Implementation of an air-entrainment model in interFoam for the course CFD with OpenSource Software

Silje Kreken Almeland

Department of Civil and Environmental Engineering,
Norwegian University of Science and Technology (NTNU),
Trondheim, Norway

January 2, 2019

Air entrainment modelling

The aim of this tutorial is to show how a sub-grid model for predicting the amount of air entrainment in an air-water system can be implemented based on `interFoam`.

Air entrainment

When the turbulence is high enough in free-surface flow, air will be entrained in the water phase and transported with the water flow

When the grid is not fine enough

- the processes at the surface will not be captured
  $\Rightarrow$ amount of air in water phase underestimated

interFoam

- Solves a single set of mass- and momentum equations

$$\nabla \boldsymbol{U} = 0$$

$$\frac{\partial \rho \boldsymbol{U}}{\partial t} + \nabla \cdot (\rho \boldsymbol{U}\boldsymbol{U}) = -\nabla p^* + \mathbf{g} \cdot \mathbf{x} \nabla \rho + \nabla \cdot \boldsymbol{\tau} + \mathbf{f}$$

- interFoam uses a VOF method with a compression term to capture the interface

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\alpha \boldsymbol{U}) + \nabla \cdot [\boldsymbol{U}_r \alpha (1 - \alpha)] = 0$$

where $\boldsymbol{U}_r = \boldsymbol{U}_1 - \boldsymbol{U}_2$ is the relative velocity

The subgrid model

The implementation shown in this tutorial will represent a try on reproduce the model introduced in Flow-3D by Hirt (2003)

$$\frac{\delta V}{\delta t} = C_{air} A_s \sqrt{2 \frac{P_t - P_d}{\rho}}$$

$$P_t = \rho k$$

$$P_d = \rho g_n L_T + \frac{\sigma}{L_T}$$

$$L_T = C_\mu \sqrt{\frac{3}{2}} \frac{k^{3/2}}{\epsilon}$$

The subgrid model - requirements

Our air-entrainment model requires the knowledge of:

- where the surface is located
- the normal of the surface
- the surface tension
- turbulence variable (k and ϵ)
- Mesh properties

`interFoam` - *immiscibleIncompressibleTwoPhaseMixture*

Integration of subgrid model

```
class immiscibleIncompressibleTwoPhaseMixture
:
    public incompressibleTwoPhaseMixture,
    public interfaceProperties
```

Takes care of transport properties and interface properties

```
//- Correct the transport and interface properties
virtual void correct()
{
    incompressibleTwoPhaseMixture::correct();
    interfaceProperties::correct();
}
```

Integration of subgrid model

`interFoam` - *immiscibleIncompressibleTwoPhaseMixture*
$\Rightarrow$ `interAirEntFoam` - *airEntrainmentTwoPhaseMixture*

- And add calculation of the air entrainment model
- Implement functions for the model variables and ρ_{mix}

Variable	Function in airEntrainmentTwoPhasMixture
L_T	volScalarField Lt(k,ϵ)
P_t	volScalarField Pt(k)
P_d	volScalarField Pd(k,ϵ)
S_g	volScalarField calcSource(U,k,ϵ)
ρ_{mix}	void updateRhomix()

interFoam → interAirEntFoam

Adding

$$\frac{\delta V}{\delta t} = C_{air} A_s \sqrt{2 \frac{P_t - P_d}{\rho}}$$

As an explicit source term

$$\frac{\partial \alpha_l}{\partial t} + \nabla \cdot (\alpha_l \mathbf{u}) + \nabla \cdot [\mathbf{u}_c \alpha_l (1 - \alpha_l)] = S_g \quad (1)$$

where

$$S_g = \frac{\frac{\delta V}{\delta t}}{mesh.V()}$$

interFoam - solution algorithm

Basic structure of the solver

- Initiate fields
- Starting runtime loop
  - Calculating time step
  - Iterating over the PIMPLE Loop nOuterCorrectors times
    - Solving the α-equation (VOF and MULES)
    - Calculate the interface and transport properties
    - Solving the momentum equation
    - PISO Loop over pressure equation
  - Calculate the turbulent properties

interFoam → interAirEntFoam

This section will provide the steps in the implementation of the air entrainment model named, interAirEntFoam.

```
cd $WM_PROJECT_USER_DIR/
mkdir -p applications/solvers/multiphase/
cd !$
cp -r $FOAM_SOLVERS/multiphase/interFoam/ .
```

