IHFOAM 初探（代码篇）

Simon Ji...
A Mentat

已关注

5 人赞同了该文章

此文承接**IHFOAM**初探（理论篇），文中将主要介绍我在本科阶段使用的IHFOAM求解器代码。若对多孔介质两相流控制方程组及**OpenFOAM**(以下简称**of**)的压力耦合求解策略尚不清楚的同学，可查看理论篇及相关文献。

友情提示：阅读文本时，建议自己打开IHFOAM代码，配合食用更佳

IHFOAM的压力速度耦合循环求解在while(pimple.loop()){...}中。pimple类的声明在src/finiteVolume/cfdTools/general/solutionControl/pimpleControl/pimpleControl.H中，该成员函数用于判断当前迭代下，计算是否已达到收敛，若达到，则返回false，结束循环

```
virtual bool loop();
```

进入while循环体内，发现include一个头文件，该文件主要为读入用户设置的参数，从字典文件system/fvSolution中读入"nAlphaCoor","nAlphaSubCycles","MULESCorr","alphaApplyPrevCoor","icAlpha"的设置参数值，参数的具体意义可暂时不管，后面将会陆续提到

```
#include "alphaControls.H"
```

接下来是一个条件判断语句，若pimple.loop()为第一次迭代或fvsolution的alphaOuterCorrectors关键字设置为true，则进入内部语句

```
if (pimple.firstIter() || alphaOuterCorrectors)
```

内部语句中仅看到使用correct()，结合后续的分析，知道该函数是对两相交界面处的曲率进行更新

```
twoPhaseProperties.correct();

/*
correct()定义在"interfaceProperties.H"中
```

```
void correct(){
     calculateK(); //recalculate the interface curvature
}
void Foam::interfaceProperties::calculateK(){

//定义相分数场的梯度场
const volVectorField gradAlpha(fvc::grad(alpha1_, "nHat"));

//插值该梯度场到网格面心处
surfaceVectorField gradAlphaf(fvc::interpolate(gradAlpha));

//计算网格面心的单位相梯度场
surfaceVectorField nHatfv(gradAlphaf/(mag(gradAlphaf) + deltaN_));

//修正边界处的接触角
correctContactAngle(nHatfv.boundaryField(),gradAlphaf.boundaryField());

//矢量点乘得到网格面心处垂直于网格面的单位相梯度场
nHatf_ = nHatfv & Sf;

//对nHatf求散度即得到界面处的曲率，实现两相交界处曲率的更新
K_ = -fvc::div(nHatf_);
}
/*
```

紧接着，进入相方程计算亚循环

```
#include "alphaEqnSubCycles.H"
```

nAlphaSubCycles为前面提及在"alphaControl.H"中读入的标量

```
if (nAlphaSubCycles > 1) //从fvsolution字典文件读入的相方程循环次数
{
    //定义带量纲标量totalDeltaT为当前时间步步长
    dimensionedScalar totalDeltaT = runTime.deltaT();
    //定义质量通量rhoPhiSum
surfaceScalarField rhoPhiSum
    (
        IOobject
        (
            "rhoPhiSum",
            runTime.timeName(),
            mesh
```

```
        ),
        mesh,
        dimensionedScalar("0", rhoPhi.dimensions(), 0)
    );
    /*这里的for循环定义一个类似Vector<>的数据类型alphaSubCycle,
     alpha1为数组中每一个元素的初始化值(注意这里的alpha1为标量场而非简单的标量)，
      nAlphaSubCycles为数组长度，等同于在当前时间下为相分数场计算建立亚循环*/
    for
(
        subCycle<volScalarField> alphaSubCycle(alpha1, nAlphaSubCycles);
        !(++alphaSubCycle).end();
){......}
```

进入该亚循环体中，可以看到主要的代码放在alphaEqn.H中,执行完alphaEqn.H后，自加之前定义的质量通量，可以猜测alphaEqn.H中应该涉及更新rhoPhi

```
#include "alphaEqn.H"
rhoPhiSum += (runTime.deltaT()/totalDeltaT)*rhoPhi;
```

