پیوست ۱:

کدهای فایل options

\ = EXE_INC

\ .I-

\ I../VoF-

\ I$(LIB_SRC)/transportModels/twoPhaseMixture/lnInclude-

\ I$(LIB_SRC)/transportModels-

\ I$(LIB_SRC)/transportModels/incompressible/lnInclude-

\ I$(LIB_SRC)/transportModels/interfaceProperties/lnInclude-

\ I$(LIB_SRC)/TurbulenceModels/turbulenceModels/lnInclude-

\ I$(LIB_SRC)/TurbulenceModels/incompressible/lnInclude-

\ I$(LIB_SRC)/finiteVolume/lnInclude-

I$(LIB_SRC)/meshTools/lnInclude-

\ = EXE_LIBS

\ ltwoPhaseMixture-

\ lturbulenceModels-

\ lincompressibleTurbulenceModels-

\ lfiniteVolume-

\ lfvOptions-

lmeshTools–

پیوست ۲:

کدهای فایل files

twoLiquidDriftMixingFoam.C

EXE = $(FOAM_APPBIN)/twoLiquidDriftMixingFoam

پیوست ۳:

تنها فایل C. ، فایلی به اسم twoLiquidDriftMixingFoam.C می‌باشد که کد آن در ذیل آورده می‌شود.

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
```

```
  \\    /   O peration     | Website:  https://openfoam.org
   \\  /    A nd           | Copyright (C) 2011-2018 OpenFOAM Foundation
    \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Application
    twoLiquidMixingFoam

Description
    Solver for mixing 2 incompressible fluids.

    Turbulence modelling is generic, i.e. laminar, RAS or LES may be selected.

\*---------------------------------------------------------------------------*/

#include "fvCFD.H"
#include "MULES.H"
#include "subCycle.H"
#include "incompressibleTwoPhaseMixture.H"
#include "turbulentTransportModel.H"
#include "pimpleControl.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{
    #include "postProcess.H"

    #include "setRootCaseLists.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "createControl.H"
    #include "initContinuityErrs.H"
    #include "createFields.H"
    #include "createTimeControls.H"
```

```
#include "CourantNo.H"
#include "setInitialDeltaT.H"

turbulence->validate();

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

Info<< "\nStarting time loop\n" << endl;

while (runTime.run())
{
    #include "readTimeControls.H"
    #include "CourantNo.H"
    #include "alphaCourantNo.H"
    #include "setDeltaT.H"

    runTime++;

    Info<< "Time = " << runTime.timeName() << nl << endl;

    mixture.correct();

    #include "alphaEqnSubCycle.H"
    #include "alphaDiffusionEqn.H"

    // --- Pressure-velocity PIMPLE corrector loop
    while (pimple.loop())
    {
        #include "UEqn.H"

        // --- Pressure corrector loop
        while (pimple.correct())
        {
            #include "pEqn.H"
        }

        if (pimple.turbCorr())
        {
            turbulence->correct();
        }
    }

    runTime.write();

    Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
        << "  ClockTime = " << runTime.elapsedClockTime() << " s"
        << nl << endl;
}

Info<< "End\n" << endl;
```

```
    return 0;
}


// ************************************************************************* //
```

پیوست ۴

فایل alphaDiffusionEqn.H که کد آن در ذیل آورده می‌شود.

```
{
        volVectorField gradAlpha=fvc::grad(alpha1);
        dimensionedScalar vf=((rhoPart-rhoInf)*(mag(g))*(pow(dPart,2)))/(18*muInf);
    fvScalarMatrix alpha1Eqn
    (
        fvm::ddt(alpha1)
      - fvc::ddt(alpha1)
      - fvm::laplacian
        (
            volScalarField("Dab", Dab + alphatab*turbulence->nut()),
            alpha1
        )
      - (vf)*(cosTeta)*(gradAlpha.component(1))
    );

    alpha1Eqn.solve();

    alpha2 = 1.0 - alpha1;
    rhoPhi += alpha1Eqn.flux()*(rho1 - rho2);
}

rho = alpha1*rho1 + alpha2*rho2;
```

به صورت explicit داره حل میکنه معادلهconvection رو و مقدار قبلی اش رو جایگذاری میکنه و لازم نیست با سعی و خطا به دست آید

پیوست ۵

کد فایلی به اسم createFields.H در ذیل آورده می‌شود.

