

## OpenFOAM中laplacian项离散格式解码



海吟吾掣

Cavitation & Multiphase flow

关注他

23 人赞同了该文章

在OF中，对于动量方程中的扩散项（二阶导数项）的离散是通过Gauss方法进行的。过程在此就不详细介绍了，具体请参见[Diffusion term](#)。

由于网格正交性的问题，将面矢量分解成两个部分，即正交部分和非正交部分。正交部分很容易处理，而非正交项无法进行直接离散；另外，如果其数值过大，会引起计算不稳定。

对于非正交部分的修正，Jasak给出了三种方法，即Minimum correction, Orthogonal correction, Over-relax correction。那么在OF代码中，到底使用的是哪种分解方法呢？

在fvScheme中，laplacian项离散关键字为：

Gauss + <interpolationScheme> + <snGradScheme>

其中各项代表的意义是什么呢？





## 1. 文件定位：

\$FOAM\_SRC/finiteVolume/finiteVolume/laplacianSchemes/**gaussLaplacianScheme.C**

\$FOAM\_SRC/finiteVolume/finiteVolume/snGradSchemes/**correctedSnGrad.C**

\$FOAM\_SRC/finiteVolume/interpolation/surfaceInterpolation/**surfaceInterpolation.C**

其中，三个文件递进关系是：

surfaceInterpolation.C  correctedSnGrad.C  gaussLaplacianScheme.C

由外向内进行分析。

## 2.gaussLaplacianScheme.C

这里主要对隐式格式进行讨论，主要涉及的成员函数有：

```
fvmLaplacianUncorrected(...)
gammaSnGradCorr(...)
fvmLaplacian(...)
```

其中，fvmLaplacian是找寻这个问题的关键，其代码和注释如下：

```
template<class Type, class GType>
tmp<fvMatrix<Type>>
gaussLaplacianScheme<Type, GType>::fvmLaplacian
(
    const GeometricField<GType, fvsPatchField, surfaceMesh>& gamma,
    const GeometricField<Type, fvPatchField, volMesh>& vf
)
{
    const fvMesh& mesh = this->mesh();
```

/\*\*\*\*\*分割线\_1\_begin\*\*\*\*\*/

```
const surfaceVectorField Sn(mesh.Sf()/mesh.magSf());
```



```
(
    SfGamma & Sn
);
const surfaceVectorField SfGammaCorr(SfGamma - SfGammaSn*Sn);
```

\*\*\*\*\*分割线\_1\_end\*\*\*\*\*

分析:

Sn: 法向量n

SfGamma: 法向量和扩散系数的内积, 考虑到材料的各向异性, gamma应该为tensor, 为简单起见, 这里可认为是个标量

SfGammaSn: 扩散系数gamma和面矢量Sf的标量积

SfGammaCorr: 一个零场, 类似于扩散项中加入速度散度, 用于修正

\*\*\*\*\*分割线\_2\_begin\*\*\*\*\*

```
tmp<fvMatrix<Type>> tfvm = fvmLaplacianUncorrected
(
    SfGammaSn,
    this->tsnGradScheme_().deltaCoeffs(vf),
    vf
);
fvMatrix<Type>& fvm = tfvm.ref();

tmp<GeometricField<Type, fvsPatchField, surfaceMesh>> tfaceFluxCorrection
= gammaSnGradCorr(SfGammaCorr, vf);
```

\*\*\*\*\*分割线\_2\_end\*\*\*\*\*

分析:

tfvm调用了fvmLaplacianUncorrected(...), 从字面意义上也可了解, 该函数是用来求取正交项部分;

\*\*\*\*\*分割线\_3\_begin\*\*\*\*\*

```
if (this->tsnGradScheme_().corrected())
{
    tfaceFluxCorrection.ref() +=
```



```
fvm.source() -= mesh.V()*fvc::div(tfaceFluxCorrection())().internalField();
```

```
*****分割线_3_end*****/
```

分析：

如果需要非正交修正，则进入if语句，非正交项显式处理，具体形式应为：

```
{gamma * Af} * {grad(vf) & unitvector_nonorth}
```

其中第一个大括号内为SfGammaSn，tsnGradScheme\_().correction(vf)就是第二个大括号内的内容了，该correction函数在**correctedSnGrad.C**文件中，后面会介绍。

```
*****分割线_4_begin*****/
```

```
if (mesh.fluxRequired(vf.name()))
{
    fvm.faceFluxCorrectionPtr() = tfaceFluxCorrection.ptr();
}

return tfvm;
}
```

```
*****分割线_4_end*****/
```

