Chapter 6
Direct Simulations Monte Carlo (DSMC) method

6.1. Basic concepts of the Direct Simulation Monte Carlo method

- Definition and major concepts of the DSMC method
- Random state variables of simulated particles
- Statistical weight
- Time discretization in the DSMC method. Time-splitting technique
- Spatial discretization in the DSMC method. Sampling and indexing

6.1. Basic concepts of the Direct Simulation Monte Carlo method

Definition and major concepts of the DSMC method

Direct simulation Monte Carlo (DSMC) method is the stochastic Monte Carlo method for simulation of dilute gas flows on the molecular level, i.e. on the level of individual molecules. To date, the DSMC method is the state-of-the-art numerical tool for the majority of applications in the kinetic theory of gases and rarefied gas dynamics.

The DSMC method is based on the following main ideas:

- The gas flow is represented by a set of **simulated particles**. Every simulated particle is considered as a *representative of real molecules* in the gas flow.
- Current state of every simulated particle is given by a set of **state variables** coinciding with phase coordinates of a molecule in the considered problem. In accordance with the general approach of the kinetic theory, these *state variables are considered as random variables with PDFs given by the solution of the Boltzmann kinetic equation.*
- The number and properties of simulated particles are not identical with the number and properties of real gas molecules. The relationship between parameters of real and simulated particles are established based on the analysis of *similarity* of gas flows described by the Boltzmann equation.
- Any dynamical process in a gas is considered as variation of state variables of individual particles due to collisions, free motion, and interaction with boundaries. The DSMC imitates this process in accordance with the Boltzmann equation and kinetic boundary conditions.
- Macroscopic gas parameters are calculated as means of corresponding random molecular quantities using *standard Monte Carlo method for calculation of integrals.*

6.1. Basic concepts of the Direct Simulation Monte Carlo method

In summary, the DSMC method is a particle-based method, where

- Gas is represented by a set of particles;
- Individual dynamic parameters of particles are random variables;
- Concept of statistical weight is used to imitate huge number of real molecules with small number of simulated particles;
- Variation of dynamical parameters is described by a random (stochastic) process which is designed based (derived from) the Boltzmann kinetic equation and kinetic boundary conditions;
- Macroscopic gas parameters are calculated as statistical means of corresponding random molecular quantities using the Monte Carlo method.

6.1. Basic concepts of the Direct Simulation Monte Carlo method

Random state variables of simulated particles

In **3D flows**, the current state of every molecule i of a simple gas can be *completely* characterized by its Cartesian coordinates x_i, y_i, z_i and velocity components v_{ix}, v_{iy}, v_{iz}, i.e. by 6 phase coordinates:

$$\mathbf{X} = (X_1, X_2, \dots, X_6) = \left(x, y, z, v_x, v_y, v_z\right).$$

In **2D flows** (distribution function does not depend on z, $f = f\left(x, y, v_x, v_y, v_z, t\right)$), the current state of every molecule i of a simple gas can be *completely* characterized by its Cartesian coordinates x_i, y_i and velocity components v_{ix}, v_{iy}, v_{iz}, i.e. by 5 phase coordinates

$$\mathbf{X} = (X_1, X_2, \dots, X_5) = \left(x, y, v_x, v_y, v_z\right).$$

In **1D flows** (distribution function does not depend on y and z, $f = f\left(x, v_x, v_y, v_z, t\right)$), the current state of every molecule i of a simple gas can be *completely* characterized by its Cartesian coordinate x_i and velocity components v_{ix}, v_{iy}, v_{iz}, i.e. by 4 phase coordinates

$$\mathbf{X} = (X_1, X_2, \dots, X_4) = \left(x, v_x, v_y, v_z\right).$$

In **spatially homogeneous problems** ($f = f\left(v_x, v_y, v_z, t\right)$), the current state of every molecule i of a simple gas can be *completely* characterized by its velocity components v_{ix}, v_{iy}, v_{iz}, i.e. by 3 phase coordinates

$$\mathbf{X} = (X_1, X_2, X_3) = \left(v_x, v_y, v_z\right).$$

In general, the set of phase coordinates of every particle can include additional variables, charactering its internal state, e.g., rotational and vibrational energy of the particle.

6.1. Basic concepts of the Direct Simulation Monte Carlo method

In DSMC, the current state at time t of a gas composed of N simulated particles, is characterized by the vector of state variables

$$\mathbf{Y} = (\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, \dots, \mathbf{X}_i, \dots, \mathbf{X}_N).$$

All these variables are considered as random variables.

Any dynamical process (flow) in a gas is considered as variation in time of random phase coordinates of individual simulated molecules (random process),:

$$[N, \mathbf{Y}] = [N(t), \mathbf{Y}(t)].$$

Initial values of the state variables at time $t = 0$ are chosen by sampling individual phase coordinates in accordance with the initial conditions for the Boltzmann equation in the considered problem.

In a computational code, it is convenient to introduce a specific structure PCL containing state variables (phase coordinates) of an individual simulated particle, e.g.

```
#define DIM      2                 // Spatial dimension of the problem
#define MAX_PCL  1000000           // Maximum number of particles in the domain

typedef struct pcl {
          double  X[DIM];
          double  V[3];
} PCL;

// Global variables defining the current state of the simulation process

int     NP;                       // Current number of simulated particles in the domain = N
PCL     P[MAX_PCL];               // Array of simulated particles = Y
```

6.1. Basic concepts of the Direct Simulation Monte Carlo method

Statistical weight

The concept of a statistical weight is the most important concept of the DSMC method. It allows one to perform simulations of gas flows using the number of simulated particles which is much smaller than the number of molecules in real gas flows.

The purpose of the DSMC is to simulate (imitate) a gas flow in accordance with the Boltzmann kinetic equation. Correspondingly, the concept of the statistical weight is based on the analysis of similarity of gas flows described by the Boltzmann equation.

Two flows (physical phenomena) are called **similar** if numerical properties of such flows are identical in reduced units. In application to the rarefied gas flows, it means that two similar flow correspond to the same solution of the Boltzmann equation in reduced units.

The Boltzmann equation in reduced units

$$\bar{\mathbf{r}} = \frac{\mathbf{r}}{L_*}, \qquad \bar{t} = \frac{t}{t_*}, \qquad \bar{\mathbf{v}} = \frac{\mathbf{v}}{v_*}, \qquad \bar{c}_r = \frac{c_r}{v_*}, \qquad \bar{\sigma} = \frac{\sigma}{\pi d_{Ref}^2}, \qquad \bar{\mathbf{F}} = \frac{\mathbf{F}}{F_*}, \qquad \bar{f} = \frac{f}{n_*/v_*^3}$$

can be written as follows (see Eq. (3.9.9))

$$Sh\frac{\partial \bar{f}}{\partial \bar{t}} + \bar{\mathbf{v}} \cdot \frac{\partial \bar{f}}{\partial \bar{\mathbf{r}}} + \frac{1}{Fr}\bar{\mathbf{F}} \cdot \frac{\partial \bar{f}}{\partial \bar{\mathbf{v}}} = \frac{1}{Kn}\int \int_0^{2\pi} \int_0^{\pi} \left(\bar{f}'\bar{f}'_1 - \bar{f}\bar{f}_1\right)\bar{c}_r\bar{\sigma}\,(\bar{c}_r, \chi) \sin\chi \, d\chi d\varepsilon d\bar{\mathbf{v}}_1 \,.$$

The solutions of this equation is not defined by individual dimensional values like L_*, and n_*, but depends on

$$Sh = \frac{L_*}{t_* v_*}, \qquad Fr = \frac{m v_*^2}{F_* L_*}, \qquad Kn = \frac{\lambda_*}{L_*} = \frac{1}{\pi d_{Ref}^2 n_* L_*}, \qquad \lambda_* = \frac{1}{\pi d_{Ref}^2 n_*}.$$

6.1. Basic concepts of the Direct Simulation Monte Carlo method

In other words, two solutions are similar if they correspond to the same Sh, Fr, and Kn. These non-dimensional numbers (parameters) are called the **criteria of similarity**.

Note: Sh, Fr, and Kn do not compose the *full* list of criteria of similarity of a gas flow. Other criteria of similarity, e.g. T_r/T_∞, are introduced by the kinetic boundary conditions.

As on can see, there is only one criterion of similarity, the Knudsen number, that is defined by the number density of molecules and molecule cross section. Thus, one can simulate a real gas flow with characteristic number density $n_{*(r)}$ composed of molecules with characteristic cross section $\sigma_{*(r)} = \pi d_{Ref}^2$ by a flow with another characteristic number density $n_{*(r)}$ and composed of simulated molecules of another size $\sigma_{*(s)}$. The real and simulated flows are similar, if the Knudsen number is the same in both flows:

(6.1.1) $$Kn = \frac{1}{\sigma_{*(r)} n_{*(r)} L_*} = \frac{1}{\sigma_{*(s)} n_{*(s)} L_*}.$$

In the DSMC method, the ratio

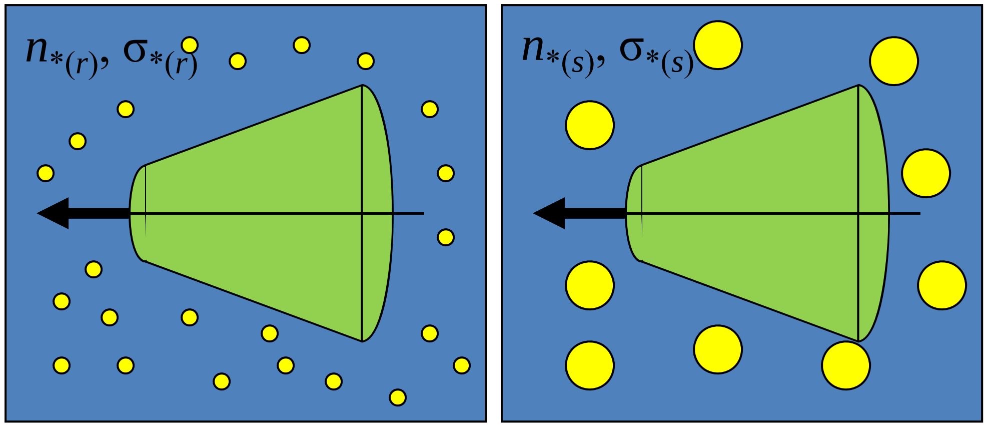
(6.1.2) $$W = \frac{n_{*(r)}}{n_{*(s)}}$$

is called the **statistical weight** of a simulated particle. Practically, *one can say that every simulated particle represents W particles in the real gas flow*. Then according to Eq. (6.1.1.), for similarity of the flow of simulated particles to the real gas flow, in all calculations of collisions between simulated particles, the cross-section of simulated particles must be equal to

(6.1.3) $$\sigma_{*(s)} = W \sigma_{*(r)}.$$

6.1. Basic concepts of the Direct Simulation Monte Carlo method

- Number of collisions between molecules is defined by the collision frequency $z \approx n\sigma C$.
- For the same velocities of gas molecules, the number of collisions depends on n and σ.
- Consider two flows



If $n_{*(r)}\, \sigma_{*(r)} = n_{*(s)}\, \sigma_{*(s)}$, then the collision frequencies are the same in both flows.

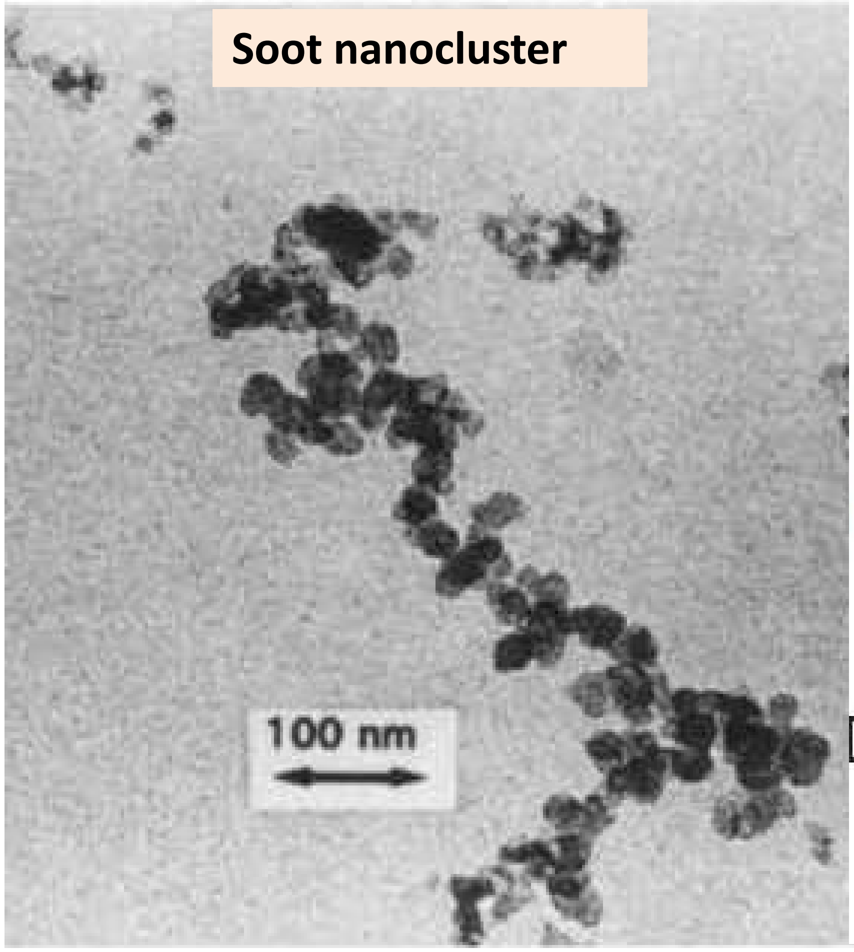
If other conditions in both flows are the same, then two flows are equivalent to each other.

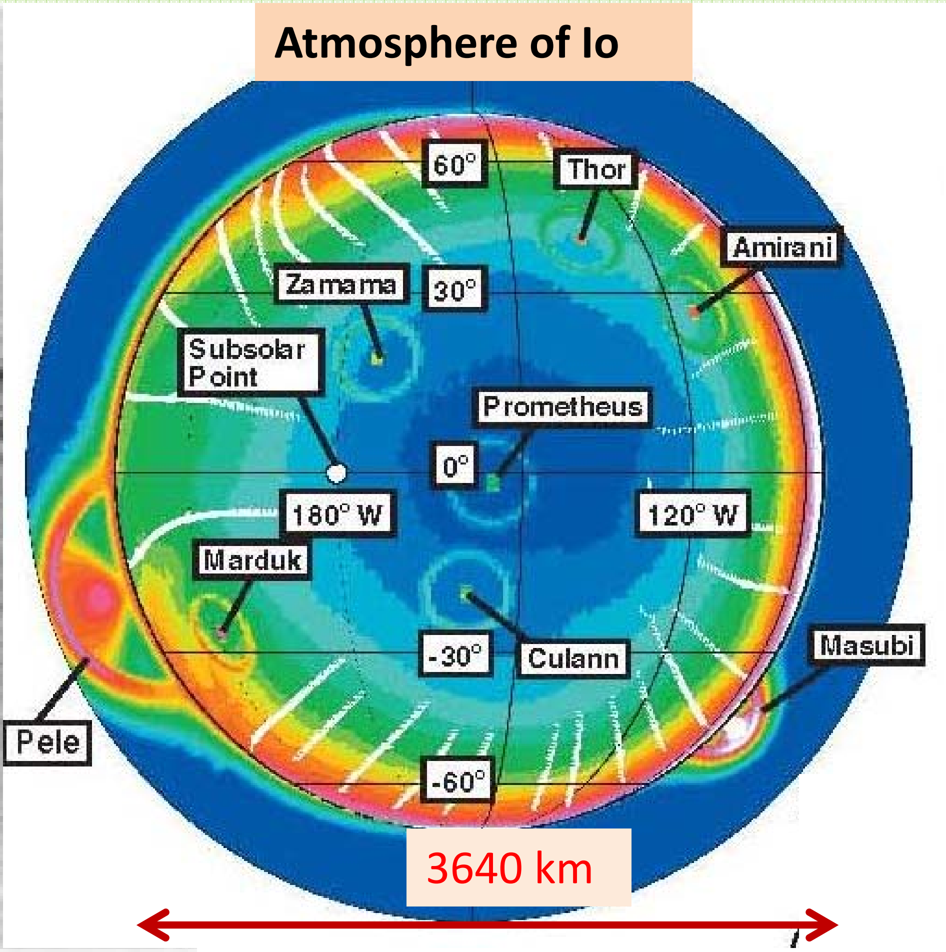
- We can simulate a huge number of small particles with small number of huge particles!
- In DSMC simulations the number of simulated molecules may be not equal to the number of molecules in real flow. This makes DSMC different from MD, where every simulated particle represents one molecule of the real system.
- By enforcing Eq. (6.1.1) is the flow of simulated particles we guarantee the same collision frequency and the same degree of rarefaction in both real and simulated gas flows.
- The statistical weight is an important **numerical parameter** of the DSMC method. Value of W affects the accuracy of simulations.