为证明猜测，进入alphaEqn.H

```
//首先定义两个离散格式的名字，word类似string类型
    word alphaScheme("div(phi,alpha)");
    word alpharScheme("div(phirb,alpha)");

//定义界面压缩系数phic；cAlpha为压缩因子,在interfaceProperties.H中声明
surfaceScalarField phic(interface.cAlpha()*mag(phi/mesh.magSf()));

//icAlpha为之前提及的、在fvSolution设置的各向同性压缩标量系数,若未设置则默认为0，此时无压缩效果
    if (icAlpha > 0)
    {
        phic *= (1.0 - icAlpha);
        phic += (interface.cAlpha()*icAlpha)*fvc::interpolate(mag(U));
}

//遍历phic处于边界区域的网格单元面，若phic的边界为非耦合边界(如常见的无滑移壁面边界)，则该边界位
    forAll(phic.boundaryField(), patchi)
    {
        fvsPatchScalarField& phicp = phic.boundaryField()[patchi];
        if (!phicp.coupled())
        {
            phicp == 0;
```

```
        }
    }
```

这里的phic写成数学表达式为：

$$phic = (1 - i_{c,\alpha})\, c_\alpha \, |u_\perp| \, S_f + (i_{c,\alpha,f} c_\alpha) \, |u_f| \tag{1}$$

需要注意的是phic用于压缩两相交界面而被人为添加到相方程中，理论篇中的相方程仅为基本形式

接下来进入MULES相分数修正，这是一个提示语句，activePorosity和outputTime均为true的情况下，提示在当前时间步下，VOF equation并未考虑孔隙率的影响

```
        if( activePorosity && runTime.outputTime() )
        {
            Info << "WARNING: The VOF equation in this loop"
                 << " does not include porosity.\n"
                 << "Set MULESCorr to 'no' in ./system/fvSolution.\n" << endl;
        }
```

构建基本相方程 $alpha1Eqn = \frac{\partial \alpha}{\partial t} + u_j \frac{\partial \alpha}{\partial x_j}$

```
    fvScalarMatrix alpha1Eqn
        (
            #ifdef LTSSOLVE
            fv::localEulerDdtScheme<scalar>(mesh, rDeltaT.name()).fvmDdt(alpha1)
            #else
            fv::EulerDdtScheme<scalar>(mesh).fvmDdt(alpha1)
            #endif
          + fv::gaussConvectionScheme<scalar>
            (
                mesh,
                phi,
                upwind<scalar>(mesh, phi)
            ).fvmDiv(phi, alpha1)
        );
```

求解相方程，这里关于solve函数的具体实现细节，可参考陈与论的回答

```
    alpha1Eqn.solve();
```

输出当前计算得到alpha1的平均相分数（以各单元体积为权重），单元中的最小alpha1和最大alpha1值；另外，不同于c++，输出流从cout改为info

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

基于alphaEqn得到的alpha1相分数场求解面通量场tphiAlphaUD,这里的tmp是of一种旨在节省存储的方式，可简单理解为指针，<>内表示所定义变量的类型

```
tmp<surfaceScalarField> tphiAlphaUD(alpha1Eqn.flux());
```

下面的MULES有1个条件判断修正，1个内循环修正，均旨在修正alpha1相分数场，这一块由于代码封装较复杂，尚未具体分析☹，

跳过这一块，来到alphaEqn.H的末尾，可以看到这样几行代码，rhoPhi变量发生更新，证实前面的猜测

```
        alpha2 = 1.0 - alpha1;
        interface.correct();
//根据VOF模型得到混合相的面流量场
        rhoPhi = tphiAlpha()*(rho1 - rho2) + phi*rho2;
//最后输出结果，额外输出的gsum为考虑多孔介质的alpha1（通常为液体）的实际体积
    Info<< "Phase-1 volume fraction = "
        << alpha1.weightedAverage(mesh.Vsc()).value()
        << "  Phase-1 total volume = "
        << gSum(alpha1*porosity*mesh.Vsc())
        << "  Min(alpha1) = " << min(alpha1).value()
        << "  Max(alpha1) = " << max(alpha1).value()
        << endl;
```

结束alphaEqn.H，返回alphaEqnSubCycle.H中，接之前提及的同一时间步下的亚循环体，此处将该时间步细分得到的面通量场代数叠加至rhoPhiSum中，并最后赋给rhoPhi

```
rhoPhiSum += (runTime.deltaT()/totalDeltaT)*rhoPhi;
}
    rhoPhi = rhoPhiSum;
}
```

当然，若不进行亚循环，可以给关键字nAlphaSubCycles赋值为0,这样将会执行下面else的语句，即alphaEqn.H在当前时间步的pimple循环体内只执行一次

```
else
{
 #include "alphaEqn.H"
}
```