Rename the solver and the files to interAirEntFoam

```
mv interFoam interAirEntFoam
cd !$
mv interFoam.C interAirEntFoam.C
```

interFoam → interAirEntFoam

Remove folders not needed

```
rm -r interMixingFoam overInterDyMFoam
```

Rename within the files

```
sed -i s/interFoam/interAirEntFoam/g Make/files
sed -i s/interFoam/interAirEntFoam/g interAirEntFoam.C
```

Change the name of the excutable path

```
sed -i s/APPBIN/USER_APPBIN/g Make/files
```

Then you have to update the Make/options file, to include the desired libraries

```
EXE_INC = \
  -I$(FOAM_SOLVERS)/multiphase/VoF \
```

Copy immiscibleIncompressibleTwoPhaseMixture

Find the location of immiscibleIncompressibleTwoPhaseMixture and copy it to your folder

```
find $WM_PROJECT_DIR -name  immiscibleIncompressibleTwoPhaseMixture
cp -r $FOAM_SRC/transportModels/immiscibleIncompressibleTwoPhaseMixture .
```

Change the name of the model and its files

```
mv immiscibleIncompressibleTwoPhaseMixture airEntrainmentTwoPhaseMixture
cd airEntrainmentTwoPhaseMixture
mv immiscibleIncompressibleTwoPhaseMixture.C airEntrainmentTwoPhaseMixture.C
mv immiscibleIncompressibleTwoPhaseMixture.H airEntrainmentTwoPhaseMixture.H
sed -i s/immiscibleIncompressible/airEntrainment/g Make/*
sed -i s/immiscibleIncompressible/airEntrainment/g airEntrainmentTwoPhaseMixture.*
```

airEntrainmentTwoPhaseMixture

And rename in the solver to call the new library

```
cd ..
sed -i s/immiscibleIncompressible/airEntrainment/g createFields.H
sed -i s/immiscibleIncompressible/airEntrainment/g interAirEntFoam.C
```

At this point you have a local version of the `immiscibleIncompressibleTwoPhaseMixture` named `airEntrainmentTwoPhaseMixture`

$\Rightarrow$ Change the name of the library path in Make/files accordingly:

```
sed -i s/LIBBIN/USER_LIBBIN/g airEntrainmentTwoPhaseMixture/Make/files
```

airEntrainmentTwoPhaseMixture

Change relative paths in
airEntrainmentTwoPhaseMixture/Make/options

```
EXE_INC = \
    -I$(LIB_SRC)/transportModels \
    -I$(LIB_SRC)/transportModels/incompressible/lnInclude \
    -I$(LIB_SRC)/transportModels/interfaceProperties/lnInclude \
    -I$(LIB_SRC)/transportModels/twoPhaseMixture/lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude
```

Compile to check that it works

```
wmake airEntrainmentTwoPhaseMixture
```

Link to airEntrainmentTwoPhaseMixture

Now you want your new solver `interAirEntFoam` to use this library

$\Rightarrow$ link to it in the Make/options file within the solver as

```
-L$(FOAM_USER_LIBBIN) \
-lairEntrainmentTwoPhaseMixture
```

Link to the new directory has to be linked to under EXE_INC

```
-IairEntrainmentTwoPhaseMixture \
```

Make sure that the name written in the Make/options file corresponds to the library it is written to

Test the model

Compile it

```
wmake airEntrainmentTwoPhaseMixture
wmake
```

When both the library and the solver compiles, you should copy the damBreak tutorial, and run it with your new solver:

```
cp -r $FOAM_TUTORIAL/multiphase/interFoam/RAS/damBreak/damBreak $FOAM_RUN
run
cd damBreak
sed -i s/interFoam/interAirEntFoam/g system/controlDict
./Allrun
```