6.1. Basic concepts of the Direct Simulation Monte Carlo method

The concept of the statistical weight is extremely important for the DMSC method. It makes possible to consider problems in both micro- and planetary scales.

By proper choice of the statistical weight, we can described both problems with the same number of simulated particles.





6.1. Basic concepts of the Direct Simulation Monte Carlo method

Time discretization in the DSMC method. Time-splitting technique

We will consider an implementation of the DSMC method for two-dimensional **steady-state** problems, i.e. problems when the distribution function of gas molecules and macroscopic gas parameters are assumed to be functions of two spatial coordinates and *do not depend on time*.

The DSMC method is an inherently unsteady method, since state variables of individual molecules always vary with time in the course of their chaotic motion.

In order to describe variation of particle parameters in time, the DSMC employs an approach similar to the approach for numerical solution of ordinary differential equations: The time of the process is discretized into short intervals of duration Δt, which is called the **time step**. The parameters of simulated molecules are defined only after end of every time step, i.e. for times

$$t^n = t^{n-1} + \Delta t, \qquad t^0 = 0, \qquad n = 1,2,\ldots$$

Here n is the **number (index) of the time step**. At time t^n, the state of the simulated system of particles is defined by parameters

$$[N^n, \mathbf{Y}^n] = [N(t^n), \mathbf{Y}(t^n)].$$

It is assumed that $[N^{n+1}, \mathbf{Y}^{n+1}]$ depends only on $[N^n, \mathbf{Y}^n]$ and the whole simulation process then looks like an iterative update of vector $[N^n, \mathbf{Y}^n]$ from step to step:

$$\begin{tikzcd}
{[N^0, \mathbf{Y}^0]} \arrow[r] & {[N^1, \mathbf{Y}^1]} \arrow[r] & \ldots \arrow[r] & {[N^{n-1}, \mathbf{Y}^{n-1}]} \arrow[r] & {[N^n, \mathbf{Y}^n]} \arrow[r] & {[N^{n+1}, \mathbf{Y}^{n+1}]} \arrow[r] & {}
\end{tikzcd}$$

Initial state
(*initial condition*)

One time step of the DSMC method

6.1. Basic concepts of the Direct Simulation Monte Carlo method

In computational codes, such iterative update of $[N^n, \mathbf{Y}^n]$ is convenient to organize in a form of a loop called the **time loop** of the DSMC method.

During a time step, the DSMC method employs the **time-splitting technique**, when variation of dynamic variables from $[N^{n-1}, \mathbf{Y}^{n-1}]$ to $[N^n, \mathbf{Y}^n]$ due to different physical processes is considered *sequentially*. Then one time step of the DSMC algorithm includes three major substeps:

Motion substep:	Collision-free motion of simulated particles under the effect of external forces (if any) during Δt.
Boundary substep:	Interaction of simulated particles with interphase boundaries and generation of new simulated particles (e.g., evaporation), i.e. implementation of *boundary conditions*.
Collision substep:	Binary collisions between simulated particles.

Then the sequence of calculations during a time step is as follows:

$$\begin{tikzcd}[N^{n-1}, \mathbf{Y}^{n-1}] \arrow[r, "\text{Free motion}"'] & [N^*, \mathbf{Y}^*] \arrow[r, "\text{Boundary conditions}"'] & [N^n, \mathbf{Y}^{**}] \arrow[r, "\text{Collisions}"'] & [N^n, \mathbf{Y}^n]\end{tikzcd}$$

Time step Δt is important **numerical parameter** of the DSMC method. Value of Δt affects the accuracy of simulations.

6.1. Basic concepts of the Direct Simulation Monte Carlo method

Spatial discretization in the DSMC method. Sampling and indexing

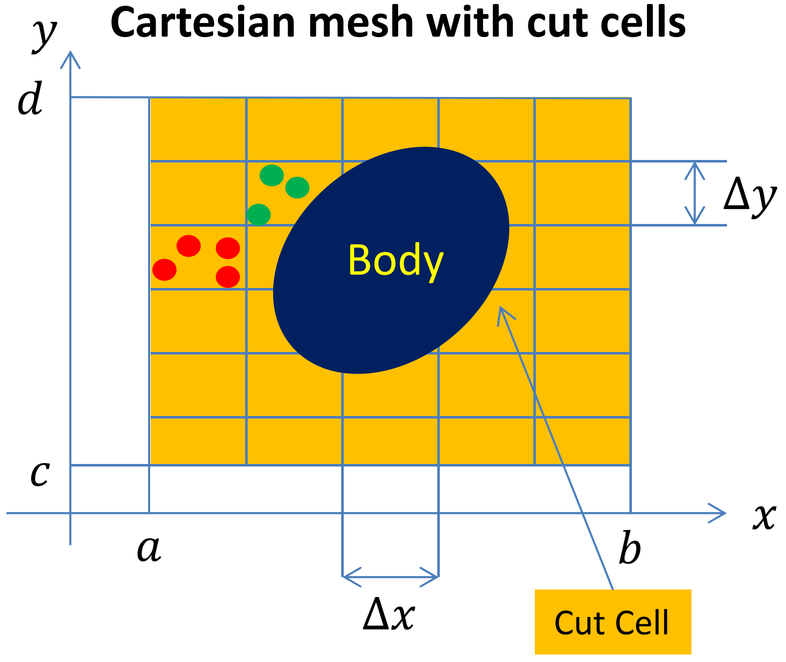
In DSMC simulations, the whole flow domain is discretized into a mesh (grid) of cells. This spatial discretization into cells is used for two purposes:

- **Collision sampling**: At every time step, binary collisions between cells are sampled (drawn) *only for particles within a cell*. No collisions between particles in different cells are accounted for.
- **Macroscopic gas parameters sampling**: Gas macroparameters in the cell center are defined as means of molecular quantities of simulated particles averaged over the cell volume .

For both purposes, at every time step of the DSMC method, it is necessary to define lists of particles belonging to every cell of the computational mesh. Usually a cell of the computational mesh is identified by one or few integer indices. The process of calculations of the cell index for every simulated particle is called the particle **indexing**.

The actual implementation of indexing depends on the approach used in order to define individual cells of the computational mesh. The DSMC method is very flexible and can be used with structured and unstructured meshes, including simple **Cartesian meshes** with **cut cells**.

Cartesian mesh with cut cells



6.1. Basic concepts of the Direct Simulation Monte Carlo method

Sampling of binary collisions and macroscopic gas parameters will be considered later on in Sections 6.3 and 6.6. Here we make only a few notes:

- Sampling of collisions is required at every time step during the whole simulation process.
- Sampling of macroscopic gas parameters usually does not affect current state of the simulated molecules $[N^n, \mathbf{Y}^n]$. Steady-state problems are usually solved by the DSMC method using the **time convergence approach**, when the steady state is approached gradually in the course of simulation started from *arbitrary* initial condition. In steady-state problems, sampling of macroscopic gas parameters is usually required only when the steady-state is achieved. The whole simulation process in this case consists of two stages:
  - **Transient stage**: Simulation of a transient process from arbitrary initial distribution of simulated particles to the steady-state during some time t_{SS}. Sampling of macroscopic gas parameters is not performed during this stage.
  - **Steady-state stage**: Simulations of the steady-state process at $t > t_{SS}$ in order to sample enough simulated molecules in every cell and calculate macroscopic gas parameters. The duration of this stage, $t_P - t_{SS}$, (here t_P is the total process time) must be chosen sufficiently large in order to ensure calculation macroscopic gas parameters in every cell of the computational mesh with small dispersion

The sizes of cells of the computational mesh are important **numerical parameters** of the DSMC method. Values of cell sizes affect the accuracy of simulations.

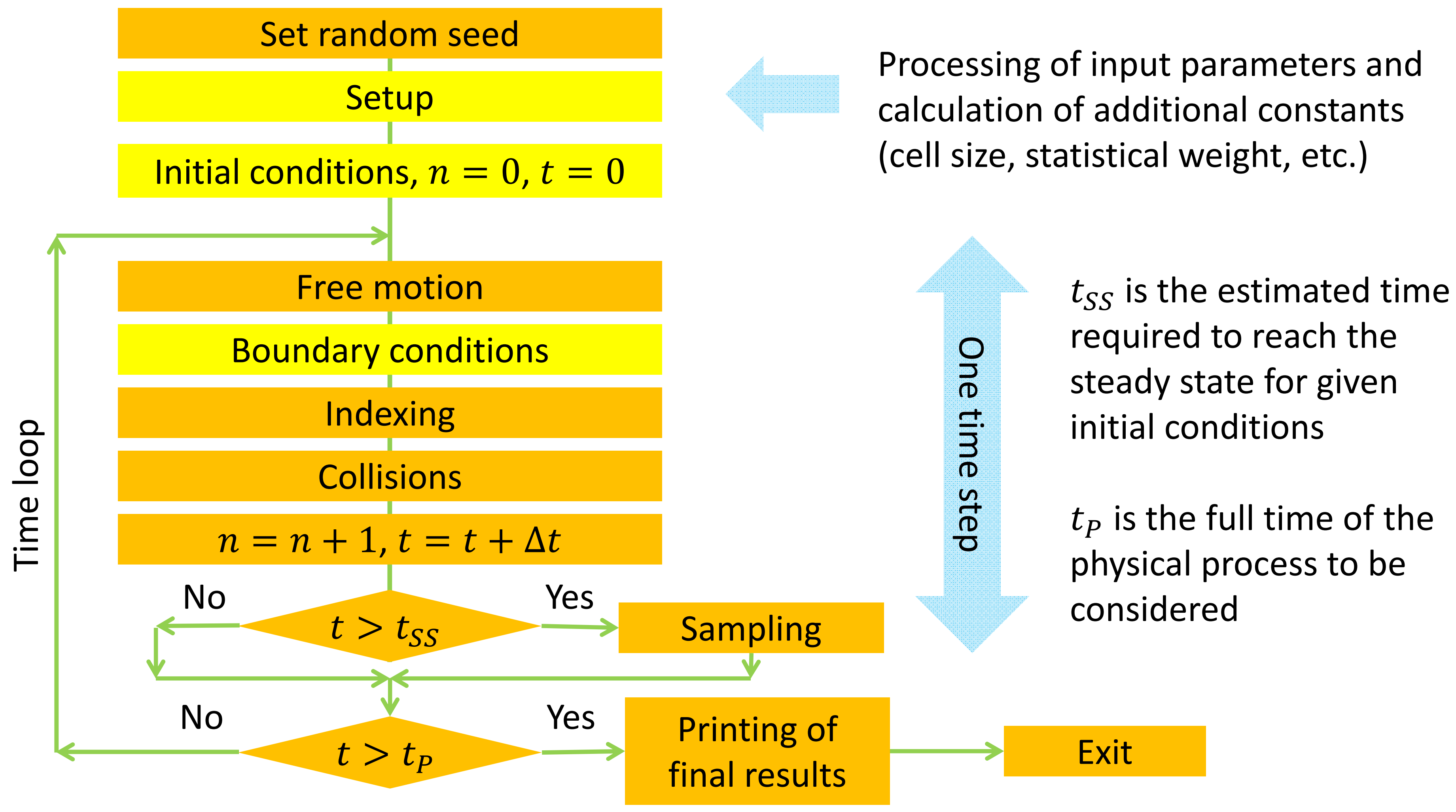
6.2. Skeleton of a DSMC-base code simulations of two-dimensional flows

- Flowchart of the DSMC algorithm
- Implementation of the DMSC time loop in a C++ code

6.2. Skeleton of a DSMC-base code simulations of two-dimensional flows

Flowchart of the DSMC algorithm

Based on the concepts considered in Section 6.1, the following flowchart of an "idealized" DSMC algorithm is used for the code development:



Set random seed
Setup
Initial conditions, $n = 0$, $t = 0$
Free motion
Boundary conditions
Indexing
Collisions
$n = n + 1$, $t = t + \Delta t$
Time loop
No
$t > t_{SS}$
Yes
Sampling
No
$t > t_P$
Yes
Printing of final results
Exit
Processing of input parameters and calculation of additional constants (cell size, statistical weight, etc.)
One time step
t_{SS} is the estimated time required to reach the steady state for given initial conditions
t_P is the full time of the physical process to be considered

6.2. Skeleton of a DSMC-base code simulations of two-dimensional flows

Implementation of the DMSC time loop in a C++ code (see DSMC2D_Template02.cpp)

```
double  Dt;                     // Time step (s)
int     NStep;                  // Total number of time steps
int     FirstSamplingStep;      // Number of the first sampling step
int     SamplingPeriod;         // Period between sampling steps
int     PrintPeriod;            // Period between printing of results
int     Step;                   // Current step number
double  Time;                   // Current time (s)

int main ( int argc, char **argv ) /////////////////////////////////////////////////////////
{
        // Set the initial seed for pseudo-random number generators
time_t  t;
        SetSeed ( unsigned ( time ( &t ) ), unsigned ( 362436069 ) );
        // Setup of the computational algorithm: Calculation of all constants
        Setup ();
        // Set initial distribution of particles in the domain
        InitialConditions ();
        // DSMC time loop
        do {
                MoveParticles ();
                BoundaryConditions ();
                Indexing ();
                CollideParticles ();
                if ( Step > FirstSamplingStep && Step % SamplingPeriod == 0 ) Sampling ();
                Step++;
                Time += Dt;
                if ( Step > FirstSamplingStep && Step % PrintPeriod == 0 ) Printing ();
        } while ( Step < NStep );
        // Printing the final results
        Printing ();
} ////////////////////////////////////////////////////////////////////////////////////////
```

6.3. Particle motion, indexing, and sampling of macroscopic gas parameters

- Particle motion in the DSMC method
- Implementation of particle motion given in a C++ code
- Indexing of particles in the DSMC method
- Implementation of particle indexing given in a C++ code
- Sampling of macroscopic gas parameters in the DSMC method
- Implementation of sampling of macroscopic gas parameters in a C++ code

6.3. Particle motion, indexing, and sampling of macroscopic gas parameters

Particle motion in the DSMC method

Free (collisionless) motion of molecules is implemented in the DSMC method in accordance with the Boltzmann kinetic equation. In this equation, the variation of the distribution function due to motion of molecules is described by the convective term in the left-hand side of this equation (see Section 3.3). This convective term is defined assuming that variation of phase coordinates of every particle i between collisions is determined by the equations of motion

