Meeting RENUDA & EDF

**Date: January 30th and 31st, 2020**

**Place: EDF R&D**

**City: Chatou**

# Participants:

EDF R&D: Jean-Pierre Minier

Renuda: Nicolas Tonnello, Panos Asproulis

INRIA: Christophe Henry

# Topic and issues:

## Executive summary

The aim of the meeting is to define precisely the work that needs to be done within the project between RENUDA and EDF R&D. This includes the six following points:

1. Turbulent dispersion models (distinction between the fluid one and the two-phase flow one);
2. Boundary conditions;
3. Adding extra terms in the SDEs;
4. Re-writing the Poisson correction;
5. Test cases (cyclone, wall-jet, Hercule, channel flow, Arnasson)
6. GUI and data model

In the following, the precise modifications needed for each of these points are specified.

## Point 1: Turbulent dispersion models

*Aim:*

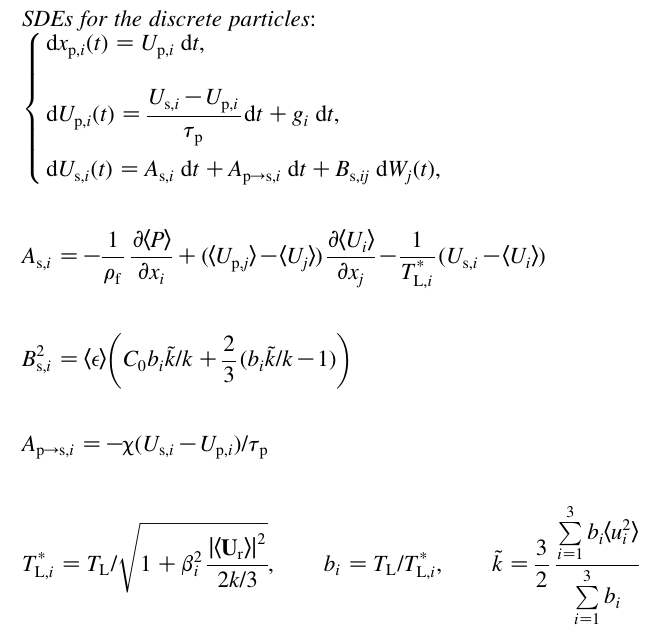
This first point is composed of two objectives related to the so-called “turbulent dispersion model” inside Code\_Saturne:

1. Generalize the “turbulence dispersion model” with the introduction of a rotation so that all the stochastic differential equations (SDEs) of particle motion are solved within the local frame of reference related to the mean relative velocity;
2. Re-arrange the current code and GUI to allow for the user to choose between the “turbulent dispersion model” (activated by default) and the limiting case of fluid-like particles.

*Theory for the particle equation of motion:*

To better understand the modifications required, we need to go back to the theory for particle motion from which the model implemented in Code\_Saturne has been derived.

The equation of motion of particles in the one-point PDF approach is given by the following SDE (see also Table 1 page 328 in [1] for more details):