Ready to implement

The solver should return the output of the source term

$$\frac{\delta V}{\delta t} = C_{air} A_s \sqrt{2 \frac{P_t - P_d}{\rho}}$$

This term is a function of several terms, that we will calculate in separate member functions within the model

$$\frac{\delta V}{\delta t} = f(P_t(k, \rho), P_d(L_T, k, g, \epsilon), \rho, C_{air}, A_s)$$

The implementations are done in `airEntrainmentTwoPhaseMixture.H` and `airEntrainmentTwoPhaseMixture.C`

Calculation of P_t

Calculate a density for the mixture, we do this by introducing a private member variable in `airEntrainmentTwoPhaseMixture.H`

```
// Private data
volScalarField rhomix_;
```

that we initiate in the constructor (in `airEntrainmentTwoPhaseMixture.C`)

```
rhomix_
(
    min(max(alpha1(), scalar(0)), scalar(1))*rho1_ +
    (scalar(1) - min(max(alpha1(), scalar(0)), scalar(1)))*rho2_
)
```

Calculation of P_t

Create an update function to update the mixture density each time we call the source term from the solver, in the header file

```
void updateRhomix();
```

We implement the calculation of this function in .C-file

```
void Foam::airEntrainmentTwoPhaseMixture::updateRhomix()
{
  rhomix_ = min(max(alpha1(), scalar(0)), scalar(1))*rho1_
          +(scalar(1) - min(max(alpha1(), scalar(0)),scalar(1)))*rho2_;
}
```

Then we are ready to implement P_t

Calculation of P_t

We declare the function for the perturbating term P_t in the .H-file

```
// - Calculates the pertubing component that makes the flow unstable
// - and returnes a volumScalarField
volScalarField Pt(const volScalarField k);
```

and implement it in the .C-file

```
Foam::volScalarField Foam::airEntrainmentTwoPhaseMixture::Pt(const
                                                    volScalarField k)
{
    updateRhomix();
    volScalarField Pt = k*rhomix_;
    return Pt;
}
```

P_d, L_T, and calcSource are implemented in a similar manner

Calculation of L_T

We declare the function in the header file

```
// Calculates Lt (the characteristic turbulent length scale)
volScalarField Lt(
                  const volScalarField k,
                  const volScalarField epsilon
);
```

And implements the function in the .C-file

```
Foam::volScalarField Foam::airEntrainmentTwoPhaseMixture::Lt(
     const volScalarField k,
     const volScalarField epsilon)
{
    dimensionedScalar dimCorr("dimCorr",dimLength/dimTime,1);
    volScalarField newepsilon = epsilon +
      dimensionedScalar("smallEpsilon",dimensionSet(0,2,-3,0,0,0,0),1e-10);
    volScalarField Lt = 0.09*sqr(1.5)*pow(k,3/2)/newepsilon*dimCorr;
    return Lt;
}
```

Additional member variables

The calculation of P_d needs the surface tension and gravity

$$P_d = \rho g_n L_T + \frac{\sigma}{L_T}$$

The calculation of the source term needs C_{air}

$$\frac{\delta V}{\delta t} = C_{air} A_s \sqrt{2 \frac{P_t - P_d}{\rho}}$$

we add these variables as private member variables in our class

```
// Private data
    const dimensionedScalar sigma_;
    uniformDimensionedVectorField g_;
    const dimensionedScalar Cair_;
```

Here we introduce a new variable type
$\Rightarrow$

```
#include "uniformDimensionedFields.H"
```

Additional member variables

We initialize these member variables in the constructor accordingly

```
sigma_
(
    "sigma", dimensionSet(1,0,-2,0,0,0,0), lookup("sigma")
),
Cair_
(
    "Cair", dimensionSet(0,0,0,0,0,0,0), lookup("Cair")
),
g_(
    IOobject
    (
        "g",
        U.time().constant(),
        U.mesh(),
        IOobject::MUST_READ_IF_MODIFIED,
        IOobject::NO_WRITE
    )
)
```