最后更新混合密度场，为下面的动量方程求解做准备

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

相较于相方程求解，动量方程和压力泊松方程的求解代码简洁直观很多，初学者可以更容易理解

进入动量方程求解UEqn.H

```
#include "UEqn.H"
```

定义面标量场 muEff（ μ_{eff} ），为雷诺应力中描述湍流而人为添加的粘滞系数

```
    surfaceScalarField muEff
    (
        "muEff",
        twoPhaseProperties.muf()
      + fvc::interpolate(rho*turbulence->nut())
);
```

定义fvMatrix类型UEqn，组建动量方程的非定常项、对流项、扩散项和多孔介质阻尼项

```
fvVectorMatrix UEqn
    (
        (1.0 + cPorField) / porosity * fvm::ddt(rho, U)
        + 1.0/porosity * fvm::div(rhoPhi/porosityF, U)
        // + turbulence->divDevRhoReff(rho, U)
        - fvm::laplacian(muEff/porosityF , U)
        - 1.0/porosity * ( fvc::grad(U) & fvc::grad(muEff) )
```

```
            // Closure Terms
            + aPorField * pow(1.0 - porosity, 3) / pow(porosity,3)
                * twoPhaseProperties.mu() / pow(D50Field,2) * U
            + bPorField * rho * (1.0 - porosity) / pow(porosity,3) / D50Field
                * mag(U) * U *
            // Transient formulation
            (1.0 + useTransMask * 7.5 / KCPorField)
    );

    //松弛该代数矩阵，主要对组建的矩阵进行变换，保证对角占优
    UEqn.relax();
```

若fvSolution中momentumPredictor设置为true，则执行下列对动量方程执行计算，注意这里的fvc::reconstruct()内的压力场（p_rgh）在当前时间步下尚未更新，因此此时求解得到的速度场并不准确

```
    //在IHFOAM的演示算例中并未采用动量预测，说明这步并不是必须的
    if (pimple.momentumPredictor())
        {
            solve
            (
                UEqn
             ==
                fvc::reconstruct
                (
                    (
                        fvc::interpolate(interface.sigmaK())*fvc::snGrad(alpha1)
                      - ghf*fvc::snGrad(rho)
                      - fvc::snGrad(p_rgh)
                    ) * mesh.magSf()
                )
            );
    }
```

下面进入压力耦合修正循环体内,correct()方法定义在pimpleControlI.H中，内部有一记录该函数调用次数的correPISO,当corrPISO小于nCorrPISO，correct()返回true，执行循环体内部语句，nCorrPISO由fvSolution中的nCorrectors读入，否则默认为1

```
    while (pimple.correct()){
         #include "pEqn"
    }
```

接下来的几行代码，实际上在整理动量方程，具体变量对应的数学表达式如下表所示：

变量名	数学表达式	补充说明
rAU	$\frac{1}{a_p}$	由该处单元体心速度构成的代数式
$rAUf$	NA	rAU插值到单元面心处
$HbyA$	$\frac{1}{a_p}\left(-\sum_N a_N\langle\bar{u}^n_{i,N}\rangle\right)$	NA
$phiHbyA$	$\frac{1}{a_p}\left(-\sum_N a_N\langle\bar{u}^n_{i,N}\rangle+\frac{\langle\bar{u}^0_{i,P}\rangle}{\Delta t}\right)$	这里代码中乘上密度是保证数学表达式中的量纲一致，由于在FVM中将待求未知量的通量作为中间变量，因此将$phiHbyA$插值到单元面心处
$phig$	$\frac{1}{a_p}\left(\sigma\kappa\frac{\partial\alpha}{\partial x_i}\cdot\frac{\mathrm{S}_f}{\lvert\mathrm{S}_f\rvert}\mathrm{n}_f+\mathbf{g}.\mathbf{h}\frac{\partial\rho}{\partial x_i}\right)$	将相方程中更新的rho和$alpha1$显式离散并插值到单元面心处；这里需要注意的，of对压力梯度项和重力项进行如下变换，统一梯度为算子符，关于该变换$cfdonline$上有专门的讨论 $\nabla p+\rho\mathbf{g}=-\nabla p_{rgh}-\mathbf{g}.\mathbf{h}\nabla\rho$ 其中定义$p_{rgh}=p-\rho\mathbf{g}.\mathbf{h}$