(6.3.1) $$\frac{d\mathbf{r}_i}{dt} = \mathbf{v}_i, \qquad m\frac{d\mathbf{v}_i}{dt} = \mathbf{F}_i,$$

where $\mathbf{F}_i$ is the external force exerted on particle i.
During the motion substep (see slide 11),

$$\left[N^{n-1}, \mathbf{Y}^{n-1}\right] \longrightarrow \left[N^{n-1}, \mathbf{Y}^{*}\right]$$

Eqs. (6.2.1) are solved numerically for a time step Δt, usually with the **Runge-Kutta methods**, e.g., of the second order.
The force $\mathbf{F}_i$ can be a gravity force, which is usually important only for planetary science/astrophysical applications, or electromagnetic force exerted on charged particles in plasma. Thus, in the majority of applications of the DSMC for neutral gas flows, $\mathbf{F}_i = 0$ and then the accurate solution of Eqs. (6.2.1) during a time step can be written in the form

(6.3.2) $$\mathbf{r}_i^* = \mathbf{r}_i^{n-1} + \Delta t \mathbf{v}_i^{n-1}, \qquad \mathbf{v}_i^* = \mathbf{v}_i^{n-1}.$$

6.3. Particle motion, indexing, and sampling of macroscopic gas parameters

Implementation of particle motion given in a C++ code

(see DSMC2D_Template03.cpp)

```
#define DIM             2         // Spatial dimension of the problem
#define MAX_PCL         1000000   // Maximum number of particles in the domain

// This structure contains dynamic state variables for an individual simulated particle
typedef struct pcl {
          double  X[DIM];         // Cartesian coordinates
          double  V[3];           // Components of the velocity vector
} PCL;

double    Dt;                     // Time step (s)
int       NP;                     // Current number of particles in the domain
PCL       P[MAX_PCL];             // Array of simulated particles

void MoveParticles () ///////////////////////////////////////////////////////////////////////
{
          for ( int i = 0; i < NP; i++ )
                    for ( int m = 0; m < DIM; m++ ) P[i].X[m] += Dt * P[i].V[m];
}
```

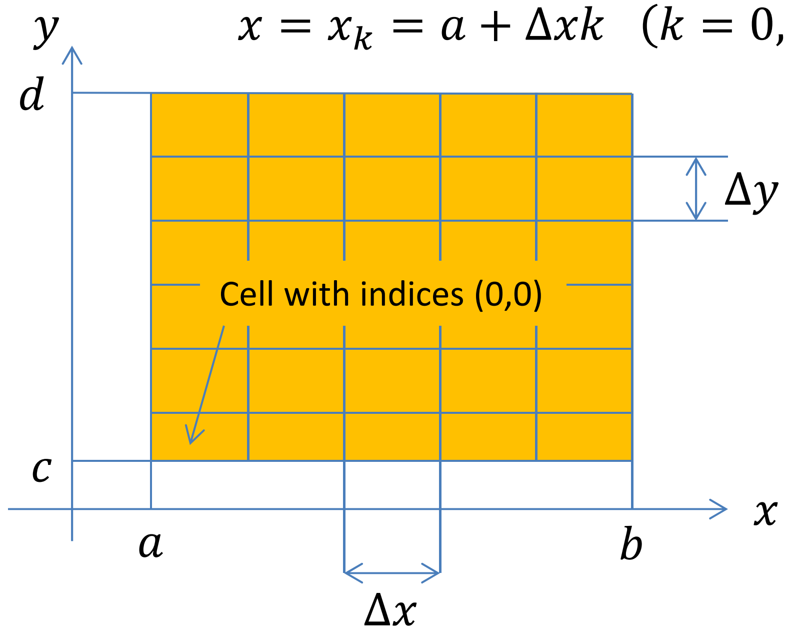
6.3. Particle motion, indexing, and sampling of macroscopic gas parameters

Indexing of particles in the DSMC method

The approach for particle indexing in DSMC depends on the type of computational mesh/grid. For structured and unstructured grids, the approaches are different. We consider only the simplest case of structured Cartesian mesh for 2D flows.

Let's assume that the flow domain is a rectangle of size $[\,a, b\,] \times [\,c, d\,]$ and we introduce in this rectangle a mesh of cells of constant sizes Δx and Δy by lines

$$x = x_k = a + \Delta x k \quad (k = 0,1,\dots), \qquad y = y_l = c + \Delta y l \quad (l = 0,1,\dots)$$



Indexing implies that we define indices of a cell (k_i, l_i), where particle every particle is located. These indices can be calculated based on current coordinates of particle i as follows

$$k_i = \left[\frac{x_i - a}{\Delta x}\right], \qquad l_i = \left[\frac{y_i - c}{\Delta y}\right], \qquad (6.3.3)$$

where $[z]$ means the integer part of z.

In the DSMC method, in every cell of the computational mesh, a special data structure (**particle list**) is introduced in order to contain information about indices of all particles, which belong to the this cell. These particle lists are updated at every time step.

6.3. Particle motion, indexing, and sampling of macroscopic gas parameters

Implementation of particle indexing given in a C++ code

(see DSMC2D_Template04.cpp)

```
#define MAX_PCL_IN_CELL 1000            // Maximum number of particles in a cell
#define MAX_CELL_X      200             // Maximum number of cells along X axis
#define MAX_CELL_Y      200             // Maximum number of cells along Y axis

double  X1, X2, Y1, Y2;                 // Domain size (m)
int     NX, NY;                         // Number of cells along X and Y axes
double  DX, DY;                         // Cell sizes (m)

// Lists of particles in cells of the computational mesh
int     NPC[MAX_CELL_X][MAX_CELL_Y];                  // Numbers of particles in cells
int     IPC[MAX_CELL_X][MAX_CELL_Y][MAX_PCL_IN_CELL]; // Indices of particles in cells

void Setup () ////////////////////////////////////////////////////////////////////
{
        DX = ( X2 - X1 ) / NX;
        DY = ( Y2 - Y1 ) / NY;
}

void Indexing () /////////////////////////////////////////////////////////////////
{
        memset ( NPC, 0, sizeof NPC ); // Set initial number of particles in every cell to zero
        for ( int i = 0; i < NP; i++ ) {// Distribute particles between cells
                // Calculate indices of the cell
                int k = int ( ( P[i].X[0] - X1 ) / DX );
                int l = int ( ( P[i].X[1] - Y1 ) / DY );
                IPC[k][l][NPC[k][l]++] = i; // Add particle to the list of particles in the cell
        }
}
```

6.3. Particle motion, indexing, and sampling of macroscopic gas parameters

Sampling of macroscopic gas parameters in the DSMC method

In the kinetic theory, all macroscopic gas parameters can be calculated in the form of averaged molecular quantities, i.e. integrals of the distribution function $f(\mathbf{r},\mathbf{v},t)$, Eq. (3.3.2):

(6.3.4) $$\langle\Phi\rangle(\mathbf{r},t) = \frac{1}{n(\mathbf{r},t)}\int \Phi(\mathbf{r},\mathbf{v},t) f(\mathbf{r},\mathbf{v},t) d\mathbf{v}\,.$$

The most important macroscopic parameters are (see Sections 3.3 and 3.10):

(6.3.5) $$\text{Number density:}\quad n = \int f(\mathbf{r},\mathbf{v},t) d\mathbf{v}\ ,$$

(6.3.6) $$\text{Gas velocity:}\ \mathbf{u} = \frac{1}{n}\int \mathbf{v} f(\mathbf{r},\mathbf{v},t) d\mathbf{v}\,,$$

(6.3.7) $$\text{Internal energy density:}\, E = \int \frac{m(\mathbf{v}-\mathbf{u})^2}{2} f(\mathbf{r},\mathbf{v},t) d\mathbf{v}\,,$$

(6.3.8) $$\text{Stress tensor:}\ \mathbf{S} = -m\int (\mathbf{v}-\mathbf{u})(\mathbf{v}-\mathbf{u})\, f(\mathbf{r},\mathbf{v},t)\, d\mathbf{v},$$

(6.3.9) $$\text{Heat flux vector:}\ \mathbf{q} = \int (\mathbf{v}-\mathbf{u})\frac{m(\mathbf{v}-\mathbf{u})^2}{2} f(\mathbf{r},\mathbf{v},t)\, d\mathbf{v}.$$

6.3. Particle motion, indexing, and sampling of macroscopic gas parameters

One can introduce the temperature T using the kinetic definition of temperature, e.g., Eq. (3.2.10):

(6.3.10)
$$\frac{k_B T}{2} = \frac{1}{3}\frac{E}{n}$$

and pressure as the negative average of diagonal components of the stress tensor:

(6.3.11)
$$p = -\frac{1}{3}\left(S_{xx} + S_{yy} + S_{zz}\right) = n k_B T.$$

In the DSMC method, any macroscopic gas parameter in the form of Eq. (6.3.4) is calculated through random state variables (coordinates and velocities) of simulated particles using the Monte Carlo method for evaluation of integrals, i.e. in the form of an arithmetic mean given by Eq. (5.4.1). In order to use such an approach, Eq. (6.3.4) must be represented in the form of an expectation of some random variable.

Let's consider a steady-state homogeneous state of a gas with distribution function $f(\mathbf{v})$. If components of the velocity vector $\mathbf{V} = \left(V_x, V_y, V_z\right)$ of some simulated particle are random variables, then what is the relationship between distribution function $f(\mathbf{v})$ and PDF $f_{\mathbf{V}}(\mathbf{v})$? The PDF must satisfy two conditions:

1. $f_{\mathbf{V}}(\mathbf{v}) \geq \mathbf{0}$: This condition is automatically satisfied for $f(\mathbf{v})$.

2. Normalization condition, e.g., Eq. (4.6.3)

(6.3.12)
$$\int f_{\mathbf{V}}(\mathbf{v}) d\mathbf{v} = 1.$$

6.3. Particle motion, indexing, and sampling of macroscopic gas parameters

Let's compare Eqs. (6.3.5) and (6.3.12). One can see that conditions given by Eq. (6.3.12) is satisfied if the PDF of random molecular velocities is equal to (see slides 33 and 34 in Chapter 5):

(6.3.13) $$f_{\mathbf{V}}(\mathbf{v}) = \frac{f(\mathbf{v})}{n}.$$

Then Eq. (6.3.4) for spatially homogeneous steady state can be represented in the form

(6.3.14) $$\langle\Phi\rangle = \frac{1}{n}\int \Phi(\mathbf{v}) f(\mathbf{v}) d\mathbf{v} = \int \Phi(\mathbf{v}) f_{\mathbf{V}}(\mathbf{v}) d\mathbf{v} = E(\Phi).$$

Eq. (6.3.14) has simple physical meaning: Macroscopic quantity $\langle\Phi\rangle$, which we introduced as an average value of molecular quantities $\Phi(\mathbf{v})$ of individual molecules, from the point of view of statistics is simply the expectation (mean) of random variables of Φ for individual molecules.
If random velocity vectors $\mathbf{v}_i$ are statistically independent, then $\Phi(\mathbf{v}_i)$ are also independent, and we can use the central limit theorem in order to calculate $\langle\Phi\rangle$ in the form of arithmetic mean given by Eq. (5.4.1), i.e.

(6.3.15) $$\langle\Phi\rangle = \frac{\Phi(\mathbf{v}_1) + \Phi(\mathbf{v}_2) + \cdots + \Phi(\mathbf{v}_N)}{N},$$

where N is the number of simulated particles in our system.
Eq. (6.3.15) allows us to calculate *all* macroscopic parameters given by Eqs. (6.3.5)-(6.3.11) if we additionally define the gas number density n. If our system of N simulated particles with statistical weights W is located in a "cell" of volume V, then

(6.3.16) $$n = \frac{WN}{V}.$$

6.3. Particle motion, indexing, and sampling of macroscopic gas parameters

Eqs. (6.3.15) and (6.3.16) are sufficient for calculation of all macroscopic parameters. Practical implementation of such calculations is based on the following approach:

1. The whole flow domain is divided into a mesh of cells and macroscopic gas parameters in the cell centers are calculated as averaged parameters over the volume of every cell.
2. It means that Eqs. (6.3.15) and (6.3.16) must be applied individually to a subsystem of particles inside every cell.
3. Calculations with Eq. (6.3.15) and (6.3.16) can be accurate only if N is very large. In order to increase the number of sampled particles, in steady-state flows, particles parameters are accumulated during many time steps.
4. In order to ensure that velocity vectors of individual simulated particles are independent random vectors, particle parameters are accumulated not every time step, but with period $\Delta t_a = \Delta K_a \Delta t$. Parameter ΔK_a is given by variable **`SamplingPeriod`** in the DSMC code shown in slide 15. In many practical problem sampling can be performed with $\Delta K_a = 1$.
5. In order to accumulate random molecular quantities during multiple time steps, special variables-**counters** are introduces for every cell of the computational mesh.

Let's assume that we have a regular (structural) mesh of cells, where every cell is identified with index (k, l), N^n is the number of simulated particles in the cell (k, l) of volume V at time t^n, $\mathbf{v}_i^n$ are velocity vectors of these particles.

Let's consider a problem when we need to calculate n, u, and E. For this purpose we will introduce in the cell (k, l) three variables-counters n_Σ^n, $\mathbf{u}_\Sigma^n$, and E_Σ^n and a counter of times steps K used for sampling of macroscopic gas parameters.

6.3. Particle motion, indexing, and sampling of macroscopic gas parameters

The calculations can be organized as follows:

1. At the beginning of simulations we set all counters to zero:

$$K = 0\,, \quad n_\Sigma^0 = 0, \quad \mathbf{u}_\Sigma^0 = 0, \quad E_\Sigma^0 = 0.$$

2. Once the steady state is reached ($t > t_{SS}$), we update values of counters at the time step after every ΔK_a steps as follows:

$$K = K + 1, \qquad n_\Sigma^n = n_\Sigma^{n-1} + N_{k,l}^n,$$

$$\mathbf{u}_\Sigma^n = \mathbf{u}_\Sigma^{n-1} + \sum_{i=1}^{N_{k,l}^n} \mathbf{v}_i^n\,, \qquad E_\Sigma^n = E_\Sigma^{n-1} + \sum_{i=1}^{N_{k,l}^n} (\mathbf{v}_i^n)^2\,.$$

3. At the end of simulation ($t > t_P$), when we need to print the macroscopic parameters in the cell (k, l), they can be calculated as follows

$$n_{k,l} = \frac{W}{V}\frac{n_\Sigma^n}{K}, \qquad \mathbf{u}_{k,l} = \frac{\mathbf{u}_\Sigma^n}{n_\Sigma^n}, \qquad E_{k,l} = \frac{W}{V}\frac{mE_\Sigma^n/2}{K} - n_{k,l}\frac{m\left(\mathbf{u}_{k,l}\right)^2}{2}.$$

Note that equation for $E_{k,l}$ is based on Eq. (3.2.8): The first term is the right-hand side is the density of the total energy and the second term is the density of kinetic energy associated with macroscopic motion of the gas. The difference between then is the density of internal energy.