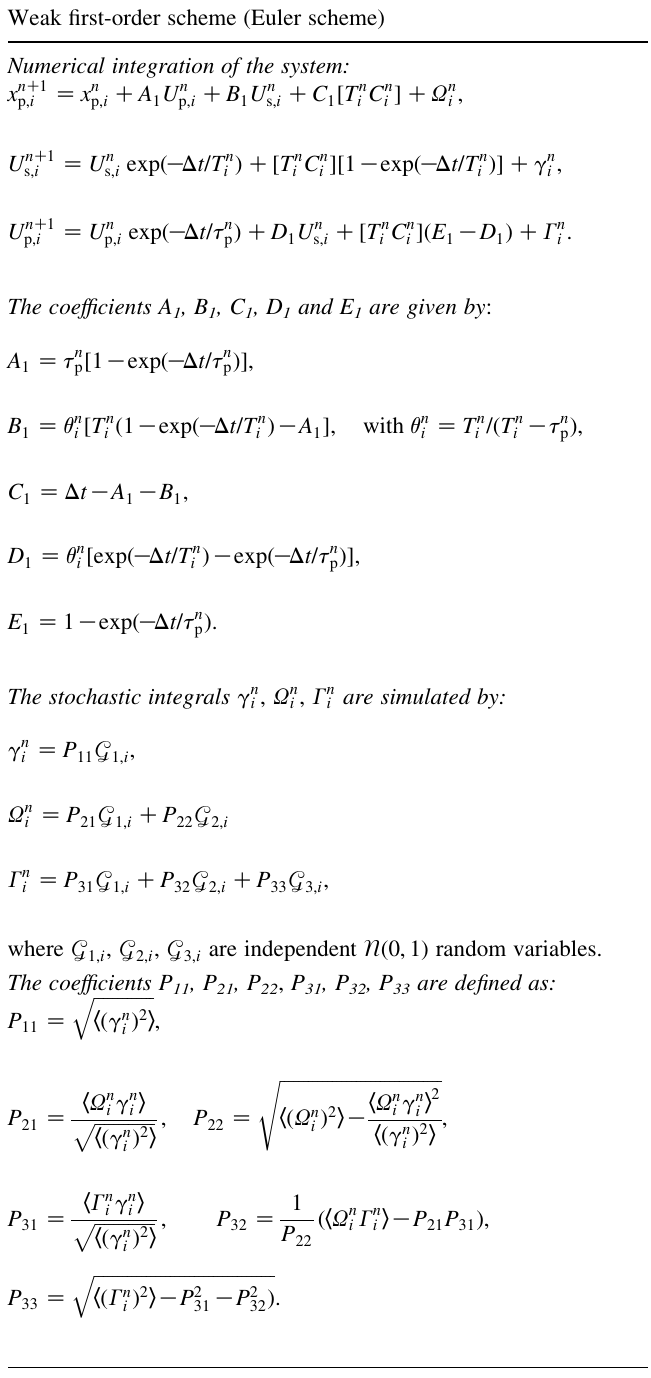


where xp is the particle position, Up the particle velocity, Us the fluid velocity seen by the particle, τp the particle relaxation time and TL the fluid Lagrangian timescale (note that the indice i denotes the coordinate). In these equations, β1=β if axis 1 is aligned with the mean drift (the mean relative velocity 〈**U**r〉) and β2=β3=2β in the transverse direction.

From these equations, it appears that, in the limit of fluid particles, we have β1= β2= β3=β and k^tilde=k which simplifies the previous equations.

*Numerical solution and implementation in Code\_Saturne:*

The numerical solution of the previous SDEs is obtained by resorting to an exponential scheme and by considering that the terms entering the equation are constant during the time step (TL, 〈**U**r〉, Up, Us, τp). This gives the following weak first-order scheme using a discrete (constant) time step Δt:



These equations are actually implemented in Code\_Saturne. More precisely, there are two routines that deal with these equations:

* cs\_lagr\_car.c:

This routine calculates all the terms needed in the equations, including:

* 1. the particle relaxation time τp,named taup (a vector of size Npart x 1)
  2. the turbulent kinetic energy k, named energ (a vector of size Ncell x 1)
  3. the turbulent dissipation ε, named dissip (a vector of size Ncell x 1)
  4. the fluid Lagrangian timescale TL\*,named tlag (a matrix of size Npart x 3)
  5. the term Bs, named bx (a matrix of size Npart x 3 x (Nordre-1))
  6. the term for pressure gradient and fluid gradient which enters TiCi, named piil (a matrix of size Npart x 3)

At the moment, there is a distinction between the case when the “complete model for turbulent dispersion” is activated or not (this corresponds to the keyword modcpl). When activated, TL\* is calculated according to the main direction that is provided by the user (given by the keyword idirla that has a value between 1 and 4).

* cs\_lagr\_sde.c:

This routine evaluates the particle positions after a time step Δt following the numerical solution of the SDEs given before. More precisely, it contains several functions:

1. \_lages1 is the function to update the particle position, velocity and velocity seen in the case of a 1st order scheme for all particles in the simulation (loop on Npart, named n\_particles).

The new particle position is computed with 5 terms entering the SDEs (ter1x, ter2x, ter3x, ter4x, ter5x) and 2 additional terms for Brownian motion (tbrix1, tbrix2).

The new particle velocity is computed with 5 terms entering the SDEs (ter1p, ter2p, ter3p, ter4p, ter5p) and 1 additional term for Brownian motion (tbriu).

The new particle velocity seen is computed with 3 terms entering the SDEs (ter1f, ter2f, ter3f).

