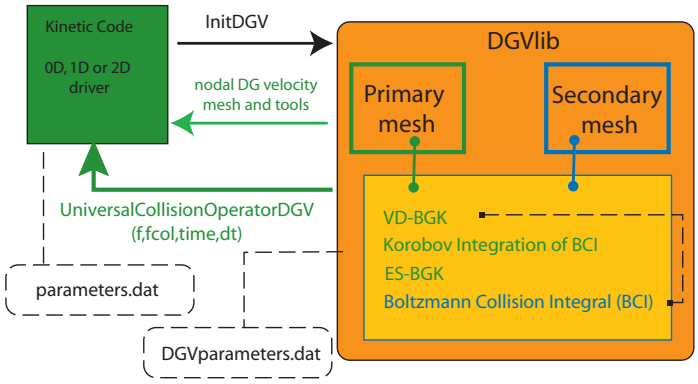

Figure 1. Functionality of DGVlib within a kinetic solver.

B.3.1 DGVlib Source Files

In Figure 2, a folder view of the DGVlib library is given in the case when the library is used with a zero dimensional (0D) driver. The DGVlib library consists of seven files: `DGV_commvar.f90`, `DGV_readwrite.f90`, `DGV_collision_mod.f90`, `DGV_miscset.f90`, `DGV_dgvtools_mod.f90`, `DGV_distributions_mod.f90`, and `DGV_sf02.f90`.

The library depends on BLAS, LAPACK, and a few additional open source modules that are available in NetLib.

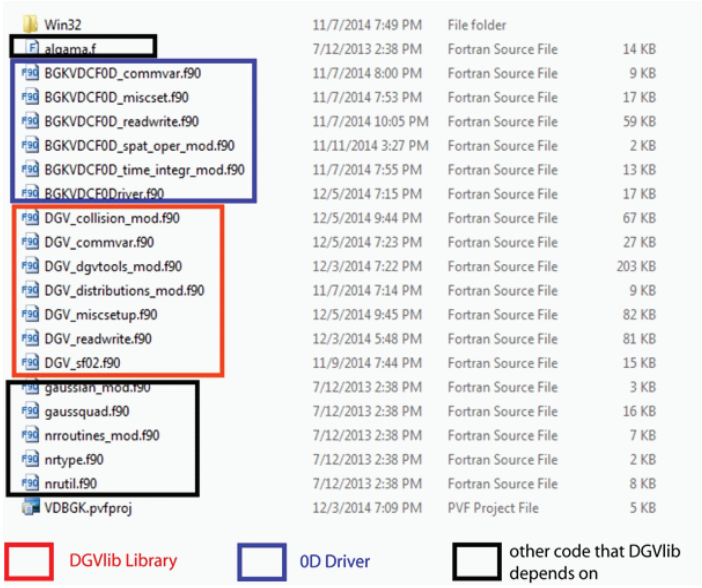

Figure 2. A folder view of DGVlib library merged with a zero dimensional driver.

B.3.2 Compiling and Running the Library

DGVlib is written in FORTRAN 90. It has been tested with PGI and Intel compilers with MPI and openMP.

The library depends on BLAS and LAPACK, therefore, proper links to BLAS and LAPACK or to MKL library should be provided for the complier. The Linux distributions of the library contain a sample Makefile, which should be modified for a particular computational environment.

We notice that adjusting the library to a desired scenario requires commenting or uncommenting lines in the code, replacing source files with copies containing the desired parameters, and adding desired features manually. As a result, it is expected that the user will have to re-compile the library frequently.

Once the library is complied and linked to the driver, it is time to try to run it. It is important to remember that the library uses its own parameter file, `DGVparameters.dat`. The library will expect that this file is present in the runtime folder. The content of the parameter file is explained in the following sections. Depending on the provided parameters, DGVlib may require a fairly developed structure of folders to be present in the run time directory. These folders are used to store copies of solutions and nDGV discretizations used to obtain these solutions. Also, the folders are used to store pre-computed collision kernels. Specifically, pre-computed values of operator $A(\vec{u}, \vec{v}, \varphi)$ (see [2]) at the nodes of the Gauss quadrature, referred in the code as `A` arrays, and at the nodal values of the Korobov quadratures, referred as `Akor` arrays need to be present in folders specified in the parameter file.

B.4 Implementation of the NDGV Discretization Using DGVlib

In this section, structure of the nDGV discretization is describes and the key subroutines implementing nDGV discretizations are identified. Also, entries of the parameter file that governing the nDGV discretization are describes.

B.4.1 Nodal-DG Discretizations

The description of the mathematical framework of the nodal-DG discretizations can be found in the Interim Technical Report PP-SAS-KY06-001-P3 01 for November 2014 [4] and also in references [2, 3]. In this section we will describe the implementation of this construction on a rectangular grid in the velocity domain.

We select a rectangular parallelepiped in the velocity space

$$K = [u_L, u_R] \times [v_L, v_R] \times [w_L, w_R]$$

where u_L, u_R, v_L, v_R, w_L, w_R are the left and right endpoints of the intervals in the u, v, and w directions,respectively. The variables determining the boundaries of the velocity domain are `u_L`, `u_R`, `v_L`, `v_R`, `w_L`, `w_R` which can be accessed from the `DGV_commvar` module (see `DGV_commvar.f90`).

A call of subroutines `SetDGVblzmmesh`, `Set3DCellsR_DGV` and `SetNodesDGV`, in this exact sequence, will establish a uniform mesh in velocity space and populate this mesh with Gauss nodal points. The parameters `Mu`, `Mv`, and `Mw` determine the number of cells in each velocity dimension. The parameters `su`, `sv` and `sw` determine the number of Gauss nodes in each velocity dimension. Note that `su` = `sv` = `sw` = 1 results in a piece-wise constant approximation with nodes located at the center of the cell, `su` = `sv` = `sw` = 2 gives tensor product of 1D linear approximation, `su` = `sv` = `sw` = 3 1D quadratic, and so on. Higher values are preferable for BGK and ES-BGK and the VD-BGK models since high order DG approximation produces more accurate integration in the velocity space. Using high values of `su`, `sv` and `sw` is difficult for the evaluation of the collision operator, both using Gauss quadratures and by using Korobov quadratures, because of the high memory requirements for pre-computed kernels. Nevertheless, quadratic approximation is still practical.

B.4.2 Primary and secondary grids

DGVlib maintains two completely independent discretizations of the velocity space, referred to as the primary and the secondary grids. The primary grid is used by all models and is expected to be the mesh on which solution is obtained. The secondary grid is reserved for handling solution temporary. Specifically, the secondary grid was introduced to evaluate full Boltzmann collision operator to estimate the relaxation rates in the VD-BGK model. For this purpose, the secondary grid is much coarser than the primary grid and the solution is temporarily projected on it only to estimate the relaxation rates. The data structure of the secondary mesh is identical to that of the primary mesh. The variables and subroutines related to the secondary mesh can be identifies by the suffix `II` in their names. For example, `MuII`, `MvII`, and `MwII` and `suII`, `svII` and `swII` determine the number of cells and the number of nodes in each cell, respectively.

B.4.2.1 Entries of the parameter file that determine the nDGV

The parameters of the nDGV discretization are specified in the parameter file. An example of parameter file requesting primary mesh to have $33 \times 33 \times 33$ cells with $2 \times 2 \times 2$ nodes in each cell and secondary mesh with $15 \times 15 \times 15$ cells with $1 \times 1 \times 1$ nodes in each cell in given in Figure 3.

Since this is the first time we discuss the parameter file, let us briefly describe its functionality. The parameter file consists of lines that either contain values of the constants or are commentaries. A commented line has the symbol "`!`" placed in the first position. All lines that start with an "`!`" are ignored by GDVlib. A line that contains a parameter value starts with a key phrase. For example "`number of cells in u`" is a key phrase that correspond to a list of values to be used as the number of cells on the 1D mesh in the direction u. Similarly the key phrases "`left endpoint in u`" and "`right endpoint in u`" designate values for parameters `u_L` and `u_L`, correspondingly. It is paramount that the key phrase is not altered. Otherwise the value of the parameter will not be read. When executing the program, the parameter reader subroutine `SetDGVParams` echoes each line that is reads successfully with the assigned value of the parameter. If it can not process the line, it sends an error message "`Can not process:`" followed by the line it can not process. Paying attention to this error message can help avoid running the code with zero or default parameters without realizing it.

```
!!!!!!!!!!!!!!!!!!!!!!!
left endpoint in u = -3.0         !! u_L
right endpoint in u = 3.0         !! u_R
left endpoint in v = -3.0         !! v_L
right endpoint in v = 3.0         !! v_R
left endpoint in w = -3.0         !! w_L
right endpoint in w = 3.0         !! w_R
uniform mesh in u = yes              !! mesh_u_uniform
uniform mesh in v = yes             !! mesh_v_uniform
uniform mesh in w = yes             !! mesh_w_uniform

use secondary velocity grid = yes  !! yes if secondary grid is in use

!set the parameters of the primary and secondary grid
number of cells in u = 33,15,4               !! Mu_list
degree of local Lagrange basis in u = 2,1,1  !! su_list
number of cells in v = 33,15,4               !! Mv_list
degree of local Lagrange basis in v = 2,1,1  !! sv_list
number of cells in w = 33,15,4               !! Mw_list
degree of local Lagrange basis in w = 2,1,1  !! sw_list
```

Figure 3. Entries of the parameter file that determine nDGV discretization

The key phrase is followed by an equal sign and the value of the parameter, which can be numerical or a sting constant, or a list of values. The numerical or a sting constants, or a list of values may be followed by an optional "`!`" which tells the subroutine to ignore the

rest of the line. The optional "!" can be used to include comments, see Figure 3

Note that parameters `u_L`, `u_R`, `v_L`, `v_R`, `w_L`, `w_R` are shared by both the primary and the secondary grids. The parameters `Mu`, `Mv`, `Mw` and `MuII`, `MvII`, `MwII` are determined from the lists titled "`number of cells in u/v/w`". The first entries in the three lists define `Mu`, `Mv`, `Mw`. The second entries in the list define `MuII`, `MvII`, `MwII`. Other entries are currently not used. The constants `su`, `sv`, `sw` and `suII`, `svII`, `swII` are defined in the same fashion.

Please note that only octree meshes are currently supported. Therefore, the values of the key phrases `uniform mesh in u/v/w` are mandatory set to "`yes`". The result of this is the uniform mesh that is at the root of octree. Later cells on this uniform mesh can be refined forming the branches and leaves of the octree mesh.

B.4.2.2 Arrays to store grids and cells in the velocity variable

Let us describe the arrays implemented in DGVlib that implement nDGV discretizations. Three types of arrays are used to describe the discretization. The first type is "grids". A grid is a collection of three 1D arrays that defines a rectangular grid with cells. Note that each refined cell forms a new grid. The second type of arrays is "cells". These arrays contain information about all cells and from all grids. While this information can be inferred from the grids arrays, it is convenient to maintain it for ease of programming. The third group of arrays is "nodes". These arrays contain information about the nodal points on all active (non-refined) cells on all grids. Again, while this information can be obtained from the cells arrays, it is extremely convenient to have the nodes readily available, albeit at the expense of using excess memory.

In DGVlib, all cells and nodes are numbered using a single index. The convention for numbering three dimensional collections of cells is that index changes most frequently in w direction, medium frequently in v direction and least frequently in x direction. Numbering of nodes in the came cell follows the same convention. For multiple cells, all nodes are enumerated in cell one, then in cell two and so on.

Below we list the array structures in more detail.

- `grids_u/_v/_w`: These arrays contain 1D meshes in variables u, v and w, whose tensor product makes up a uniform rectangular grid. In other words, `grids_u` contains values of u components of grid vertices, `grids_v` contains values of v components and `grids_w` contains values of w components of grid vertices.
- `grids_cap_u/_v/_w`: store the total number of points for each one-dimensional mesh points which includes both endpoints, i.e. $[u_L, u_R]$ (e.g., one cell has two points, two cells have three and so on). In particular, different 1D meshes can be stored in the array `grids_u` recorded one after another. The array `grids_cap_u` stores the total

number of points in these 1D meshes. Thus points of a 1D mesh can be recovered from `grids_u` by using the information contained in `grids_cap_u`.

- `cells_lu/_lv/_lw/_ru/_rv/_rw`: these arrays determine and store the six values u_L^j, u_R^j, v_L^j, v_R^j, w_L^j, w_R^j that define the cell $K_j = [u_L^j, u_R^j] \times [v_L^j, v_R^j] \times [w_L^j, w_R^j]$. The index j of the cell K_j is also the index in the arrays `cells_xx`. For example $u_L^j =$ `cells_lu(j)` and $u_R^j =$ `cells_ru`(j)

- `cells_pgrid`: this array contains the number of the parent grid to which this cell belongs. In particular, in the case of a single grid, `cells_pgrid` $= [1, 1, ..., 1]$ or `cells_pgrid(j)` $= 1$ for all j. Array `cells_cgrid` stores the information about the child grids for cells.

- `cells_ugi/_vgi/_wgi`: these arrays define the value of the index of the intervals $[u_L^j, u_R^j]$, $[v_L^j, v_R^j]$, $[w_L^j, w_R^j]$ on grids arrays `grids_u/_v/_w`. This value is also defined as the relative addresses of cells on the parent grid. The cells are numbered in the following order. If l, m, n are the related addresses on the 1D grids, then

$$j = (l-1)s_v s_w + (m-1)s_w + n$$

. We note that arrays `cells_ugi/_vgi/_wgi` are populated by calling the subroutine `SetCellsUGI_DGV`.

- `cells_gou/_gov/_gow`: these arrays carry the information about the number of Gauss nodes in cells for each velocity dimension, i.e., they describe the nodal-DG descretization on the cells. Specifically, `cells_gou(j)` gives the number of Gauss nodes in component u, on the cell K_j. Similarly, `cells_gov(j)` gives the number of Gauss nodes in component v, and `cells_gow(j)` gives the number of Gauss nodes in component w on the cell K_j. The 3D nodes are obtained by taking the tensor products of 1D Gauss nodes. If l, m, n are the number of the Gauss nodes on 1D intervals, then the corresponding number of the 3D Gauss node v_j^i on cell K_j is

$$i = (l-1)\texttt{cells\_gov}(j)\texttt{cells\_gow}(j) + (m-1)\texttt{cells\_gow}(j) + n$$

- `cells_refu/_refv/_refw`: these arrays describe the refinement process and the development of octree structure. In particular, when a cell is refined to produce sub cells, `cells_refu` tells in how many even pieces the cell is divided in the direction of the u component, `cells_refv` in v and `cells_refw` in w. The 1D grids that make this sub-division are assigned a number and added to the grids arrays, and the `grids_cap_u` will be given the total number of points on the 1D grids (including the endpoints).

  For example, suppose that the cell $K = [u_L, u_R] \times [v_L, v_R] \times [w_L w_R]$ is refined into 8 equal pieces, *i.e.* it is refined in 2 pieces in each velocity direction. In the process of

dividing the cells, the following 1D grids will be created:

$$\texttt{grids\_ofu} = \left[u_L, \frac{u_L + u_R}{2}, u_R\right]$$
$$\texttt{grids\_ofv} = \left[v_L, \frac{v_L + v_R}{2}, v_R\right]$$
$$\texttt{grids\_ofw} = \left[w_L, \frac{w_L + w_R}{2}, w_R\right] .$$

Hence each 1D grid will receive an addition and the `drids_cap_u/_v/_w` arrays will be appended an extra record: `grids_cap_u(i+1)` $= 3$, `grids_cap_v(i+1)=3` and `grids_cap_w(i+1)` $= 3$ respectively.

Now, suppose cell K_j is refined in 3 pieces in direction u, in 4 pieces in direction v and in 6 pieces in direction w. Also, suppose this is the very first refinement of the first grid. Then the subroutine will assign `cells_refu(j)` $= 3$, `cells_refv(j)` $= 4$, and `cells_refw(j)` $= 6$.

A 1D grid in the direction of u will be created

$$\texttt{grids\_ofu}^j = \left[u_L^j, u_L + \frac{\triangle u^j}{3}, u_L + \frac{2 \triangle u^j}{3}, u_R^j\right] .$$

The array `grids_u` will be extended by appending the new grid $grids\_ofu^j$ after the first grid

$$\begin{aligned} \textbf{grids\_u} &= [\texttt{grids\_u}, \texttt{grids\_ofu}] \\ &= \left[[u_L, \ldots, u_R][u\_L^j, \ldots, u\_R^j]\right] \end{aligned}$$

The array `grids_cap_u` $= [[3]]$ is updated to become `grids_cap_u` $= [[3][4]]$. Also, `cells_cgrid(j)` $= 2$.

One can refine any cell into subcells using the library subroutine `CellsRefineDGV(cellnum, refu,refv,refw)`, where `cellnum` is the number of the cell to be refined, and `refu`, `refv`, `refw` are the number of subdivisions for each velocity dimension (e.g., `refu` $= 2$, `refv` $= 2$, and `refw` $= 2$ in the case of the octree refinement). When a cell is refined, a new grid is added, where the boundaries of the new grid correspond to the boundaries of the original cell. This grid is identified as a child grid of the refined cell, and the 1D meshes of this grid will be appended to the arrays `grids_u/_v/_w`, and the arrays `grids_cap_u/_v/_w` will be updated. The arrays `cells_lu_lv/_lw/_ru/_rv/_rw` will be extended to include the information about the subcells of the refined cell and `nodes_u/_v/_w/_gwts` will be appended with the nodes that are created on the new cells.

B.4.2.3 Implementation of the Secondary Mesh

The DGVlib library supports two nodal-DG velocity discretizations, which are referred to as the primary and the secondary discretizations. The primary and the secondary discretizations occupy the same rectangular domain in the velocity space. The variables described in the previous section belong to the primary discretization. The variables of the secondary discretization are identical in their structure to the variables of the primary discretization. To distinguish the variables of the secondary velocity discretization, their names are given the suffix "`II`". For example, `nodes_uII/_vII/_wII` are the arrays that contain components of the velocity nodes of the secondary meshes. Similarly, names of subroutines working with the secondary mesh are given the suffix "`II`" as well, *e.g.*, `SetDGVblzmmeshII`, `Set3DCellsR_DGVII`, and `SetNodesDGVII`. On the other hand secondary grid will use the same boundaries as the primary grid, therefore `u_L`, `u_R`, `v_L`, `v_R`, `w_L`, `w_R` will stay unchanged. The subroutine `ProjectDGVtoSecMesh` is provided to project the discrete solution from the primary to the secondary grid. The primary and secondary meshes and their applications are indentified in Figure 3.