6.3. Particle motion, indexing, and sampling of macroscopic gas parameters

Implementation of sampling of macroscopic gas parameters in a C++ code (see full version in DSMC2D_Template05.cpp)

```
typedef struct cell { ////////////////////////////////////////////////////////////////////////
        double  CountNP;                // Particle number -> Number density
        double  CountV[3];              // Velocity vector V -> Macroscopic gas velocity
        double  CountV2;                // Velocity square -> Density of internal energy
} CELL;
int     SampleStep;                     // Number of sampled steps
CELL    C[MAX_CELL_X][MAX_CELL_Y];      // Sample counters in cells
#define sqr3( V ) ( V[0] * V[0] + V[1] * V[1] + V[2] * V[2] )
void Sampling () { ///////////////////////////////////////////////////////////////////////////
        SampleStep++;
        for ( int k = 0; k < NX; k++ )
                for ( int l = 0; l < NY; l++ ) {
                        C[k][l].CountNP += NPC[k][l];
                        for ( int i = 0; i < NPC[k][l]; i++ ) {
                                C[k][l].CountV2 += sqr3 ( P[IPC[k][l][i]].V );
                                for ( int m = 0; m < 3; m++ ) C[k][l].CountV[m] += P[IPC[k][l][i]].V[m];
                        }
                }
}
void Printing () { ////////////////////////////////////////////////////////////////////////////
        for ( int k = 0; k < NY; k++ )
                for ( int l = 0; l < NX; l++ ) {
                        double X = X1 + ( k + 0.5 ) * DX;
                        double Y = Y1 + ( l + 0.5 ) * DY;
                        double N = Weight * C[k][l].CountNP / SampleStep / DV; // DV is the cell size
                        double U[3];
                        for ( int m = 0; m < 3; m++ ) U[m] = C[k][l].CountV[m] / C[k][l].CountNP;
                        double E = Weight * ParticleMass * C[i][j].CountV2 / 2.0 / SampleStep / DV
                                   - N * ParticleMass * sqr3 ( U ) / 2.0;
                        fprintf ( F, "…", X, Y, N, U[0], U[1], E );
                }
}
```

6.4. 2D test problem: Flow past a thin wing at an attack angle

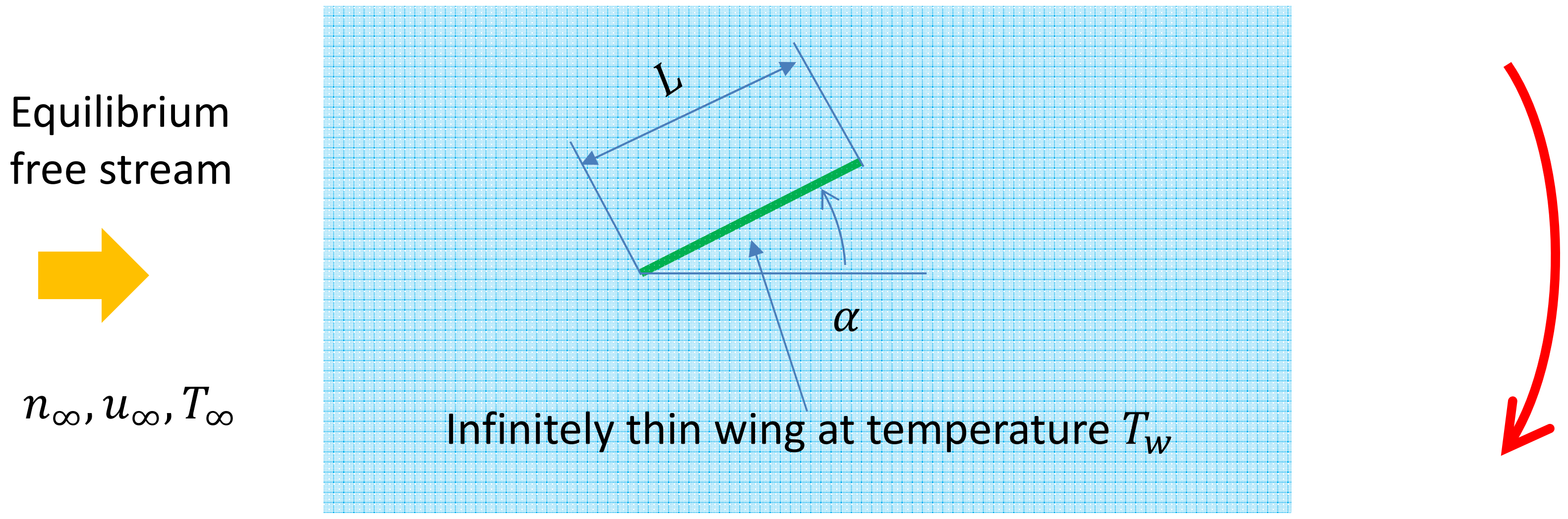
- Two-dimensional test problem
- Simulation parameters in 2D flow past a thin wing in a C++ code

6.4. 2D test problem: Flow past a thin wing at an attack angle

Implementation of initial and boundary conditions in the DSMC simulation is problem-specific. We consider implementation of typical initial and boundary conditions in the rarefied gas aerodynamics.

Two-dimensional test problem

Let's consider a flow past an infinitely thin wing at the angle of attack α.



Equilibrium free stream
$n_\infty, u_\infty, T_\infty$
L
α
Infinitely thin wing at temperature T_w

Molecular model: VHS molecules of mass m with the total cross-section

$$\sigma_T(C_r) = \pi d^2\,(C_r) = \sigma_{T,Ref}\left(\frac{C_{r,Ref}}{C_r}\right)^{\varpi}.$$

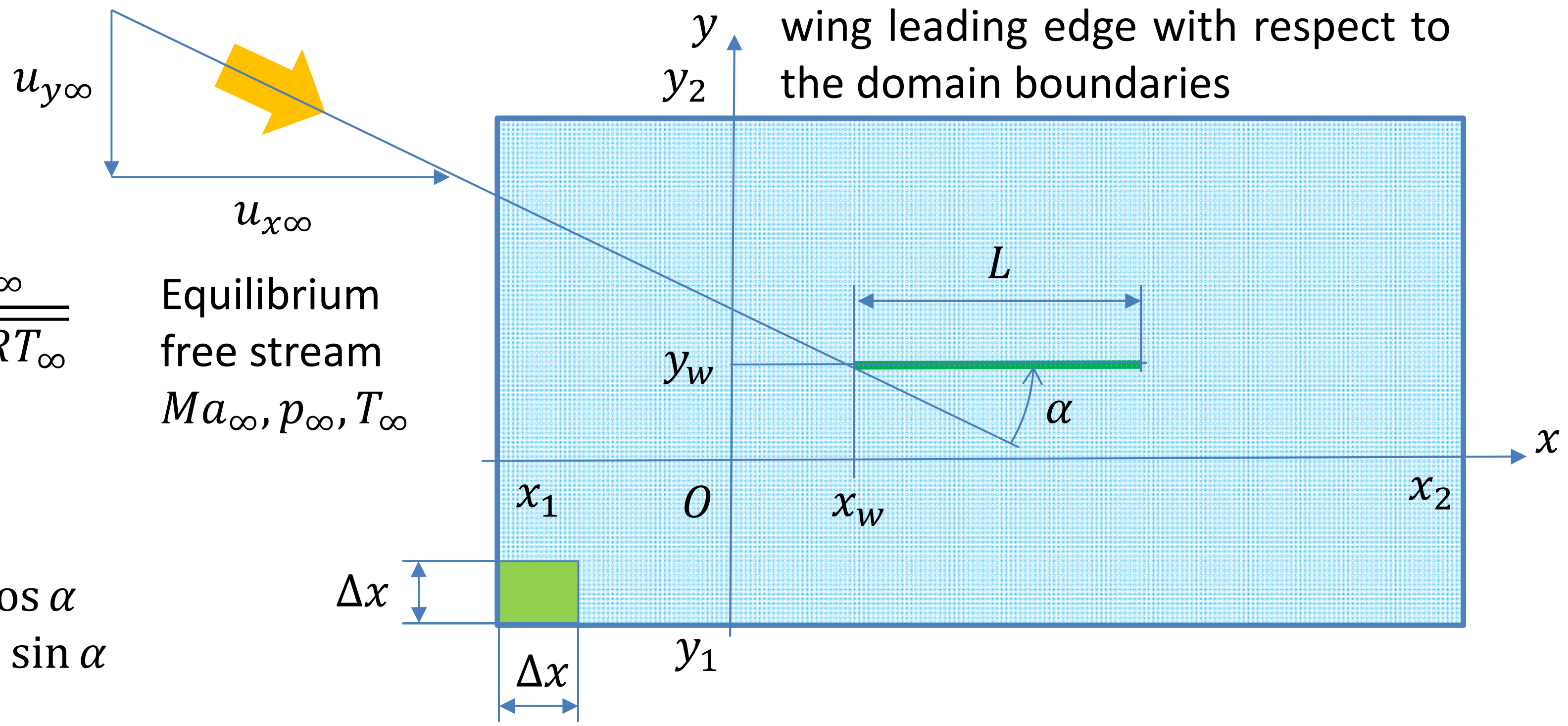
Wing : Plane wing of length L at constant temperature T_w and **angle of attack** α; Diffuse scattering of gas molecules from the wing surface.

Free stream : Equilibrium Maxwell-Boltzmann flow at given $n_\infty, u_\infty, T_\infty$.

6.4. 2D test problem: Flow past a thin wing at an attack angle

We will solve this test problem in rotated frame of reference in order to use the simplest rectangular computational mesh.

(x_w, y_w) specifies position of the wing leading edge with respect to the domain boundaries



Equilibrium free stream $Ma_\infty, p_\infty, T_\infty$

$$Ma_\infty = \frac{u_\infty}{\sqrt{\gamma R T_\infty}}$$

$$n_\infty = \frac{p_\infty}{k_B T_\infty}$$

$$u_{x\infty} = u_\infty \cos\alpha$$
$$u_{x\infty} = -u_\infty \sin\alpha$$

Statistical weight is usually calculated based on desired number of simulated particles N_{c0} in a cell, which is placed in the free stream:

$$W = \frac{\Delta x^2 \Delta z n_\infty}{N_{c0}}.$$

Here Δz is the size of the computational domain along Oz axis. In the developed C++ code, N_{c0} and Δz are stored in variables **`NPCFree`** and **`DZ`.**

6.4. 2D test problem: Flow past a thin wing at an attack angle

Simulation parameters in 2D flow past a thin wing in a C++ code (I)
(see full version in DSMC2D_Template06.cpp)

```
double  Dt                      = 1.0e-06;      // Time step (s)
int     NStep                   = 100000;       // Total number of time steps
int     FirstSamplingStep       = 10000;        // Number of the first sampling step
int     SamplingPeriod          = 1;            // Period between sampling steps
int     PrintPeriod             = 10000;        // Period between printing of results
double  X1      = -1.0;         // (m)
double  X2      = 1.0;          // (m)
double  Y1      = -1.0;         // (m)
double  Y2      = 1.0;          // (m)
double  DZ      = 0.1;          // Domain (cell) size in z direction for 2D problem (m)

int     NX      = 100;          // Number of cells along X axis
int     NY      = 100;          // Number of cells along Y axis

double  NPCFree = 10;           // Average number of particles in a cell of the free stream
double  DL      = 0.1;          // Size of the auxiliary domains implementing the free stream (m)

double  MolarMass = 0.040;      // Molar mass of gas (kg/mole)

double  MaFree = 4.0;           // Free stream Mach number
double  PFree = 0.1;            // Free stream pressure (Pa)
double  TFree = 200.0;          // Free stream temperature (K)

double  WingX = -0.25;          // X coordinate of the wing leading edge (m)
double  WingY = 0.0;            // Y coordinate of the wing leading edge (m)
double  WingLength = 0.5;       // Length of the wing (m)
double  AttackAngle = 30.0;     // Angle of attack (degree)
double  Tw = 300.0;             // Temperature of the wing surface (K)
```

6.4. 2D test problem: Flow past a thin wing at an attack angle

Simulation parameters in 2D flow past a thin wing in a C++ code (II)
(see full version in DSMC2D_Template06.cpp)

```
double  DX, DY;          // Cell sizes (m)
double  DV;              // Cell volume (m^3)
double  Weight;          // Statistical weight of simulated particles
double  MoleculeMass;    // Mass of a molecule (kg)

double  NFree;           // Number density in the free stream (1/m^3)
double  UFree;           // Velocity in the free stream (m/s)
double  UxFree, UyFree;  // X and Y components of the gas velocity vector in the free stream

void Setup () ////////////////////////////////////////////////////////////////////////////////
{
        MoleculeMass = MolarMass / AVOGADRO_CONSTANT;
        NFree = PFree / ( BOLTZMANN_CONSTANT * TFree );
        UFree = MaFree * sqrt ( ( 5.0 / 3.0 ) * BOLTZMANN_CONSTANT * TFree / MoleculeMass );
        UxFree = UFree * cos ( M_PI * AttackAngle / 180.0 );
        UyFree = - UFree * sin ( M_PI * AttackAngle / 180.0 );
        DX = ( X2 - X1 ) / NX;
        DY = ( Y2 - Y1 ) / NY;
        DV = DX * DY * DZ;
        Weight = DV * NFree / NPCFree;
}
```

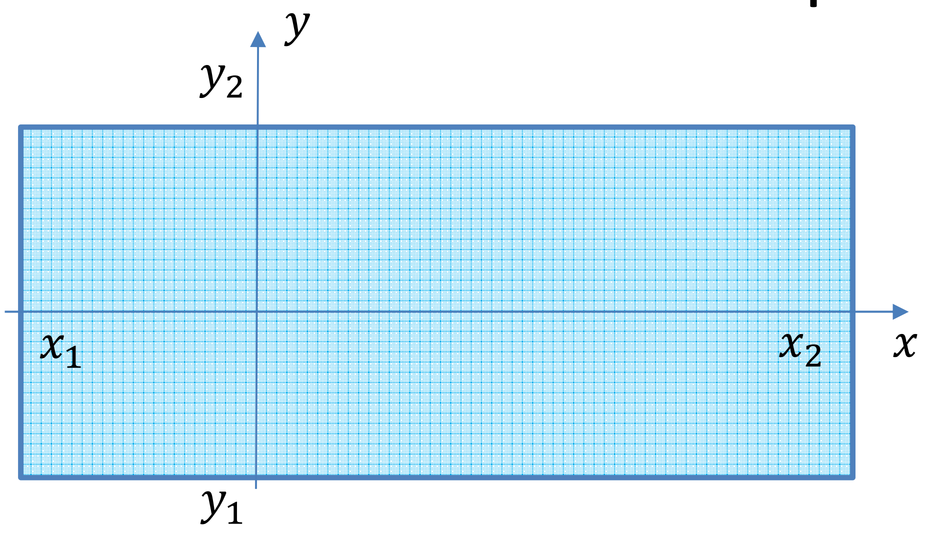
6.5. Initial and boundary conditions for the 2D test problem

- Generation of molecules with equilibrium distribution in a rectangular domain
- Implementation of generation of particles in a rectangular domain in a C++ code
- Initial conditions
- Implementation of initial conditions in 2D flow past a thin wing in a C++ code
- Free stream boundary conditions at the external boundary
- Implementation of free stream boundary conditions in the C++ code
- Boundary condition of diffuse scattering of gas molecules from the wing surface
- Implementation of boundary conditions of diffuse scattering on the wing surface in a C++ code
- Example: Simulation of free molecular (collisionless) flow

6.5. Initial and boundary conditions for the 2D test problem

Initial conditions and boundary conditions in the free stream in the test problem reduce to generation of simulated particles with Maxwell-Boltzmann distribution in a rectangular domain.

Generation of molecules with equilibrium distribution in a rectangular domain



$$f(\mathbf{v}) = \frac{n}{(2\pi k_B T/m)^{3/2}} \exp\left[-\frac{m(\mathbf{v}-\mathbf{u})^2}{2k_B T}\right].$$

Average number of simulated molecules to be generated:

$$\langle N \rangle = \frac{(x_2 - x_1)(y_2 - y_1)\Delta z n}{W}.$$

- If $\langle N \rangle$ is large ($\langle N \rangle > \sim 20$), then the random number of particles N can be chosen as

$$N = \begin{cases} [\langle N \rangle] + 1, & \gamma < \langle N \rangle - [\langle N \rangle] \\ [\langle N \rangle], & \text{otherwise} \end{cases}.$$

- If $\langle N \rangle$ is small ($\langle N \rangle < \sim 20$), then the random number of particles N can be chosen based on the Poisson distribution with parameter $a = \langle N \rangle$ (see Sections 4.5 and 5.3).
- Random coordinates of particles are distributed uniformly.
- Components of the random velocity vector have Gaussian distribution with means (u_x, u_y, u_z) and variance $k_B T/m$ (see slides 33 and 34 in Chapter 5).