Note that, at the moment, a rotation is only applied in the case of non-spherical particles (flag shape inside the cs\_glob\_lagr\_model attributes).

1. \_lages2 is the function to update the particle position, velocity and fluid velocity seen in the case of a 2nd order scheme. It is similar to lages1 (note that lages1 is actually called for the first passage through the function).
2. \_lagesd and \_lagdep are function to compute the particle position, velocity and fluid velocity seen when the deposition submodel is applied (flag deposition in cs\_glob\_lagr\_model).
3. cs\_lagr\_sde is the main function called in cs\_lagr.c. It creates temporary arrays to store the value of the particle density (named romp, size Npart x 1), the values of the random number for the stochastic terms in the SDEs (named vagaus, size Npart x 3 x 3), the values of the random numbers for Brownian motion (named brgaus, size Npart x 6) and the value of the extra forces acting on particles (named force\_p, size Npart x 3). It also switches between the various functions for 1st order and 2nd order schemes (note here that the second loop through the SDE equation for 2nd order schemes is done inside cs\_lagr.c and not inside cs\_lagr\_sde to update the values of the fluid ).

*Modifications to be done:*

According to the aforementioned aim, the two following modifications are needed here:

* Rotation to a local frame of reference

The rotation to the local frame of reference associated with the mean relative velocity in each cell requires several changes:

1. Inside \_lages1, a rotation to the local frame of reference is needed at the beginning of the function and a rotation back to the original frame of reference is needed at the end of the function (see what is done currently for non-spherical particles, i.e. shape = 1 or 2).
2. Inside cs\_lagr\_sde, the rotation matrix needs to be computed for each cell. The best solution is to allocate an extra matrix (size Ncell x 3 x 3) at the beginning of cs\_lagr\_sde and to pass it as an argument to the other functions. The value of the relative velocity is simply obtained by getting the mean particle velocity and the mean fluid velocity in each cell.

* Limit for fluid-particles (also called inertia-less particles or tracer-particles):

In that case, it has been seen before that the term TL\* and Bs both become isotropic (no differences between longitudinal and transversal directions).

In terms of implementation, this is actually already done in line 533 for tlag and 544-546 for bx. The only thing that is required here is to activate this model with the new choice in the GUI to activate the fluid-particle limit model.

*Test cases:*

The first step is to check that the current modifications do not affect the verification test case of the point-source dispersion (see [2]).

Then, a second test case is needed to verify that the rotation is correctly implemented. For that purpose, the idea is to perform two numerical simulations similar to the case of a point-source dispersion, except that a constant relative velocity is imposed inside cs\_lagr\_car (〈**U**r〉 = 1.0 along the x direction in the 1st simulation and along a direction tilted at 45° from x, y and z directions) while keeping the mean fluid velocity 〈**U**f〉 = 0 (inside cs\_lagr.c).

We expect in that case that the particle dispersion will look like a rugby ball (with more particles being dispersed along the direction of the relative velocity) instead of a sphere (as is the case in the original point-source dispersion case). The verification consists then into comparing that the dispersion obtained in the two simulations are the same statistically speaking (both simulations can be compared by applying the proper rotation of the axes).

## Point 2: Boundary conditions

*Aim:*

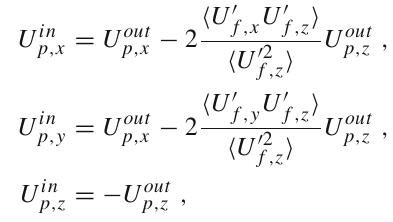
The specific aim here is to implement the wall function model for the fluid velocity seen during the impact of a particle on a wall.

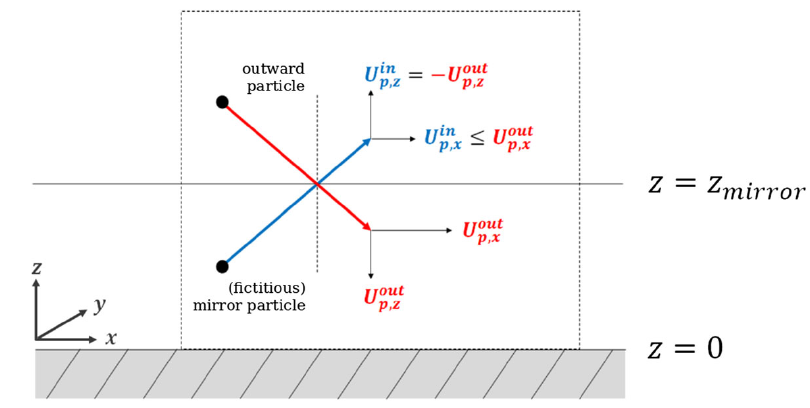
*Theory for wall-boundary conditions*

The formulation of wall-boundary conditions in particle stochastic approaches requires a specific treatment for the case of a rebound with a surface.