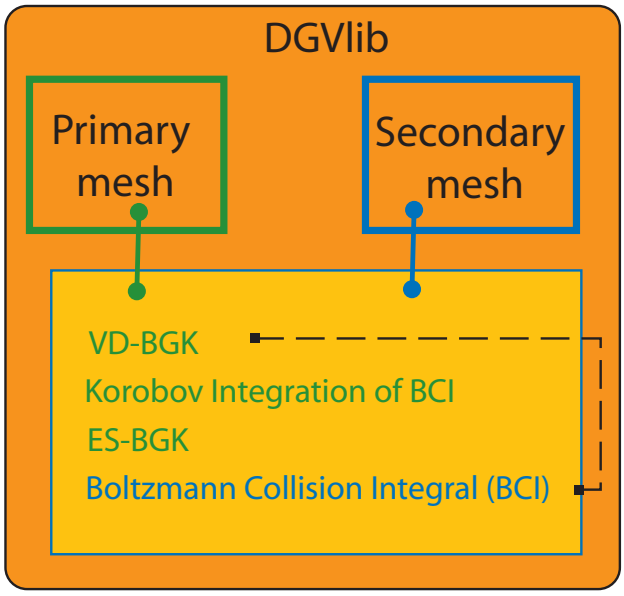

Figure 4. The structure and the functionality of the DGVlib library. The VD-BGK, ES-BGK and the Korobov quasi-statistic integration of the Boltzmann collision integral are using the primary mesh. The secondary mesh is used for evaluating the Boltzmann collision operator and for estimating the moments' relaxation rates (to be used by the VD-BGK model).

The development of a secondary velocity discretization was not planned originally. However, after a few attempts to use the BGK model with velocity-dependent collision frequency

(VD-BGK), it became evident that the evaluation of the Boltzmann collision integral represented a major challenge to the efficiency of the method. It was also observed that reasonable estimates on the relaxation rates of the low to moderate order moments can be obtained using low resolutions in the velocity space. As a result, low resolution secondary velocity discetization was introduced in the library to enable efficient estimation of moment relaxation rates, while using high resolution primary discretization with VD-BGK and ES-BGK models.

B.4.2.4 Arrays to Store Nodes in the Velocity Variable

The following is the characterization of the arrays that store the velocity nodes of the nodal-DG discretization.

- `nodes_u/_v/_w`: are the coordinates of the node on the cell. The nodes are necessarily listed for each cell in the order cells are listed. These arrays are most frequently used in the subroutines and store the u, v and w components of the velocity nodes. The velocity nodes stored in arrays `nodes_u/_v/_w` can be used as substitutes to discrete ordinates in a solver based on a discrete ordinate discretization. `nodes_ui/_vi/_wi` are the values of the local indices that determine addresses of nodes within the parent cell.

  The kinetic solution is represented by its values at the velocity nodes. Thus, the size of the unknown discrete solution at each spatial point is exactly the same as the number of elements in `nodes_u` array.

- Similar to `cells_pgrid`, which is the number of the parent grid, `nodes_pcell` contains the numbers of the parent cells in which nodes are contained.

- `nodes_gwts` contains the values of products of weights of Gauss quadrature formulas to be used in 3D integration. For example, integral

$$\int_{u_L}^{u_R}\int_{v_L}^{v_R}\int_{w_L}^{w_R} uf(u,v,w)\,du\,dv\,dw$$

  can be computed by the command

```
sum(f*nodes_u*nodes_gwts)
```

B.5 Subroutines for Evaluating the Collision Operator in Kinetic Equations

The following model collision operators are currently implemented in DGVlib:

- The evaluation of the full Boltzmann collision operator with hard spheres collision potential. The model is mplemented in subroutine `EvalCollisionPeriodicAPlus_DGV` and requires pre-computation of the integral kernel);
- The BGK model with velocity-dependent collision frequency. The model is implemented in the subroutines `EvalColVelES0D`, `EvalColVelES1Donecell`). The model requires periodic evaluation of full Boltzmann collision operator on secondary grid;
- The BGk and the ES-BGK models. The models are implemented in the subroutine `EvalColESBGK`;
- The evaluation of the Boltzmann collision integral using quasi-stochastic Korobov quadratures. The model is implemented in the subroutine `EvalCollisionPeriodicAKor_DGV` and requires pre-computation of the integral kernel.

All models are supported for the primary velocity discretization. Currently, all models only support uniform velocity discretization.

In addition to subroutines implementing individual models, subroutines `UniversalCollisionOperator0DDGV` and `UniversalCollisionOperator1DonecellDGV` are provided to enable an automatic selection of the desired kinetic model based on the degree of deviation of the solution from continuum.

B.5.1 Entries in the parameter file that determine the use of particular models

Entries that determine which model is used in `UniversalCollisionOperator0DDGV` and `UniversalCollisionOperator1DonecellDGV` are shown in Figure 5. The library distinguishes five modes, 0,. . . ,4, which are determined bases on the values of the deviation of the solution at thegiven point from the local Maxwellian. Specifically, the following quantity is being computed on every time step

$$\epsilon(t,\vec{x}) = \frac{\int_{R^3} |f(t,\vec{x},\vec{v}) - f_M(t,\vec{x},\vec{v})|\, dv}{\int_{R^3} f(t,\vec{x},\vec{v})\, dv},$$

where $f_M(t,\vec{x},\vec{v})$ is the Maxwellian distribution function that has the same density, bulk velocity and temperature as $f(t,\vec{x},\vec{v})$. The computed value is compared to a number of threshold constants, `decom_lev`, `linear_lev`, `vel_lev`, and `ES_lev` which are given in the order of decreasing magnitudes. The mode corresponding to a particular threshold is selected if the value of $\epsilon(t,\vec{x})$ is below that threshold, but above the next threshold.

The value of the current mode is stored in the variables `run_mode` in the case of 0D and `run_mode_1D` in the case of 1D, 2D and 3D.

The entries in the parameter file that determine the values of `decom_lev`, `linear_lev`, `vel_lev`, and `ES_lev` are shown in Figure 5. The values of the constants should be deter-

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Model threshholds: The order in which the models are evoked are
! 1) the Boltzmann equation, non-split formulation
! 2) the Boltzmann equation, split f=f_M + df, formulation
! 3) the linearized Boltzmann equation
! 4) the BGK-type Model with velocity dependent collision frequency
! 5) the ES-BGK or BGK model
!
! the parameters below define the thresholds for switching between
! models. They represent the maximum allowed values of the L1-norm
! of the deviation of the solution from the local Maxwellian. When
! the L1-norm of the deviation drops below the threshold, the
! appropriate model is invoked.
! NOTICE: keep linear_lev an order of magnitude less than decomp_lev
!
! VERY IMPORTANT: the thresholds for the models above should be given
! in decreasing order. Otherwise the algorithms will make a mistake.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

error maxwell ESBGK = 5.0E-3              ! ES_lev
error maxwell Vel ESBGK = 5.1E-3          ! vel_lev - VDBGK regime
error maxwell linearization = 0.01        ! linear_lev,
error maxwell decomposition = 0.05        ! decomp_lev
```

Figure 5. Entries of the parameter file that determine use of different models for evaluating the collision integral

mined experimentally and adjusted as needed to accomplish the desired transition between different models.

B.5.2 Merging DGVlib with 0D, 1D and 2D (and higher) Dimensional Kinetic Solvers

The DGVlib does not contain subroutines for time discretization and space discretization of kinetic equations. Except for the cases of 0D and 1D, such subroutines are cumbersome and are difficult to implement. Since, there already exist a considerable number of kinetic solvers that use discrete ordinate approximations with which DGVlib can be merged. To facilitate this, interfaces were designed to link DGVlib to other solvers. Two types of interfaces have been implemented: 0D corresponding to no spatial dimension and 1D corresponding to one/two/three spatial dimensions. In the 1D interface, it is assumed that points in the physical space where the kinetic solution is being evaluated are numbered by a single index. This way one, two and three dimensional solvers can communicate with the library using

the same interfaces.

The general scheme for merging DGVlib library with a kinetic code or driver is given in Figure 1. To construct the velocity discretization, the driver calls the initialization subroutine; in the case of zero spatial dimensions, the initialization subroutine is `InitDGV0D`. For one, two or three spatial dimensions, the initialization subroutine is `InitDGV1D`. The initialization subroutine reads parameters from the parameter file (`DGVparameters.dat`). The parameter file sets the number of velocity cells and the order of DG polynomial approximation, and also sets the parameters related to the gas and the description of the dimensionless formulation. Lastly, the parameter file also contains directives on where the pre-computed kernel is located. The initialization subroutine calls a number of library subroutines that develop the primary and the secondary velocity discretizations and sets up additional variables and structures that will be used during the solver's execution. The final stages of the velocity discretization are performed inside the driver. The driver accesses the key variables of the velocity discretization, e.g., `nodes_u/_v/_w/_gwts`, the key constants and, as necessary, tools for performing operations with nodal-DG discretizations. To initialize the solution, it is generally sufficient to evaluate the initial data at the velocity nodes. The (explicit) time iteration then proceeds according to the driver's construction. However, the evaluation of the collision operator at each time step is replaced with a call of the universal collision operator, `UniversalCollisionOperator0DDGV` for zero spatial dimensions and `UniversalCollisionOperator1DonecellDGV` for one, two and three spatial dimensions.

B.6 The Subroutines Implementing the VD-BGK Model.

The VD-BGK model is a distinguishing feature of the implemented DGVlib library. The algorithm was developed to make the VD-BGK model efficient and convenient to use. In this section, the architecture of the software is discussed. For most uses, VD-BGK model is invoked automatically inside the subroutines `UniversalCollisionOperator0DDGV` and `UniversalCollisionOperator1DonecellDGV`. However, the user may also access the model directly through subroutines `EvalColVelES` for 0D problems and `EvalColVelES1Donecell` for 1D, 2D and 3D problems.

The scheme of `EvalColVelES1Donecell` is given in Figure 6. (The structure of `EvalColVelES` is similar.) The discrete velocity solution at a single spatial point needs to be passed to the subroutine. The subroutine returns the value of the dimensionless collision operator. The value of the current solution time and the number of the collocation points are provided to the subroutine for use in calculating the relaxation rates for moments. It is assumed that all collocation points in 1D, 2D or 3D applications are numbered using a single index.

The first step is to determine the relaxation rates for the specified moments. This is accom-

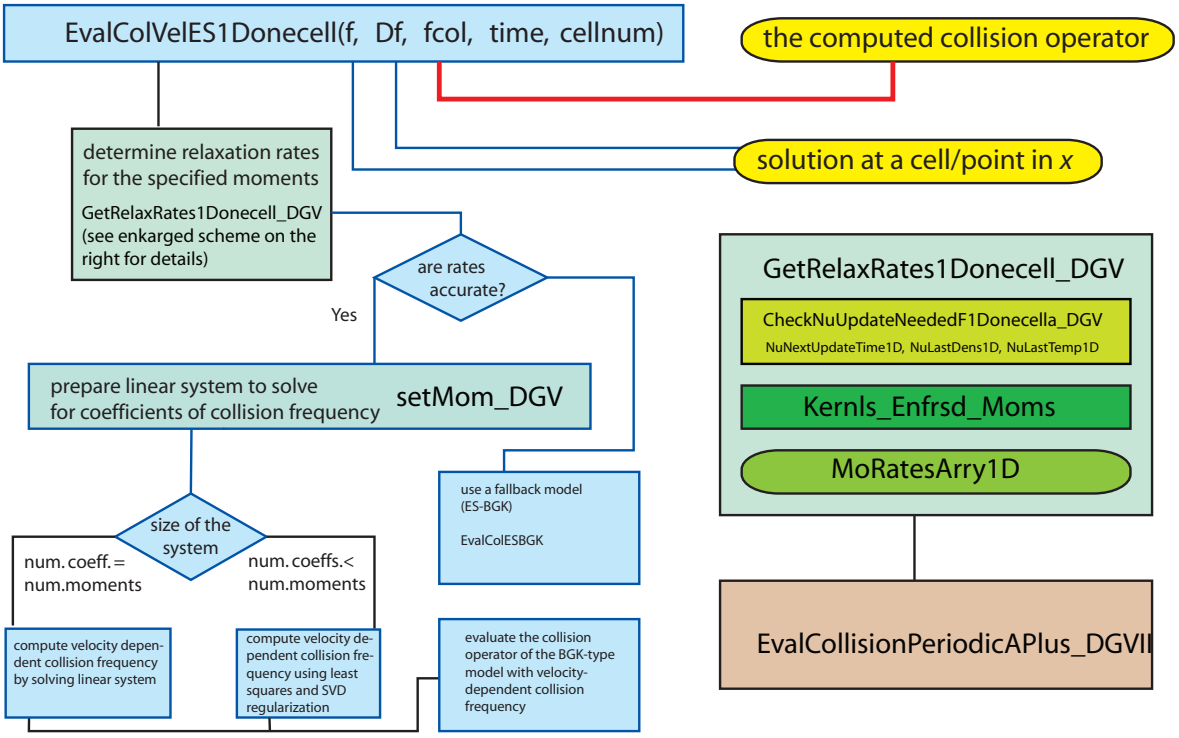

Figure 6. Structure and function of the subroutine implementing VD-BGK model.

plished by executing `GetRelaxRates1Donecell_DGV`, which, in turn, evaluates the Boltzmann collision operator on the secondary mesh. The design of `GetRelaxRates1Donecell_DGV` is schematically depicted in Figure 6. First, the subroutine `CheckNuUpdateNeededF1Donecell,a_DGV` is called to determine whether the rates need to be updated. Since evaluation of the full Boltzmann collision operator is expensive, rates are evaluated intermittently based on the time of the last update and the change in the macroparameters of density and temperature. Arrays `NuNextUpdateTime1D`, `NuLastDens1D`, `NuLastTemp1D` store the values of the solution time for the next update, the values of the density at the last update and the values of the temperature at the last update for each cell/point in x. The parameter `mft_coeff` determines the time period between the updates in multiples of the local mean free time. The parameters `dens_trshld` and `tempr_trshld` determine the threshold values of the changes in local density and temperature required to initiate the update. The update of the relaxation rate can be triggered by tests using any of time, density or temperature. If the update was not triggered, rates stored in the array `MoRatesArray1D` are used. If the update is triggered, rates are computed by evaluating the full Boltzmann collision operator on the secondary velocity grid by calling `EvalCollisionPeriodicAPlus_DGVII`, and then by computing the moments of the solution and of the collision operator. The computed

rates are then stored in the `MoRatesArray1D` array.

Relaxation rates are computed by evaluating the, moments of the Boltzmann collision integral. The kernels of enforced moments are encoded in subroutine `Kernls_Enfrsd_Moms` (see Figure 7). This subroutine determines the a set of moments to be enforced in simulations. The parameter `Order` is used to determine how many kernels are used in a particular simulation. In other words, while many kernels may be encoded in the subroutine `Kernls_Enfrsd_Moms`, only $1, \ldots,$ `Order` will be used in simulations.

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! kernls_enfrsd_moms(n,nodes_u,nodes_v,nodes_w,ubar,vbar,wbar) result (y)
!
! This function is used to evaluate kernels of the enforced moments.
!
! The subroutine is desiged to provide a flexibility in which moments to enforce.
! Change the kernel functions below to determined the enformced moment.
!
! Moments are enforced beginning from 1 and finishing with some K < MaxNumEnforcedMoments
! with relaxation of all moments between
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
function kernls_enfrsd_moms(n,nodes_u,nodes_v,nodes_w,ubar,vbar,wbar) result (y)

integer (I4B), intent (in) :: n ! the number of the basis function.
real (DP), dimension(:) :: nodes_u,nodes_v,nodes_w ! the nodes at which the basis function is evaluated.
real (DP), intent (in) :: ubar,vbar,wbar ! values of the bulc velocity to be used in the evaluation of the moment kernels
real (DP), dimension(1:size(nodes_u,1)) :: y ! the value of the function
!
!integer (I4B) :: j ! some local counter

!!
select case (n)
 case(1)
  y = 1 + nodes_u*0.0d0 ! kernel density
 case(2)
  y = nodes_u           ! u component of the kernel of momentum
 case(3)
  y = nodes_v           ! v component of the kernel of momentum
 case(4)
  y = nodes_w           ! w component of the kernel of momentum
 case(5)
  y = ((nodes_u-ubar)**2+(nodes_v-vbar)**2+(nodes_w-wbar)**2)/3.0_DP*2.0_DP     ! kernel of tempearture, please notice the dimensi
 case(6)
  y = (nodes_u-ubar)**2/3.0_DP*2.0_DP                                          ! directional temperature in direction u
 case(7)
  y = (nodes_u-ubar)**3
 case(8)
  y = (nodes_u-ubar)**4
 case(9)
  y = (nodes_u-ubar)**5
 case(10)
  y = (nodes_u-ubar)**6
 case(11)
  y = (nodes_u-ubar)**7
 case(12)
  y = (nodes_u-ubar)**8
 case(13)
  y = (nodes_u-ubar)**9
 case(14)
  y = (nodes_u-ubar)**10
 case(15)
  y = (nodes_u-ubar)**11
 case default
  print *, "kernls_enfrsd_moms: Can not process: n=", n
  stop
end select
end function kernls_enfrsd_moms
```

Figure 7. Subroutine `Kernels_Enfrsd_Moms` that defines rates for which moments are enforced.

Because the Boltzmann collision integral is evaluated on the secondary (usually coarser) velocity grid, it may be that computed relaxation rates are heavily biased by the truncation rates. To avoid the use of incorrect rates, error indicators based on the violation of the

energy conservation law are used to predict when the rates are not reliable. If all of the computed rates are unreliable, a fallback model is invoked, in which the collision integral is computed, not by the VD-BGK model, but by a simpler and more reliable model. Currently, the fallback model is ES-BGK. If, however, at least one computed relaxation rate is reliable, the velocity-dependent collision frequency is evaluated.

The number of enforced moments must be greater than or equal to the number of unknown coefficients in the representation of the collision frequency. When the number of the enforced moments is less equal to the number of coefficients, the coefficients of the velocity dependent collision frequency are determined by solving a linear system. When the number of enforced moments is larger than the number of unknown coefficients, the coefficients are determined by solving an l^2 minimization problem using singular value decomposition (SVD) and SVD regularization. Once the coefficients are determined, the VD-BGK collision operator is computed and returned to the calling subroutine.

B.6.0.1 Calling Interfaces of `EvalColVelES`

The evaluation of the collision integral using VD-BGK model can be performed by calling the subroutines

```
EvalColVelES(f,Df,fcol,time)
```

in the case of 0D problems and

```
EvalColVelES1Donecell(f,Df,fcol,time,cellnum)
```

in the cases of 1D, 2D, and 3D problems. The following is the description of the variables that are used in these subroutines.

- `f` is an array of double real numbers that contains the discrete velocity solution at a single spatial point.
- `Df` is an array of the same size and type as `f` that contains deviation of the solution from the local Maxwellian. `Df` is usually computed by the subroutines `CheckSolutionModeSimple1Donecell_DGV` or `CheckSolutionMode_DGV`.
- `fcol` is an array of the same size and type as `f` in which the computed values of the collision operator are stored.
- `time` is a double real value of the dimensionless time at which the evaluation is called.
- `cellnum` is an long integer (1D,2D and 3D applications only) that gives the number of the spatial point on the spatial mesh where the collision integral is evaluated.