6.5. Initial and boundary conditions for the 2D test problem

Then generation of coordinates and velocities of a single particle can be perform using the following fragment of C++ code:

```
PCL P;
v2rand_uniform_rect ( P.X[0], P.X[1], X1, X2, Y1, Y2 ); // Generate position
vrand_MB ( P.V, MoleculeMass, Ugas, Tgas ); // Generate velocity
```

where

```
void vrand_MB ( double *v, double m, double *u, double T ) //////////////////////////////////
// This function generates random velocity vector from Maxwell-Boltzmann distribution.
// It was adopted from solution of problem 6 in homework 4.
{ /////////////////////////////////////////////////////////////////////////////////////
double  RT = BOLTZMANN_CONSTANT * T / m;
        v[0] = frand_Gaussian ( u[0], RT );
        v[1] = frand_Gaussian ( u[1], RT );
        v[2] = frand_Gaussian ( u[2], RT );
}
double frand_Gaussian ( double E, double V ) ////////////////////////////////////////////
// Here E is the mean, V is the variance
{ /////////////////////////////////////////////////////////////////////////////////////
        return E + sqrt ( - 2.0 * V * log ( brng () ) ) * cos ( 2.0 * M_PI * brng () );
}
void v2rand_uniform_rect ( double &X, double &Y, double a, double b, double c, double d ) //
// (X,Y) are Cartesian coordinates of a point inside the rectangle [a,b]x[c,d]
{ /////////////////////////////////////////////////////////////////////////////////////
                X = a + ( b - a ) * brng ();
                Y = c + ( d - c ) * brng ();
}
```

6.5. Initial and boundary conditions for the 2D test problem

Implementation of generation of particles in a rectangular domain in a C++ code (see DSMC2D_Template06.cpp)

```
void GenerateNewParticles ( double X1, double Y1, double X2, double Y2, double Ngas,
                            double Ugasx, double Ugasy, double Tgas, int MoveFlag )
{
        // Here we calculate the number of particles to be added to the list (NP,P)
double  V = ( X2 - X1 ) * ( Y2 - Y1 ) * DZ; // Volume of the rectangular region (m^3)
double  Nnew_avg = Ngas * V / Weight; // Average number of particles to be added
int     Nnew; // Random number of particles to be added
        if ( Nnew_avg > 20.0 ) { // Number if large; use standard method
                Nnew = int ( Nnew_avg );
                if ( brng () < Nnew_avg - Nnew ) Nnew++;
        } else { // Number is not large; use Poisson distribution
                Nnew = irand_Poisson ( Nnew_avg );
        }
        // Now we add new particles one by one
double  Ugas[3] = { Ugasx, Ugasy, 0.0 };
        for ( int i = NP; i < NP + Nnew; i++ ) {
                // Generate position
                v2rand_uniform_rect ( P[i].X[0], P[i].X[1], X1, X2, Y1, Y2 );
                // Generate velocity
                vrand_MB ( P[i].V, MoleculeMass, Ugas, Tgas );
                // Move particles during time step, if necessary
                if ( MoveFlag == 1 ) { // Move particle during time step
                        P[i].X[0] += Dt * P[i].V[0];
                        P[i].X[1] += Dt * P[i].V[1];
                }
        }
        NP += Nnew;
}
```

6.5. Initial and boundary conditions for the 2D test problem

Initial conditions

Since we are interested in the steady-state solution, the initial condition is arbitrary. But it is useful to use the initial condition which is as much close as possible to the final steady-state solution. In this case we can reduce time t_{SS} (see slide 13 in this Chapter) required for the transient process converging to the steady-state solution.

In aerodynamics, the initial conditions are often used in the form, corresponding of the undisturbed free stream. In this case, the initial distribution of simulated molecules corresponds to the Maxwell-Boltzmann distribution function in the free stream.

In this case, the transient process can be viewed as propagation of disturbances introduced into the free stream by inserting the body

(6.5.1) $$f(\mathbf{v}) = \frac{n_\infty}{(2\pi k_B T_\infty / m)^{3/2}} \exp\left[-\frac{m(\mathbf{v}-\mathbf{u}_\infty)^2}{2k_B T_\infty}\right].$$

Implementation of initial conditions in 2D flow past a thin wing in a C++ code (see DSMC2D_Template06.cpp)

```
void InitialConditions () ///////////////////////////////////////////////////////////////////
{
        Step = 0;
        Time = 0.0;
        NP = 0;
        // Set to zero all counters of macroscopic properties
        SampleStep = 0;
        SampleTime = 0.0;
        memset ( C, 0, sizeof C );
        // Here we distribute initial particles according to the distribution in the free stream
        GenerateNewParticles ( X1, Y1, X2, Y2, NFree, UxFree, UyFree, TFree, 0 );
}
```

6.5. Initial and boundary conditions for the 2D test problem

Free stream boundary conditions at the external boundary

According to the formulation of boundary value problems for the Boltzmann kinetic equation, the boundary conditions at any boundaries of the computational domain must be imposed only for molecules that move from outside into the domain (see Section 3.9 and Eqs. (3.9.4) and (3.9.5)). There are no constrains on the distribution function of molecules that leave the domain through the boundary. Distribution function of such molecules must be obtained as a result of solution of the problem.

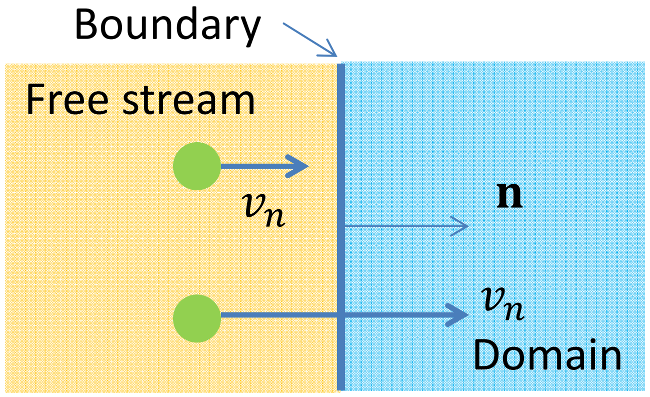
Corresponding boundary conditions in the DSMC simulations imply that at every time step

1. New particles are generated at the external boundary of the domains. These particles simulate the inflow of particles into the computational domain from the free stream, where

$$f(\mathbf{v}) = \frac{n_\infty}{(2\pi k_B T_\infty / m)^{3/2}} \exp\left[-\frac{m(\mathbf{v} - \mathbf{u}_\infty)^2}{2 k_B T_\infty} \right].$$

2. All existed simulated particles that left the domain through the external boundary are excluded from further consideration.

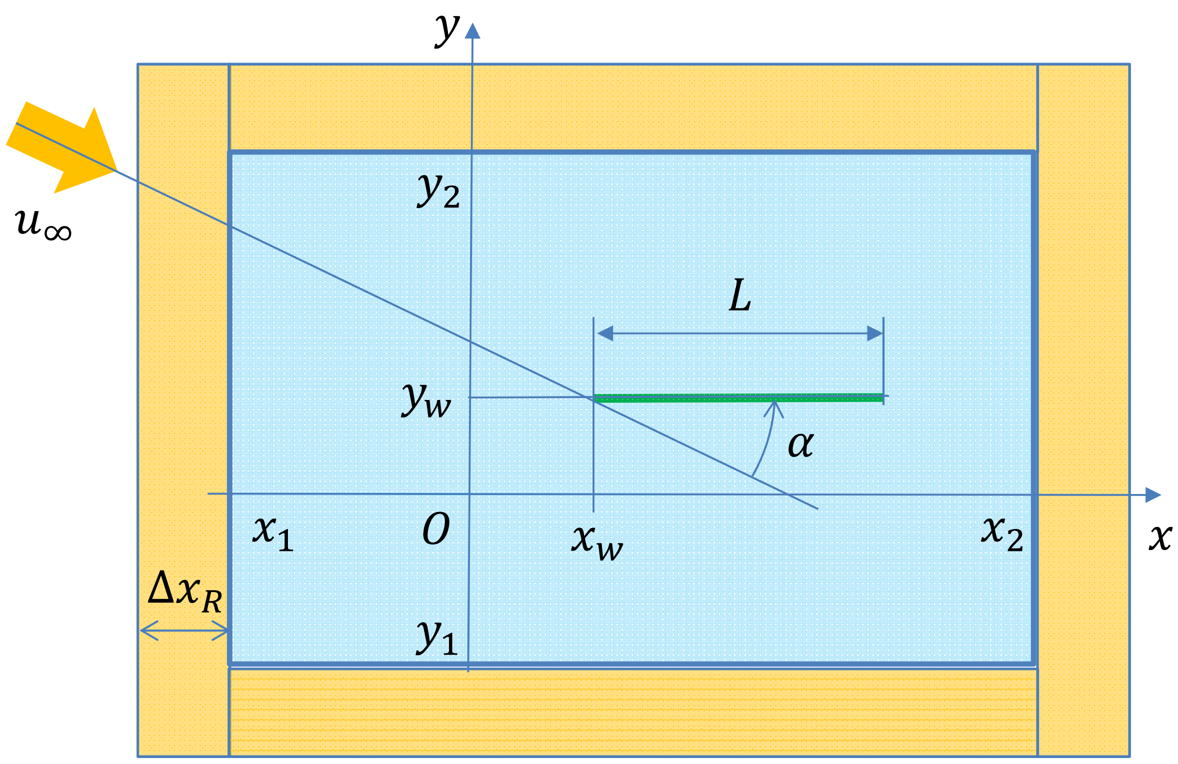
Important note: Particles entering the domain from the free stream do not have Maxwell-Boltzmann distribution of velocities. It happens, because the molecules that have large velocity v_n in the direction normal to the boundary, have large chance to move through the boundary when molecules with smaller v_n.



6.5. Initial and boundary conditions for the 2D test problem

Although the inflow of molecules at the free stream boundaries can be simulated by the direct approach, it is often (especially in the case of boundaries of a complex geometrical shape) simulated indirectly based on the acceptance and rejection method as follows:

1. The domain is surrounded by auxiliary subdomains or **reservoirs**.
2. In every subdomain at every time step new particles are generated using the approach considered before for implementation of the initial conditions (See slides 30-32).
3. Every particle from a reservoir moves during Δt. If the particle enters the domains, it is added to the list of particles and used for further calculations, otherwise it is rejected.



- In the test problem, four rectangular reservoirs are required. The reservoir size Δx_R is stored in variable **`DL`**.
- In every reservoir, new particles can be generated using function **`GenerateNewParticles`** (slide 32) with **`MoveFlag = 1`**.
- The size of the reservoir Δx_R is an *important numerical parameter*: It must be sufficiently large in order to allow high-velocity molecules enter the domain during the time step. Value of Δx_R depends on u_∞, RT_∞, and Δt.

6.5. Initial and boundary conditions for the 2D test problem

Implementation of free stream boundary conditions in the C++ code (see DSMC2D_Template06.cpp)

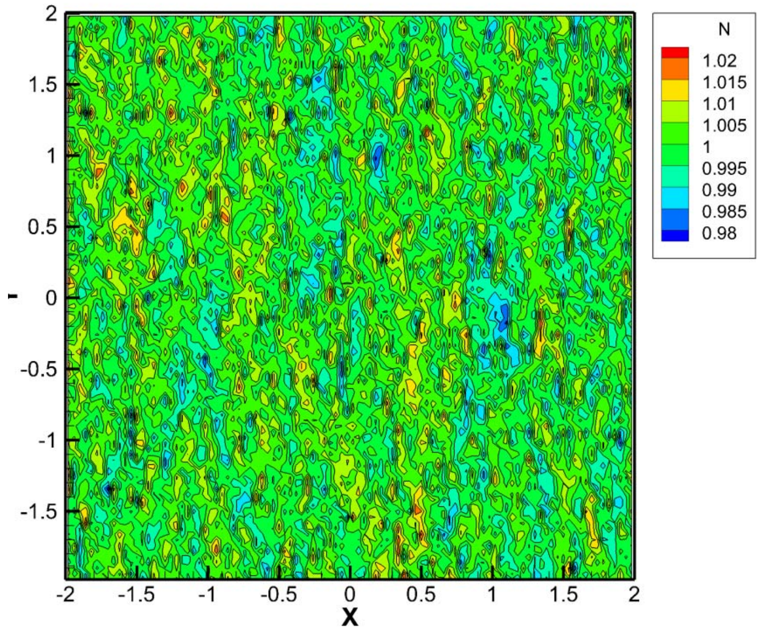
```cpp
void BoundaryConditions () //////////////////////////////////////////////////////////////////
{
        // Here we generate new particles at the external boundaries of the computational domain
        // where the distribution function of molecules entering domain is the free stream
        // equilibrium distribution function
        // Left boundary
        GenerateNewParticles ( X1 - DL, Y1 - DL, X1, Y2 + DL, NFree, UxFree, UyFree, TFree, 1 );
        // Right boundary
        GenerateNewParticles ( X2, Y1 - DL, X2 + DL, Y2 + DL, NFree, UxFree, UyFree, TFree, 1 );
        // Bottom boundary
        GenerateNewParticles ( X1, Y1 - DL, X2, Y1, NFree, UxFree, UyFree, TFree, 1 );
        // Top boundary
        GenerateNewParticles ( X1, Y2, X2, Y2 + DL, NFree, UxFree, UyFree, TFree, 1 );

        // Here we remove all particles that are located outside the domain
        for ( int i = 0; i < NP; ) {
                if ( P[i].X[0] <= X1 || P[i].X[0] >= X2 || P[i].X[1] <= Y1 || P[i].X[1] >= Y2 ) {
                        // Particle i is outside the domain, so we replace P[i] with P[NP-1]
                        if ( i < NP - 1 ) memmove ( &P[i], &P[NP-1], sizeof ( PCL ) );
                        NP--;
                } else {
                        i++;
                }
        }
}
```

6.5. Initial and boundary conditions for the 2D test problem

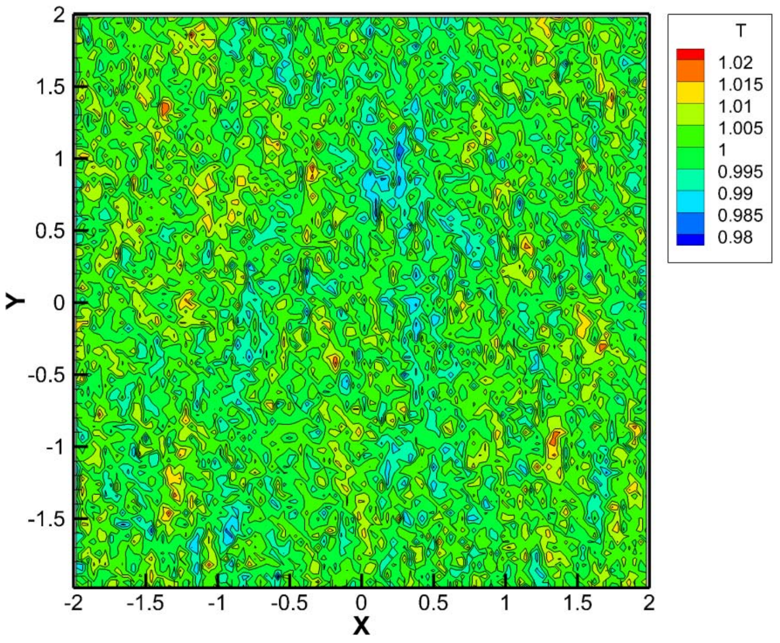
- Once the free stream boundary conditions are implemented, the code (DSMC2D_Template06.cpp) can be used in order to simulate the equilibrium free stream.
- It is a good practice to check that the code is capable of reproducing simple flows with known properties.

