Modification of Stochastic Model in Lagrangian Tracking Method
for the course
CFD with OpenSource Software

Jundi Xu

Naval Architecture and Ocean Engineering,
Chalmers University of Technology,
Gothenburg, Sweden

2016-12-05
Agenda

1. Background and Motivation
   - Introduction
   - Lagrangian Particle Tracking
   - Improvement

2. Insight of Solvers
   - Solvers
   - Tracking Mechanism
   - Stochastic Model

3. Modification of Solver and Stochastic Model
   - Create New Solver
   - Create New Stochastic Model

4. Test case
   - Primitive preparation
   - Test icoUncoupledKinematicNocollisionFoam with myStochasticModel
   - Result
Lagrangian and Eulerian

- **Lagrangian**
- **Eulerian**
Different Phases

1. **One-way coupling**, highly dilute dispersed flows, \( \alpha_p \leq 10^{-6} \),
2. **Two-way coupling**, dilute dispersed flows, \( 10^{-3} < \alpha_p < 10^{-6} \),
3. **Two-way coupling**, dilute dispersed flows, \( 10^{-3} < \alpha_p < 10^{-6} \),
4. **Four-way coupling**, dense dispersed flows, \( \alpha_p \geq 10^{-3} \).
Fundamental Assumption: For the isothermal, incompressible, turbulent flow of a Newtonian fluid, the continuity and momentum equation of RANS as follows:

\[ \nabla \cdot \overline{U_c} = 0 \]

\[ \frac{\partial \overline{U_c}}{\partial t} + \rho_c (\overline{U_c} \cdot \nabla) \overline{U_c} = -\nabla \bar{p} + \mu \Delta \overline{U_c} - \nabla \cdot \tau^{RS} + \bar{f}_D \]

Forces Acting on Particles: A particle P is defined as: \( x_P, D_P, U_P, \rho_P, \) and \( m_P = \frac{1}{6} \rho_P \pi D_P^3 \). The force balance:

\[ \frac{dx_P}{dt} = U_P \]

\[ m_P \cdot \frac{dU_P}{dt} = \sum F_i \]

\[ \sum F_i = F_D + F_G = F_D + m_P g \]
Drag force, Relaxation time $\tau_P$, Drag coefficient $C_D$:

\[ F_D = m_P \frac{U_c - U_P}{\tau_P} \]

\[ \tau_P = \frac{3}{4} \frac{\rho_P D_P}{\rho_c C_D U_c - U_P} \]
Lagrangian particle tracking (LPT)

In RANS model, the instantaneous variables of turbulent flow is to be decomposed into a mean value and a fluctuating value, i.e.

\[ U_i = \bar{U}_i + U'_i \]
\[ p = \bar{p} + p' \]

The fluctuating velocity \( U' \) has to be estimated by stochastic model in order to model turbulent dispersion phenomenon. Discrete Random Walk Model (DRW) assume that the eddies are isotropic:

\[ \bar{U}'_1 = \bar{U}'_2 = \bar{U}'_3 \]

This assumption is not true instantaneously. This equation introduces a large over-prediction for the wall normal component \( U'_2 \), y-dir, for the boundary layer in turbulence model.
Improvement

For inside the boundary layer ($y^+ < 80$): Taking into account anisotropy and using DNS data in channel data results, three functions $f_1$, $f_2$ and $f_3$ introduced for the determination of the local fluctuating velocity:

$$U_1' = f_1 N_1 \sqrt{\frac{2k}{3}}, \quad U_2' = f_2 N_2 \sqrt{\frac{2k}{3}}, \quad U_3' = f_2 N_2 \sqrt{\frac{2k}{3}}$$

where $N_1$, $N_2$ and $N_3$ are random numbers generated from a Gaussian probability density function and the functions are expressed as:

$$f_1 = 1 + 0.285(y^+ + 6) \exp[-0.455(y^+ + 6)^{0.53}],$$
$$f_2 = 1 - \exp(-0.02y^+),$$
$$f_3 = \sqrt{3 - f_1^2 - f_2^2},$$

as $y^+$:

$$y^+ = \frac{y \cdot \sqrt{\nu \cdot (\partial u/\partial y)_{y=0}}}{\nu}$$
Agenda