```
Info<< "Reading field p_rgh\n" << endl;
volScalarField p_rgh
(
    IOobject
    (
        "p_rgh",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
```

```
    mesh
);

Info<< "Reading field U\n" << endl;
volVectorField U
(
    IOobject
    (
        "U",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

#include "createPhi.H"

Info<< "Reading transportProperties\n" << endl;
incompressibleTwoPhaseMixture mixture(U, phi);

volScalarField& alpha1(mixture.alpha1());
volScalarField& alpha2(mixture.alpha2());

const dimensionedScalar& rho1 = mixture.rho1();
const dimensionedScalar& rho2 = mixture.rho2();

dimensionedScalar Dab("Dab", dimViscosity, mixture);
dimensionedScalar dPart("dPart", dimLength, mixture);
dimensionedScalar rhoPart("rhoPart", dimDensity, mixture);
dimensionedScalar cosTeta("cosTeta", dimless, mixture);
dimensionedScalar nuInf("nuInf", dimViscosity, mixture);
dimensionedScalar rhoInf("rhoInf", dimDensity, mixture);
dimensionedScalar muInf=nuInf*rhoInf;

// Read the reciprocal of the turbulent Schmidt number
dimensionedScalar alphatab("alphatab", dimless, mixture);

// Need to store rho for ddt(rho, U)
volScalarField rho("rho", alpha1*rho1 + alpha2*rho2);
rho.oldTime();


// Mass flux
// Initialisation does not matter because rhoPhi is reset after the
// alpha1 solution before it is used in the U equation.
surfaceScalarField rhoPhi
(
    IOobject
```

```
    (
        "rhoPhi",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::NO_WRITE
    ),
    rho1*phi
);

// Construct incompressible turbulence model
autoPtr<incompressible::turbulenceModel> turbulence
(
    incompressible::turbulenceModel::New(U, phi, mixture)
);

#include "readGravitationalAcceleration.H"
#include "readhRef.H"
#include "gh.H"

volScalarField p
(
    IOobject
    (
        "p",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    p_rgh + rho*gh
);

label pRefCell = 0;
scalar pRefValue = 0.0;
setRefCell
(
    p,
    p_rgh,
    pimple.dict(),
    pRefCell,
    pRefValue
);

if (p_rgh.needReference())
{
    p += dimensionedScalar
    (
        "p",
        p.dimensions(),
```

```
        pRefValue - getRefCellValue(p, pRefCell)
    );
    p_rgh = p - rho*gh;
}

mesh.setFluxRequired(p_rgh.name());
mesh.setFluxRequired(alpha1.name());
```

پیوست۶ :

فایلی به اسم alphaControls.H می‌باشد که کد آن در ذیل آورده می‌شود.

```
const dictionary& alphaControls = mesh.solverDict(alpha1.name());

label nAlphaSubCycles(readLabel(alphaControls.lookup("nAlphaSubCycles")));
```

فایلی به اسم UEqn.H می‌باشد که کد آن در ذیل آورده می‌شود.

```
fvVectorMatrix UEqn
    (
        fvm::ddt(rho, U)
      + fvm::div(rhoPhi, U)
      + turbulence->divDevRhoReff(rho, U)
    );

    UEqn.relax();

    if (pimple.momentumPredictor())
    {
        solve
        (
            UEqn
         ==
            fvc::reconstruct
            (
                (
                  - ghf*fvc::snGrad(rho)
                  - fvc::snGrad(p_rgh)
                ) * mesh.magSf()
            )
        );
         }
```

فایلی به اسم alphaEqnSubCycle.H می‌باشد که کد آن در ذیل آورده می‌شود.

```
#include "alphaControls.H"

if (nAlphaSubCycles > 1)
{
    dimensionedScalar totalDeltaT = runTime.deltaT();
```

```
    surfaceScalarField rhoPhiSum
    (
        IOobject
        (
            "rhoPhiSum",
            runTime.timeName(),
            mesh
        ),
        mesh,
        dimensionedScalar("0", rhoPhi.dimensions(), 0)
    );

    for
    (
        subCycle<volScalarField> alphaSubCycle(alpha1, nAlphaSubCycles);
        !(++alphaSubCycle).end();
    )
    {
        #include "alphaEqn.H"
        rhoPhiSum += (runTime.deltaT()/totalDeltaT)*rhoPhi;
    }

    rhoPhi = rhoPhiSum;
}
else
{
    #include "alphaEqn.H"
}

rho == alpha1*rho1 + alpha2*rho2;
```

فایلی به اسم alphaEqn.H می‌باشد که کد آن در ذیل آورده می‌شود.