  Typical results with parameters specified in DSMC2D_Template06.cpp are shown below

Number density n/n_∞



Temperature T/T_∞



- These fields are obtained after sampling of gas parameters during 90,000 time steps with 10 simulated molecules per cell in average.
- These parameters provides constant distributions with the statistical scattering (fluctuations) at the level of 2%.

6.5. Initial and boundary conditions for the 2D test problem

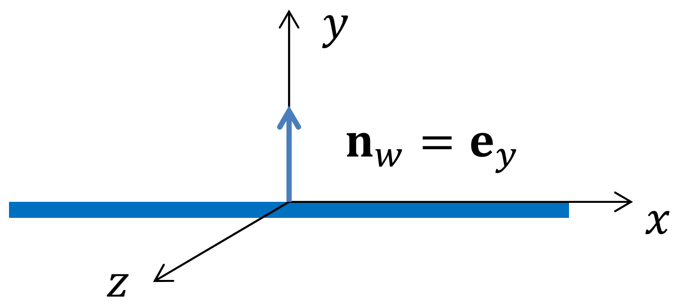
Boundary condition of diffuse scattering of gas molecules from the wing surface

According to the model of diffuse scattering, the scattering probability density function $S(\mathbf{v}|\mathbf{n}_w)$ is equal to (here $T_r = T_w$, see Section 3.8 and Eq. (3.8.9))

(6.5.2) $$S(\mathbf{v}|\mathbf{n}_w) = \frac{|\mathbf{v}\cdot\mathbf{n}_w|}{2\pi(RT_w)^2}\exp\left[-\frac{\mathbf{v}^2}{2RT_w}\right].$$

This equation defines the PDF of components of the velocity vector $\mathbf{v}$ of an individual molecule after reflection from the body surface at $\mathbf{v}\cdot\mathbf{n}_w \geq 0$.

In the test problem, the $\mathbf{n}_w$ is equal to $\pm\mathbf{e}_y$. Let's consider the case $\mathbf{n}_w = \mathbf{e}_y$ (another case with $\mathbf{n}_w = -\mathbf{e}_y$ can be considered using the same approach) and re-write Eq. (6.5.2) for individual components of $\mathbf{v}$:



(6.5.3)
$$S(\mathbf{v}|\mathbf{n}_w) = \frac{v_y}{2\pi(RT_w)^2}\exp\left[-\frac{v_x^2+v_y^2+v_z^2}{2RT_w}\right] = S_t(v_x)S_n(v_y)S_t(v_z)\,.$$
$$S_n(v) = \frac{v}{RT_w}\exp\left[-\frac{v^2}{2RT_w}\right], \qquad S_t(v) = \frac{1}{\sqrt{2\pi RT_w}}\exp\left[-\frac{v^2}{2RT_w}\right].$$

It means that the normal component of velocity v_y has Rayleigh distribution with parameter $\sigma = \sqrt{RT_w}$ (see Section 5.4, slide 28 in Chapter 5) and the tangential components v_x and v_z have Gaussian distribution with zero mean and variance RT_w.

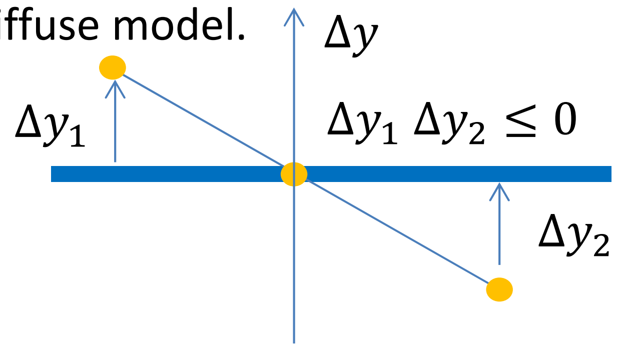
6.5. Initial and boundary conditions for the 2D test problem

Then the generation of a random velocity vector of a molecule after diffuse scattering from the wing can be implemented in the following C++ function (here the variable **Ny** defines the direction of the normal and can be equal to ± 1):

```
void DiffuseScattering ( double *v, double m, double Tw, double Ny ) //////////////////////
{
double  RTw = BOLTZMANN_CONSTANT * Tw / m;
        v[0] = frand_Gaussian ( 0.0, RTw );
        v[1] = Ny * frand_Rayleigh ( sqrt ( RTw ) );
        v[2] = frand_Gaussian ( 0.0, RTw );
}
```

For every simulated particle, the following sequence of calculations must be performed:

1. Check whether the particle trajectory during Δt intersects the body surface or not.
2. If yes, calculate the position of the particle on the surface and time Δt_i from the beginning of the time step until interaction.
3. Replace particle velocity with the new random velocity according to the model of interaction of gas molecules with the surface, e.g., Eqs. (6.5.4) for the diffuse model.
4. Move particle from the body during time $\Delta t - \Delta t_i$.



In order to implement # 2, we need to know positions of the particle in the beginning and at the end of the time step and then to perform **linear interpolation** to the point of interaction. This can be easily done along with displacement of particles in function **MoveParticles ()**.

$$\Delta t - \Delta t_i = \Delta t - \Delta t \Delta y_1 / (\Delta y_1 - \Delta y_2) \quad (6.5.4)$$

6.5. Initial and boundary conditions for the 2D test problem

Implementation of boundary conditions of diffuse scattering on the wing surface in a C++ code (see DSMC2D_Template07.cpp)

```
void MoveParticles () ///////////////////////////////////////////////////////////////////////
{
        for ( int i = 0; i < NP; i++ ) {
                double X0 = P[i].X[0];
                double Y0 = P[i].X[1];
                for ( int m = 0; m < DIM; m++ ) P[i].X[m] += Dt * P[i].V[m];
                // Here we implement diffuse scattering of gas molecules from the wing
                if ( ( Y0 - WingY ) * ( P[i].X[1] - WingY ) < 0.0 ) {
                        // Linear interpolation to point Y = WingY
                        double Xw=( X0*(WingY-P[i].X[1])+P[i].X[0]*(Y0-WingY))/(Y0-P[i].X[1]);
                        if ( Xw > WingX && Xw < WingX + WingLength ) {
                                // Molecule interacts with the wing during the time step
                                // Linear interpolation of the time of scattering, Eq. (6.5.4)
                                double Dt1 = Dt - Dt * ( Y0 - WingY ) / ( Y0 - P[i].X[1] );
                                // Generate velocity vector of the reflected molecule
                                DiffuseScattering(P[i].V,MoleculeMass,Tw,(Y0-WingY>0)?1.0:(-1.0));
                                // Move the reflected molecule
                                P[i].X[0] = Xw + Dt1 * P[i].V[0];
                                P[i].X[1] = WingY + Dt1 * P[i].V[1];
                        }
                }
        }
}
```

Here we first check that particle trajectory crosses the line $y = y_w$: See sketch in slide 31 in this Chapter

Next we check that the coordinate x of the point, where the particle crossed the line $y = y_w$, belongs to the wing, i.e. $x_w \le x \le x_w + L$

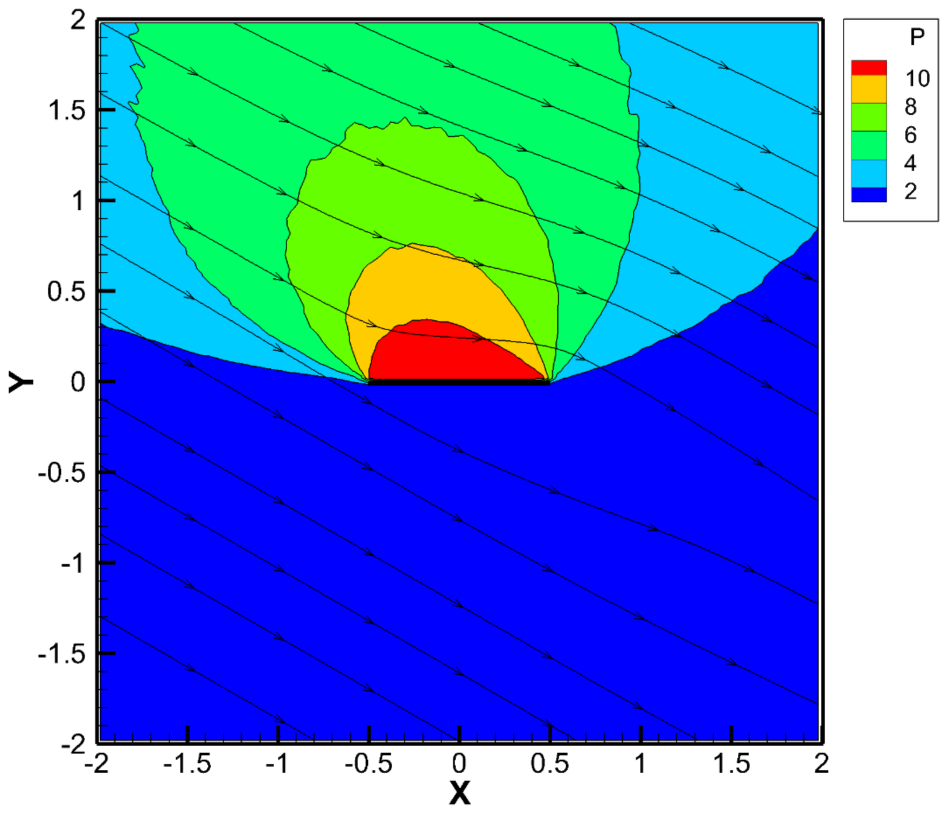
6.5. Initial and boundary conditions for the 2D test problem

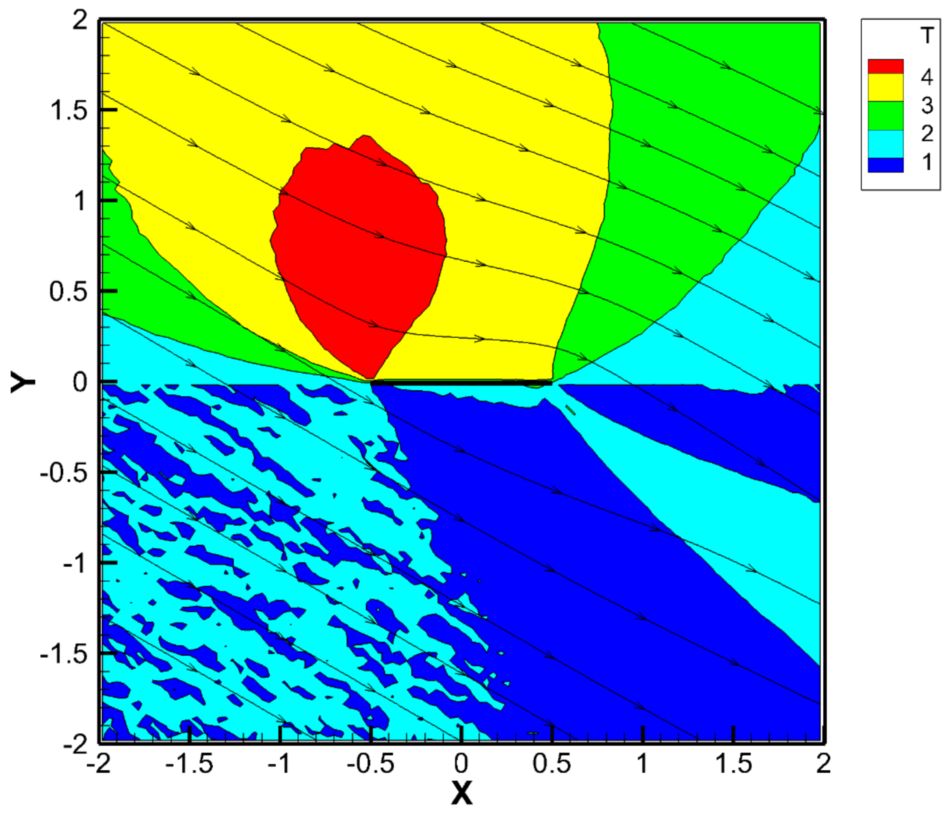
Example: Simulation of free molecular (collisionless) flow

- Once all boundary conditions are implemented, the code (DSMC2D_Template07.cpp) can be used in order to simulate the free molecular (collisionless) flow over a wing.
- This flow can be considered as a limit case when n_∞ is such small that $Kn_\infty \ll 1$.

Typical results with parameters specified in DSMC2D_Template07.cpp are shown below

$$Ma_\infty = 4, \qquad \alpha = 30^o$$





6.6. Sampling of binary collisions

- Sampling of post-collisional velocities for the VHS molecular model
- Sampling of colliding pairs
- Primitive scheme
- The No Time Counter scheme
- Transitional flow past a wing

6.6. Sampling of binary collisions

As a results of indexing, the list of simulated particles located in every cell of the computational mesh is known. **Sampling of binary collisions** implies

- **Sampling of colliding pairs**: Selection of random pairs of colliding particles from the particle list in the cell.
- **Sampling of post-collisional velocities**: Replacement of velocities of colliding pairs of molecules with post-collisional velocities.

Both parts of this approach must be performed in agreement with the structure of collisional term in the Boltzmann kinetic equation and chosen molecular model (model of a binary collision).

Sampling of post-collisional velocities for the VHS molecular model

Post-collisional velocities after a binary collisions of molecules i and j are given by Eqs. (2.4.26):

$$\mathbf{v}'_i = \mathbf{v}_j + \left[\left(\mathbf{v}_j - \mathbf{v}_i\right) \cdot \mathbf{n}\right]\mathbf{n}, \qquad \mathbf{v}'_j = \mathbf{v}_j - \left[\left(\mathbf{v}_j - \mathbf{v}_i\right) \cdot \mathbf{n}\right]\mathbf{n}.$$

In the case of the VHS molecular model, however, it is easier to use equations that follow from Eqs. (2.4.6) and (2.4.9):

$$\mathbf{v}'_i = \mathbf{v}_c - \frac{1}{2}\mathbf{c}'_r, \qquad \mathbf{v}'_j = \mathbf{v}_c + \frac{1}{2}\mathbf{c}'_r, \qquad \mathbf{v}_c = \frac{\mathbf{v}_i + \mathbf{v}_j}{2}, \qquad \mathbf{c}'_r = |\mathbf{v}_j - \mathbf{v}_i|\mathbf{e}'_{cr}. \tag{6.6.1}$$

where the unit vector $\mathbf{e}'_{cr}$ defines the direction of relative velocity $\mathbf{c}'_r$ after the collision. Distribution of directions of $\mathbf{e}'_{cr}$ is given by the differential collision cross section $\sigma(\chi, C_r)$ (see Eq. (2.5.10)). But according to the VHS model, $\sigma = \sigma(C_r)$ (see Eq. (2.6.5)) and, consequently, $\mathbf{e}'_{cr}$ is an *isotropic random vector*. Random components of $\mathbf{e}'_{cr}$ can be sampled using the approach considered in Section 5.4 (slides 41 and 42 in Chapter 5).