These variables are read and changed from the input case files

Calculation of P_d

declare the function in the header file

```
// Calculates the stabilizing surface force Pd
volScalarField Pd(
        const volScalarField k,
const volScalarField epsilon
);
```

And implements the function in the .C-file

Calculation of P_d

```
Foam::volScalarField Foam::airEntrainmentTwoPhaseMixture::Pd(
                                                    const volScalarField k,
                                                    const volScalarField epsilon)
{
    // Calculate surface normal
        // Cell gradient of alpha
    const volVectorField gradAlpha(fvc::grad(alpha1(), "nHat"));
    // Unit interface normal
    volVectorField nHat(gradAlpha/(mag(gradAlpha) + deltaN()));
    // Gravitation forces normal to surface
    volScalarField gn = g_ & nHat;
    volScalarField lLt = Lt(k,epsilon);
    // Calculates the stabilizing surface force Pd
    updateRhomix();
    volScalarField Pd = rhomix_*mag(gn)*lLt + sigma_/lLt;
    return Pd;
}
```

Calculation of P_d

The implementation of P_d make use of the namespace `fvc`

$\Rightarrow$ Include the fvCFD.H in airEntrainmentTwoPhaseMixture.H

```
#include "fvCFD.H"
```

and link to meshTools in airEntrainmentTwoPhaseMixture/Make/options as

```
-I$(LIB_SRC)/meshTools/lnInclude \


-lmeshTools \
```

Calculation of S_g

Finally the source term S_g can be calculated as a volume fraction term

- We declare the function in the header file

```
//- Calculates source term, returned as volum fraction rate
volScalarField calcSource(
        const volVectorField& U,
        const volScalarField k,
        const volScalarField epsilon
);
```

And implement the function in the .C-file

Calculation of S_g

```
Foam::volScalarField Foam::airEntrainmentTwoPhaseMixture::calcSource(
     const volVectorField& U,
     const volScalarField k,
     const volScalarField epsilon)
{
    Info << "calcSource" << endl;

    // Initialize the volume fraction rate term, as 0 or 1 for on interface
    volScalarField dvdt(nearInterface());
    volScalarField Afs(dvdt);

    // Estimate cell surface area as V^(2/3)
    forAll(Afs, cellI)
    {
        Afs[cellI] = pow(U.mesh().V()[cellI],0.67);
    }

    volScalarField Pdl = Pd(k,epsilon);
    volScalarField Ptl = Pt(k);
    updateRhomix();
```

Calculation of S_g

```
    // Calculate the volume fraction rate term
    forAll(dvdt, cellI)
    {
     if((Ptl[cellI] > Pdl[cellI])){
         scalar air2cell = dvdt[cellI]*Cair_.value()*Afs[cellI]*
          sqr(2*(Ptl[cellI]-Pdl[cellI])/rhomix_[cellI])
          /U.mesh().V()[cellI];
         dvdt[cellI] = -min(air2cell,0.5);
     }else{
         dvdt[cellI] = scalar(0.0);
     }
    }

    forAll(dvdt.boundaryField(), patchI)
    {
forAll(dvdt.boundaryField()[patchI], faceI )
{
    dvdt.boundaryFieldRef()[patchI][faceI] = scalar(0.0);
}
    }

    dimensionedScalar dimCorr("dimCorr",dimVelocity/dimLength,1);

    return dvdt*dimCorr;
}
```

Add source term to α_l-equation

Add source term to the α_l-equation as the explixit source term S_u in `alphaSuSp.H`. Creates this field in `createFields.H`

```
// Source term for visualization
volScalarField Su
(
    IOobject
    (
        "Su",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT, //NO_READ,
        IOobject::AUTO_WRITE //NO_WRITE
    ),
    mesh,
    dimensionedScalar("Su",dimVelocity/dimLength, 0)

);
```

Let it be updated for every time step by adding a call to `calcSource` from `alphaSuSp.H`

```
Su = mixture.calcSource(U,turbulence->k(),turbulence->epsilon());
```

