OpenFOAM 中源项的实现

2019-10-12 · Algorithm · 2231 words · 5 mins read · 12963 times read

考虑如下所示的关于 ψ 的输运方程：

$$\frac{\partial \psi}{\partial t}+\nabla\cdot(\mathbf{u}\psi)-\nabla\cdot\gamma\nabla\psi=S_\psi \tag{1}$$

其中，S_ψ 为源项。

源项线性化

实际情况下 S_ψ 是比较复杂的，可能包含非线性项。为了求解稳定，通常对其进行线性化处理

$$S_\psi=S_u+S_p\psi \tag{2}$$

其中，S_u 和 S_p 可与 ψ 有关，也可与 ψ 无关。

将输运方程离散成 $[\mathbf{A}][\mathbf{x}]=[\mathbf{b}]$ 后，S_u 将进入 $[\mathbf{b}]$，而 S_p 将变成 $-S_p$ 进入 $[\mathbf{A}]$ 的对角元素中。若 S_p 为正，则将削弱 $[\mathbf{A}]$ 的对角占优，很可能造成线性方程组求解发散。因此在对源项进行线性化时必须保证 S_p 为负或零。

源项线性化的选择并不是唯一的，不同的选择有不同的精度和稳定性。举个例子说明，考虑如下形式的源项

$$S_\psi=10-2\psi^3 \tag{3}$$

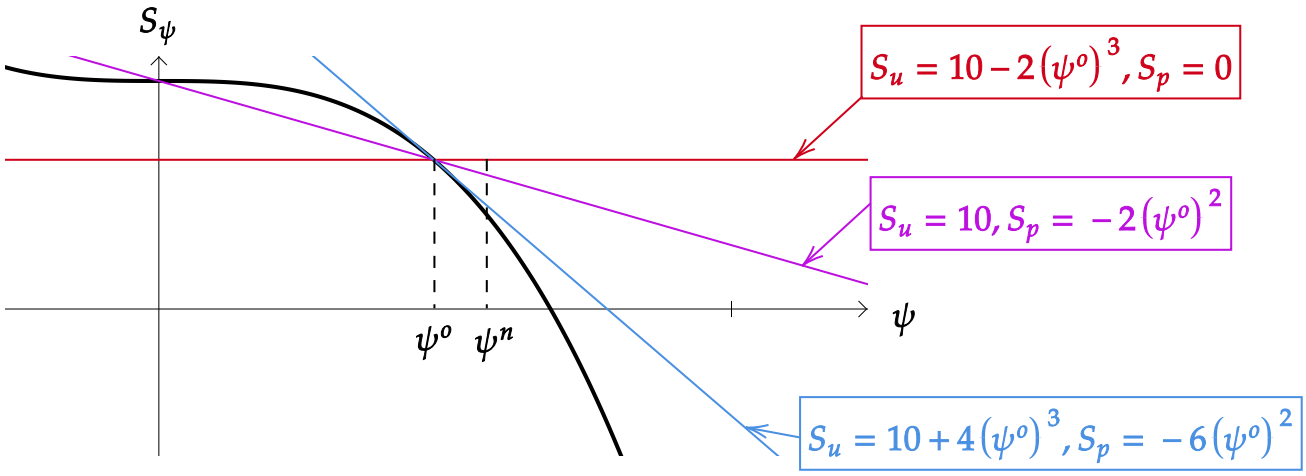
上式中包含非线性项 $-2\psi^3$，需要对其进行线性化处理。这里有好几种方法可以选择：

- 方法一：$S_u=10-2(\psi^o)^3$，$S_p=0$
- 方法二：$S_u=10$，$S_p=-2(\psi^o)^2$
- 方法三（Picard's method）：对 $S_\psi(\psi)$ 在 ψ^o 处进行泰勒展开，只保留一阶项

$$\begin{aligned}S_\psi&\approx S_\psi(\psi^o)+\frac{\partial S_\psi}{\partial\psi}\Big|_{\psi=\psi^o}(\psi-\psi^o)\\&=10-2(\psi^o)^3+[-6(\psi^o)^2](\psi-\psi^o)\\&=10+4(\psi^o)^3-6(\psi^o)^2\psi\end{aligned} \tag{4}$$