附上一个关于p_rgh的讨论链接

```
volScalarField rAU("rAU", 1.0/UEqn.A());
    surfaceScalarField rAUf("rAUf", fvc::interpolate(rAU));

    volVectorField HbyA("HbyA", U);
    HbyA = rAU*UEqn.H();

    surfaceScalarField phiHbyA
    (
        "phiHbyA",
        (fvc::interpolate(HbyA) & mesh.Sf())
      + fvc::interpolate(rho*rAU)*fvc::ddtCorr(U, phi)
    );

    adjustPhi(phiHbyA, U, p_rgh);

    surfaceScalarField phig
    (
        (
            fvc::interpolate(interface.sigmaK())*fvc::snGrad(alpha1)
          - ghf*fvc::snGrad(rho)
        )*rAUf*mesh.magSf()
    );
    adjustPhi(phiHbyA, U, p_rgh);

    surfaceScalarField phig
    (
```

```
        (
            fvc::interpolate(interface.sigmaK())*fvc::snGrad(alpha1)
          - ghf*fvc::snGrad(rho)
        )*rAUf*mesh.magSf()
);
```

自加后的phiHbyA即为式压力泊松方程左侧项，现在万事俱备只欠东风

$$phiHbyA = \nabla \cdot \left(\frac{1}{na_p} H\left(\langle \bar{u} \rangle \right) \right) \tag{2}$$

```
phiHbyA += phig;
```

等等，在正式计算前，类似相方程对非耦合边界的处理方式，这里也需要对一些特别的边界条件进行修正，这里特指fixedFluxPressure,该边界处的压力梯度受速度边界影响，具体表达式为：

$$\nabla p_{rgh,boundary} = \left(\frac{H\left(\langle \bar{u}_i^{n\&0} \rangle \right)_f - \mathrm{u}_i \cdot \mathrm{S}_f \mathrm{a}_{p,f}}{|\mathrm{S}_f|} \right)_{boundary} \tag{3}$$

```
    setSnGrad<fixedFluxPressureFvPatchScalarField>
    (
        p_rgh.boundaryField(),
        (
            phiHbyA.boundaryField()
          - (mesh.Sf().boundaryField() & U.boundaryField())
        )/(mesh.magSf().boundaryField()*rAUf.boundaryField())
);
```

下面正式进入压力泊松求解，同样也是一个while循环体，correctNonOrthogonal和前面的pimple.correct()作用相同，关键字为fvSolution中的nNonOrthogonalCorrectors,默认为0，此时while循环体只执行一次

```
    while (pimple.correctNonOrthogonal())
{
//组建压力泊松表达式p_rghEqn，==为数学表达式中的减号，注意此时未建立方程
        fvScalarMatrix p_rghEqn
        (
            fvm::laplacian(rAUf, p_rgh) == fvc::div(phiHbyA)
        );
        //设置压力参考点，简单来说这里p_rgh的取值可以以空间某点处的压力值作为参考点
```

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

    //这里是泊松方程求解的代码本体
    p_rghEqn.solve(mesh.solver(p_rgh.select(pimple.finalInnerIter())));
    //在循环体的最后一次循环时，更新面通量phiHbyA（压力场完成更新），更新速度场，更新速度场的表达式为
        if (pimple.finalNonOrthogonalIter())
        {
            phi = phiHbyA - p_rghEqn.flux();

            p_rgh.relax();

            U = HbyA + rAU*fvc::reconstruct((phig - p_rghEqn.flux())/rAUf);
            U.correctBoundaryConditions();
            fvOptions.correct(U);
        }
    }
    #include “continuityErrs.H”
```

最后，利用已更新的P_rgh更新压力场p，若存在压力参考点的设置，则按参考点的pRefValue与实际该点处压力的偏差值，依次修正p和p_rgh；应当注意的是p_rgh本身是为了统一梯度算子符而引人的变量，与压力参考点无关，因此即使在fvSolution/PIMLPE中不设置pReference，p_rgh变量依然会生成

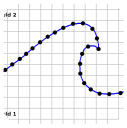
```
    p == p_rgh + rho*gh;

    if (p_rgh.needReference())
    {
        p += dimensionedScalar
        (
            "p",
            p.dimensions(),
            pRefValue - getRefCellValue(p, pRefCell)
        );
        p_rgh = p - rho*gh;
    }
}
//压力泊松方程求解结束，输出必要的变量，时间步向前步进...
```

湍流模拟的控制方程在此暂未展开讨论，至此IHFOAM求解主体流程结束

最后感谢前辈们的文章提供的极大帮助，这里一并附上

大于治水
www.zhihu.com/column/haidong

interFoam解析
dyfluid.com/interFoam.html



有所思
www.zhihu.com/column/whataphd

以上