B.6.0.2 Entries in the Parameter File that Control VD-BGK Model

Entries in the parameters file that determine constants used by `EvalColVelES` are shown in Figure 8

```
! directory and the base name for the solution
current solution save directory = sol080909/   !!
current solution base name = M155_testKor_     !! used to create A/Akor
current A operator base name = OpA/1xN/exp0912_    !! A on primary mesh
secondary operator A base name = OpA/1xN/exp0912_  !! A on secondary mesh
current AKor operator base name = AkorE21_         !! for reading Akor
!
num of chunks for Aarray = 1     !! numchnks  #files Aarray on pri. mesh
num of chunks for sec Aarray = 1 !! numchnksII #files Aarray on sec. mesh
number of nets chunks = 1,5      !! Akor  numbers of chunks in each net
!
! Parameters of the gas and ES-BGK
the ordinary gas constant = 208.1322 !! gasR
gas alpha = .5                 !! gas_alpha exp. in power viscosity law
gas viscosity = 2.4459e-005    !! reference gas_viscosity
Temperature reference = 420    !! gas_T_reference for viscosity power law
ESBGK modifier = -0.5          !! alpha = in the ES-BGK model.
use VSS diam in relax rates = no  !! flag_use_VSS_diam !
!
!! Parameters VD-BGK (BGK model with vel. dependent collision frequency
N of enforced moments = 6       !! Order -- # of moments to enforce
N of coefficients in VDCF = 6   !! Order_nu -- N of coeff in vel.-dep.
!                               !! collision frequency
mean free time rates update = 1.0  !! mft_coeff -- update period in mft
```

Figure 8. Entries of the parameter file that determine use of VD-BGK model

The number of the enforced moments is stored in the variable `Order` and it is identified by the key phrase "`N of enforced moments`". The kernels of the enforced moments are encoded in the subroutine `Kernls_Enfrsd_Moms`, see Figure 7. Moments corresponding to kernels with numbers $1, \ldots,$ `Order` are enforced.

The key phrase "`N of coefficients in VDCF`" is designated for the value of constant `Order_nu`, which determines how many basis functions is used in representation of the velocity-dependent collision frequency. The basis functions are encoded in the function `psi_basis_funcs`. Basis function with numbers $1, ldots,$ `Order_nu` will be used in the ex-

pression for the velocity-dependent collision frequency. The larger is the number of bass-functions that are used, the more freedom the library has to enforce relaxation rates for the selected moments. However, a larger number of basis function usually means that high order polynomials have to be used. This may have undesirable results to the simulations, for example, but loosing stability and restricting the time step.

Note that at all times, `Order_nu` has to be less than or equal to `Order`.

The VD-BGK model relies on the evaluation of Boltzmann collision operator invoked by the subroutines `EvalCollisionPeriodicAPlus_DGVII` and `EvalCollisionPeriodicMixedTermsA_DGVII` inside the subroutine `GetRelaxRates1Donecell_DGV` that determines the relaxation rates for the enforced moments. To ensure correct work of the subroutines `EvalCollisionPeriodicAPlus_DGVII` and `EvalCollisionPeriodicMixedTermsA_DGVII`, the pre-computed kernel of the collision operator should be provided to the library. The key phrase that identifies the header of the file name where the kernel is stored is textttsecondary operator A base name. As is seen in Figure 8, this header can contain directories followed by the first portion of the file name. The remainder of the file name is created automatically by subroutine `MakeBaseNameAoperDGVII` based on the parameters of the secondary velocity discretization. One should be mindful that each pre-computed kernel corresponds to a specific DG velocity discretization. The velocity discretization that was used to create a pre-computed kernel is encoded in the name of the file that is used to store the components of pre-computed kernel. As an implicit consistency check, subroutine `MakeBaseNameAoperDGVII` will create the file name automatically based on the parameters of the secondary mesh. This will make sure that a proper file will be read from the identified folder. Please, note, however, that not all DG discretization parameters are encoded in the name. In particular, the boundaries of the velocity domain are not encoded in the name. Therefore, care must be taken to use the appropriate pre-computed kernel. A successful strategy would be to keep the parameter file used to compute the kernel and the output file produced while computing the kernel in the same folder as the file storing the kernel. This way, the consistency can be verified by reviewing those files.

The default location of the kernel file is the solution folder identified by the key phrase `current solution save directory`.

It is possible that the relaxation rates for the moments can not be computed reliably on the selected secondary grid, when, for example, the grid is not resolving the solution. The subroutines `GetRelaxRates1Donecell_DGV` and `GetRelaxRates0D_DGV` have built-in indicators that help determin when the rates are not reliable. If not a single computed rate is reliable, instead of using the VD-BGK model, the subroutine `EvalColVelES` will invoke the *fall back model*, which is currently the ES-BGK model. Entries in the parameter file that correspond to the ES-BGK model can be seen in Figure 8. Here "`gas alpha`" indicates the value of variable `gas_alpha` that stands for the exponent in the viscosity power

law; "`gas viscosity`" indicates the value of the variable `gas_viscosity` that corresponds to the reference viscosity of the gas; "`Temperature reference`" indicates the value of `gat_T_reference` that gives the value of the reference temperature for which the reference viscosity was measured. The key phrase "`ESBGK modifier`" stands for the parameters α in the ES-BGK model that is used to adjust the model Prandtl number.

The key phrase "`use VSS diam in relax rates`" expects "`yes`" or "`no`" entry and indicates whether the diameter of the hard sphere model should be calculated from the viscosity of the gas. If this parameter is set to "`yes`", then VSS model will be used to calculate the diameter of molecules based on the viscosity of the gas. This values will override the diameter of the molecules in the hard spheres model in variable `mol_diam`. The value of `mol_diam` is indicated by the key phrase "`ref molecular diameter`" given among the parameters defining the dimensionless reduction in Figure 9

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Parameters for dimensionless reduction
!
! C_inf = thermal velocity in m/s
! L_inf = characteristic length in m
! N_inf = the total number of molecules in the volume
! T_inf =  the normalization for time is selected
!          from the condition T_inf*C_inf = L_inf
!          calculated automatically
! Temp_inf = the characteristic temperature
! mol_diam = molecular diameter in hard shperes model
!!!!!!!!!!!!!!!
ref termal velocity = 645.1854419808672 ! C_inf m/s
ref characteristic length = 1.0 ! L_inf m
ref number of molecules = 1.0E+21 ! N_inf
ref molecular diameter = 3.76E-10  ! mod_diam m
ref molecular mass = 6.633520884527004d-26 ! mol_mass molecular mass in kg
!!!!!!!!!!!!!!!
```

Figure 9. Entries of the parameter file that determine the parameters of dimensionless reduction and also the hard spheres model

Note, however that if the hard spheres model is used for the ES-BGK calculation, the viscosity will not be calculated automatically. Thus correct values of the viscosity of hard spheres gas should be provided in the parameter file along with the correct values of the exponent `gas_alpha`=0.5. This way of handling the parameters is historical and will be optimised in the future versions of the library.

B.6.0.3 Selecting the Moments to Enforce

Function `Kernls_Enfrsd_Moms` defines what moments can be enforced. The function has the following interface.

```
function kernls_enfrsd_moms(n,nodes_u,nodes_v,nodes_w,ubar,vbar,wbar)
                                        result (y)
```

Here is a brief description of the variables.

- `n` is a number of type `INTEGER` that gives the number of the moment kernel.
- `nodes_u/_v/_w` are one index arrays of type `DP` (double precision real numbers) that contain the values of the u, v and w components of velocity points at which the kernels need to be evaluated.
- `ubar, vbar, wbar` are numbers of type `DP` that contain the values of the bulk velocity of the solution. The bulk velocity is provided to the kernels, rather then being directly computed from the solution.
- `y` is an array of type `DP` and having the same size as `nodes_u` that contains the values of the moment kernel evaluated at the provided velocity nodes.

It is expected that the user will modify subroutine `Kernls_Enfrsd_Moms` to enforce the moments that are desired. In Figure 7 examples of kernels are provided. Other kernels can be programmed in a very straightforward fashion. One just have to remember that arrays `nodes_u/_v/_w` contains u, v and w components of all points where the kernels need to be evaluated. The FORTRAN convention for handling arithmetic operations on arrays element by element can be used to code any desired kernel.

B.6.0.4 Selecting the Basis Functions to Express Velocity-Dependent Collision Frequency

Function `psi_basis_funcs` defines the basis functions used in the representation of the velocity dependent collision frequency. The function has the following interface.

```
function psi_basis_funcs(n,nodes_u,nodes_v,nodes_w) result (y)
```

Here is a brief description of the variables.

- `n` is a value of type `INTEGER` that gives the number of the basis function to be evaluated.
- `nodes_u/_v/_w` are one index arrays of type `DP` (double precision real numbers) that contain the values of the u, v and w components of velocity points at which the basis functions needs to be evaluated.

- `y` is an array of type `DP` and having the same size as `nodes_u` that contains the values of the basis function evaluated at the provided velocity nodes.

It is expected that the user will modify subroutine `psi_basis_funcs` to use basis functions that better suite the purpose for a particular problem. The process of modifying basis function is identical to the process of changing the moment kernels in subroutine `Kernls_Enfrsd_Moms`. Again, one have to keep in mind that arrays `nodes_u/_v/_w` contains u, v and w components of all points where the kernels need to be evaluated. The FORTRAN convention for handling arithmetic operations on arrays element by element can be used to code a desired basis function.

B.6.1 Evaluation of the Boltzmann Collision Integral Using Korobov Quadratures

Subroutines `EvalCollisionPeriodicAKor_DGV` and `EvalCollisionPeriodicAKorMixed-Terms_DGV` implement evaluation of the Galerkin projection of the Boltzmann collision integral using Korobov quadratures. Subroutine `EvalCollisionPeriodicAKor_DGV` evaluates the integral

$$Q_{\phi_p^j}(f,f) = \frac{8}{\Delta v^j \omega_p} \int_{R^3} \int_{R^3} f(t,x,v) f(t,x,v_1) A(v,v_1;\phi_p^j) dv_1\, dv.$$

and subroutine `EvalCollisionPeriodicAKorMixedTerms_DGV` evaluates the integral

$$Q_{\phi_p^j}(f,g) = \frac{8}{\Delta v^j \omega_p} \int_{R^3} \int_{R^3} f(t,x,v) g(t,x,v_1) A(v,v_1;\phi_p^j) dv_1\, dv. \tag{1}$$

The latter integral is extremely useful for evaluation of the collision integral when the solution $f(t,x,v)$ is close to Maxwellian. The implemented approach was proposed in [2] and consists of introducing a decomposition $f(t,x,v) = f_M(t,x,v) + \Delta f(t,x,v)$, where $f_M(t,x,v)$ is a Maxwellian distribution with the same density, bulk velocity and temperature as $f(t,x,v)$. If $\Delta f(t,x,v)$ is small (but not negligible), the evaluation of $Q_{\phi_p^j}(f,f)$ is replaces with

$$Q_{\phi_p^j}(f,f) = 2Q_{\phi_p^j}(f_M,\Delta f) + Q_{\phi_p^j}(\Delta f,\Delta f).$$

In this case, `EvalCollisionPeriodicAKorMixedTerms_DGV` is used to evaluate the first term and `EvalCollisionPeriodicAKor_DGV` is used to evaluate the second.

B.6.1.1 Interfaces of the Subroutines `EvalCollisionPeriodicAKorMixedTerms_DGV` and `EvalCollisionPeriodicAKor_DGV`

A brief description of the interfaces of subroutine `EvalCollisionPeriodicAKor_DGV` is given next.

```
subroutine EvalCollisionPeriodicAKor_DGV(f,fc)
```

Here

- `f` is a one index array of `DP` (double precision real) numbers that contain the solution.
- `fc` is the one index array of the same length as `f` that contains the values of the collision operator.

The interface of `EvalCollisionPeriodicAKorMixedTerms_DGV` is very similar.

```
subroutine EvalCollisionPeriodicAKorMixedTerms_DGV(f,fm,fc)
```

Here

- `f` is a one index array of `DP` (double precision real) numbers that contain the perturbation of the solution from the local Maxwellian;
- `fm` is a one index array of the same length as `f` that contains the values of the local Maxwellian;
- `fc` is the one index array of the same length as `f` that contains the values of the collision operator.

B.6.1.2 Entries of the Parameter File that Determine Evaluation of Collision Integral Using Korobov Quadratures

Entries that control the evaluation of the collision integral using subroutines `EvalCollisionPeriodicAKorMixedTerms_DGV` and `EvalCollisionPeriodicAKor_DGV` can be found in Figures 8, 3 and 5. These entries are briefly explained next.

- `current AKor operator base name`. This key phrase identifies the values of the variable `current_AKoroperator_base_name` that keeps the header of the names of the files in which the pre-computed values of the kernel of the collision integral at the nodes of Korobov quadrature are stored. The values of the pre-computed kernel can be stored in multiple files, therefore the library will modify the header by adding `net00_` and `ch000_` to the name with zeros replaced with a relevant number to produce a name of the file that contains a portion of the pre-computed kernel that will be used in integration.

  The convention accepted in DGVlib, is that a full collection of Korobov nodes for the evaluation of six dimensional integrals (1) constitute one Korobov net. In the following we will define the nodes and give a table of parameters that produce Korobov nets that are covering the six dimensional domain somewhat uniformly. In DGVlib a collection of nets may be used in succession to evaluate the collision integral. This way, one can hope to avoid the growth of spurious modes that are generated by deficiencies in a single Korobov net.

Values of the kernel of the collision integral computed at all nodes of the Korobov quadrature constitute values of the kernel computed on one Korobov net. If the number of nodes is large, the pre-computed values can be divided in two or more files. These files should be named according to the following rules:

  - File names have the same header. For example the header of the file name could be `myHSkernel_`. Note that underscore symbols are not mandatory in the file name header.
  - File name header is follows by the net number addition, `netXX_`, which has to be the same for all files sharing the values of the kernel for a particular Korobov net. For example, `myHSkernel_net02_`. We note that the underscore symbol is **mandatory** in `netXX_`. Nets are enumerated starting from `00`.
  - File name header and the net number are followed by the chunk number addition `chXXX_`. For example the values of the kernel can be divided between files whose names begin with `myHSkernel_net02_ch000_`, `myHSkernel_net02_ch001_`, and so on. The underscore symbol in `chXXX_` is mandatory. Also, the numbering of chunks start from `000`

    The default location of the kernel files is in the solution directory identified by the key phrase `current solution save directory`.
  - The remaining parts of the file names are identical and encode the information about the velocity mesh for which these values were computed.

- `number of nets chunks`. This key phrase identifies values of the one index array `KorNetsChunks`. `KorNetsChunks` is an array of integers that have the following interpretation. The total number of provided entries indicates the number of nets that will be used by DGVlib. Then each number indicates the number of chunks that store the pre-computed values of the kernels for a particular net. For example, an entry

  `number of nets chunks = 2,3,4,1`

  indicates that these will be four Korobov nets in use and that the pre-computed kernel is divided between two files for the net 0, three files for net 1, four files for net 2 and one file for net 3. Using the header form the above example, it is expected that the files with the following names are located in the solution directory: `myHSkernel_net00_ch000_...`, `myHSkernel_net00_ch001_...`, `myHSkernel_net01_ch000_...`, `myHSkernel_net01_ch001_...`, `myHSkernel_net01_ch002_...`, `myHSkernel_net02_ch000_...`, `myHSkernel_net02_ch001_...`, `myHSkernel_net02_ch002_...`, `myHSkernel_net02_ch003_...`, `myHSkernel_net03_ch000_...`.

- `left endpoint in u/v/w` and `right endpoint in u/v/w`. These key phrases indicate values of the variables `u_L/v_L/w_L` and `u_R/v_R/w_R`, respectively. These vari-

ables determine the boundary of the velocity domain and determine the velocity discretization. Care must be taken to ensure that the pre-computed values of the collision kernel correspond to the same velocity domain.

- `number of cells in u/v/w` and `degree of local Lagrange basis in u/v/w`. The evaluation of the collision integral using Korobov quadratures is using the primary velocity discretization. Also, values of the kernels are pre-computed for a specific nDGV discretization. The parameters affecting the construction of the primary mesh also will be used to create a name for the file that stores the pre-computed values. This way only the pre-computed values that are consistent with the current nDGV discretization will be used by the library.
- `error maxwell decomposition` and `error maxwell linearization`. The evaluation of the collision integral using Korobov quadratures will be performed in run modes 0, 1, and 2. Appropriate values need to be provided for variables `decomp_lev` and `linear_lev` indicated by the key phrases `error maxwell decomposition` and `error maxwell linearization` in order to achieve desired evaluation of the collision operator.

B.6.1.3 Computing and Storing Values of Kernels on Korobov Nodes

DGVlib uses pre-computed values of the kernel of the collision integral to evaluate the collision operator. A small database of files storing pre-computed values of kernels at Korobov nodes is provided with the distribution of DGVlib. These values where computed for particular Korobov nets and for particular nDGV discretizations. If the user wishes to use a different Korobov net or the user wishes to use different velocity domain, or the user wishes to use a particular numbers of cells and gauss nodes on each cell, or the user wishes to use a different generating cell (will be defined later), then the user will have to pre-compute values of the kernels with new parameters. In this section, we will discuss how this can be done.

Remark. We note that evaluation of the pre-computed kernels can be slow, especially for large numbers of Korobov nodes and for large numbers of nDGV discretization points. As a result, the user is advised to use OpenMP parallelization. The library subroutines contain OpenMP pragmas and can be compiled with OpenMP capabilities. Some MPI subroutines are available in the code that preceded the DGVlib for computing values of kernels at the nodes of nDGV discretization. A copy of the MPI source files can be obtained from Dr. A. Alekseenko (CSUN) at `alexander.alekseenko@csun.edu`. There are currently no MPI parallel algorithms implemented for computing kernels of the collision integral on Korobov nodes. Nevertheless, for meshes up to 50^3 nodes and 10^6 Korobov integration points, the kernels can be produced using the subroutines provided with the library.

B.6.1.4 Modifications of `InitDGV0D` and `InitDGV1D` to Compute Values of the Collision Kernel at Korobov Nodes

The subroutine `SetAKorobov_DGV` is provided in DGVlib to compute values of the collision kernel at the nodes of a given Korobov net. Once the values are computed, they are stored in the following arrays that can be accessed in the `DGV_commvar` module:

- `Akor`. This is a single index array of `DP` (double precision real) values that contains non-zero values of the collision kernel at the nodes of a Korobov net. We note that only non-zero values are stored for the sake of compactness.
- `Akor_k`. This is a single index array of `I4B` (long integer) values that contain the number k of the Korobov node used to evaluate the collision kernel. Arrays `Akor_k` and `Akor` have the same number of records. For example, array `Akor` has 400 records and `Akor(39)` is a non-zero value that was obtained at a node of Korobov quadrature. The number of that note is recorded in `Akor_k(39)`. We note that the nodes itself are not stored since they can be computed from its number in a simple fashion. In other words, an entry `Akor_k(i)` is the number of the Korobov node for which the value of `Akor(i)` was computed.
- `Akor_phi` This is a single index array of `I4B` of the same size as arrays `Akor` and `Akor_k` that contains the number of the basis function for which the value of the kernel was computed. The meaning of this value will be explained later. It is used to ensure correct evaluation of the convolution form of the collision integral.
- `Akor_capphi` This is a single index array of `I4B` of the size of the total number of points in the selected nDGV discretization. One entry corresponds to one basis function of the nDGV discretization. Each entry states how many non-zero values were pre-computed and stored for the kernel for this particular basis function. The convention used in DGVlib is that entries in Akor appear in the order of increasing basis function `Akor_phi`. Under this convention, information contained in `Akor_capphi` can be used to locate records in `Akor`.
- `korob_net_param` This is an array of seven `I4B` values that keeps the parameters of the Korobov net. The use of these parameters will be discussed in the next section.

Perhaps the simplest way to compute the values of the collision kernel at Korobov nodes is un-commenting a handful of lines in subroutines `InitDGV0D` and `InitDGV1D`. Specifically, the following lines were inserted in `InitDGV0D` and `InitDGV1D` that enable computation of the kernel values:

```
! call SetA_DGV          ! call evaluation of A on Gauss nodes
call SetAKorobov_DGV    ! call evaluation of A on Korobov nodes
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!
⋮
! call WriteAarraysDGV     ! write A on Gauss nodes on hard drive
call WriteAKorArraysDGV  ! write A on Korobov nodes on hard drive
!!! uncomment the stop directive here is only computing A or Akor arrays....
stop
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

By un-commenting the lines containing calls of `SetAKorobov_DGV` and `WriteAKorArraysDGV` the user will cause DGVlib to compute the values of the collision kernel at Korobov nodes. Notice that the subroutine `SetAKorobov_DGV` contains OpenMP pragmas and can be computed in parallel.