这个就是在fvScheme文件中的fluxRequired，视用户要求而定。

### 3.correctedSnGrad.C

前面提及，这个文件中函数起到的作用是，给出grad(vf) & unitvector\_nonorth项。包含两个函数类：

```
fullGradCorrection(...)
correction(...)
```

在fullGradCorrection中，核心代码片段是：



```
(
    mesh.nonOrthCorrectionVectors(),
    gradScheme<Type>::New
    (
        mesh,
        mesh.gradScheme("grad(" + vf.name() + ')')
    )().grad(vf, "grad(" + vf.name() + ')')
);
```

其中mesh.nonOrthCorrectionVectors()就是unitvector\_nonorth项，它包含在surfaceInterpolation.C中；grad(vf,...)就是场的梯度；dotInterpolate就是将二者求内积后插值到面上。

在correction函数中，调用了fullGradCorrection(..)，并且令新定义的ssf，在三个方向上存储{grad(vf) & unitvector\_nonorth}的面上的数据，具体代码片段如下：

```
GeometricField<Type, fvsPatchField, surfaceMesh>& ssf = tssf.ref();

for (direction cmpt = 0; cmpt < pTraits<Type>::nComponents; cmpt++)
{
    ssf.replace
    (
        cmpt,
        correctedSnGrad<typename pTraits<Type>::cmptType>(mesh)
        .fullGradCorrection(vf.component(cmpt))
    );
}
```

## 4.surfaceInterpolation.C

实现非正交修正的核心函数类数是两个：

```
makeNonOrthDeltaCoeffs()
makeNonOrthCorrectionVectors()
```



Minimum correction:  $\cos(\theta)$

Orthogonal correction: 1

Over-relax correction:  $1/\cos(\theta)$

代码显示，用的是Over-relax correction，其余两种方法均被注释掉了，已标出位置，具体如下：

```
forAll(owner, facei)
{
    vector delta = C[neighbour[facei]] - C[owner[facei]];
    vector unitArea = Sf[facei]/magSf[facei];

    // Standard cell-centre distance form
    //NonOrthDeltaCoeffs[facei] = (unitArea & delta)/magSqr(delta);

    // Slightly under-relaxed form
    //NonOrthDeltaCoeffs[facei] = 1.0/mag(delta);

    // More under-relaxed form
    //NonOrthDeltaCoeffs[facei] = 1.0/(mag(unitArea & delta) + VSMALL);

    // Stabilised form for bad meshes
    /**就是这句**/
    nonOrthDeltaCoeffs[facei] = 1.0/max(unitArea & delta, 0.05*mag(delta));
}

forAll(nonOrthDeltaCoeffs.boundaryField(), patchi)
{
    vectorField delta(mesh_.boundary()[patchi].delta());

    nonOrthDeltaCoeffs.boundaryField()[patchi] =
        1.0/max(mesh_.boundary()[patchi].nf() & delta, 0.05*mag(delta));
}
```

有了非正交修正系数( $1/\cos(\theta)$ )后，将面向量分解后的非正交单位向量的代码为：

```
forAll(owner, facei)
{
    vector unitArea = Sf[facei]/magSf[facei];
```



```
/**就是这句**/  
corrVecs[facei] = unitArea - delta*NonOrthDeltaCoeffs[facei];  
}
```

至此分析完毕，其他如对流项的套路也基本如此，以此为模版便可举一反三。

编辑于 2017-02-21

「真诚赞赏，手留余香」

赞赏

还没有人赞赏，快来当第一个赞赏的人吧！

openfoam    计算流体力学 (CFD)

▲ 赞同 23 ▼    2 条评论    分享    喜欢    收藏    申请转载    ...

文章被以下专栏收录



大于治水  
Cavitation and multiphase flow

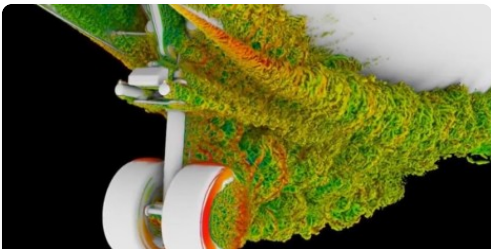
关注专栏



OpenFOAM学习记录  
OpenFOAM在超算平台上的移植优化，欢迎交流

关注专栏

推荐阅读



网格划分的五

尽管现在FEA及C  
网格软件，然而对  
用者来訪 各  
常重要的

2 条评论

切换为时间排序

写下你的评论...



Micro

10-05

深度好文

👍 赞



Micro

10-18

您好，也就是说 非正交修正这个功能是fv::laplacian 自带的功能是吗，哪和ioFoam中的非正交循环是一个概念吗

👍 赞