6.5.Initial and boundary conditions for DSMC simulations of the test problem

Then generation of random velocity vectors of molecules after binary collision can be implemented in the following C++ function:

```
void ElasticCollision ( double *V1, double *V2, double Cr ) ///////////////////////////////
// Elastic collisions of molecules with the same masses used for both HS and VHS models
{ ///////////////////////////////////////////////////////////////////////////////////
double  N[3];
        v3rand_isotropic ( N );
        Cr *= 0.5;
double  VC[3] = { 0.5 * ( V2[0] + V1[0] ), 0.5 * ( V2[1] + V1[1] ), 0.5 * ( V2[2] + V1[2] ) };
double  VCr[3] = { Cr * N[0], Cr * N[1], Cr * N[2] };
        V1[0] = VC[0] + VCr[0];
        V1[1] = VC[1] + VCr[1];
        V1[2] = VC[2] + VCr[2];
        V2[0] = VC[0] - VCr[0];
        V2[1] = VC[1] - VCr[1];
        V2[2] = VC[2] - VCr[2];
}
```

In this function, it is assumed that the absolute relative velocity $c_r = |\mathbf{v}_j - \mathbf{v}_i|$ is calculated preliminary and passed to this function through the input parameter **`Cr`**.

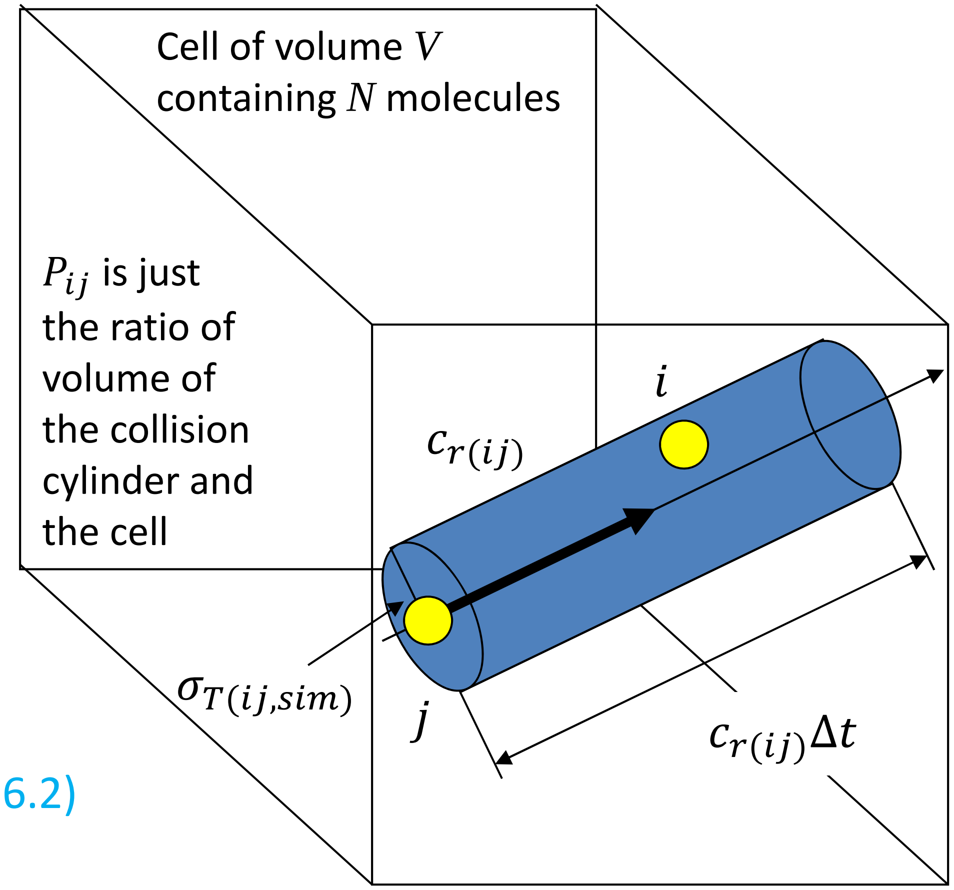
Sampling of colliding pairs

Sampling of colliding pairs is the central part of any DSMC method. Different DSMC methods are different by the approach used for sampling of colliding pairs. We consider two approaches:

- **Primitive scheme** that follows from the Bernoulli trial.
- **No Time Counter (NTC)** scheme proposed by G.A. Bird.

6.6. Sampling of binary collisions

Let's consider a cell of volume V where N simulated molecules are located. These N molecules form $N(N-1)/2$ of different pairs. *In the "standard" DSMC technique, molecules are assumed to be distributed homogeneously inside the cell* and relative position of particles with respect to each other is not taken into account. Any binary collision between particles with indices i and j is considered as a random event that happens during time step Δt with some probability P_{ij}.


Cell of volume V containing N molecules
P_{ij} is just the ratio of volume of the collision cylinder and the cell
i
$c_{r(ij)}$
$\sigma_{T(ij,sim)}$
j
$c_{r(ij)}\Delta t$

Relative velocity of molecules i and j:

$$c_{r(ij)}, = |\mathbf{c}_{r(ij)}|, \qquad \mathbf{c}_{r(ij)} = \mathbf{v}_j - \mathbf{v}_i,$$

Total collision cross section of these molecules:

$$\sigma_{T(ij)} = \sigma_T(C_{r(ij)})$$

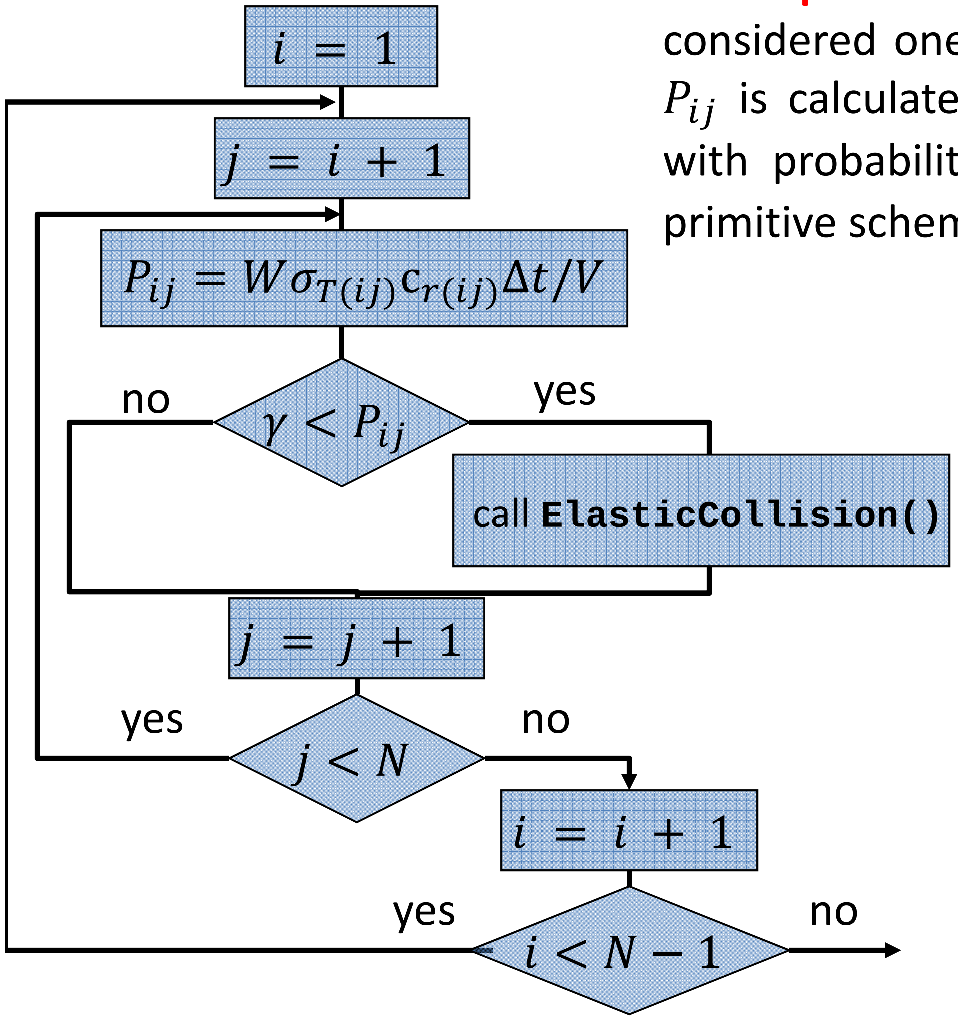
Probability of a random collision between molecules i and j during time step (assuming that Δt is sufficiently small and $P_{ij} < 1$:

$$P_{ij} = \frac{\sigma_{T(ij,sim)} c_{r(ij)} \Delta t}{V} = \frac{W \sigma_{T(ij)} c_{r(ij)} \Delta t}{V} \qquad (6.6.2)$$

6.6. Sampling of binary collisions

Primitive scheme

In the **primitive scheme**, all pairs of molecules in the cell are considered one by one, for every pair the collision probability P_{ij} is calculated and then random collision event is sampled with probability P_{ij} (see Section 5.5). The flowchart of this primitive scheme can be formulated as follows:



Calculation of the collision probability

Does the collision occur?

Sampling of post-collisional velocities

Are there other pairs of molecules in the cell?

Go to the next cell

The disadvantage of the primitive scheme is the large number of arithmetic operation, which is proportional to the N^2 and increases fast with increasing N.

6.6. Sampling of binary collisions

The No Time Counter scheme

The **NTC scheme** by Bird utilizes the acceptance and rejection Monte Carlo method and is based on the introduction of a **majorant** $[\sigma_T c_r]_{\max}$, i.e. such quantity that

$$[\sigma_T C_r]_{\max} \geq \sigma_{T(ij)} c_{r(ij)} \qquad \text{for pairs of particles } (i,j) \text{ in the cell.}$$

Probability of a binary collision during Δt: $$P_{ij} = \frac{W \sigma_{T(ij)} c_{r(ij)} \Delta t}{V}$$

Collision frequency for a pair of molecules: $$z_{ij} = \frac{P_{ij}}{\Delta t} = \frac{W \sigma_{T(ij)} c_{r(ij)}}{V}$$

Collision frequency in the cell: $$Z_{cell} = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} z_{ij} = \frac{W}{V} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \sigma_{T(ij)} c_{r(ij)}$$

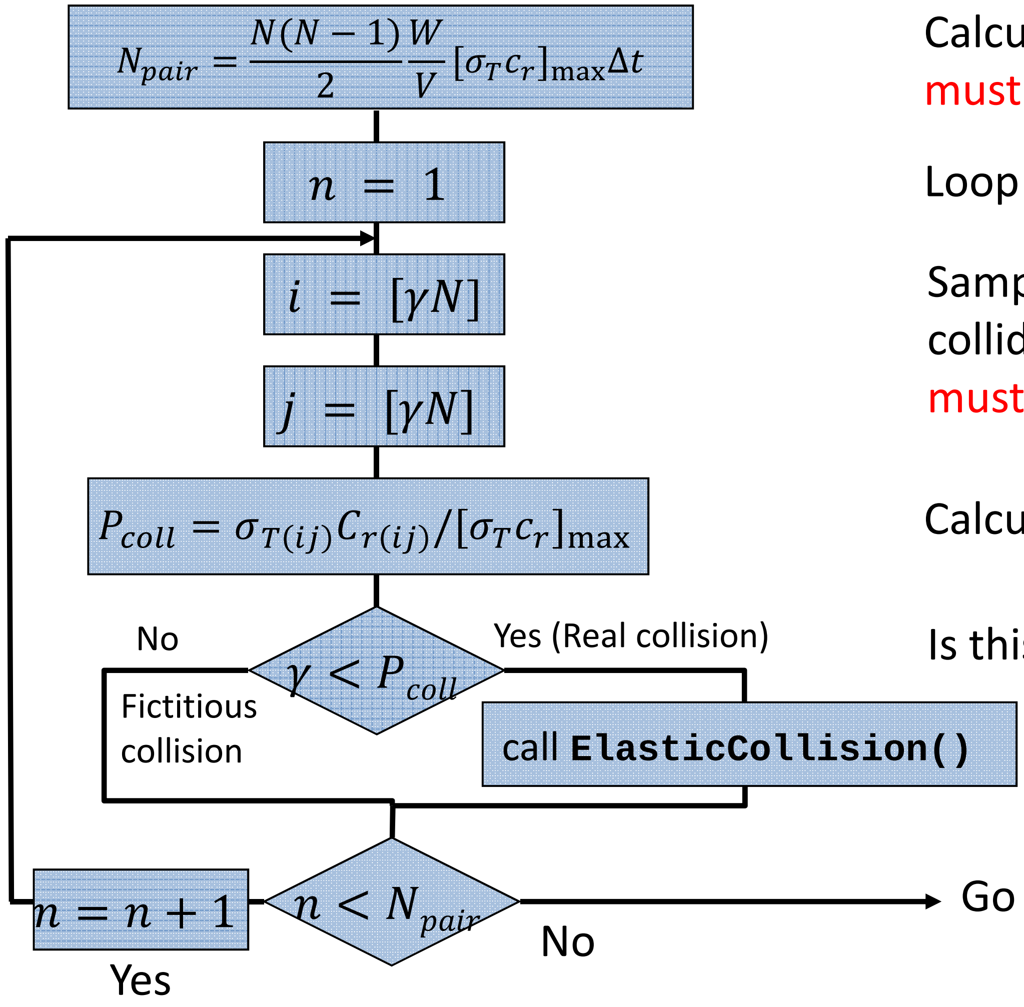
Majorant collision frequency in the cell: $$Z_{max} = \frac{W}{V} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} [\sigma_T C_r]_{\max} = \frac{N(N-1)}{2} \frac{W}{V} [\sigma_T C_r]_{\max}$$

In the NTC scheme, $N_{pair} = Z_{max} \Delta t$ random pairs (i,j) of molecules is selected, but for every pair collision happens with probability $P_{coll} = \sigma_{T(ij)} C_{r(ij)} / [\sigma_T C_r]_{\max}$. In this case, the number of arithmetic operations is proportional to $N \log N$ and grows with N much slower than in the primitive scheme. Majorant $[\sigma_T C_r]_{\max}$ is important **numerical parameter** of the NTC scheme.

6.6. Sampling of binary collisions

- $N_{pair} = Z_{max}\Delta t$ collisions are sampled during time step.
- **Real** collision (accepted collision) occurs with probability $P_{coll} = \sigma_{T(ij)}c_{r(ij)}/[\sigma_T c_r]_{\max}$.
- All other collisions are **fictitious** (rejected collisions).

The *simplified* flowchart of the NTC scheme can be formulated as follows:



Calculation of the majorant collision number, must be reduced to an integer value

Loop over all collisions

Sampling of random indices of colliding molecules, must ensure that $i \neq j$

Calculation of probability of real collision

Is this a real collision?

Sampling of post-collisional velocities

Go to the next cell

6.6. Sampling of binary collisions

NTC scheme is proven to provide correct collision statistics and to be in agreement with the Boltzmann equation.