It is recommended that evaluation of the kernels be performed as a separate operation. As a result, a `stop` command is inserted to terminate execution after the computed values were saved on the hard drive.

If the user desires to invoke subroutines `SetAKorobov_DGV` and `WriteAKorArraysDGV` directly from the driver, the user needs to be aware that these subroutines rely on data structures estabilshed by other subroutines. These subroutines appear (in a correct sequence) in `InitDGV0D` and `InitDGV1D` above the call of `SetAKorobov_DGV`. All this subroutines need to be executed prior to calling `SetAKorobov_DGV` to ensure the correct result.

B.6.1.5 Definition of Korobov Nodes

Definition of Korobov quadrature rules for evaluation of multidimensional integrals as well as some theoretical results about their accuracy can be found in [6]. Here we only state the definition as it applies for the evaluation of (1). The classical Korobov quadrature is used to approximately evaluate a integral of a periodic function on a multidimensional cube $[0,1]^d$ where d is the dimension of the cube. In the case of (1), $d = 6$. Thus the classical quadrature rule is defined as

$$\int_{[0,1]^6} f(\vec{x})\, d\vec{x} \approx \frac{1}{p} \sum_{k=1}^{p} f(\vec{x}_k)\,,$$

where the Korobov nodes x_k are given by

$$\vec{x}_k = \left\{ \{\frac{a_1 k}{p}\}, \{\frac{a_2 k}{p}\}, \ldots, \{\frac{a_6 k}{p}\} \right\}\,.$$

Here p is the total number of integration nodes in the quasi-stochastic quadrature and $a_1, \ldots, a_6$ are the coefficients that are used to construct the quadrature nodes, and $\{\ \}$

denotes the truncation fo the fractional part. Scaling and shifting is used to produce the values fo the quadrature nodes on the velocity domain.

We note that the Korobov nodes x_k can be computed from their definition based on the number of the node k and the combination of seven parameters p, $a_1, \ldots, a_6$. As a result, DGVlib does not store the individual nodes. Instead, DGVlib stores the parameters p, $a_1, \ldots, a_6$ for each Korobov net that is used and tracks the number of the node, k, when necessary.

It turns out that not all combinations of parameters p, $a_1, \ldots, a_6$ will produce nodes that cover the cube $[0,1]^6$ uniformly. In fact, it is a challenging mathematical problems to identify the parameters that make a good map. Some results, however, can be found in literature. In Table 1 [5], combinations of p, $a_1, \ldots, a_6$ are listed that provide a good approximation to the integral

$$I = \int_{[0,1]^6} e^{-\|\vec{x}-0.\vec{5}\|^2/T} d\vec{x},$$

where $0.\vec{5} = (.5, .5, .5, .5, .5, .5)$. The relative error of the integration using Korobov nets is given in the last column of the Table 1.

B.6.1.6 Definition of the Collision Kernel and the Convolution Form of the Collision Integral

Presently, evaluation of the collision operator in DGVlib is only implemented for uniform nDGV discretizations. In uniform DG discretizations, the velocity mesh is uniform, and the same DG basis functions used on all cells of the velocity mesh. The key advantage of uniform Galerkin discretizations is that they naturally give rise to a convolution form of the Galerkin projection of the collision integral. The idea of the convolution form is summarized below. In particular, the notions of generating (or canonical) velocity cell and generating (or canonical) basis function are explained. These notions are important for pre-computing collision kernel in the DGVlib.

Let the velocity elements K_j be uniform and let the same DG basis functions be used on each element. We can select a cell K_c and designate this cell as the generating cell and the basis functions $\phi_i^c(\vec{v})$ on K_c as the generating basis functions of the nDGV discretization. Because the velocity mesh is uniform, basis functions $\phi_i^j(\vec{v})$ on other velocity cells can be restored using a substitution of the velocity variables, namely $\phi_i^j(\vec{v}) = \phi_i^c(\vec{v} + \vec{\xi^j})$ where $\vec{\xi^j} \in \mathbb{R}^3$ is the vector that connects the centers of K_c and K_j.

We recall that operator $A(\vec{v}_1, \vec{v}, \phi_i^c)$ is invariant with respect to translations [3], and rewrite

the Galerkin projection of the collision integral as follows:

$$I_{\phi_i^j} = \frac{8}{\omega_i \Delta \vec{v}^j} \int_{\mathbb{R}^3} \int_{\mathbb{R}^3} f(t, \vec{x}, \vec{v} - \vec{\xi}^j) f(t, \vec{x}, \vec{v}_1 - \vec{\xi}^j) A(\vec{v}_1, \vec{v}; \phi_i^c(\vec{u}))\, d\vec{v} d\vec{v}_1 \,.$$

We note that this formula gives the foundation to the method employed in DGVlib for evaluation of the collision operator. Specifically,

- Values of the collision kernel $A(\vec{v}_1, \vec{v}; \phi_i^c(\vec{u}))$ are computed for all basis function of the generating cell using subroutimes `SetAkor_DGV` or `SetA_DGV`.
- Values of displacements ξ_j are determined for each nDGV basis function using subroutine `SetCellsUGI_DGV`
- Subroutines `EvalCollisionPeriodicAKor_DGV`, `EvalCollisionPeriodicAKorMixedTerms_DGV`, `EvalCollisionPeriodicAPlus_DGVII`, and `EvalCollisionPeriodicMixedTermsA_DGV`

B.6.2 Entries in the Parameter File that Determine the Evaluation of the Collision Kernel on Korobov Nodes

Entries in the parameter file that determine the evaluation of the collision appear in Figures 10, 3 and 8. Parameters affecting the evaluation of the collision kernel are discussed below.

```
! Parameters for Calculating the A-array:

cutoff radius of A-array = 3.00       ! Trad
error in integral in Chi = 1.0E-4     !  ErrChi
error in integral in Epsilon = 1.0E-5 ! ErrEps
cutoff values for A-array = 1.0E-11   ! min_sens
list of basis functions for A-array = 17969,1,1,14896 ! I1_list
parameters of Korobov net = 262144,3851,150137,150067,142641,118811,99881 !
range of node numbers I2 = 1,262144 ! I2_from and I2_to

! OpenMP parameters: (can ignore if not using OpenMP)
number of OMP threads = 4  ! Num_OMP_threads
```

Figure 10. Entries of the parameter file for computation of values of the collision kernel

- `left endpoint in u/v/w` and `right endpoint in u/v/w`. These key phrases indicate values of the variables `u_L/v_L/w_L` and `u_R/v_R/w_R`, respectively. These variables determine the boundaries of the velocity domain and determine the velocity discretization. We note that the pre-computed values of the collision kernel can only be used on the velocity domain for which they were computed.

- `number of cells in u/v/w` and `degree of local Lagrange basis in u/v/w`. The subroutine for computing the values of the collision kernel at the Korobov node uses the primary velocity discretization. Thus only the first values on these lists are affecting the evaluation of the kernel. The parameters affecting the construction of the primary mesh also will be used to create a name for the file that stores the pre-computed values. This way only pre-computed values that are consistent with the constructed nDGV discretization will be used by the library.
- `current solution save directory` indicates the value of the variable `current_solution_dir`. This is the name of the local folder where the computed values will be saved.
- `current solution base name` this key phrase indicates the values of the variable `current_solution_base_name` that stores the header of the file names for different output files created by the library. Note that this name header will be used to record the computed values of the kernel on the hard drive. However different variables will be used to read the pre-computed values of the kernel from the hard drive. The name additions `net00_` and `ch000_` discussed in the previous sections should be added to the name header manually in the cases when only a portion of the Korobov net is being computed or when the values of kernels on several nets are being computed. If, for any reason, the file was named incorrectly, it is always possible to re-name it manually following the conventions used in the library.
- `cutoff radius of A-array`. This key phrase indicates the value of the variable `Trad` which is the largest allowed distance between $\vec{v}$ and $\vec{v}_1$ for pre-computing the values of $A(\vec{v}, \vec{v}_1; \phi_i^j)$. DGVlib implements a dimensionless reduction of the collision operator. The strategy used in the library is to choose a reference velocity of the dimensionless reduction so that the support of the solution lies well within the boundaries of the velocity domain. Moreover, it is assumed that the non-negligible part of the solution is contained within a ball of **diameter** `Trad`. In this case, evaluating $A(\vec{v}, \vec{v}_1; \phi_i^j)$ for pairs $(\vec{v}, \vec{v}_1)$ such that $\|\vec{v} - \vec{v}_1\| \geq$ `Trad` is not necessary. It turns out that, in this case, products $f(t, \vec{x}, \vec{v}) f(t, \vec{x}, \vec{v}_1) A(\vec{v}, \vec{v}_1; \phi_i^j)$ are small and can be neglected in the evaluation of the collision integral. When deciding what values to use for `Trad`, please keep in mind that too small of a value will make it too restrictive and the evaluation of the collision operator will be incorrect if the support of the solution can not be contained within a ball of diameter `Trad`. On the other hand, using large values of `Trad` will increase the storage requirement for the pre-computed values.
- `error in integral in Chi` and `error in integral in Epsilon`. These key phases indicate values of the variables `ErrChi` and `ErrEps`, respectively. These errors determine how accurate the values of the collision kernel will be evaluated. Evaluation of the collision kernel consists of computing a double integral over the impact pa-

rameters variables. Both integrals are computed using adaptive quadratures. The variables `ErrChi` and `ErrEps` give the tolerance for the error indicator in the adaptive quadrature. The user advised to use as low values for these parameters as possible. However, lower values will require longer computation. Also, the adaptive integration subroutine may exceed an internal stack if adaptive quadrature will fail to converge. (This is guaranteed to happen for small enough values of tolerance. In this case the library will send the following error message: `Number of nodes in chi is too big`. This will indicate that the desired accuracy can not be achieved. The user may continue execution of the code. However, at this point accuracy is not guaranteed. The user is advised to increase the tolerance and repeat the calculation if this error message appears. Lastly, it is recommended that `ErrEps` is kept one order of magnitude smaller than `ErrChi`.

- `cutoff values for A-array` is the key phrase that indicates the values of the variable `min_sens` which is a threshold for accepting a computed values of the kernel. Values with magnitudes less than `min_sens` will not be recorded by subroutine `SetAKorobov_DGV`.
- `list of basis functions for A-array` is a key phrase to indicate the list of values to be stored in array `I1_list`. This array contains the numbers of the generating basis functions on the generating cell for which the values of the kernel are being computed. In nDGV discretization one velocity node corresponds to exactly one basis function. Moreover, in DGVlib, the number of that basis function is the same as the number of the corresponding velocity node. Thus, to determine the numbers of the basis functions for which the kernel needs to be computed, it is sufficient to determine the number of velocity nodes that fall into the canonical cell. The simplest way to determine these numbers is to use the numbering convention of the nodes on uniform velocity mesh discussed in previous sections. Also, one can use array `nodes_pcell` to identify velocity nodes that belong to a cell with the given number. As a rule, evaluation of the kernel for just one basis function is quite time consuming. Therefore, presently, the option of using more than one node is disabled in the library and only the first entry of the list is used.
- `parameters of Korobov net` is the key phase to indicate the seven integer values that define a Korobov net. These values are stored in the array `korob_net_param` and used to construct the nodes of the Korobov quadrature.
- `range of node numbers I2` is the key phrase to indicate the values of the variables of `I2_from` and `I2_to`. If more than two integer numbers are provides in the list following the key phrase, all values starting from the third are ignored. The first two values are used to restrict evaluation of the collision kernel only to a portion of the Korobov nodes. The smallest values for `I2_from` is 1 and the largest values for `I2_to`

is p, the total number of the Korobov nodes in the net. The library will evaluate kernels beginning from the node with the number `I2_from` and ending with the node with the number `I2_to`. By dividing the entire number of the Korobov nodes into portions and by using variables `I2_from` and `I2_to`, the values of the kernel on a Korobov net can be computed in portions and saved in different files.

- `number of OMP threads` is the key phrase to indicate the values of the variable `Num_OMP_threads`. This value is used to override the environmental variable that determines the number of OpenMP threads in parallel regions.

B.6.2.1 Suggested Practices

Files that store values of the collision kernel computed on Korobov nodes can be recognized by the suffix `_Akor_arrs.dat`. It is recommended that the user stores these files along with copies of the parameters files used for their evaluation. Also, it is recommended that outputs of the library be saved in text files and stored in the same directory where the files `_Akor_arrs.dat` are stored. This way, the user will always be able to determine all parameters for which these arrays were computed.

B.6.3 Computing and Storing Values of Kernels on nDGV Nodes

DGVlib uses pre-computed values of the kernel of the collision integral to evaluate the collision operator. A small database of files storing pre-computed values of kernels at the nodes of the nDGV discretization is provided with the distribution of DGVlib. These values where computed for particular velocity domains and for particular nDGV discretizations. If the user wishes to use a different velocity domain, or the user wishes to use a particular numbers of cells and gauss nodes on each cell, or the user wishes to use a different generating velocity cell (defined in the previous sections), then the user will have to pre-compute values of the kernels with new parameters. The approach to compute the values of the kernel on the nodes of the nDGV discretization is very similar to the approach to compute the values of the kernel on Korobov nodes. As a result, we will omit most of the explanation and refer the user to the previous section. In this section, we will only highlight the aspects that are different.

B.6.3.1 Modifications of `InitDGV0D` and `InitDGV1D` to Compute Values of the Collision Kernel at Korobov Nodes

Subroutine `SetA_DGV` is provided in DGVlib to compute values of the collision kernel at the nodes of the nDGV discretizations. Once the values are computed, they are stored in the following arrays that can be accessed in the `DGV_commvar` module:

- `A`. This is a single index array of `DP` (double precision real) values that contains non-

zero values of the collision kernel at the nodes of an nDGV discretization. We note that only non-zero values are stored for the sake of compactness.

- `A_xi`. This is a single index array of `I4B` (long integer) values that contain the number j of the velocity node (`nodes_u(j)`, `nodes_v(j)`, `nodes_w(j)`) that is used as the value of $\vec{v}$ when evaluating $A(\vec{v}, \vec{v}_1, \phi_l)$. Arrays `A_xi` and `A` have the same number of records.
- `A_xi1`. This is a single index array of `I4B` (long integer) values that contain the number k of the velocity node (`nodes_u(k)`, `nodes_v(k)`, `nodes_w(k)`) that is used as the value of $\vec{v}_1$ when evaluating $A(\vec{v}, \vec{v}_1, \phi_l)$. Arrays `A_xi1`, `A_xi` and `A` have the same number of records.
- `A_phi` This is a single index array of `I4B` of the same size as array `A` that contains the number l of the basis function ϕ_l for which the value of the kernel $A(\vec{v}, \vec{v}_1, \phi_l)$ was computed. The meaning of this value was explained earlier.
- `A_capphi` This is a single index array of `I4B` of the size of the total number of points in the selected nDGV discretization. One entry corresponds to one basis function of the nDGV discretization. Each entry states how many non-zero values were pre-computed and stored for the kernel for this particular basis function. The convention used in DGVlib is that entries in A appear in the order of increasing basis function `A_phi`. Under this convention, information contained in `A_capphi` can be used to locate records in `A`.