Add source term to α_l-equation

The solution algorithm in `alphaEqn.H` solves Equation (1) successively using MULES compression. This results in a limited flux:

$$\phi_{\alpha,l} = \phi_{\alpha,l}^{UD} + \phi_{\alpha,l}^{lim,Corr}$$

where

$$\phi_{\alpha,l}^{lim,Corr} = \theta_l \cdot \phi_{\alpha,l}^{Corr} = \theta_l \cdot (\phi_{\alpha,l}^{HO} + \phi_{\alpha,l}^{C} - \phi_{\alpha,l}^{UD})$$

The MULES compression, that are included to make the interface sharp, is possible to turn off by the user by setting the user parameters accordingly (`MULESCorr` to `false` and `cAlpha` to 0). Then the flux is calculated to be

$$\phi_{\alpha,l} = \phi_{\alpha,l}^{HO}$$

which means that only the higher order fluxes, using the discretization scheme given for `div(phi,alpha)` in `divSchemes` in the `system/fvSchemes` input file, is used for calculating the fluxes of α_l.
If `MULESCorr` is set to `false` and `cAlpha` $\neq 0$, then $\phi_{\alpha,l}$ is calculated as:

$$\phi_{\alpha,l} = \theta_l \cdot (\phi_{\alpha,l}^{HO} + \phi_{\alpha,l}^{C})$$

Add source term to α_l-equation

S_u added in the calculation of the upwind flux if MULES is activated

```
if (MULESCorr)
{
    #include "alphaSuSp.H"

    fvScalarMatrix alpha1Eqn
    (
        (
            LTS
          ? fv::localEulerDdtScheme<scalar>(mesh).fvmDdt(alpha1)
          : fv::EulerDdtScheme<scalar>(mesh).fvmDdt(alpha1)
        )
      + fv::gaussConvectionScheme<scalar>
        (
            mesh,
            phiCN,
            upwind<scalar>(mesh, phiCN)
        ).fvmDiv(phiCN, alpha1)
    // - fvm::Sp(fvc::ddt(dimensionedScalar("1", dimless, 1), mesh)
    //           + fvc::div(phiCN), alpha1)
      ==
        Su + fvm::Sp(Sp + divU, alpha1)
    );

    alpha1Eqn.solve();
```

Add source term to α_l-equation

If MULES = `false`, S_u to added in MULES::explicitSolve

```
MULES::explicitSolve
(
    geometricOneField(),
    alpha1,
    phiCN,
    alphaPhi10,
    Sp,
    (Su + divU*min(alpha1(), scalar(1)))(),
    1,
    0
);
```