1. Background and Motivation
   - Introduction
   - Lagrangian Particle Tracking
   - Improvement

2. Insight of Solvers
   - Solvers
   - Tracking Mechanism
   - Stochastic Model

3. Modification of Solver and Stochastic Model
   - Create New Solver
   - Create New Stochastic Model

4. Test case
   - Primitive preparation
   - Test icoUncoupledKinematicNocollisionFoam with myStochasticModel
   - Result
Solvers

**Target:** Find a promising solver which could be modified the stochastic model for one-way coupling without collision.

```plaintext
sol
cd lagrangian
```

**Methodology:** Merge two solvers together for a new one which only for `singlePhaseTransportModel` and `basicKinematicCloud`.

```plaintext
# include "fvCFD.H"
# include "singlePhaseTransportModel.H"
# include "turbulentTransportModel.H"
# include "basicKinematicCollidingCloud.H"
```

```plaintext
# include "fvCFD.H"
# include "psiThermo.H"
# include "turbulentTransportModel.H"
# include "basicKinematicCloud.H"
```
Comparison (1/4):

basicKinematicCollidingCloud.H and basicKinematicCloud.H

```c
#include "Cloud.H"
#include "KinematicCloud.H"
#include "CollidingCloud.H"
#include "basicKinematicCollidingParcel.H"
```

```c
#include "Cloud.H"
#include "KinematicCloud.H"
#include "CollidingCloud.H"
#include "basicKinematicParcel.H"
```
Comparison (2/4):

basicKinematicCollidingCloud.H and basicKinematicCloud.H

```cpp
basicKinematicCloud.H:

```cpp
41 namespace Foam 
42 {
43 typedef KinematicCloud<Cloud<basicKinematicParcel>> basicKinematicCloud;
44 }
```

```cpp
basicKinematicCollidingCloud.H:

```cpp
42 namespace Foam 
43 {
44 typedef CollidingCloud
45   <KinematicCloud<Cloud<basicKinematicCollidingParcel>>> basicKinematicCollidingCloud;
46 }
```
Comparison (3/4):

basicKinematicCollidingParcel.H and basicKinematicParcel.H

```
#include "contiguous.H"
#include "particle.H"
#include "KinematicParcel.H"
#include "CollidingParcel.H"
```

John Doe

Modification of Stochastic Model in Lagrangian Tracking Method

2016-12-05
Comparison (4/4):

**basicKinematicCollidingParcel.H, basicKinematicParcel.H and CollidingParcel.H**

```cpp
typedef CollidingParcel<KinematicParcel<particle>> basicKinematicCollidingParcel;
```

```cpp
typedef KinematicParcel<particle> basicKinematicParcel;
```

**CollidingParcel.H**

Description

Wrapper around kinematic parcel types to add collision modelling;
Tracking Mechanism (1/3)

```
icoUncoupledKinematicParcelFoam.C:

63       while (runTime.loop())
64       {
65           Info<< "Time = " << runTime.timeName() << nl << endl;
66           Info<< "Evolving " << kinematicCloud.name() << endl;
67           laminarTransport.correct();
68           mu = laminarTransport.nu()*rhoInfValue;
69           kinematicCloud.evolve();
70           runTime.write();
71           Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
72               << " ClockTime = " << runTime.elapsedClockTime() << " s"
73               << nl << endl;
74       }
```

The core member function is `kinematicCloud<evolve>`. 
Comparison (1/4):

basicKinematicCollidingCloud.H and basicKinematicCloud.H

```c
#include "Cloud.H"
#include "KinematicCloud.H"
#include "CollidingCloud.H"
#include "basicKinematicCollidingParcel.H"
```

```c
#include "Cloud.H"
#include "KinematicCloud.H"
```
Tracking Mechanism (2/3)

kinematicCloud<evolve>:

1. Line 683: evolve calls solve(td), td is reference of TrackData;
2. Line 89-126: solve(td) calls evolveCloud(td);
3. Line 174 -215: evolveCloud(td) is if-else loop:
   - Line 183-215, when solution_.transient() is true, line 202, 204 will be executed, otherwise,
   - Line 213 CloudType::move will be performed.
4. CloudType is equivalent to Cloud<basicKinematicParcel>, investigate the member function CloudType::move in Cloud.C
   - Line 242-378, ”while loop”: how the particles transfer through move member function;
   - Line 251-295, shows how to move the particle
5. Line 197: injectors_.inject(td) injects particles into flow.
6. Line 202: td.cloud().motion(td) calls motion function in class KinematicParcel.
7. Line 204: stochasticCollision().update(solution_.trackTime()) updates the model under particles collision.
Tracking Mechanism (3/4)

KinematicParcel.H:

```cpp
    281   // Calculate new particle velocity
    282   template<class TrackData>
    283   const vector calcVelocity
    284   (  
    285       TrackData& td,
    286       const scalar dt,       // timestep
    287       const label celli,    // owner cell
    288       const scalar Re,      // Reynolds number
    289       const scalar mu,      // local carrier viscosity
    290       const scalar mass,    // mass
    291       const vector& Su,     // explicit particle momentum source
    292       vector& dUTrans,     // momentum transfer to carrier
    293       scalar& Spu          // linearised drag coefficient
    294   ) const;
```
Tracking Mechanism (4/4)

KinematicParcel.C:
Eventually, `td.cloud().dispersion().update` is called to update:

```c
69     Uc_ = td.cloud().dispersion().update
70     ( 
71         dt, // time step
72         celli, // cell number
73         U_,  // particle mean velocity
74         Uc_, // flow mean velocity
75         UTurb_, // flow fluctuating velocity in turbulence
76         tTurb_  // time fluctuating velocity in turbulence
77     );
```
Stochastic Model

stochasticCollision is called to update the model if the collision happens, but the collision scenarios does not happen in our case; Instead, StochasticDispersionRAS model is a promising candidate to update the model, with $UTurb = \zeta d\sigma$

```
97  const scalar sigma = sqrt(2*k/3.0);
102 const scalar theta = rnd.sample01<scalar>()*twoPi;
103 const scalar u = 2*rnd.sample01<scalar>() - 1;
105 const scalar a = sqrt(1 - sqr(u));
106 const vector dir(a*cos(theta), a*sin(theta), u);
108 UTurb = sigma*mag(rnd.GaussNormal<scalar>())->dir;
```

Line 97: calculate $\sigma$ with isotropic turbulence assumption, introduce a large over-prediction for the wall normal component $v'$, y-dir,: 

$$\sigma = \sqrt{\frac{2k}{3}}, \sigma = \sqrt{w'^2} = \sqrt{v'^2} = \sqrt{w'^2}$$
Agenda

1. Background and Motivation
   - Introduction
   - Lagrangian Particle Tracking
   - Improvement

2. Insight of Solvers
   - Solvers
   - Tracking Mechanism
   - Stochastic Model

3. Modification of Solver and Stochastic Model
   - Create New Solver
   - Create New Stochastic Model

4. Test case
   - Primitive preparation
   - Test icoUncoupledKinematicNocollisionFoam with myStochasticModel
   - Result
Create New Solver–icoUncoupledKinematicNocollisonFoam

Prototype: icoUncoupledKinematicParcelFoam.C;
Header file: #include "basicKinematicCollidingCloud.H" replaced by #include "basicKinematicCloud.H" (no collision occurs).

```
35 #include "fvCFD.H"
36 #include "singlePhaseTransportModel.H"
37 #include "turbulentTransportModel.H"
38 #include "basicKinematicCloud.H"
```

Make/files:

```
icoUncoupledKinematicNocollisionFoam.C
EXE = $(FOAM_USER_APPBIN)/icoUncoupledKinematicNocollisionFoam
```

Then compile the new solver.
New Stochastic Model – myStochasticModel (1/8)

Prototype: StochasticDispersionRAS;
Renamed as myStochasticModel.