Perhaps the simplest way to compute the values of the collision kernel at the nodes of nDGV discretization is to use lines inserted into subroutines `InitDGV0D` and `InitDGV1D`. Specifically, the following lines were inserted in `InitDGV0D` and `InitDGV1D` that enable computation of the kernel values:

```
call SetA_DGV         ! call evaluation of A on Gauss nodes
! call SetAKorobov_DGV   ! call evaluation of A on Korobov nodes
!!!!!!!!!!!!!!!!!!!!!!!!!!!
⋮
call WriteAarraysDGV     ! write A on Gauss nodes on hard drive
! call WriteAKorArraysDGV  ! write A on Korobov nodes on hard drive
!!! uncomment the stop directive here is only computing A or Akor arrays....
stop
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

By un-commenting the lines containing the calls of `SetA_DGV` and `WriteAarraysDGV` the user will cause DGVlib to compute the values of the collision kernel at the nodes of nDGV

discretization. Notice that the subroutine `SetA_DGV` contains OpenMP pragmas and can be computed in parallel.

It is recommended that evaluation of the kernels be performed as a separate operation. As a result, a `stop` command is inserted to terminate execution after the computed values were saved on the hard drive.

If the user desires to invoke subroutines `SetA_DGV` and `WriteAarraysDGV` directly from the driver, the user needs to be aware that these subroutines rely on data structures established by other subroutines. These subroutines appear (in a correct sequence) in `InitDGV0D` and `InitDGV1D` above the call of `SetAKorobov_DGV`. All this subroutines need to be executed prior to calling `SetA_DGV` to ensure the correct result.

B.6.4 Entries in the Parameter File that Determine the Evaluation of the Collision Kernel on Nodes of NDGV Discretization

Most of the entries in the parameter file that control evaluation of the collision kernel at nodes of an nDGV discretization are the same as those that control evaluation of the kernel at the Korobov notes. In this section we only state the options that are different. The user can use description of the previous section for guidance ignoring the only the key phrase `parameters of Korobov net`. The use of the entries is identical for the kernel on nodes of nDGV discretization with exceptions listed below. The user can refer to Figures 10, 3 and 8 for guidance.

- `range of node numbers I2` is the key phrase to indicate the values of the variables of `I2_from` and `I2_to`. The use of these variables is the same as for the evaluation on Korobov nodes with the exception that the numbers of nodes of nDGV discretization are used. In particular, the smallest values for `I2_from` is 1 and the largest values for `I2_to` is the number of the last velocity node ($= M_u M_v M_w s_u s_v s_w$). As before, by dividing the entire collection of nDGV discretization nodes into portions and by using variables `I2_from` and `I2_to`, the values of the kernel can be computed in portions and saved in different files.

B.6.4.1 Suggested Practices

Files that store values of the collision kernel computed on Korobov nodes can be recognized by the suffix `_Aarrs.dat`. It is recommended that the user stores these files along with copies of the parameters files used for their evaluation. Also, it is recommended that outputs of the library be saved in text files and stored in the same directory where the files `_Aarrs.dat` are stored. This way, the user will always be able to determine all parameters for which these arrays were computed.

B.7 Sample OD Driver

To illustrate capabilities of DGVlib library, it was merged with a driver that implements solution of kinetic equations in zero spatial dimensions. The resulted software was used to simulate problem of spatially homogeneous relaxation in gases.

The basic scheme of the driver is shown in Figure 1. The 0D driver implements numerical solution of kinetic equations in the case when spatial derivatives in the transport part of the kinetic equations vanish. As a result, discrete kinetic equations are reduced to a system of ordinary differential equations coupled through the collision operator. The driver implements numerical integration of the zero dimensional equations in time. The order of accuracy in the temporal discretization can be varied from one to five. The time integration is performed using explicit Adams-Bashforth multi step methods.

The driver uses the nDGV discretization of the library to construct the system of coupled ODEs. Then the driver uses the library module sf02_mod to compute initial values of the solution. After that, at each time step, the driver calls the universal collision operator to determine the values of the collision operator.

The entries of the parameter file that describe the work of the universal collision operator are explained in previous sections. Below we will identify the values of the driver parameter file, `parameters.dat`.

B.7.1 Entries of the Parameter File of the 0D Driver

A sample parameter file is provided with a distribution of the 0D driver. Below we list the entries of the parameter file.

- `current solution save directory` indicates the value of the variable `current_solution_dir`. This is the name of the local folder where the computed solutions will be saved. Note that a variable with the same name exists in DGVlib. However, the driver does not have access to the variable of DGVlib and uses its own copy.
- `current solution base name` this key phrase indicates the values of the variable `current_solution_base_name` that stores the header of the file names for different output files created by the driver. Again, DGVlib has a variable with the same name, however, this copy is local to the driver and is used by subroutines of the driver.
- `initial time` is the key phrase to indicate the value of the initial moment of time, stored in the variable `t_L`.
- `final time` is the key phrase to indicate the final value of time, `t_R`.
- `time step` is the phrase to indicate the values of the time step.
- `instances to evaluate error` is the key phrase to indicate the values of `num_eval_error`

that determines how many times the driver will perform diagnostic operations on the solution. A typical example of such diagnostic operation is evaluation of moments. The diagnostic operations will be performed at equal intervals in time. The maximum number of operations during the run time is 1000.

- `instances to save solution` is the key phrase to indicate the value of `num_save_sol` that determines how many times the solution needs to be saved.
- `solution restart` is the key phrase that identifies the value of the boolean flag `need_to_restart`. When the value of the entry is "yes", the flag is set to be `.true.` and the diver will attempt to restart the solution using the time value provided in the next entry.
- `restart time` is the entry indicating the values of the string variable `restart_time_txt` that carries the values of the time for which the solution was previously computed and which should be used as the initial data for the solution. The value of time variable, at which the solution was saved to the hard drive is encoded into the name of the solution file.

The 0D driver maintains a number of local folders in which various output files are placed. Below is the summary of the folders' names as the files they will contain.

- `sol080909` is the folder where DGVlib will write files containing information about nDGV.
- `results` is the folder where the 0D solutions are placed when the solution is saved on the hard drive.
- `moments` is the folder where the driver will place text files with values of moments that were computed from the solution.
- `L1_errs` is the folder where additional diagnostic information about the solution is recorded.

B.8 Sample 1D Driver

To illustrate capabilities of DGVlib library, it was merged with a driver that implements solution of kinetic equations in one spatial dimension. The obtained software was used to simulate the problem of normal shock wave in dilute gases.

The basic scheme of the driver is shown in Figure 1. The 1D driver implements numerical solution of kinetic equations in the case when spatial derivatives in y and z directions in the transport part of the kinetic equations vanish. As a result, discrete kinetic equations are reduced to a system of first order transport equations coupled through the collision operator. The driver implements high order DG discretizations in the spatial variable and

high order temporal integration using explicit Adams-Bashforth multi step methods.

The driver invokes the nDGV discretization of the library to construct the system of coupled transport equations. The driver uses the library module sf02_mod to compute initial values of the solution and for computing the exact boundary values of the solution. At each time step, on each spatial cell, the driver calls the evaluation of the universal collision operator to determine the values of the collision operator at Gauss quadrature nodes that are used to compute projections of the collision operator on DG basis functions on that spatial cell.

The entries of the parameter file that describe the work of the universal collision operator are explained in previous sections. Below we will identify the values of the driver parameter file, `parameters.dat`.

B.8.1 Entries of the Parameter File of the 1D Driver

A sample parameter file is provided with a distribution of the 1D driver. Below we list entries of the parameter file that affect the execution of the driver.

- `scheme order in x` is the key phrase to indicate the degree of the polynomial basis in the DG method. In particular the value of zero gives a piece-wise constant DG discretization method while the value 4 gives a discretization using the DG basis of first five Legendre polynomials
- `left endpoint in x` and `right endpoint in x` indicate the left and right boundaries of the spatial interval.
- `uniform mesh in x` indicate the values of the boolean variable `mesh_x_uniform`. While in principle non-uniform meshes can be used with this driver, the benchmark solutions for this project were calculated using uniform meshes in variable x. The entry "yes" should be provided here.
- `number of cells in x` indicated the value of the number of cells in x
- `order of Runge Kutta method` is the value determining the order of the method used for time integration. For stability reasons, the value provided here has to be one grater than the value provided in the line `scheme order in x`.
- `conditions on the left boundary` and `conditions on the right boundary` are values determining what boundary conditions to use. The value 2 for both entries corresponds to exact boundary conditions on both left and right boundaries. This option is used for simulation of normal shock waves.
- `current solution save directory` indicates the value of the variable `current_solution_dir`. This is the name of the local folder where the computed solutions will be saved. Note that a variable with the same name exists in DGVlib. However, the driver does not have access to the variable of DGVlib and uses its own copy.

- `current solution base name` this key phrase indicates the values of the variable `current_solution_base_name` that stores the header of the file names for different output files created by the driver. Again, DGVlib has a variable with the same name, however, this copy is local to the driver and is used by subroutines of the driver.
- `initial time` is the key phrase to indicate the value of the initial moment of time, stored in the variable `t_L`.
- `final time` is the key phrase to indicate the final value of time, `t_R`.
- `time step` is the phrase to indicate the values of the time step.
- `instances to evaluate error` is the key phrase to indicate the values of `num_eval_error` that determines how many times the driver will perform diagnostic operations on the solution. A typical example of such diagnostic operation is evaluation of moments. The diagnostic operations will be performed at equal intervals in time. The maximum number of operations during the run time is 1000.
- `instances to save solution` is the key phrase to indicate the value of `num_save_sol` that determines how many times the solution needs to be saved.
- `solution restart` is the key phrase that identifies the value of the boolean flag `need_to_restart`. When the value of the entry is "yes", the flag is set to be `.true.` and the diver will attempt to restart the solution using the time value provided in the next entry.
- `restart time` is the entry indicating the values of the string variable `restart_time_txt` that carries the values of the time for which the solution was previously computed and which should be used as the initial data for the solution. The value of time variable, at which the solution was saved to the hard drive is encoded into the name of the solution file.
- `number of OMP threads` is the key phrase to indicate the values of the variable `Num_OMP_threads`. This value is used to override the environmental variable that determines the number of OpenMP threads in parallel regions.

B.9 Additional Topics

B.9.1 Linking DGVlib to C/C++ Codes

Many advanced CFD solvers are written in C/C++. DGVlib can provide additional capability to these codes if DGVlib subroutines are made accessible from the C/C++ software. In this section we provide very brief guidelines on compiling DGVlib and linking the library to C/C++ software. We would like to note that we only document one possible approach for linking the code, however, other solutions may be possible.

There are three key difficulties in linking Fortran 90 code to C/C++. The first difficulty is related to the conventions about variable types and to the differences how C and Fortran pass variables in subroutines. This difficulty is very well address in a number of textbooks and manuals. We therefore do not discuss it here.

The second difficulty is the difference in naming conventions in objective files. In particular, name of subroutines generated by C compilers usually are case sensitive. The names generated by the Fortran compilers can be low case (default option for many compilers), or can be mixed case. Because subroutine calls in C are case sensitive, the user needs to make sure that C calls a correct name, otherwise the linker will not be able to locate the function. The easiest way to determine the name is to open objective file in text editor and search for the name of the function. Additional difficulty in naming is that Fortran adds underscores or symbols "@" in front and after the names. Please use available compiler option to control this behaviour. The discussion below applies to the case when the compiler adds underscores to the name.

The third difficulty of linking Fortran 90 subroutines with C is related to the fact that Fortran 90 subroutines are usually encapsulated in modules. Solutions can be found online that explain how to call Fortran 77 subroutines, which are not included in modules. It turns out that these solutions do not apply to Fortran 90 subroutines that are included in modules. The authors were able to locate one reference that that describes this case [1]. We discuss this solution here. In our case, DGV lib was complied with `gfortran` and successfully linked to a C++ code compiled with `gcc`.

The solution found in [1] is based on matching the names of subroutines in the C/C++ calls. Specifically, let us assume that a Fortran 90 subroutine `initDGV` that does not take or returns any parameters is located in the module `DGV_miscset`, e.g.,

```
module DGV_miscset
contains
subroutine initDGV
...
end subroutine
end module DGV_miscset
```

Then the external C declaration for it in C++ is shown in Figure 11. You may also need to add `attribute((stdcall))` after the extern. By default, C assumes `cdecl` which stacks the parameters differently.

```
extern "C"
{
 //         ,-- 2 Leading underscores to start
 //         | ,-- then the module name
 //         | |     ,-- then _MOD_
 //         | |     |    ,-- then the subroutine name
 //         V V     V    V
extern void __DGV_miscset_MOD_initDGV();
}
```

Figure 11. Name of the Fortran 90 subroutine that is included in a module in external C declaration.

B.10 Library Subroutines and Functions

B.10.1 Functions of DGV_dgvtools_mod

B.10.1.1 lagrbasfun

Function `lagrbasfun` evaluates the Lagrange basis function $\phi(x)$ given the x and the array of nodes κ_i as follows

$$\phi_i(x) = \prod_{j \neq i} \frac{\kappa_j - x}{\kappa_j - \kappa_i}$$

The interface of the function is the following

```
function lagrbasfun(i,x,kappa) result (y)

integer (I4B) :: i
real (DP) :: x
real (DP), dimension(:) :: kappa
real (DP) :: y
```

Here is a brief description of the variables.

- `i` is a number of type `INTEGER` that gives the number of the node where the Lagrange basis function is one.
- `x` is a point of type `DP` (double precision real numbers) where the function needs to be evaluated.
- `kappa` is a number of type `DP` that represents the nodes of the lagrange basis functions.
- `y` is the value of the function of type `DP`.

B.10.1.2 lagrbasfun_xvec

Function `lagrbasfun_xvec` evaluates the Lagrange basis function $\phi(x)$ given the x and the array of nodes κ_i as follows

$$\phi_i(x) = \prod_{j \neq i} \frac{\kappa_j - x}{\kappa_j - \kappa_i}$$

The interface of the function is the following

```
lagrbasfun_xvec(i,x,kappa) result (y)
integer (I4B) :: i
real (DP) :: x
real (DP), dimension(:) :: kappa
real (DP) :: y
```

Here is a brief description of the variables.

- `i` is a number of type `INTEGER` that gives the number of the node where the Lagrange basis function is one.
- `x` is a point of type `DP` (double precision reals) where the function needs to be evaluated.
- `kappa` is a number of type `DP` that represents the nodes of the lagrange basis functions.
- `y` is the value of the function of type `DP`.

B.10.1.3 EvalLagrBasisFunByNdsDGblzm

Function `EvalLagrBasisFunByNdsDGblzm` evaluates the Lagrange basis function $\phi(x)$ identified by a node for a given value of velocity. To do this, first the cell is found to which this velocity belongs. Then the cell is found to which the basis function belongs, and the numbers of the 1D-basis functions are identified. Then if the basis function is defined on this cell, it is evaluated using the function `lagrbasfun` mentioned above. If the velocity and the basis function belong to different cells, then the value of the basis function is zero.

The interface of the function is the following

```
EvalLagrBasisFunByNdsDGblzm(u,v,w,i) result (y)
real (DP) :: u,v,w
integer (I4B) :: i
real (DP) ::  y
```

Here is a brief description of the variables.

- `u,v,w` are the components of the given velocity of type `DP` (double precision real), where the basis function needs to be evaluated.
- `i` is of type `INTEGER` that represents the number of the node in the nodes array. This is the number of the node associated with the given Lagrange basis function.
- `y` is a number of type `DP` that gives the value of the Lagrange basis function $\phi(x)$ on the given velocity.

B.10.1.4 EvalLagrBasisFunByNdsDGblzmEZ

Function `EvalLagrBasisFunByNdsDGblzmEZ` evaluates the Lagrange basis function $\phi(x)$ identified by a node for a given value of velocity. To do this, first the cell is found to which this velocity belongs. Then the cell is found to which the basis function belongs, and the numbers of the 1D-basis functions are identified. Then if the basis function is defined on this cell, it is evaluated using the function `lagrbasfun` mentioned above. If the velocity and the basis function belong to different cells, then the value of the basis function is zero.

The interface of the function is the following

```
EvalLagrBasisFunByNdsDGblzmEZ(u,v,w,i) result (y)

ireal (DP) :: u,v,w
integer (I4B) :: i
real (DP) ::  y
```

Here is a brief description of the variables.

- `u,v,w` are the components of the given velocity of type `DP`, where the basis function needs to be evaluated.
- `i` is of type `INTEGER` that represents the number of the node in the nodes array. This is the number of the node associated with the given basis function.
- `y` is a number of type `DP` (double precision real) that gives the value of the basis function $\phi(x)$ on the given velocity.

B.10.1.5 EvalLagrBasisFunByNdsDGblzmEZAccyCheck

Function `EvalLagrBasisFunByNdsDGblzmEZAccyCheck` can be used instead of the above function (EvalLagrBasisFunByNdsDGblzmEZ) to estimate the conservative properties of the computed operator A. We use this function instead of the above in SetA_DGV or SetAKorobov_DGV to obtain indicators of integration errors.

The interface of the function is the following

```
EvalLagrBasisFunByNdsDGblzmEZAccyCheck(u,v,w,i) result (y)
ireal (DP) :: u,v,w
integer (I4B) :: i
real (DP) ::  y
```

Here is a brief description of the variables.

- `u,v,w` are the components of the given velocity of type `DP` (double precision real), where the basis function needs to be evaluated.
- `i` is of type `INTEGER` that represents the number of the node in the nodes array. This the number of the node associated with the given basis function.
- `y` is a number of type `DP` that gives the value of the basis function on the given velocity.

B.10.1.6 IphiDGV

Function `IphiDGV` evaluates the moment of the collision integral I_{ϕ_i} for Lagrange basis functions. The basis functions are defined for each velocity node. The Collision Information Operator is calculated for all (or few basis functions). The subroutine will calculate the moment for each basis function recorded in A.

The interface of the function is the following

```
IphiDGV(f) result (Iphi)
real (DP), dimension(:) :: f
real (DP), dimension (size(A_capphi,1)) :: Iphi
```

Here is a brief description of the variables.

- `f` is the main variable of type `DP` (double precision real) in the distribution function.
- `Iphi` is a number of type `DP` that represents the result of integration.

B.10.1.7 IphiDGV_decomp

Function `IphiDGV_decomp` is a diagnostic subroutine that evaluates the moment of the collision integral I_{ϕ_i} for basis functions using the decomposition method. It is to test the advantage of decomposition if there are any. The collision operator is evaluated. The

Lagrange basis functions are defined for each velocity node. The Collision Information Operator is calculated for all (or few basis functions). The subroutine will calculate the moment for each basis function recorded in A.

The interface of the function is the following

```
IphiDGV_decomp(f) result (Iphi)
real (DP), dimension(:) :: f
real (DP), dimension (size(A_capphi,1)) :: Iphi
```

Here is a brief description of the variables.

- `f` is the main variable of type `DP` (double precision real) in the distribution function.
- `Iphi` is a number of type `DP` that represents the result of integration.

B.10.1.8 EvalPhiPostCollIntegrand

Function `EvalPhiPostCollIntegrand` helps to calculate the post collision velocities $\overrightarrow{\xi}'$ and $\overrightarrow{\xi}'_1$ and calls evaluation of the Lagrange basis function on the post collision velocities. The function is mainly to reduce the number of repeating lines in the code.

The interface of the function is the following

```
EvalPhiPostCollIntegrand(xiu,xiv,xiw,xi1u,xi1v,xi1w,ugu,ugv,ugw, ugux2,ugvx2,ugwx2,
ugux3,ugvx3,ugwx3,epsil,sinchi,coschi,g1,g2,dsph,i1,varphi_xi,varphi_xi1) result
(y)
real (DP), intent (in) :: sinchi, coschi, g1,g2
real (DP), intent (in) :: ugu,ugv,ugw,ugux2,ugvx2,ugwx2,ugux3,ugvx3,ugwx3
real (DP), intent (in) :: epsil
real (DP), intent (in) :: dsph,xiu,xiv,xiw,xi1u,xi1v,xi1w
integer (I4B), intent (in) :: i1
real (DP), intent (in) :: varphi_xi,varphi_xi1
real (DP) :: y
```

Here is a brief description of the variables.

- `sinchi, coschi, g1,g2` are some useful numbers of type `DP` (double precision real).
- `ugu/v/w, ugux2/vx2/wx2, ugux3/vx3/wx3` are some useful coefficients of type `DP`.
- `epsil` is the angle for which the integral must be evaluated, and is of type `DP`.

- `dsph, xiu/v/w, xi1u/1v/1w` are the pre-collision velocities of type `DP`.
- `i1` is a number of type `INTEGER` that represents the number of the basis function to evaluate.
- `varphi_xi, varphi_xi1` are the values of the basis function on xi and xi1 of type `DP`.
- `y` is the result of the calculation of type `DP`.

B.10.1.9 A_IntEpsilon

Function `A_IntEpsilon` evaluates

$$\sin\chi \int_0^{2\pi} (\varphi(\xi') + \varphi(\xi_1'))d\varepsilon.$$

The result is the integral for this particular angle χ. This portion implements the integration in ε in the evaluation of operator A.

The interface of the function is the following

