laplacianFoam解析

李东岳

1. 引言

laplacianFoam是OpenFOAM里面3个最基本的求解器之一（另外两个求解器为scalarTransportFoam和potentialFoam），其求解的为瞬态椭圆形拉普拉斯方程，主要用于对固态热传导问题进行分析。laplacianFoam中植入的瞬态热传导方程为：

$$\frac{\partial T}{\partial t} - \nabla^2(D_\mathrm{T} T) = 0 \tag{1}$$

其中D_T为扩散系数，T表示温度，那么D_T表示热导率。方程(1)仅仅存在一个未知数，因此对其直接求解即可。

2. 代码分析

首先我们进入createFields.H：

```
Info<< "Reading field T\n" << endl;
//在终端输出“Reading field T”，endl在OpenFOAM中为空行，也可以用nl替代，Info在Oper

volScalarField T //声明volScalarField类型，名称为T
(
    IOobject
    (
        "T", //在求解器中注册的名字，例如，可以通过lookupObject<volScalarField>("
        runTime.timeName(), //存储在运行时间
        mesh, //注册于网格
        IOobject::MUST_READ, //必须进行读取，如果某个场为计算而来，可以不进行读取
        IOobject::AUTO_WRITE //自动写场，按照controlDict中的定义来确定在哪个时间
    ),
    mesh //场定义在网格
); //本代码段表示表示创建一个标量场volScalarField，类似的代码段在其他求解器的crea


Info<< "Reading transportProperties\n" << endl;

IOdictionary transportProperties //声明IOdictionary类型，命名为transportProperti
(
    IOobject
    (
        "transportProperties",
        runTime.constant(),
        mesh,
        IOobject::MUST_READ_IF_MODIFIED, //在字典文件被更改的时候进行读取
        IOobject::NO_WRITE //不写入字典文件
    )
);


Info<< "Reading diffusivity DT\n" << endl;

dimensionedScalar DT
(
    transportProperties.lookup("DT") //lookup意思为在上文创建的transportProperti
); //dimensionedScalar为有单位的标量
```

然后进入laplacianFoam.C：

```
  #include "fvCFD.H"
  //在OpenFOAM的所有求解器中都可以看到这个头文件，它涉及到构建时间、组建矩阵、有限体积
  //建议在自定义的求解器中必备此项。初学可以忽略此头文件（下文中用“略”来表示，意义为初

  #include "simpleControl.H"
  //包含SIMPLE循环文件头，使用SIMPLE循环必须包含此文件，本求解器没有使用SIMPLE算法，但

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

  int main()
  {
      #include "setRootCase.H" //设置算例的根目录，略
      #include "createTime.H" //创建时间对象，略
      #include "createMesh.H" //创建网格对象，略

      simpleControl simple(mesh);
      //对于采用SIMPLE算法的算例，创建simple对象，略

    #include "createFields.H" //包含上文分析过的createFields.H头文件

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

    Info<< "\nCalculating temperature distribution\n" << endl;
    //终端输出信息

    while (simple.loop()) //开始SIMPLE循环，也即时间步进
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;
        //输出当前时间步

        while (simple.correctNonOrthogonal())
        //是否非正交修正？如果用户在fvSolution中定义为0，
        //那么求解下面的方程一次，如果设置为n，则求解下述方程n+1次
        {
            solve
            (
                fvm::ddt(T) - fvm::laplacian(DT, T)
            ); //公式(1)
        }

        #include "write.H" //输出T的梯度,参见下文

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

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

    return 0;
}
```

在这里讨论一下上文中的fvm::ddt(T)以及fvc::ddt(T)。fvm::在OpenFOAM中代表隐性离散，fvc::代表显性离散。隐性离散未知项进入离散方程的左边，显性离散未知项进入方程的右边并且用当前时间步的量来计算。另一种理解的思路为显性离散可用于被求解的变量和其他变量，隐性离散只能用于被求解的变量。

下面分析上文中的write.H：

```
    if (runTime.outputTime())
    //判断是不是所需要输出的时间步，是，则执行下面语句
    {
        volVectorField gradT(fvc::grad(T));
        //创建gradT场，其值为fvc::grad(T)，即为∇T显性离散的值

        volScalarField gradTx
        (
            IOobject
            (
                "gradTx",
                runTime.timeName(),
                mesh,
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
            ),
            gradT.component(vector::X)
        );//创建gradTx场，其值为矢量gradT的x的值

        volScalarField gradTy
        (
            IOobject
            (
                "gradTy",
                runTime.timeName(),
                mesh,
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
            ),
            gradT.component(vector::Y)
        );

        volScalarField gradTz
        (
            IOobject
            (
                "gradTz",
                runTime.timeName(),
                mesh,
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
            ),
            gradT.component(vector::Z)
        );

        runTime.write();//写需要写的场
    }
```

更新历史

*2018.03.31*小修格式及内容/*2017.07.01*创立页面