For instance, a first idea consists in applying specular rebound: every particle that crosses a wall boundary and exits the computational domain is reflected back with the same streamwise velocity component as the incoming one, i.e. Up,xin = Up,xout and Up,yin = Up,yout (see also the next figure). Yet this amounts to writing a zero-flux boundary condition for the streamwise velocity component, which fails to account for the momentum exchange with the wall.

Following the same spirit of the one used to derive so-called wall-function boundary conditions in computational fluid dynamics (CFD) codes, where a boundary condition is applied in the logarithmic region near the wall rather than at the wall itself [5], a better representation of wall-boundary conditions to capture the constant-stress boundary layer is (assuming that z is the wall-normal direction)



**

*Numerical implementation in Code\_Saturne*

The boundary condition is applied in the cs\_lagr\_tracking routine, which computes where the particles are according to their position and the mesh used. When a particle is detected as colliding with a boundary face, the outcome is treated in the function \_boundary\_treatment within the cs\_lagr\_tracking.c file.

Currently, when the flag associated to the face is a rebound (CS\_LAGR\_REBOUND) or a symmetry (CS\_LAGR\_SYM), a specular rebound is applied.

*Modifications to be done:*

Here, the modification needed is to change the specular rebound into the wall-boundary condition described previously. This requires only a modification of the fluid velocity seen by a particle impacting the surface (the position and velocity are unchanged).

The piece of code replacing the velocity seen is given below:

// CH: modification of rebound properties for particles

cs\_real\_t ustar = extra->uetbor[face\_id];

tmp = cs\_math\_3\_dot\_product(particle\_velocity\_seen, face\_norm);

tmp1 = cs\_math\_3\_dot\_product(particle\_velocity, face\_norm);

int iprev = 0; // Use fields at current time step

if ( extra->iturb == 30

|| extra->iturb == 31

|| extra->iturb == 32) {

if (extra->cvar\_rij == NULL) {

particle\_velocity\_seen[0] -= - 2. \* tmp \* face\_norm[2];

particle\_velocity\_seen[2] -= 2. \* tmp \* face\_norm[2];

}

else {

int stat\_type = cs\_lagr\_stat\_type\_from\_attr\_id(CS\_LAGR\_VELOCITY\_SEEN);

cs\_field\_t \*var\_vel\_part =

cs\_lagr\_stat\_get\_moment( stat\_type, CS\_LAGR\_MOMENT\_VARIANCE, 0, -1);

particle\_velocity\_seen[0] -= 2. \* tmp \* face\_norm[2]

\* (-pow(ustar,2)) / extra->cvar\_rij->vals[iprev][6 \* cur\_cell\_id + 2];

particle\_velocity\_seen[1] -= 2. \* tmp \* face\_norm[2]

\* (-pow(ustar,2)) / extra->cvar\_rij->vals[iprev][6 \* cur\_cell\_id + 2];

particle\_velocity\_seen[2] -= 2 \* tmp \* face\_norm[2];

}

}

*Test cases:*

Here, the best verification case is the channel flow simulation described in [3]. The only issue is that we still have an inconsistency between the Eulerian calculation and the Lagrangian one. As a result, the comparison between Eulerian and Lagrangian values is not perfect yet (see Figure 8 in [3]).

The Eulerian calculation still needs fixing for the Rotta model.

Due to this issue, the best practice is probably to compare the numerical results obtained with a specular rebound to those obtained with the wall-boundary condition.

*Future developments:*

Several future developments are need on the GUI since there are currently a number of issues, among which:

* Boundary conditions:

When a face is identified as a symmetry for the fluid phase, the boundary condition for the particle phase should be either ‘mirror’ or ‘outlet’ (by default a mirror).

When a face is identified as inlet for the fluid phase, then it can be an inlet / outlet for the particle (by default an inlet).