```
A_IntEpsilon(chi,ires_e,xiu,xiv,xiw,xi1u,xi1v,xi1w,ugu,ugv,ugw,ugux2,ugvx2,ugwx2,
ugux3,ugvx3,ugwx3,g1,g2,dsph,i1,ErrEps,varphi_xi,varphi_xi1) result (y)

real (DP), intent (in) :: chi,g1,g2,ires_e
real (DP), intent (in) :: ugu,ugv,ugw,ugux2,ugvx2,ugwx2,ugux3,ugvx3,ugwx3
real (DP), intent (in) :: dsph,xiu,xiv,xiw,xi1u,xi1v,xi1w
integer (I4B) :: i1
real (DP), intent (in) :: ErrEps
real (DP), intent (in) :: varphi_xi,varphi_xi1
real (DP) :: y
```

Here is a brief description of the variables.

- `chi, g1, g2, ires_e` are some useful numbers of type `DP` (double precision real).
- `ugu/v/w, ugux2/vx2/wx2, ugux3/vx3/wx3` are some useful coefficients of type `DP`.
- `dsph, xiu/v/w, xi1u/1v/1w` are the pre-collision velocities of type `DP`.
- `ErrEps` is the max set error of integral evauation, and is of type `DP`.
- `i1` is a number of type `INTEGER` that represents the number of the basis function to evaluate.
- `varphi_xi, varphi_xi1` are the values of the basis function on ξ and ξ_1 of type `DP`.
- `y` is the result of the calculation of type `DP`.

B.10.1.10 MassCheck

Function `MassCheck` calculates the mass of the distribution function $f(t, \overline{x}, \overline{v})$. This is a debug subroutine.

The interface of the function is the following

```
MassCheck (f) result (y)

real (DP), dimension (:), intent (in) :: f
real (DP) :: y
```

Here is a brief description of the variables.

- `f` is a variable of type `DP` (double precision real), and is the vector of nodal values of the distribution function.
- `y` is the value of the function of type `DP`.

B.10.1.11 FindCellContainsPoint_DGV

Function `FindCellContainsPoint_DGV` finds the number of the active cell in the cell arrays that contains the point (u,v,w). The function returns zero if the velocity point is not on any cells. With a given velocity, the function identifies the cell where it came from and its 1D numbers. First it looks in course cells. If a cell is found that has that velocity, then it checks if the cell is refined. If it is, then the subroutine looks into the corresponding grid and seeks the smaller cell that has it, and so on.

The interface of the function is the following

```
FindCellContainsPoint_DGV(u,v,w) result (y)

real (DP) :: u,v,w
integer (I4B) ::  y
```

Here is a brief description of the variables.

- `u, v, w` are numbers of the components of the given velocity point, of type `DP` (double precision real).
- `y` the value of the index in the cell arrays that corresponds to the active cell containing the point (u,v,w).

B.10.1.12 FindCellContainsPoint_DGVII

This function is a copy of the above subroutine `FindCellContainsPoint_DGV` except it works with the secondary mesh.

B.10.1.13 linleastsqrs

Function `linleastsqrs` solves linear least squares problem

$$|Ac - b|^2 \to min,$$

where A is a rectangular matrix with # rows > # columns. The solution consists of performing SVD decomposition, regularization by eliminating small singular values and solving the least squares problem for the regularized system.

The interface of the function is the following

```
linleastsqrs(A,b) result (c)

real (DP), dimension(:,:), intent(in) :: A
real (DP), dimension(:), intent(in) :: b
real (DP), dimension(size(A,2)) :: c
```

Here is a brief description of the variables.

- `A` is a rectangular matrix with # rows > # columns, of type `DP`.
- `b` is a useful constant of type `DP`.
- `c` is a 2-dimensional array of type `DP` that gives the result of the calculation.

B.10.1.14 kernls_enfrsd_moms

Function `kernls_enfrsd_moms` defines what moments can be enforced.

The function has the following interface

```
kernls_enfrsd_moms(n,nodes_u,nodes_v,nodes_w,ubar,vbar,wbar) result (y)

integer (I4B), intent (in) :: n
real (DP), dimension(:) :: nodes_u,nodes_v,nodes_w
real (DP), intent (in) :: ubar,vbar,wbar
real (DP), dimension(1:size(nodes_u,1)) :: y
```

Here is a brief description of the variables.

- `n` is a number of type `INTEGER` that gives the number of the moment kernel.
- `nodes_u/_v/_w` are one index arrays of type `DP` (double precision real) that contain the values of the velocity nodes.
- `u/v/wbar` are numbers of type `DP` that contain the values of the bulk velocity of the solution. The bulk velocity is provided to the kernels, rather then being directly computed from the solution.
- `y` is an array of type `DP` and have the same size as `nodes_u` that contains the values of the moment kernel evaluated at the provided velocity nodes.

B.10.2 Functions of DGV_distributions_mod

B.10.2.1 maxwelveldist_T_vector_u_vectors

Function `maxwelveldist_T_vector_u_vectors` is a re-loadable function that evaluates the 1D maxwellian equilibrium distribution with given temperature and average velocity. This function evaluates

$$f_M(t,\overline{x},\overline{u}) = (\pi T(t,\overline{x}))^{-3/2} e^{(-\frac{(u-\bar{u})^2}{T})}.$$

The interface of the function is the following.

```
maxwelveldist_T_vector_u_vectors (T,u_0,v_0,w_0,n,u,v,w) result (y)

real (DP), dimension (:), intent (in) :: T
real (DP), dimension (:), intent (in) :: n
real (DP), dimension (:), intent (in) :: u_0,v_0,w_0
real (DP), dimension (:), intent (in) :: u,v,w
real (DP), dimension (size(T),size(u))  :: y
```

Here is a brief description of the variables.

- `T` is a number of type `DP` (double precision real) that represents the temperature parameter.
- `n` is the density parameter of type `DP`.
- `u_0, v_0, w_0` are the average velocities, and is of type `DP`.
- `u, v, w` are the values of velocity variables. Here these values are of type `DP`.
- `y` is a number of type `DP` value of the density for this values of u and T.

B.10.2.2 maxwelveldist_T_vector

This is the copy of the above subroutine when T is vector and u is scalar

B.10.2.3 maxwelveldist_u_vector

This is the copy of the above subroutine when T is scalar and u is vector

B.10.2.4 maxwelveldist

This is the copy of the above subroutine when both T and u are scalars

B.10.2.5 ESBGK_f0

Function `ESBGK_f0` evaluates the 3D ES-BGK equilibrium distribution with given temperature and average velocity.
$f_0(t,u) = \frac{n(t)}{\sqrt{(\pi^3 det\mathbb{T}(t))}} e^{(-c^{\vec{T}}\mathbb{T}^{-1}c)}$.
The parameters are dependent on time and velocity, but not in the spacial variable x.

The interface of the function is the following.

```
ESBGK_f0 (TensorInv,Determinant,n,u_0,v_0,w_0,nodes_u,nodes_v,nodes_w) result
(f0)

real (DP), dimension(3,3), intent (in) :: TensorInv
real (DP), intent (in) :: u_0, v_0, w_0
real (DP), intent (in) :: Determinant
real (DP), intent (in) :: n
real (DP), dimension(:), intent (in) :: nodes_u,nodes_v,nodes_w
real (DP), dimension (1:size(nodes_u,1)) :: f0
```

Here is a brief description of the variables.

- `TensorInv` is the inverse of the tensor matrix of type `DP`.
- `u_0, v_0, w_0` are the bulk velocities of type `DP`.
- `Determinant` is the Determinant of the Tensor matrix, and is of type `DP`.
- `n` is the number of type `DP` that represents the density.
- `nodes_u/_v,/_w` is a number of type `DP` that represents the velocity nodes.
- `f0` is the result of the ES-BGK distribution function of type `DP`.

B.10.2.6 Shakhov_f0

Function `Shakhov_f0` evaluates the dimensionless 3D Shakhov equilibrium distribution function with given temperature and average velocity and the given $Prandtl$ number. This function is

$$f_0(t,u) = f_M(t,u)(1 + (4/5)(1 - Pr)S_a c^a (c_a c^a - 5/2))$$

where
$c_a = (u_a - \bar{u}_a)/\sqrt{T}$,
$S_a = q_a/(nT^{3/2})$,
$q_a = \int f(u_a)(u_a - (\bar{u})_a)|u - \bar{u}|^2 du$.

The interface of the function is the following.

```
Shakhov_f0 (alpha,n,u_0,v_0,w_0,T,S_u,S_v,S_w,nodes_u,nodes_v,nodes_w) result (f0)
real (DP), intent (in) :: alpha
real (DP), intent (in) :: T
real (DP), intent (in) :: n
real (DP), intent (in) :: u_0, v_0, w_0
real (DP), dimension(:), intent (in) :: S_u,S_v,S_w
real (DP), dimension(:), intent (in) :: nodes_u,nodes_v,nodes_w
real (DP), dimension (1:size(nodes_u,1)) :: f0
```

Here is a brief description of the variables.

- `alpha` is the Prandtl number of type `DP`.
- `T` is a number of type `DP` that represents the temperature parameter.
- `n` is the number of type `DP` that represents the density of the model.
- `u_0, v_0, w_0` are the bulk velocities of the local Maxwellian of type `DP`.
- `S_u/_v/_w` are the components of the vector S in the Shakhov model, and are of type `DP`.
- `nodes_u/_v/_w` is a number of type `DP` that represents the velocity nodes.
- `f0` is the result of the Shakhov distribution function of type `DP`.

B.10.3 Functions of DGV_sf02

B.10.3.1 f_1D3D

Function `f_1D3D` evaluates the a bimodal Maxwellian distribution. This function is for the first and second variable scalars.

The interface of the function is the following.

```
f_1D3D (x, u, v, w, t) result (y)

real (DP), intent (in)  :: x
real (DP), intent (in)  :: u,v,w
real (DP), intent (in)  :: t
real (DP)  :: y
```

Here is a brief description of the variables.

- `u, v, w` are the vectors of values of variable (u,v,w) of type `DP` where the function needs to be evaluated.
- `x` is a number of type `DP` that represents vector of values of variable x where the function needs to be evaluated.
- `t` is the number of type `DP` that represents the value of time parameter.
- `y` is the result of the function of type `DP` that gives the value of time parameter.

B.10.3.2 f_1D3D_x_vector_u_vectors

Function `f_1D3D_x_vector_u_vectors` is a copy of the function above (`f_1D3D`) for the vector first variable and vector second variable.

B.10.3.3 f_1D3D_u_vectors

Function `f_1D3D_u_vectors` is a copy of the function above (`f_1D3D`) for the scalar first variable and vector second variable.

B.10.3.4 f_1D3D_x_vector

Function `f_1D3D_x_vector` is a copy of the function above (`f_1D3D`) for the vector first variable and scalar second variable.

B.10.3.5 fade

Function `fade` makes the fading coefficients. It uses $x1$ and $x2$ constants of the module and
if $x < x1$ it returns 1
if $x > x2$ it returns 0 otherwise, it returns $\frac{x2-x}{x2-x1}$.

The interface of the function is the following.

```
fade (x) result (y)

real (DP), intent (in)  :: x
real (DP)  :: y
```

Here is a brief description of the variables.

- `x` is a number of type `DP` (double precision real) that represents the value of variable where the function needs to be evaluated.
- `y` is the result of the function of type `DP`.

B.10.4 Subroutines of DGV_dgvtools_mod

B.10.4.1 finduiviwi

Subroutine `finduiviwi` looks through three one-dimensional meshes and checks whether u is in the mesh $ugrid$, v is in the mesh $vgrid$, w is in the mesh $wgrid$. If it is, it returns the number greater than or equal to 1. If a zero is returned, the number is not on the mesh.

The interface of the subroutine is the following.

```
finduiviwi(ugrid,vgrid,wgrid,u,v,w,ui,vi,wi)

real (DP), dimension (:), intent (in) :: ugrid,vgrid,wgrid
real (DP), intent (in) :: u,v,w
integer (I4B), intent (out) :: ui,vi,wi
```

Here is a brief description of the variables.

- `u, v, w` are the values of velocity variables of type `DP` (double precision real).
- `u/v/wgrid` are the grids in the three variables of type `DP`.
- `ui, vi, wi` are numbers of type `DP` that represents the the coordinates of the right endpoint in each dimension.

B.10.4.2 ExtendAarraysDGV

Subroutine `ExtendAarraysDGV` extends arrays A, A_phi ,A_xi, A_xi1 for additional ni records.

The interface of the subroutine is the following.

```
ExtendAarraysDGV(An,ni)
integer (I4B), intent (out) :: An
integer (I4B), intent (in) :: ni
```

Here is a brief description of the variables.

- `An` is an `INTEGER` array that represents the length of the arrays that will be updated.
- `ni` is the number of records to add to the array, and is of type `Integer`.

B.10.4.3 ExtendAarraysDGVII

This is the copy of the subroutine above (`ExtendAarraysDGV`) with the only change that works for a secondary meshes, which is accomplished by re-naming variable in the `USE commvar, only:` statement.

B.10.4.4 ShrinkAarraysDGV

Subroutine `ShrinkAarraysDGV` shrinks arrays A, A_phi ,A_xi, A_xi1 by ni records.

The interface of the subroutine is the following.

```
ShrinkAarraysDGV(ni)
integer (I4B), intent (in) :: ni
```

The description of the variable is the following.

- `ni` is the new size of the arrays to be shrunk, and is of type `Integer`.

B.10.4.5 ShrinkAarraysDGVII

This is the copy of the subroutine above (`ShrinkAarraysDGV`) with the only change that it works for a secondary meshes, which is accomplished by re-naming variable in the `USE commvar, only:` statement.

B.10.4.6 ExtendAKorAarraysDGV

Subroutine `ExtendAKorAarraysDGV` extends arrays *Akor*, *Akor_phi*, *Akor_k* for additional *ni* records.

The interface of the subroutine is the following.

```
ExtendAKorAarraysDGV(An,ni)
integer (I4B), intent (out) :: An
integer (I4B), intent (in) :: ni
```

Here is a brief description of the variables.

- `An` is an `INTEGER` array that represents the length of the arrays that will be updated.
- `ni` is the number of records to add to the array, and is of type `Integer`.

B.10.4.7 ExtendAKorAarraysDGVII

This is the copy of the subroutine above (`ExtendAKorAarraysDGV`) with the only change that it works for a secondary mesh, which is accomplished by re-naming variable in the `USE commvar, only:` statement.

B.10.4.8 ShrinkAKorAarraysDGV

Subroutine `ShrinkAKorAarraysDGV` shrinks arrays *Akor*, *Akor_phi*, *Akor_k* by *ni* records.

The interface of the subroutine is the following.

```
ShrinkAKorAarraysDGV(ni)
integer (I4B), intent (in) :: ni
```

The description of the variable is the following.

- `ni` is the new size of the arrays to be shrunk, and is of type `Integer`.

B.10.4.9 ShrinkAKorAarraysDGVII

This is the copy of the subroutine above (`ShrinkAKorAarraysDGV`) with the only change that it works for a secondary mesh, which is accomplished by re-naming variable in the

`USE commvar, only:` statement.

B.10.4.10 FindI1sByCellNum_DGV

Subroutine `FindI1sByCellNum_DGV` goes through the nodes arrays and finds the numbers of the nodes that belong to cell with the number i. The numbers will be recorded in the array of results with the first number indicating the number of useful records. These numbers will be used to build the A array. If there is no cell with the number i, the program returns zero records and prints a warning.

The interface of the subroutine is the following.

```
FindI1sByCellNum_DGV(i,y)

integer (I4B), intent (in) :: i
integer (I4B), dimension (:), intent (out) :: y
```

Here is a brief description of the variables.

- `i` is the number of the cell where I1s need to be looked at and is of type `Integer`.
- `y` is the numbers of the nodes that belong to the cell with number i.

B.10.4.11 TruncAarrsTrhls_DGV

Subroutine `TruncAarrsTrhls_DGV` truncates the A arrays, which means that all entries of A that are below the provided threshold value (`trhld`) are being cut. Here the arrays Axi, $Axi1$, $Aphi$, A_capphi are updated correspondingly.

The interface of the subroutine is the following.

```
TruncAarrsTrhls_DGV (trhld)

real (DP), intent (in) :: trhld
```

Here is a brief description of the variable.

- `trhld` is the threshold of type `DP` (double precision real) at which to cut A.

B.10.4.12 TruncAarrsTrhls_DGVII

Subroutine `TruncAarrsTrhls_DGVII` is the copy of the subroutine above (TruncAarrsTrhls_DGV) with the only change that it works for a secondary mesh, which is accomplished by renaming variable in the `USE commvar, only:` statement.

B.10.4.13 TruncAarrsRadius_DGV

Subroutine `TruncAarrsRadius_DGV` truncates the A arrays based on the distance between ξ and $\xi 1$. If the distance is larger than the specified radius (`rad`), the record is discarded from A arrays Axi, $Axi1$, $Aphi$, A_capphi are updated correspondingly.

The interface of the subroutine is the following.

```
TruncAarrsRadius_DGV (rad)
real (DP), intent (in) :: rad
```

Here is a brief description of the variables.

- `rad` is the threshold of type `DP` at which to cut A.

B.10.4.14 TruncAarrsRadius_DGVII

Subroutine `TruncAarrsRadius_DGVII` is the copy of the subroutine above (`TruncAarrsRadius_DGV`) that works for a secondary mesh.

B.10.4.15 MakeTensor

Subroutine `MakeTensor` produces the tensor

$$\mathbb{T} = (1-\alpha)TI + \frac{\alpha}{n}\Theta$$

associated with the ES-BGK distribution, where $\Theta = \frac{2}{n}\int cc^T f(t,u)du$

The interface of the subroutine is the following.

```
MakeTensor (f,Tensor)
real (DP), dimension (:), intent (in) :: f
real (DP), dimension (3,3), intent (out) :: Tensor
```

Here is a brief description of the variables.

- `f` is the solution at the current of type `DP`.
- `tensor` represents both the tensor and the inverse tensor for the ES-BGK model,

where T is a 3 by 3 matrix

$$T = \begin{vmatrix} T11 & T12 & T13 \\ T21 & T22 & T23 \\ T31 & T32 & T33 \end{vmatrix}$$

B.10.4.16 getnu

Subroutine `getnu` assembles, reconstructs and evaluates the velocity dependent collision frequency (VDCF).

The interface of the subroutine is the following.

```
getnu(c_0,Cco,nodes_u,nodes_v,nodes_w,nuB)
```

```
real (DP), intent(in) :: c_0
real (DP), dimension(:), intent(in) :: Cco
real (DP), dimension(:), intent (in) :: nodes_u, nodes_v, nodes_w
real (DP), dimension(:), intent (out) :: nuB
```

Here is a brief description of the variables.

- `c_0` is a number of type `DP` (double precision real) that contains ν_BGK.
- `Cco` are the coefficients of the velocity dependent collision frequency (VDCF) of type `DP`.
- `nodes_u/v/w` are the velocity nodes where VDCF needs to be evaluated.

B.10.4.17 ComputeRelaxRatesBCI_DGV

Subroutine `ComputeRelaxRatesBCI_DGV` computes relaxation rates $\nu_\phi(t)$ for a selected group of moments from the provided values of the Boltzmann collision integral (BCI) I_{ϕ_i}. Values of the collision integral I_{ϕ_i} are evaluated on the secondary DG mesh. Values of the solution will be on the primary mesh. The relaxation speeds will be computed from the following definition:

$$\nu_\phi = -\frac{\int Q(f)(u)\phi(u)du}{f_\varphi(t) - f_\varphi^M(t)}$$

The interface of the subroutine is the following.