Implementation of the NTC scheme in the C++ code (see DSMC2D_Template08.cpp)

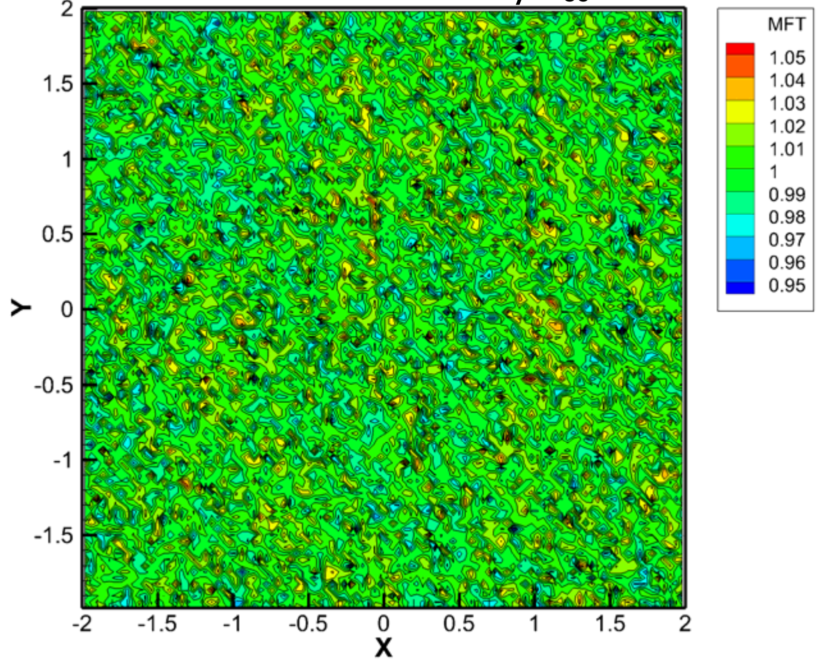
```
int BinaryCollisionsInCell ( double CellVolume, int NPC, int *IPC ) ///////////////////////
{
        if ( NPC < 2 ) return 0; // No collisions if there are less than 2 simulated particles
double  SCMax = 9.0 * SigmaRef * sqrt ( BOLTZMANN_CONSTANT * TFree / MoleculeMass );
double  Npair = 0.5 * NPC * ( NPC - 1 ) * Weight * SCMax * Dt / CellVolume;
        // Random number of collisions/pairs to be dranw in the cell during current time step
int     NN = int ( Npair );
double  N1 = Npair - NN;
        NN = ( brng () < N1 ) ? NN + 1 : NN;
int     i, j;
int     NC = 0; // Counter of collisions
        for ( int k = 0; k < NN; k++ ) { // Sampling of collisions
                i = irand_uniform ( NPC );
                do { j = irand_uniform ( NPC ); } while ( j == i );
                // Indices of colliding molecules in the global list of particles (NP,P)
                i = IPC[i];
                j = IPC[j];
                double VCr[3]={ P[j].V[0]-P[i].V[0], P[j].V[1]-P[i].V[1], P[j].V[2]-P[i].V[2] };
                double Cr = sqrt ( VCr[0] * VCr[0] + VCr[1] * VCr[1] + VCr[2] * VCr[2] );
                double Sigma = SigmaRef * pow ( CrRef / Cr, Omega ); // VHS total cross section
                if ( brng () < Sigma * Cr / SCMax ) { // Real collision
                        ElasticCollision ( P[i].V, P[j].V, Cr );
                        NC++;
                }
        }
        return NC;
}
```

6.6. Sampling of binary collisions

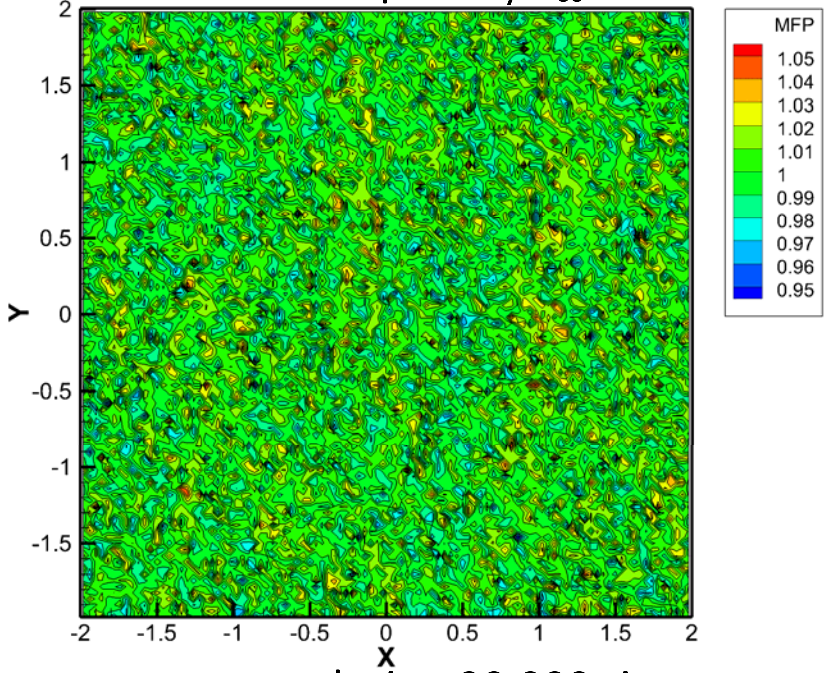
- Once collision sampling is implemented, the code (DSMC2D_Template08.cpp) can be used to simulate equilibrium free stream with collisions.
- It is a good practice to check that the code is capable of reproducing simple flows with known properties.

Typical results with parameters specified in DSMC2D_Template08.cpp are shown below

Mean free time τ/τ_∞



Mean free path λ/λ_∞



- These fields are obtained after sampling of gas parameters during 90,000 time steps with 10 simulated molecules per cell.
- These parameters provides constant distributions with the statistical scattering (fluctuations) at the level of 5%.

6.6. Sampling of binary collisions

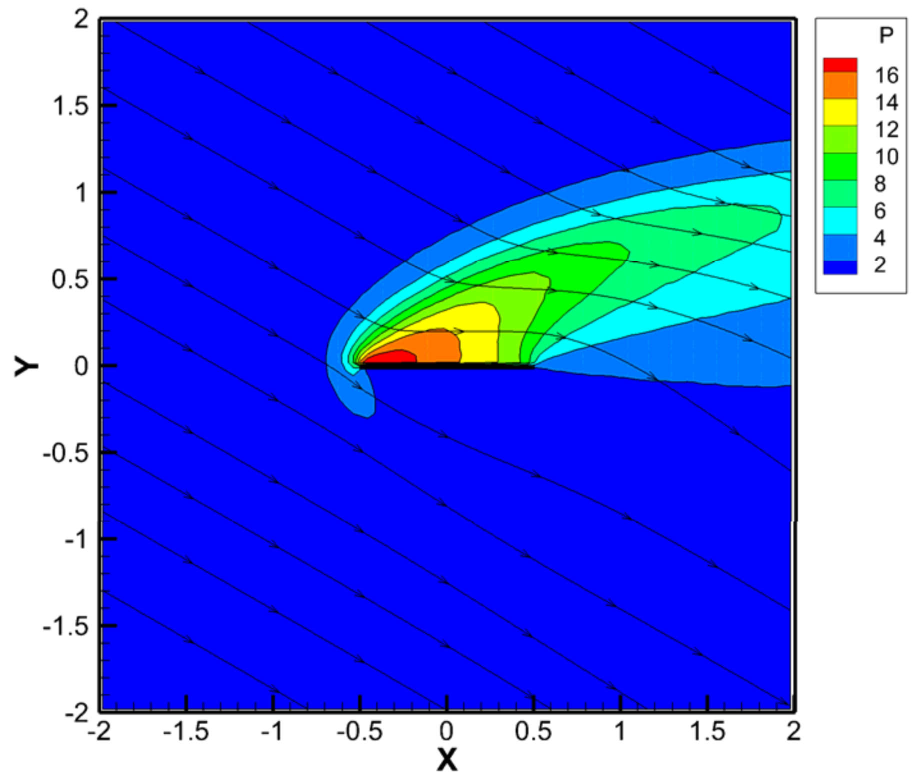
Transitional flow past a wing

DSMC2D_Template08.cpp is a fully functional version of the DSMC code.

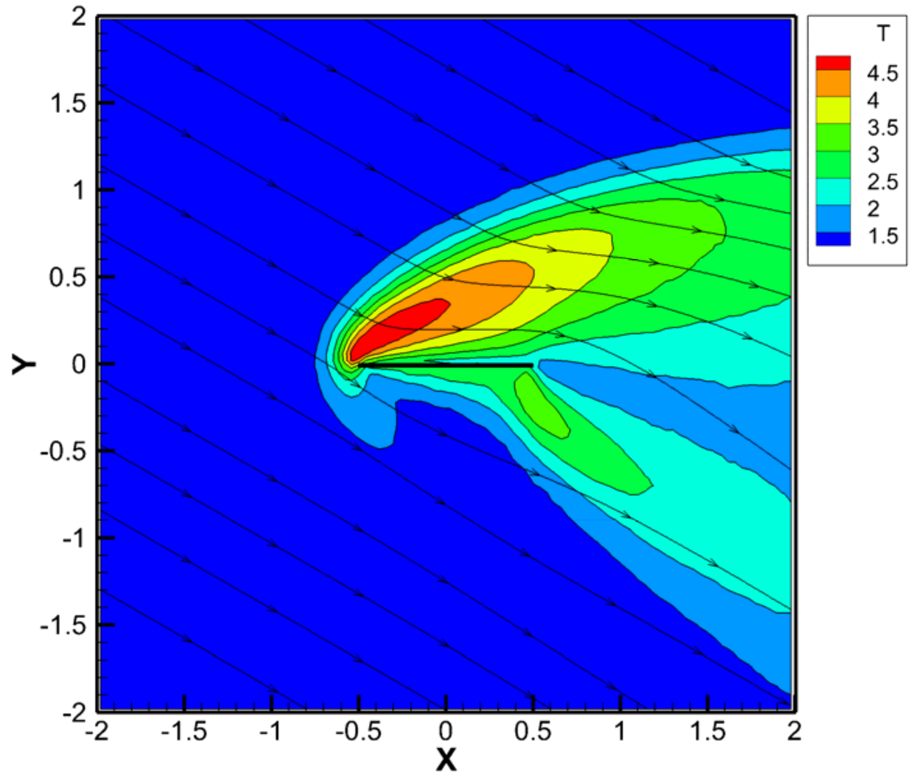
Typical results with parameters specified in DSMC2D_Template08.cpp are shown below.

$$Ma_\infty = 4, \qquad \alpha = 30^o, \qquad Kn_\infty = \frac{\lambda_\infty}{L} = 0.065$$

Pressure p/p_∞



Temperature T/T_∞



6.6. Sampling of binary collisions

One can introduce partial “temperatures” that characterize energies of chaotic motion of individual degrees of freedom $\beta = x, y, z$ (compare with equations in slides 23 and 24):

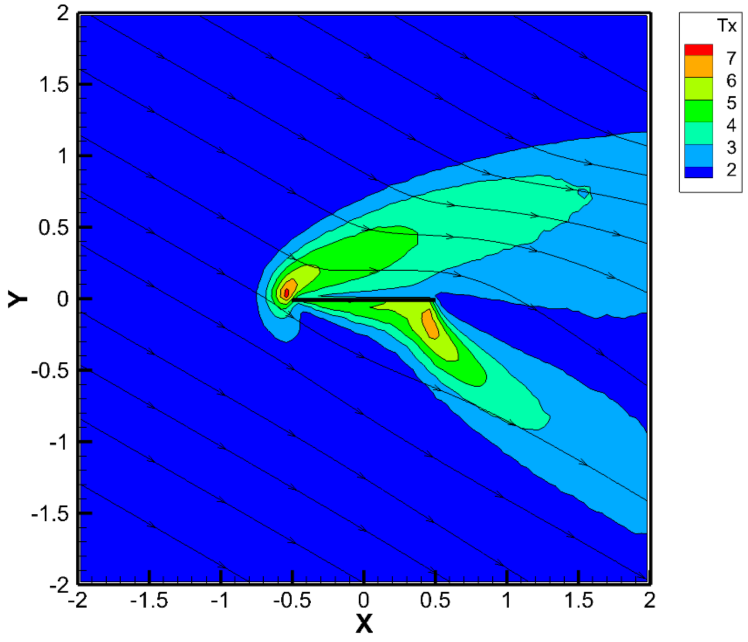
$$\frac{k_B T_\beta}{2} = \frac{1}{n} \int \frac{m\left(v_\beta - u_\beta\right)^2}{2} f(\mathbf{r}, \mathbf{v}, t) d\mathbf{v}, \qquad T = \frac{1}{3}\left(T_x + T_y + T_z\right).$$

Under conditions of equilibrium, equipartition of energy between different degrees of freedom must be established and

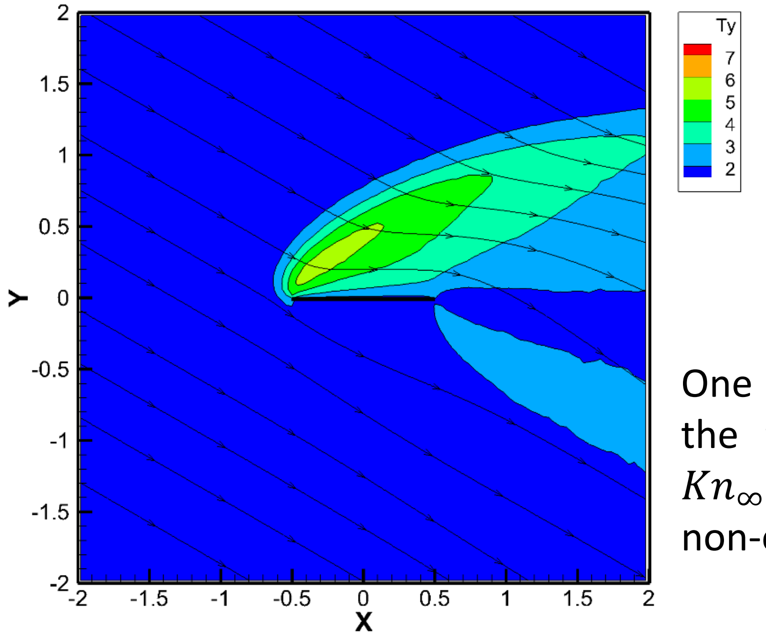
$$T = T_x = T_y = T_z.$$

The differences between T, T_x, T_y, T_z can be used as measures of degree of non-equilibrium.

T_x / T_∞



T_y / T_∞



See also comments in slide 42 of Chapter 1

One can conclude that the transitional flow at $Kn_\infty = 0.065$ is strongly non-equilibrium

6.7. Numerical parameters of the DSMC method

- Sources of errors in the DSMC simulations
- Major numerical parameters of the DSMC simulations

6.7. Numerical parameters of the DSMC method

Sources of errors in the DSMC simulations

We have a solution. What is its value?

Three sources of errors in the DSMC simulations:

- **Poor physical models:**

  Collision cross-sections, gas-surface interactions, chemical reactions, etc.

- **Finite dispersion of gas parameters obtained as averages of random values:**

  Intrinsic statistical noise (scattering) in gas parameters obtained in the DSMC simulations reduces with increasing sample size N_{sample} (or increasing $t_P - t_{SS}$), i.e. increasing the number of simulated particles whose molecular quantities are averaged in every cell for calculation of macroscopic gas parameters.

- **Numerical (discretization) errors:**

  Controlled by the primary numerical parameters of the DSMC method (cell size Δx, statistical weight W, time step Δt) and secondary numerical parameters like the reservoir size Δx_R, majorant $[\sigma_T C_r]_{\max}$ in the NTC scheme, and some other parameters (position of the free stream boundaries).

6.7. Numerical parameters of the DSMC method

Major numerical parameters of the DSMC simulations

Cell size Δx:

Should be small as compared to the local mean free path of gas molecules, λ, in order to ensure homogeneous distribution of molecules in cell

$\frac{\Delta x}{\lambda} < 0.1 \div 0.5$ (some problems can be solved at $\Delta x = \lambda$).

Statistical weight W:

Should be small enough in order to provide sufficient number of simulated molecules in every cell, N, for correct collision statistics and reduced statistical dependence between simulated molecules (the Boltzmann equation is obtained assuming molecular chaos, i.e. complete statistical independence)

$N > 10 - 30$ (some simulation can be performed even at $N = 1$).

Time step Δt :

Should be small enough as compared to the mean free time between collisions, τ, in order to make possible time-splitting of real evolution into the sequence of collisionless motion and "motionless" collisions and use of collision probability in the form of Eq. 6.6.2)

$\frac{\Delta t}{\tau} < 0.1 \div 0.5,$ $\frac{U \Delta t}{\Delta x} < 0.5,$ (U, characteristic velocity of gas molecules).