Modification as:
- change the class name, from StochasticDispersionRAS to myStochasticModel using sed command in .H .C
- declare the object yP_ as class volScalarField in .H:
  volScalarField yP_;
New Stochastic Model – myStochasticModel (2/8)

- Initialize the object \texttt{yP\_} in \texttt{.C}:

```cpp
template<class CloudType>
Foam::myStochasticModel<CloudType>::myStochasticModel
(const dictionary& dict,
 CloudType& owner):
DispersionRASModel<CloudType>(dict, owner),
yP_
{
  IOobject
  ("yP",
   this->owner_.mesh().time().timeName(),
   this->owner_.mesh(),
   IOobject::NO_READ,
   IOobject::AUTO_WRITE
  ),
  this->owner_.mesh(),
dimensionedScalar("yP", dimless, 0.0)
}
```
New Stochastic Model – myStochasticModel (3/8)

- initialize the object `yP_` in `.C`:

```cpp
template<class CloudType>
Foam::myStochasticModel<CloudType>::myStochasticModel
(
    const myStochasticModel<CloudType>& dm
)
:
    DispersionRASModel<CloudType>(dm),
    yP_(dm.yP_)
{}```
New Stochastic Model – myStochasticModel (4/8)

- Setup the variables will be used in $y^+$ calculation:
  - obtain the turbulence model:

```
//setup the patches
const fvPatchList& patches = this->owner().mesh().boundary();
const objectRegistry& obr = this->owner().mesh();
const word turbName = IOobject::groupName(
  turbulenceModel::propertiesName,
  this->owner().U().group()
);
const turbulenceModel& turbModel =
  obr.lookupObject<turbulenceModel>(turbName);
```
New Stochastic Model – myStochasticModel (5/8)

- Setup the variables will be used in $y^+$ calculation:
  - use `forAll` loop for the all patches:

```cpp
forAll(patches, patchi)
{
  // the main code for calculation for turbulent fluctuating velocity UTurb
}
```

- obtain the current Patch

```cpp
// obtain the current Patch
const fvPatch& currPatch = patches[patchi];
```
New Stochastic Model – myStochasticModel (6/8)

- Setup the variables will be used in $y^+$ calculation:
  - use if loop for calculation;
  - setup the variables for $y_{Plus}$ calculation;

```c++
// calculation
if (typeid(currPatch) == typeid(wallFvPatch))
{
const scalarField& y_ = turbModel.y()[patchi];
const fvPatchVectorField& Uw =
  turbModel.U().boundaryField()[patchi];
const tmp<scalarField> tnuw = turbModel.nu(patchi);
const scalarField& nuw = tnuw();
```
Setup the variables will be used in $y^+$ calculation:

- calculate the $y^+$ for the current patch;
- use `const_cast` to release the `const` limitation

Calculate $y^+$;

```c++
// yPlus for the current patch
fvPatchField<scalar>& yPpatch = 
  const_cast<fvPatchField<scalar>&>(yP_.boundaryField()[patchi]);

yPpatch = (y_*sqrt(nuw*mag(Uw.snGrad())))/nuw;
```
New Stochastic Model – myStochasticModel (8/8)

- update the turbulence fluctuating velocity for particle when $y^+ < 80$:

```cpp
const scalar sigma = sqrt(2*k/3.0);
const scalar theta = rnd.sample01<scalar>()*twoPi;
const scalar u = 2*rnd.sample01<scalar>() - 1;
const scalar a = sqrt(1 - sqr(u));

f_x = 1 + 0.285*(yPpatch[patchi]+6)*exp(-0.455*pow(yPpatch[patchi]+6, 0.53));
 htonl  f_y = 1 - exp(-0.02*yPpatch[patchi]);
 htonl  f_z = sqrt(3 - sqr(f_x) - sqr(f_y));
// update the turbulence fluctuating velocity for particle