```
ComputeRelaxRatesBCI_DGV(fcolII,f,momsrates,L1_SENS_TRSHLD)
```

```
real (DP), dimension(:), intent(in) :: fcolII
```

```
real (DP), dimension(:), intent(in) :: f
real (DP), dimension(:), intent(out):: momsrates
real (DP), intent (in) :: L1_SENS_TRSHLD
```

Here is a brief description of the variables.

- `fcolII` is the value of the collision integral of type `DP` (double precision real). It is the evaluated collision operator on the secondary velocity mesh.
- `f` is the solution on one spatial cell. It is the value of the solution on the primary mesh of type `DP`.
- `momsrates` is the array of type `DP` that contains the enforced relaxation rates.
- `L1_SENS_TRSHLD` stores the local Maxwellian and the derivative of the local Maxwellian of type `DP`.

B.10.4.18 ProjectDGVtoSecMesh

Subroutine `ProjectDGVtoSecMesh` takes a solution on the primary mesh and projects it to the secondary mesh, on the same velocity domain. It is done by using nodal-DG interpolation that is built in into the DG velocity discretization. Essentially, given the values of the solution on the primary mesh, needs to be produced the values on the secondary mesh.

The interface of the subroutine is the following.

```
ProjectDGVtoSecMesh(f,fII)
```

```
real (DP), dimension (:), intent (in) :: f
real (DP), dimension (:), intent (out) :: fII
```

Here is a brief description of the variables.

- `fII` values of the solution on the secondary mesh of type `DP`.
- `f` values of the solution on the primary mesh of type `DP`.

B.10.4.19 setMom_DGV

Subroutine `setMom_DGV` computes the matrix that is used to determine the coefficients of the velocity dependent collision frequency (VDCF) in the BGK model, and the vector of differences between moments and their local equilibrium values.

The interface of the subroutine is the following.

```
setMom_DGV(Df,Mom,DifMom,ubar,vbar,wbar)

real (DP), dimension (:), intent (in) :: Df
real (DP), dimension (:,:), intent (out) :: Mom
real (DP), dimension (:), intent (out) :: DifMom
real (DP), intent (in) :: ubar,vbar,wbar
```

Here is a brief description of the variables.

- `Df` is a number of type `DP` that is the difference between the solution and the local Maxwellian. Here $Df = f - f_M$.
- `Mom` is the moments matrix that needs to be evaluated.
- `u/v/wbar` are the values of the local macro parameters of type `DP`.

B.10.4.20 SetA_DGV

Subroutine `SetA_DGV` sets up the dimensionless collision information operator

$$A(\xi, \xi_1; \varphi_p^j) = \frac{d^2|g|}{8} \int_0^{\pi} \int_0^{2\pi} (\varphi_p^j(\xi') + \varphi_p^j(\xi_1')) d\varepsilon \, \sin\chi d\chi - \frac{d^2|g|\pi}{2} (\varphi_p^j(\xi) + \varphi_p^j(\xi_1))$$

The subroutine computes values of the collision kernel at the nodes of the nDGV discretizations. Once the values are computed, they are stored in the arrays A, A_xi and A_xi1. This is a symmetric operator in ξ and ξ_1, therefore we are only interested in the records for unique un-ordered pairs (ξ, ξ_1). Because all velocities are indexed by one index i, we produce records for pairs (ξ_i, ξ_j), $i > j$, since in the case $\xi = \xi_i$ gives value 0.

Evaluation of each entry of A involves a two dimensional integral. If the entry is smaller than some given number, then the entry is neglected/nullified by force.

B.10.4.21 SetAKorobov_DGV

Subroutine `SetAKorobov_DGV` sets up the collision information operator

$$A(\xi, \xi_1; \varphi_p^j) = \frac{|\bar{g}|}{2} \int_0^{2\pi} \int_0^{b_*} (\varphi_p^j(\xi') + \varphi_p^j(\xi_1') - \varphi_p^j(\xi) - \varphi_p^j(\xi_1)) b db d\varepsilon$$

for the case when ξ and ξ_1 run over the Korobov nodes. The Korobov nodes are obtained by translating the input region to the unit cube and approximating it there by an expression of the form

$$\int_0^1 ... \int_0^1 f(x_1, x_2, ...x_s) dx_1 ... dx_s = \frac{1}{P} \sum_{k=1}^{P} f(\{\frac{a_1 k}{P}\}\{\frac{a_2 k}{P}\}...\{\frac{a_s k}{P}\}) - R_p[f] \qquad (2)$$

where $\{\frac{a_s k}{P}\}$ are the nodes of the quadrature formula.

B.10.5 Subroutines of DGV_collision_mod

B.10.5.1 EvalCollisionPeriodicA_DGV

Subroutine `EvalCollisionPeriodicA_DGV` assumes a uniform velocity grid and a periodic structure of the basis functions. Operator A is only evaluated for one basis function. The values of A for other basis functions are obtained by a transposition from operator A on the canonical cell.

The interface of the subroutine is the following.

```
EvalCollisionPeriodicA_DGV(f,fc)

real (DP), dimension (:), intent (in) :: f
real (DP), dimension (:), intent (out) :: fc
```

Here is a brief description of the variables.

- `f` is the components of the solution at the current time step of type `DP`.
- `fc` is a number of type `DP` which is the value of the collision operator for each component of the solution.

B.10.5.2 EvalCollisionLinear

Subroutine `EvalCollisionLinear` evaluates the linearized collision kernel. To call this subroutine, a vector of the linearized kernel must be prepared. The vector is

$$2\int f_m(\bar{v})A(\bar{v},\bar{v}_1,\phi)dv$$

which is calculated in the subroutine `PrepareFMA_DGV`.

The interface of the subroutine is the following.

```
EvalCollisionLinear(f,fc)

real (DP), dimension (:), intent (in) :: f
real (DP), dimension (:), intent (out) :: fc
```

Here is a brief description of the variables.

- `f` is the components of the solution at the current time step of type `DP`.

- `fc` is a number of type `DP` which is the value of the collision operator $A(\bar{v}, \bar{v}_1, \phi)$ for each component of the solution.

B.10.5.3 PrepareFMA_DGV

Subroutine `PrepareFMA_DGV` prepares the derivative of the collision integral,

$$2 \int f_m(\bar{v}) A(\bar{v}, \bar{v}_1, \phi) dv.$$

It is used in the subroutine `EvalCollisionLinear` that evaluates the linearized collision kernel.

B.10.5.4 EvalCollisionPeriodicAKorPlus_DGV

Subroutine `EvalCollisionPeriodicAKorPlus_DGV` evaluates the bilinear collision operator $A(\bar{v}, \bar{v}_1; \phi_i^j)$ using Korobov quadratures. This subroutine assumes a uniform velocity grid and a periodic structure of the basis functions. Operator *Akor* is only evaluated for one basis function. The values of A for other basis functions are obtained by a transposition from operator A on the canonical cell.

The interface of the subroutine is the following.

```
EvalCollisionPeriodicAKorPlus_DGV(f,fc)

real (DP), dimension (:), intent (in) :: f
real (DP), dimension (:), intent (out) :: fc
```

Here is a brief description of the variables.

- `f` is the components of the solution at the current time step of type `DP`.
- `fc` is a number of type `DP` which is the value of the collision operator $A(\bar{v}, \bar{v}_1, \phi)$ for each component of the solution.

B.10.5.5 EvalColESBGK

Subroutine `EvalColESBGK` evaluates the kinetic collision operator using the `ES-BGK` model.

$$Q(f) = \nu(f_0 - f)$$

where ν is the collision frequency and f_0 is ES-BGK distribution.

The interface of the subroutine is the following.

```
EvalColESBGK(f,RHS)
```

```
real (DP), dimension (:), intent (in) :: f
real (DP), dimension (:), intent (out) :: RHS
```

Here is a brief description of the variables.

- `f` is the components of the solution at the current time step of type `DP`.
- `RHS` is a number of type `DP` which is the value of the collision operator for each component of the solution.

B.10.5.6 EvalColShakov

Subroutine `EvalColShakov` evaluates the kinetic collision operator using the Shakhov model. Here, the

$$Q(f) = \nu(f_0^s - f)$$

where $\nu = \frac{P}{\mu}$ is the collision frequency and f_0^s is the Shakhov model of the equilibrium distribution.

The interface of the subroutine is the following.

```
EvalColShakov(f,RHS)
```

```
real (DP), dimension (:), intent (in) :: f
real (DP), dimension (:), intent (out) :: RHS
```

Here is a brief description of the variables.

- `f` is the components of the solution at the current time step of type `DP`.
- `RHS` is a number of type `DP` which is the value of the collision operator for each component of the solution.

B.10.5.7 EvalColVelES

Subroutine `EvalColVelES` evaluates the collision integral using VD-BGK model with velocity dependent collision frequency.

The interface of the subroutine is the following.

```
EvalColVelES(f,Df,fcol,time)
```

```
real (DP), dimension (:), intent (in) :: f,Df
real (DP), dimension (:), intent (out) :: fcol
real (DP), intent (in) :: time
```

Here is a brief description of the variables.

- `f` is an array of double real numbers that contains the discrete velocity solution at a single spatial point.
- `Df` is an array of the same size and type as f that contains deviation of the solution from the local Maxwellian. Here $Df = f - f_M$. `Df` is usually computed by the subroutines `CheckSolutionModeSimple1Donecell_DGV` or `CheckSolutionMode_DGV`.
- `fcol` is an array of the same size and type as f in which the computed values of the collision operator are stored.
- `time` is the current time of type `DP`

B.10.5.8 EvalColVelES1Donecell

Subroutine `EvalColVelES1Donecell` is a copy of the above subroutine `EvalColVelES`, and is adjusted to be used with solutions on multiple spatial cells.

The interface of the subroutine is the following.

```
EvalColVelES1Donecell(f,Df,fcol,time,cellnum)
```

```
real (DP), dimension (:), intent (in) :: f,Df
real (DP), dimension (:), intent (out) :: fcol
real (DP), intent (in) :: time
integer (I4B) :: cellnum
```

Here is a brief description of the variables.

- `f` is the components of the solution at the current time step of type `DP`.
- `Df` is a number of type `DP` that is the difference between the solution and the local Maxwellian. Here $Df = f - f_M$.
- `fcol` is an array of the same size and type as f in which the computed values of the collision operator are stored.
- `cellnum` is a long integer (1D,2D and 3D applications only) that gives the number of the spatial point on the spatial mesh where the collision integral is evaluated.

B.10.5.9 GetRelaxRates0D_DGV

Subroutine `GetRelaxRates0D_DGV` returns the relaxation rates to be used in the model with velocity-dependent collision frequency.

The interface of the subroutine is the following.

```
GetRelaxRates0D_DGV(momsrates,f,Df,time,L1_err,LocUbar,LocVbar,LocWbar,nu,momsrates_reli
dimension(:), intent(in) :: f
real (DP), dimension(:), intent(in) :: Df
real (DP), dimension(:), intent(out):: momsrates
real (DP), intent(in) :: time
real (DP), intent(out) :: L1_err
real (DP), intent (out) :: LocUbar,LocVbar,LocWbar
real (DP), intent (out) :: nu
logical, intent (out) :: momsrates_reliab
```

Here is a brief description of the variables.

- `f` the solution (velocity distribution) on one spatial cell of type `DP`.
- `Df` is a number of type `DP` that is the difference between the solution and the local Maxwellian. Here $Df = f - f_M$.
- `momsrates` is a number of type `DP` which is the array that contains the enforced relaxation rates.
- `time` is the value of the dimensionless time variable of type `DP`.
- `L1_err` is the value of the relative L_1 norm of the difference between the solution and the local Maxwellian of type `DP`.
- `LocUbar/Vbar/Wbar` is the values of the local bulk velocity of type `DP`.
- `nu` is the value of the default relaxation rate of type `DP`. It will be assigned to moments for which the relaxation rates can not be calculated form the Boltzmann collision operator.
- `momsrates_reliab` is a `LOGICAL` operator that is true if at least one rate has calculated from the Boltzmann collision operator.

B.10.5.10 GetRelaxRates1Donecell_DGV

Subroutine `GetRelaxRates1Donecell_DGV` returns the relaxation rates to be used in the model with velocity-dependent collision frequency. The subroutine will check if the rates

need to be updated. If the rates are not updated, the stored rates are returned. If the rates need to be updated, the Boltzmann collision integral is evaluated and the rates are determined from the Boltzmann collision integral.

The interface of the subroutine is the following.

```
GetRelaxRates1Donecell_DGV(momsrates,f,Df,time, L1_err,LocUbar,
LocVbar,LocWbar,nu,momsrates_reliab,cellnum)

dimension(:), intent(in) :: f
real (DP), dimension(:), intent(in) :: Df
real (DP), dimension(:), intent(out):: momsrates
real (DP), intent(in) :: time
real (DP), intent(out) :: L1_err
real (DP), intent (out) :: LocUbar,LocVbar,LocWbar
real (DP), intent (out) :: nu
logical, intent (out) :: momsrates_reliab
integer (I4B), intent (in) :: cellnum
```

Here is a brief description of the variables.

- `f` the solution (velocity distribution) on one spatial cell of type `DP`.
- `Df` is a number of type `DP` that is the difference between the solution and the local Maxwellian. Here $Df = f - f_M$.
- `momsrates` is a number of type `DP` which is the array that contains the enforced relaxation rates.
- `time` is the value of the dimensionless time variable of type `DP`.
- `L1_err` is the value of the relative L_1 norm of the difference between the solution and the local Maxwellian of type `DP`.
- `LocUbar/Vbar/Wbar` is the values of the local bulk velocity of type `DP`.
- `nu` is the value of the default relaxation rate of type `DP`. It will be assigned to moments for which the relaxation rates can not be calculated form the Boltzmann collision operator.
- `momsrates_reliab` is a `LOGICAL` operator that is true if at least one rate has calculated from the Boltzmann collision operator.
- `cellnum` is an long integer (1D,2D and 3D applications only) that gives the number of the spatial point on the spatial mesh where the collision integral is evaluated.

B.10.5.11 UniversalCollisionOperator1DonecellDGV

Subroutine `UniversalCollisionOperator1DonecellDGV` evaluates the contribution due to particle collisions to the evolution of the velocity distribution function in a kinetic equation. This is a copy of the above subroutine `GetRelaxRates1Donecell_DGV` to use in multidimensional applications.

The interface of the subroutine is the following.

```
UniversalCollisionOperator1DonecellDGV(f,fcol,time,dt,cellnum)
real (DP), dimension (:), intent (in) :: f
real (DP), dimension (:), intent (out) :: fcol
real (DP), intent (in) :: time
real (DP), intent (in) :: dt
integer (I4B), intent (in) :: cellnum
```

Here is a brief description of the variables.

- `f` the solution of the current time step of type `DP`.
- `fcol` is an array of the same size and type as f in which the computed values of the right side collision operator are stored.
- `time` is the value of the current time of type `DP`.
- `dt` is the time step of type `DP`.
- `cellnum` is the number of the spatial cell for which this distribution f function corresponds of type `INTEGER`.

B.10.5.12 CheckSolutionMode_DGV

Subroutine `CheckSolutionMode_DGV` evaluates macro-parameters of the solution, then it subtracts from the solution a Maxwellian with the computed macro-parameters. This difference is placed into Df then a norm of Df is assessed. This subroutine also checks the solution and calls for the update of the linearized operator $f_M A$.

The interface of the subroutine is the following.

```
CheckSolutionMode_DGV(f,Df)
dimension(:), intent(in) :: f
real (DP), dimension(:), intent(in) :: Df
```

Here is a brief description of the variables.

- `f` the solution of the current state of type `DP`.
- `Df` is a number of type `DP` that is the difference between the solution and the local Maxwellian. Here $Df = f - f_M$.

B.10.5.13 CheckSolutionModeSimple1Donecell_DGV

Subroutine `CheckSolutionModeSimple1Donecell_DGV` evaluates macro-parameters of the solution, then it subtracts from the solution a Maxwellian with the computed macro-parameters. This difference is placed into Df then a norm of Df is assessed. This subroutine also checks the solution and calls for the update of the linearized operator $f_M A$.
This subroutine is a simplified version of the above subroutine `CheckSolutionMode_DGV`.

The interface of the subroutine is the following.

```
CheckSolutionModeSimple1Donecell_DGV(f,Df,cellnum,slt)

dimension(:), intent(in) :: f
real (DP), dimension(:), intent(in) :: Df
integer (I4B), intent(in) :: cellnum ! the number of the cell for which run_mode is calc
logical, intent (in) :: slt
```

Here is a brief description of the variables.

- `f` the solution of the current state of type `DP`.
- `Df` is a number of type `DP` that is the difference between the solution and the local Maxwellian. Here $Df = f - f_M$.
- `cellnum` is the number of type `INTEGER` that represents the number of the cell for which `run_mode` is calculated.
- `slt` is a `LOGICAL` operator for which when $if = true$, no printout is generated.

B.10.5.14 CheckNuUpdateNeededF0D_DGV

Subroutine `CheckNuUpdateNeededF0D_DGV` checks if the coefficients of the velocity dependent collision frequency need to be updated. The subroutine uses the current solution, current time, some records and some set parameters to decide if the coefficients need to be updated.

The interface of the subroutine is the following.

```
CheckNuUpdateNeededF0D_DGV(updateNulcl,time,f,Df,LocDens,
LocUbar,LocVbar,LocWbar,LocTempr,L1_err)

logical, intent (out) :: updateNulcl
real (DP), intent(in) :: time
real (DP), dimension(:), intent (in) :: f
real (DP), dimension(:), intent (out) :: Df
real (DP), intent (out) :: LocDens,LocUbar,LocVbar,LocWbar,LocTempr
real (DP), intent (out) :: L1_err
```

Here is a brief description of the variables.

- `updateNulcl` is a `LOGICAL` operator where if the coefficient needs to be updated returns flag $if = true$.
- `time` is the value of the current time of type `DP`.
- `f` the current solution of the primary mesh of type `DP`.
- `Df` is a number of type `DP` that is the difference between the solution and the local Maxwellian. Here $Df = f - f_M$.
- `LocUbar/Vbar/Wbar, LocTempr, LocDens` are numbers of type `DP` that represent the values of the local macro-parameters.
- `L1_err` is the L1-norm of Df of type `DP`.

B.10.5.15 CheckNuUpdateNeededF0Da_DGV

Subroutine `CheckNuUpdateNeededF0D_DGV` is a copy of the above subroutine `CheckNuUpdateNeededF0D_DG` with the exeption that `Df` ($f - f_M$, the difference between f and the local Maxwellian) is already provided, therefore no need to compute it.

B.10.6 Subroutines of DGV_miscsetup

B.10.6.1 SetGDVGnodes

Subroutine `SetGDVGnodes` sets up the nodes used in various integrations and used as ordinates/nodal values. This subroutine uses parameters set by SetDGVParams and sets arrays used in other programs.

B.10.6.2 SetDGVblzmmesh

Subroutine `SetDGVblzmmesh` sets up the meshes. It is highly dependent on the main program and created mainly to organize the main program.

B.10.6.3 SetDGVblzmmeshII

Subroutine `SetDGVblzmmeshII` sets up the secondary meshes. It is highly dependent on the main program and created mainly to organize the main program. The suffix II in the name suggests that the subroutine works with the secondary mesh. This is a copy of the above subroutine (`SetDGVblzmmesh`), with the only change that it works with secondary meshes. This is accomplished by re-naming variable in the `USE commvar, only:` statement.

B.10.6.4 OldCreateUmesh

Subroutine `OldCreateUmesh` is used to generate 1D meshes in the velocity variables.

The interface of the subroutine is the following.