Vertical plunging jet

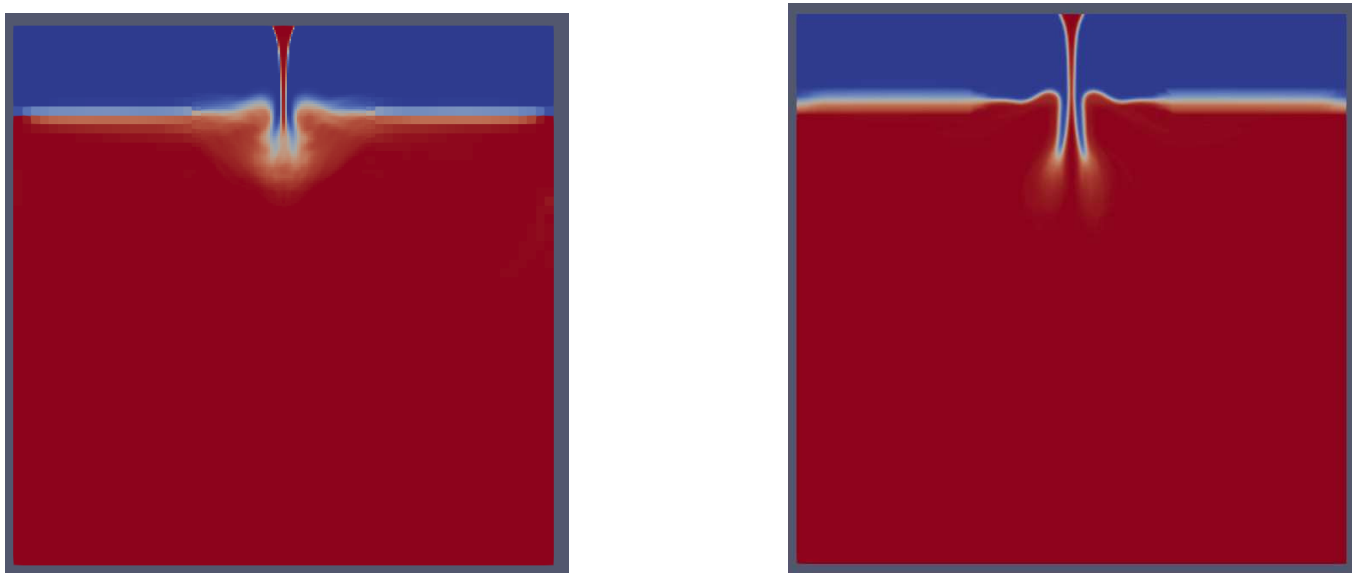
Download the test case to your user directory. Then go into the case directory and run the Allrun-script

```
cd $FOAM_RUN/jetIntoPool
./Allrun &
```

Then copy the case, to a new folder, and run it with `interFoam` to compare the behavior

```
cd ..
cp -r jetintopool jetintopool_interFoam
sed -i s/interAirEntFoam/interFoam/g system/controlDict
./Allrun &
```

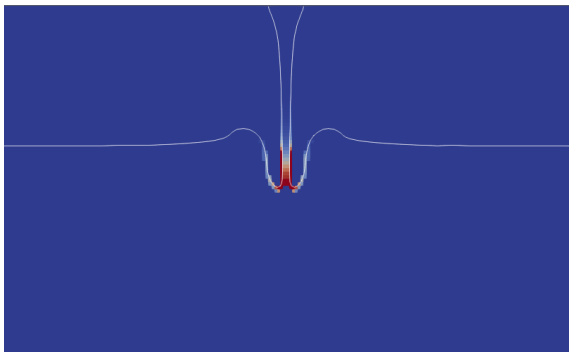
Vertical plunging jet – results



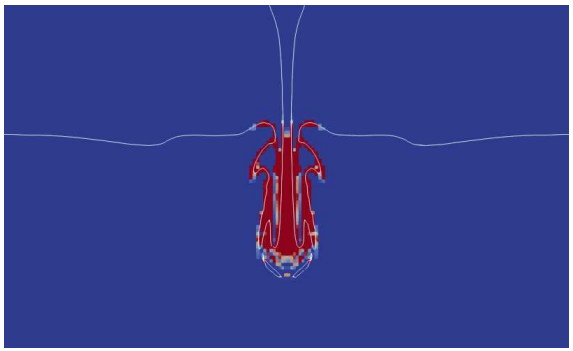
(a) `interAirEntFoam` (b) `interFoam`

Figure: Vertical plunging jet, 2D

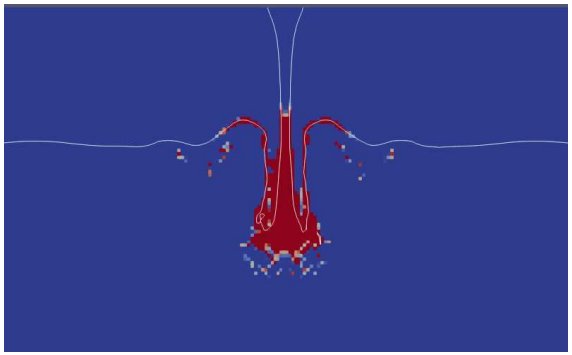
Vertical plunging jet – results



(a) t=0.2 sec



(b) t=0.5 sec



(c) t=0.8 sec

Figure: Indication of where the source term is active