dir_x = f_x*a*cos(theta);
dir_y = f_y*a*sin(theta);
dir_z = f_z*u;
const vector dir(dir_x, dir_y, dir_z);
UTurb = sigma*dir;
```
Compile New Class myStochasticModel (1/3)

Make/files:

PARCELS=parcels
DERIVEDPARCELS=$(PARCELS)/derived

KINEMATICPARCEL=$(DERIVEDPARCELS)/basicKinematicParcel
$(KINEMATICPARCEL)/makeBasicKinematicParcelSubmodels.C

LIB = $(FOAM_USER_LIBBIN)/liblagrangianTurbulence

The compile file makeBasicKinematicParcelSubmodels.C:

26 #include "basicKinematicCloud.H"
27 #include "makeParcelTurbulenceDispersionModels.H"

31 namespace Foam
32 {
33     makeParcelTurbulenceDispersionModels(basicKinematicCloud);
34 }
Compile New Class myStochasticModel (2/3)

`makeParcelTurbulenceDispersionModels.H` includes the options under sub-models for `DispersionModel`;

```cpp
#include "GradientDispersionRAS.H"
#include "StochasticDispersionRAS.H"
#include "myStochasticModel.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#define makeParcelTurbulenceDispersionModels(CloudType) 
  typedef Foam::CloudType::kinematicCloudType kinematicCloudType;
defineNamedTemplateTypeNameAndDebug
  ( 
    Foam::DispersionRASModel<kinematicCloudType>, 
    0 
  );
makeDispersionModelType(GradientDispersionRAS, CloudType);
makeDispersionModelType(StochasticDispersionRAS, CloudType);
makeDispersionModelType(myStochasticModel, CloudType);
```

Jundi Xu
Modification of Stochastic Model in Lagrangian Tracking Method
2016-12-05
Compile New Class myStochasticModel (3/3)

Until now, the modification of class myStochasticModel is done. Go to compile to add the new model as a submodel of DispersionModel:

wclean
wmake
**Agenda**

1. **Background and Motivation**
   - Introduction
   - Lagrangian Particle Tracking
   - Improvement

2. **Insight of Solvers**
   - Solvers
   - Tracking Mechanism
   - Stochastic Model

3. **Modification of Solver and Stochastic Model**
   - Create New Solver
   - Create New Stochastic Model

4. **Test case**
   - Primitive preparation
   - Test icoUncoupledKinematicNocollisionFoam with myStochasticModel
   - Result
Before implement the new solver with the new model, a steady flow with \( k \) and \( \varepsilon \) value should be obtained previously. Therefore, \( k-\varepsilon \) model will be implemented using \texttt{simpleFoam} in a pipe.

- One tutorial case under \texttt{ras} directory will be used as a base case for the modification;
- Create a pipe;
- Modification of the setting;
- Obtain a steady flow
Create a Pipe

The pipe model is created by Star-CCM+, then use ccm26ToFoam to convert into OpenFOAM, the profile of the pipe:

**Radius:** 0.01m  
**Length:** 1.5m

**Mesh stats**
- **points:** 7469  
- **faces:** 20760  
- **internal faces:** 18456  
- **cells:** 6688  
- **faces per cell:** 5.86364  
- **boundary patches:** 3  
- **point zones:** 0  
- **face zones:** 0  
- **cell zones:** 0

**Patch**  | **Faces** | **Points**
--- | --- | ---
shroud | 2128 | 2156  
outlet | 88 | 97  
inlet | 88 | 97
Modification in the 0 directory (1/2)

0/p:

```plaintext
dimensions [0 2 -2 0 0 0 0];
internalField uniform 0;
boundaryField
{
    inlet
    {
        type zeroGradient;
    }
    outlet
    {
        type fixedValue;
        value uniform 0;
    }
    shroud
    {
        type zeroGradient;
    }
}
```

0/U:

```plaintext
dimensions [0 1 -1 0 0 0 0];
internalField uniform (0 0 0);
boundaryField
{
    inlet
    {
        type fixedValue;
        value uniform (0 0 -12);
    }
    outlet
    {
        type zeroGradient;
    }
    shroud
    {
        type fixedValue;
        value uniform (0 0 0);
    }
}
```
Modification in the 0 directory (2/2)

0/k:

```
dimensions [0 2 -2 0 0 0 0];
internalField uniform 0.0006;
boundaryField
{
    inlet
    {
        type fixedValue;
        value uniform 0.0006;
    }
    outlet
    {
        type zeroGradient;
    }
    shroud
    {
        type kqRWallFunction;
        value uniform 0.00375;
    }
}
```

0/epsilon:

```
dimensions [0 2 -3 0 0 0 0];
internalField uniform 0.00754;
boundaryField
{
    inlet
    {
        type zeroGradient;
    }
    outlet
    {
        type zeroGradient;
    }
    shroud
    {
        type epsilonWallFunction;
        value uniform 0.00754;
    }
}
```
Modification in the constant directory

To set similar conditions at those used by A. Dehbi, the transportProperties should be changed for the same Re number.

For flow in a pipe or tube, the Reynolds number is generally defined as:

\[
Re = \frac{QD_H}{\nu A} = \frac{vD}{\nu} \approx 12000
\]

```plaintext
rhoInf [1 -3 0 0 0 0 0] 1.2;
transportModel Newtonian;
nu [0 2 -1 0 0 0 0] 1e-05;
```
Modification in the system directory (1/2)

system/controlDict

```
application    simpleFoam;
startFrom      startTime;
startTime      0;
stopAt         endTime;
endTime        600;
deltaT         1;
writeControl   timeStep;
writeInterval  50;
purgeWrite     0;
writeFormat    ascii;
writePrecision 6;
writeCompression off;
timeFormat     general;
timePrecision  6;
runTimeModifiable true;
```
Modification in the system directory (2/2)

system/fvScheme:

```plaintext
ddtSchemes
{
    default steadyState;
}
gradSchemes
{
    default Gauss linear;
}
divSchemes
{
    default none;
    div(phi,U) bounded Gauss linearUpwind grad(U);
    div(phi,k) bounded Gauss upwind;
    div(phi,epsilon) bounded Gauss upwind;
    div(phi,nuTilda) Gauss limitedLinear 1;
    div((nuEff*dev2(T(grad(U))))) Gauss linear;
}
```
Run the simulation

Obtain the steady flow:

ccm26ToFoam mesh.ccm

simpleFoam

Remain the last time directory, 600, deleted the rest;
Test icoUncoupledKinematicNocollisionFoam with myStochasticModel

Copy the default setting from another test case:

```bash
tut
cp -r lagrangian/icoUncoupledKinematicParcelFoam/hopper/hopperInitialState
  ➜  $FOAM_RUN/caseStudy
cd  $FOAM_RUN/myPipe
cp  ../caseStudy/constant/kinematicCloudPositions constant
    cp  ../caseStudy/constant/kinematicCloudProperties constant
    cp  ../caseStudy/constant/g constant
    cp  ../caseStudy/system/controlDict system
    cp  ../caseStudy/system/fvSchemes system
    cp  ../caseStudy/system/fvSolution system
```
Modification under 0 directory

Change the directory name: 600 to be 0 including the files;

    cd 0
    sed -i s/"600"/"0"/g epsilon
    sed -i s/"600"/"0"/g k
    sed -i s/"600"/"0"/g nut
    sed -i s/"600"/"0"/g p
    sed -i s/"600"/"0"/g phi
    sed -i s/"600"/"0"/g U
Modification under constant directory

Create some random particle positions in `kinematicCloudPositions`;
Change the simulation setting in `kinematicCloudProperties` as:

```plaintext
dispersionModel myStochasticModel;
patchInteractionModel none;
surfaceFilmModel none;
stochasticCollisionModel none;
collisionModel none;

inlet
{
  ...
}
outlet ...
shroud ...
```
Result: In Transition
Comparison: myStochastisModel vs StochasticDispersionRAS

UTurb (magnitude):

![UTurb Magnitude Graph](image1)

![UTurb Magnitude Graph](image2)
Comparison: myStochastisModel vs StochasticDispersionRAS

UTurb-Y (magnitude):
Thank you!

Questions?