```
OldCreateUmesh(ugrid,ugrid_cap,mesh_u_uniform,u_nonuniform_mesh_type,u_L,u_R)
real (DP), dimension (:), intent (out)  :: ugrid
integer (I4B), dimension (:), intent (out)  :: ugrid_cap
logical, intent(in) :: mesh_u_uniform
integer (I4B), intent (in) :: u_nonuniform_mesh_type
real (DP), intent (out) :: u_L, u_R
```

Here is a brief description of the variables.

- `ugrid` is an array that contains a 1D mesh in a variable `u`.
- `ugrid_cap` is a number of type `INTEGER` that stores the total number of points for each 1D mesh points which includes both endpoints, i.e. [`u_L`,`u_R`].
- `mesh_u_uniform` is a logical operator that returns true if the 1D mesh in `u` is uniform.
- `u_L, u_R` are numbers of type `DP` that represent the left and right endpoints of the interval in 1D mesh `u`.

B.10.6.5 Set3DCellsR_DGV

Subroutine `Set3DCellsR_DGV` creates a 3D velocity mesh. It looks up the `u, v, w` 1D grids arrays and creates 3D cells from these grids arrays.

B.10.6.6 Set3DCellsR_DGVII

Subroutine `Set3DCellsR_DGVII` creates a 3D velocity mesh. The suffix II in the name suggests that the subroutine works with the secondary mesh. The subroutine looks up the `u, v, w` 1D grids arrays and creates 3D cells from these grids arrays.

B.10.6.7 FillCellsArrsDGV

Subroutine `FillCellsArrsDGV` populates the certain portion of the cells_arrays.

The interface of the subroutine is the following.

```
FillCellsArrsDGV(ibeg,iend,umesh,vmesh,wmesh,pgrid,gou,gov,gow)
integer (I4B), intent(in) :: ibeg,iend
real (DP), dimension (0:), intent (in) :: umesh,vmesh,wmesh
integer (I4B), intent (in) :: pgrid
integer (I4B), intent (in) :: gou,gov,gow
```

Here is a brief description of the variables.

- `ibeg, iend` are numbers of type `INTEGER` that are the beginning and end of the range to fill the cells array.
- `u/v/wmesh` the 1D meshes in variables `u, v` and `w`, whose tensor product makes up the 3D grid.
- `pgrid` is a number of type `INTEGER` which represents the number of the parent grid to which these cells belong.
- `gou/v/w` are numbers of type `INTEGER` where they carry the information about the number of Gauss nodes in cells for each velocity dimension. Specifically, cells `gou(j)` gives the number of Gauss nodes in component u, on the cell `Kj`.

B.10.6.8 FillCellsArrsDGVII

Subroutine `FillCellsArrsDGVII` populates the certain portion of the cells_arrays. This is a copy of the above subroutine (`FillCellsArrsDGV`), with the only change that it works with secondary meshes, which is accomplished by re-naming variable in the `USE commvar, only:` statement.

B.10.6.9 CellsRefineDGV

Subroutine `CellsRefineDGV` refines cells that are on the supplied list into subcells.

The interface of the subroutine is the following.

```
CellsRefineDGV(reflist,refu,refv,refw)

integer (I4B), dimension (:), intent (in) :: reflist
integer (I4B), intent(in) :: refu,refv,refw
```

Here is a brief description of the variables.

- `reflist` is of type `INTEGER` that is the number of cell to be refined.
- `refu/v/w` are the refinement factors in dimension `u, v`, and `w`, of type `INTEGER`

B.10.6.10 ExtendGridsDGV

Subroutine `ExtendGridsDGV` extends the `grids_cap_u/_v/_w` and the `grids_u/_v/_w` arrays.

The interface of the subroutine is the following.

```
ExtendGridsDGV(dumu,dumv,dumw,cgrid)

real (DP), dimension (:), intent (in) :: dumu,dumv,dumw
integer (I4B), intent (out) :: cgrid
```

Here is a brief description of the variables.

- `dumu/v/w` are the new velocity values that will be added to grids, and is of type `DP`.
- `cgrid` is a number of type `INTEGER` that gives the number of the child grid that will be created.

B.10.6.11 SetNodesDGV

Subroutine `SetNodesDGV` sets all nodes in velocity that will ever be used.

B.10.6.12 SetNodesDGVII

Subroutine `SetNodesDGVII` sets all nodes in velocity in secondary grid, that will ever be used.

B.10.6.13 SetCellsUGINdsSftArrs

Subroutine `SetCellsUGINdsSftArrs` sets up the arrays `cells_gui/_gvi/_gwi` that stores the relative integer address of the cell on the grid. It only works if there is only one grid in the mesh. Also this subroutine sets up arrays that gives the address shift for different nodes to read staff from the `A`-array.

The interface of the subroutine is the following.

```
SetCellsUGINdsSftArrs(ccell)
integer (I4B), intent(in) :: ccell
```

Here is a brief description of the variables.

- `ccell` is a number of type `DP` which is a number of the canonical cell. This is a cell in the middle of the domain for which `A` is computed.

B.10.6.14 SetCellsUGINdsSftArrsII

Subroutine `SetCellsUGINdsSftArrsII` is a copy of the above subroutine, with the only change that it works with secondary meshes.

B.10.6.15 InitDGV0D

Subroutine `InitDGV0D` performs initialization of the parameters, constants and the variables of the DGV Library.

B.10.6.16 InitDGV1D

Subroutine `InitDGV1D` is a copy of the above subroutine (`InitDGV0D`) that performs initialization of the parameters, constants and the variables of the DGV Library, except it creates some arrays to support multiple spatial cells.

The interface of the subroutine is the following.

```
InitDGV1D(nspatcells)
integer (I4B), intent (in) :: nspatcells
```

Here is a brief description of the variable.

- `nspatcells` is a number of type `INTEGER` which is the total number of the spatial

cells. Arrays will be created to store information for each spatial cell.

B.10.6.17 prepare_sDGVpMap_DGV

Subroutine `prepare_sDGVpMap_DGV` sets some supplementary arrays that will be used to project the solution from the primary to the secondary meshes. In particular, this subroutine sets the array `nodesII_primcell`.

B.10.7 Subroutines of DGV_readwrite

B.10.7.1 SetDGVParams

Subroutine `SetDGVParams` reads the variables from the `parameter.dat` file on the hard drive. The variables from the common variable block are accessed directly.

The interface of the subroutine is the following.

```
SetDGVParams(pfname,slt)

character (len=*), intent (in) :: pfname
integer, intent (in) :: slt
```

Here is a brief description of the variables.

- `pfname` is the name of the file of type `CHARACTER` where the parameters are stored.
- `slt` is a number of type `INTEGER` which is the parameter determining if the printout is generated
  (If == 0 then the printout is generated).

B.10.7.2 WriteGridsDGV

Subroutine `WriteGridsDGV` writes the `grids_`arrays on the disk.

B.10.7.3 WriteCellsDGV

Subroutine `WriteCellsDGV` writes the *cells*_arrays on the disk.

B.10.7.4 WriteNodesDGV

Subroutine `WriteNodesDGV` writes the *nodes*_arrays on the disk.

B.10.7.5 WriteAarraysDGV

Subroutine `WriteAarraysDGV` writes the A_arrays from the disk.

B.10.7.6 WriteAKorArraysDGV

Subroutine `WriteAKorArraysDGV` writes the $AKor$_arrays from the disk.

B.10.7.7 WriteDGV2dArray

Subroutine `WriteDGV2dArray` writes the 2D arrays on the hard drive.

The interface of the subroutine is the following.

```
WriteDGV2dArray(A,suffix)

real (DP), dimension (:,:), intent (in) :: A
character (len=10) :: suffix
```

Here is a brief description of the variable.

- `A` is a number of type `DP` that represents the incoming array of doubles.
- `suffix` is the string of type `CHARACTER` that holds the suffix, which is a string containing a combination of letters that will be attached to the name of the file.

B.10.7.8 WriteI1DGV

Subroutine `WriteI1DGV` writes a sequence of node indices to be the later used in the evaluation of `A` operator. I1 is the array of integers that needs to be written on the disk. Notice that the first element in the array I1(1) contains the total number of records. So then only 2..I1(1)+1 cells in the array are important.

The interface of the subroutine is the following.

```
WriteI1DGV(u,v,w)

real (DP), intent (in) :: u,v,w
```

Here is a brief description of the variable.

- `u,v,w` are numbers of type `DP` that represent the components of the velocity.

B.10.7.9 WriteErr_DGV

Subroutine `WriteErr_DGV` writes some error arrays in the 0D solution on the disk.

The interface of the subroutine is the following.

```
WriteErr_DGV (time_a,ndens_a,ubar_a,vbar_a,wbar_a,tempr_a)

real (DP), dimension (:), intent (in) time_a,ndens_a,ubar_a,vbar_a,wbar_a,tempr_a
```

Here is a brief description of the variables.

- `time_a, ndens_a, u/v/wbar_a, tempr_a` are arrays of type `DP` that contain the time when the errors are recorded and the errors themselves.

B.10.7.10 ReadAarraysDGV

Subroutine `ReadAarraysDGV` reads the A_arrays from the disk.

B.10.7.11 ReadAarraysDGVII

Subroutine `ReadAarraysDGVII` reads the A_arrays for secondary velocity mesh from the disk. This is a copy of the above subroutine `ReadAarraysDGV`, with the only change that it works with secondary meshes, which is accomplished by re-naming variables in the `USE commvar, only:` statement. The suffix II in the name suggests that the subroutine works with the secondary mesh.

B.10.7.12 ReadAKorArraysOneNet_DGV

Subroutine `ReadAKorArraysOneNet_DGV` will read *Akor* arrays for a single net from hard drive. Here the chunks of *Akor* arrays will be read and pieced together in the sequence imbedded in the file names. The information stored in the chunks will be in the right order. Once the first chunk is read into the *Akor* arrays we proceed on reading the rest of the chunks. *do* $j = 2, numchks$ array will loop until all chunks are read in a given net.

The interface of the subroutine is the following.

```
ReadAKorArraysOneNet_DGV (netnum, numchks)

integer (I4B), intent (in) :: netnum
integer (I4B), intent (in) :: numchks
```

Here is a brief description of the variable.

- `netnum` is a number of type `INTEGER` that gives the number of the net from which the *Akor* is read.
- `numchks` is a number of type `INTEGER` which is the number of chunk in which *Akor*_array is divided for the given Korobov net.

B.10.7.13 ReadAKorArraysOneNet_DGVII

Subroutine `ReadAKorArraysOneNet_DGVII` will read `Akor` arrays for a single net from hard drive.

This is a copy of the above subroutine `ReadAKorArraysOneNet_DGV`, with the only change that it works with secondary meshes, which is accomplished by re-naming variables in the `USE commvar, only:` statement. The suffix II in the name suggests that the subroutine works with the secondary mesh.

B.10.7.14 ReadAKorArraysAllNets_DGV

Subroutine `ReadAKorArraysAllNets_DGV` loops through all the nets and reads *Akor* arrays for all nets from the hard drive, where each net is read separately in the subroutine `ReadAKorArraysOneNet_DGV`.

B.10.7.15 MakeBaseNameDGV

Subroutine `MakeBaseNameDGV` creates a recognizable name for the file. It contains values of important parameters of the DG discretization, therefore lots of parameter values is cited in the file name. The subroutine sets up the basic file name that is common for the numerical simulation and is used by many output procedures.

The interface of the subroutine is the following.

```
MakeBaseNameDGV (file_name)
character (len=132), intent (out) :: file_name
```

Here is a brief description of the variable.

- `file_name` is the variable of type `CHARACTER` that stores the file name.

B.10.7.16 MakeBaseNameAoperDGV

Subroutine `MakeBaseNameAoperDGV` creates a recognizable name for the file to store A-operator in one file. It contains values of important parameters of the DG discretization,

therefore lot of parameter values is cited in the file name. The subroutine sets up the basic file name that is common for the experiment and is used by many output procedures.

The interface of the subroutine is the following.

```
MakeBaseNameAoperDGV (file_name)
character (len=132), intent (out) :: file_name
```

Here is a brief description of the variable.

- `file_name` is the variable of type `CHARACTER` that stores the file name.

B.10.7.17 MakeBaseNameAoperDGVII

Subroutine `MakeBaseNameAoperDGVII` creates a recognizable name for the file to store operator A in one file. It contains values of important parameters of the DG discretization, therefore lot of parameter values is cited in the file name. The subroutine sets up the basic file name that is common for the experiment and is used by many output procedures.

This is a copy of the above subroutine `MakeBaseNameAoperDGV`, with the only change that it works with secondary meshes, which is accomplished by re-naming variables in the `USE commvar, only:` statement. The suffix II in the name suggests that the subroutine works with the secondary mesh.

B.10.7.18 MakeBaseNameAoperChnksDGV

Subroutine `MakeBaseNameAoperChnksDGV` creates a recognizable name for the files to store chunks of operator A. It is used when the data is written in several chunks. The file name is formed by adding the chunk number $00X$ to the base name.

The interface of the subroutine is the following.

```
MakeBaseNameAoperChnksDGV (file_name,i)
integer (I4B) :: i
character (len=132), intent (out) :: file_name
```

Here is a brief description of the variable.

- `i` is a number of type `INTEGER` that represents the index of the chunk (starts from zero).
- `file_name` is the variable of type `CHARACTER` that stores the file name.

B.10.7.19 MakeBaseNameAoperChnksDGVII

Subroutine `MakeBaseNameAoperChnksDGVII` creates a recognizable name for the files to store chunks of operator A. It is used when the data is written in several chunks. The file name is formed by adding the chunk number $00X$ to the base name.

This is a copy of the above subroutine `MakeBaseNameAoperChnksDGV`, with the only change that it works with secondary meshes, which is accomplished by re-naming variables in the `USE commvar, only:` statement. The suffix II in the name suggests that the subroutine works with the secondary mesh.

C. REFERENCES

[1] How to call a fortran90 function included in a module in c++ code? http://stackoverflow.com/questions/15557439/how-to-call-a-fortran90-function-included-in-a-module-in-c-code, March 2013. Stakoverflooooooow Blog, Stack Exchange Q&A Communities.

[2] A. Alekseenko and E. Josyula. Deterministic solution of the Equation using a discontinuous Galerkin velocity discretization. In *28th International Symposium on Rarefied Gas Dynamics, 9-13 July 2012, Zaragoza, Spain*, AIP Conference Proceedings, pages 279–286. American Institute of Physics, 2012.

[3] A. Alekseenko and E. Josyula. Deterministic solution of the spatially homogeneous boltzmann equation using discontinuous galerkin discretizations in the velocity space. *Journal of Computational Physics*, 272(0):170 – 188, 2014.

[4] Alexander Alekseenko. Interim Technical Report for the Project PP-SAS-KY06-001-P3, november 2014, 2014.

[5] Seymour Haber. Parameters for integrating periodic functions of several variables. *Mathematics of Computation*, 41(163):pp. 115–129, 1983.

[6] Korobov. *Exponential sums and their applications*. Mathematics and its applications. Soviet series; V. 80. Kluwer Academic Publishers, 1992.

Table 1. Values of the parameters p and $a_1, \ldots, a_6$ in Korobov quadrature nets [5]

p	b or a_1	a_2	a_3	a_4	a_5	a_6	Error
256	123	25	3	113	75	9	3.9E-3
1,536	341	1,081	1,517	1,201	965	361	6.2E-4
2,048	443	1,689	707	1,905	139	137	4.6E-4
3,072	1,095	945	2,583	2,145	1,767	2,577	9.5E-4
4,096	1,271	1,617	3,111	1,441	599	3,569	2.1E-4
6,144	1,477	409	1,981	1,393	5,365	4,489	1.4E-4
8,192	67	4489	1655	3219	4389	7343	1.1E-4
12,288	5,685	729	869	785	1,093	393	1.4E-4
24,576	3771	15513	8643	4977	16779	14985	1.0E-4
32,769	4 335	16 161	32 719	16 961	27 311	2 401	1.0E-4
49,152	1,509	16,089	46,365	21,489	35,733	1,353	4.5E-4
5,536	24,565	49,273	6,861	47,409	27,365	18,673	5.2E-6
98,304	40,709	13,849	5,501	3,697	96,063	81,673	3.1E-6
131,072	33,269	54,393	20,685	41,265	128,229	50,217	1.5E-6
753,456	15,723	79,161	692,547	723,825	501,051	642,393	3.8E-7
196,608	5,209	1,777	1,5817	12,001	188,473	92,113	1.1E-5
262,144	3,851	150,137	150,067	142,641	118,811	99,881	1.6E-6
393,216	4,513	313,153	41,185	26,953	114,721	263,617	5.1E-6
24,288	67,469	444,225	434,401	138,881	245,793	394,689	1.2E-6
786,432	8,629	535,033	443,917	635,953	702,373	531,625	1.6E-6
1,048,576	9,199	735521	645327	374,337	2,479	784,225	4.8E-8
1,572,864	9,781	1,296,121	75,661	794,161	886,309	934,825	2.0E-8
2,097,152	11,027	2,057,065	459,723	549,137	855,875	549,625	8.8E-10
3,145,728	12,641	2,508,481	770,081	1,711,489	1,760,993	1,541,185	3.9E-6
4,194,304	11,587	136,841	1,263,755	2,904,313	294,915	1,267,345	5.0E-8
6,291,456	8,969	4,945,489	1,326,041	2,409,889	3,143,081	4,570,609	8.0E-10
8,388,608	13,997	2,978,025	422,773	3,585,041	7,554,429	938,873	4.2E-8
12,582,912	14,639	390,817	8,528,015	6,541,633	7,005,167	10,489,825	2.8E-12
16,777,216	15,241	14,184,273	8,076,633	1,529,761	11,534,377	3,770,609	2.0E-10
25,165,824	16,981	11,530,297	5,862,637	22,604,977	800,965	1,164,1705	3.9E-17
33,554,423	40,709	13,055,954	26,325,489	23,169,927	9,727,713	29,722,694	6.6E-16
50,331,648	19,211	16,740,985	42,163,763	20,839,729	14,105,627	47,939,113	5.1E-11
67,108,864	63,727	34,598,689	8,927,183	20,750,913	13,267,631	1,743,201	1.0E-15
100,663,296	32,119	24,997,201	61,758,887	29,889,441	26,655,959	54,969,073	1.5E-15