When a face is identified as a wall for the fluid phase, then the user should have a choice between inlet (to be kept?), rebound or deposition (then further ask if it is with elimination or sticking).

* The wall function is defined inside the turbulence model panel whereas the user has not yet defined whether there is a wall or not !

This should rather be asked during the boundary condition, if there is a wall boundary condition for the fluid.

## Point 3: Extra terms to the SDEs

*Aim:*

The aim of these developments is to add external terms to the SDEs described previously.

*Underlying equations and implementation:*

Jean-Pierre is going to provide a pdf document describing the additional terms to be implemented (including the gradient of TL\*).

To compute the gradient of TL\*, an extra statistics for the fluid Lagrangian timescale can be activated (see for instance user\_examples/cs\_user\_lagr\_model.c where a statistics can be activated using the function cs\_lagr\_stat\_activate).

*Test cases:*

The test case here will be the point-source dispersion (see Point 1). The idea is to check that the modifications do not change the results obtained.

## Point 4: extra Poisson correction

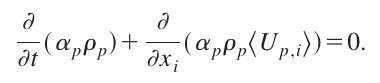
*Aim:*

The aim of these developments is to correct the pressure correction step performed at the end of the Lagrangian module.

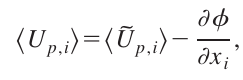
*Underlying equations and implementation:*

A pressure correction step is added at the end of the Lagrangian module to ensure that the particle stochastic model is consistent with the Eulerian model [4].

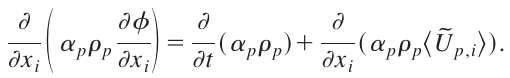
With the definition of the mean particle velocity field given previously, the corresponding particle continuity equation is (density is constant):



For each time step in the Lagrangian solver, the mean fields αp and 〈**Up**〉 are computed from the particle location and velocity. Here, we propose to modify particle velocities (not locations) so as to enforce the mean continuity constraint, by adding a pressure-correction field as a potential Φ. The corrected particle velocity field is then



where Φ is calculated from the Poisson equation



This mean velocity correction term is then applied to each particle velocity

*Modifications to be done:*

Here, the first step is to check what is now inside the routine cs\_lagr\_poisson.c to see if it corresponds to the previous formula. If not, we need to see what is wrong and make appropriate corrections (probably all-together).

*Test cases:*

Hercule test case [4] ?

## Point 5: test cases

Apart from the previously described test cases, a few other test cases need to be either updated or re-created. This includes the following test cases: Anarsson, Hercule, the wall-jet and the cyclone.

## Point 6: GUI and data model

During the meeting, we have discussed some of the possible modifications to the GUI. However, since it requires deep knowledge of the model inside Code\_Saturne, it was decided to keep the discussion on the data model and global modifications of the Lagrangian module inside the GUI for later (so that P. Asproulis can follow and provide useful inputs).

Besides, due to the tremendous amount of work related to such modifications, the idea is to make a tentative proposal for such a data model (and it can look like in the GUI) only in the final report.

# Future possible developments:

In the following, we list some developments that need to be done (within future collaborations):

* Develop a fractional time-step algorithm for particles.

The idea is the following: when a particle crosses an interface and changes cell, the particle is placed at the interface and the particle motion during the time remaining is computed with the values of the fluid inside this new cell. This is repeated every time a particle change cell during the time step until each particle has travelled for a time equal to the time step (or stopped before by depositing to a wall or going out through an outlet face).

# Actions:

Todo

* Next meeting using zoom?
* J.P. Minier send N. Tonnello and P. Asproulis a pdf document describing the theory of point 3.

Done

* C. Henry send the detailed notes to N. Tonnello and P. Asproulis.

Bibliography

1. E. Peirano, S. Chibbaro, J. Pozorski, J.-P. Minier (2006). Progress in Energy and Combustion Science, Vol. 32, pp. 315-371.
2. J.-P. Minier, E. Peirano, S. Chibbaro (2003). Monte Carlo Methods and Applications, Vol. 9, pp. 93-133.
3. M. Bahlali, C. Henry, B. Carissimo (2019). Boundary-Layer Meteorology, Vol. , pp. 1-22.
4. J.-P. Minier, E. Peirano, S. Chibbaro (2004), Physics of Fluids, Vol. 16, pp. 2419-2431.
5. S.B. Pope, Turbulent Flows, Cambridge University Press