```
{
    word alphaScheme("div(phi,alpha)");

    surfaceScalarField alphaPhi
    (
        phi.name() + alpha1.name(),
        fvc::flux
        (
            phi,
            alpha1,
            alphaScheme
        )
    );

    MULES::explicitSolve
    (
        geometricOneField(),
```

```
        alpha1,
        phi,
        alphaPhi,
        oneField(),
        zeroField()
    );

    rhoPhi = alphaPhi*(rho1 - rho2) + phi*rho2;

    Info<< "Phase-1 volume fraction = "
        << alpha1.weightedAverage(mesh.Vsc()).value()
        << "  Min(" << alpha1.name() << ") = " << min(alpha1).value()
        << "  Max(" << alpha1.name() << ") = " << max(alpha1).value()
        << endl;
}
```

فایلی به اسم pEqn.H می‌باشد که کد آن در ذیل آورده می‌شود.

```
{
    volScalarField rAU("rAU", 1.0/UEqn.A());
    surfaceScalarField rAUf("rAUf", fvc::interpolate(rAU));
    volVectorField HbyA(constrainHbyA(rAU*UEqn.H(), U, p_rgh));
    surfaceScalarField phiHbyA
    (
        "phiHbyA",
        fvc::flux(HbyA)
      + fvc::interpolate(rho*rAU)*fvc::ddtCorr(U, phi)
    );
    adjustPhi(phiHbyA, U, p_rgh);

    surfaceScalarField phig
    (
        - ghf*fvc::snGrad(rho)*rAUf*mesh.magSf()
    );

    phiHbyA += phig;

    // Update the pressure BCs to ensure flux consistency
    constrainPressure(p_rgh, U, phiHbyA, rAUf);

    while (pimple.correctNonOrthogonal())
    {
        fvScalarMatrix p_rghEqn
        (
            fvm::laplacian(rAUf, p_rgh) == fvc::div(phiHbyA)
        );

        p_rghEqn.setReference(pRefCell, getRefCellValue(p_rgh, pRefCell));

        p_rghEqn.solve(mesh.solver(p_rgh.select(pimple.finalInnerIter())));
```

```
        if (pimple.finalNonOrthogonalIter())
        {
            phi = phiHbyA - p_rghEqn.flux();

            U = HbyA + rAU*fvc::reconstruct((phig - p_rghEqn.flux())/rAUf);
            U.correctBoundaryConditions();
        }
    }

    #include "continuityErrs.H"

    p == p_rgh + rho*gh;

    if (p_rgh.needReference())
    {
        p += dimensionedScalar
        (
            "p",
            p.dimensions(),
            pRefValue - getRefCellValue(p, pRefCell)
        );
        p_rgh = p - rho*gh;
    }
}
```

فایلی به اسم alphaCourantNo.H می‌باشد که کد آن در ذیل آورده می‌شود.

```
*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Copyright (C) 2011-2018 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.
```

```
Global
    alphaCourantNo

Description
    Calculates and outputs the mean and maximum Courant Numbers.

\*---------------------------------------------------------------------------*/

scalar maxAlphaCo
(
    readScalar(runTime.controlDict().lookup("maxAlphaCo"))
);

scalar alphaCoNum = 0.0;
scalar meanAlphaCoNum = 0.0;

if (mesh.nInternalFaces())
{
    scalarField sumPhi
    (
        pos0(alpha1 - 0.01)*pos0(0.99 - alpha1)
       *fvc::surfaceSum(mag(phi))().primitiveField()
    );

    alphaCoNum = 0.5*gMax(sumPhi/mesh.V().field())*runTime.deltaTValue();

    meanAlphaCoNum =
        0.5*(gSum(sumPhi)/gSum(mesh.V().field()))*runTime.deltaTValue();
}

Info<< "Interface Courant Number mean: " << meanAlphaCoNum
    << " max: " << alphaCoNum << endl;

// ************************************************************************
//
```