即 $S_u=10+4(\psi^o)^3$，$S_p=-6(\psi^o)^2$。

下图给出了以上三种不同源项线性化方法的示意图，可以看出，第一种方法精度最低，第二种其次，第三种方法的精度最高。



不同源项线性化方法示例

源项线性化的用途

指定网格单元的值

对某一单元 P，若需要指定该处的值为 $\psi_{P,desired}$，则可对该单元施加如下线性化后的源项

$$S_P = A_{large}\psi_{P,desired} - A_{large}\psi_P \tag{5}$$

其中 A_{large} 为一个很大的数，如 10^{10}。

对于单元 P，将除源项外的所有其他项离散后的方程表示为

$$A_P\psi_P + \sum_N^{nb} A_N\psi_N = b_P \tag{6}$$

加上源项后，上式变为

$$(A_P + A_{large})\psi_P + \sum_N^{nb} A_N\psi_N = b_P + A_{large}\psi_{P,desired} \tag{7}$$

若满足 $A_{large} \gg A_P$ 、 $A_{large} \gg A_N$ 且 $A_{large}\psi_{P,desired} \gg b_P$ ，则求解得到的 $\psi_P \approx \psi_{P,desired}$ 。

在需要修改某个网格单元的值时而不影响其他单元时（如重叠网格的插值单元赋值），这种方法格外有用。

保证物理量为非负

当源项为负数的时候，求解方程得到的值也可能为负数，而这对一些不可能为负数的物理量（例如温度 T、湍动能 k、特定耗散率 ω）来说是没有意义的。

为了避免这种情况发生，可以将源项做如下处理：

$$S_\psi = S_{const} \approx S_{const}\frac{\psi}{\psi^o} \tag{8}$$

其中，S_{const} 为负的源项。经过以上处理后，$-S_{const}/\psi^o$ 将作为正数计入到系数矩阵的对角元素中，既增强了对角占优，又可保证 ψ 为非负。

OpenFOAM 的实现

方程 (1) 用代码可以表示为：

C++

```
fvScalarMatrix psiEqn
(
    fvm::ddt(psi)
  + fvm::div(phi, psi)
  - fvm::laplacian(gamma, psi)
 ==
    S_psi
);
```

这里，`S_psi` 为 `fvScalarMatrix` 对象，一般通过 `fvm::Su` / `fvm::Sp` / `fvm::SuSp` 、fvOptions 等得到[1][2]。

`==` 运算优先级低于 `+` 和 `-`，因此 `==` 最后计算。上面这段代码执行时，先计算 `==` 左右的表达式，得到两个 `fvScalarMatrix` 对象，再将这两个对象进行 `==` 运算。`==` 操作符已经被重载，实际为 `-` 操作符：

C++

```
// src/finiteVolume/fvMatrices/fvMatrix/fvMatrix.C
template<class Type>
Foam::tmp<Foam::fvMatrix<Type>> Foam::operator==
(
    const fvMatrix<Type>& A,
    const fvMatrix<Type>& B
)
{
    checkMethod(A, B, "==");
    return (A - B);
}
```

最后做的是类似 fvmA - fvmB 的操作，源项放在 fvmB 中。

OpenFOAM 对源项的实现比较灵活，有多种实现方式，这里介绍比较常见的 SuSp。

Su 和 Sp

对于 fvmB 而言，S_u 将从 $[\mathbf{b}]$ 中被减去，相应代码如下：

C++

```
// src/finiteVolume/finiteVolume/fvm/fvmSup.C
template<class Type>
Foam::tmp<Foam::fvMatrix<Type>>
Foam::fvm::Su
(
    const DimensionedField<Type, volMesh>& su,
    const GeometricField<Type, fvPatchField, volMesh>& vf
)
{
    const fvMesh& mesh = vf.mesh();

    tmp<fvMatrix<Type>> tfvm
    (
        new fvMatrix<Type>
        (
            vf,
            dimVol*su.dimensions()
        )
```

```
    );
    fvMatrix<Type>& fvm = tfvm.ref();

    fvm.source() -= mesh.V()*su.field();

    return tfvm;
}
```

同样，对于 fvmB，S_p 将加到 $[\mathbf{A}]$ 的对角上，相应代码如下：

C++

```
// src/finiteVolume/finiteVolume/fvm/fvmSup.C
template<class Type>
Foam::tmp<Foam::fvMatrix<Type>>
Foam::fvm::Sp
(
    const volScalarField::Internal& sp,
    const GeometricField<Type, fvPatchField, volMesh>& vf
)
{
    const fvMesh& mesh = vf.mesh();

    tmp<fvMatrix<Type>> tfvm
    (
        new fvMatrix<Type>
        (
            vf,
            dimVol*sp.dimensions()*vf.dimensions()
        )
    );
    fvMatrix<Type>& fvm = tfvm.ref();

    fvm.diag() += mesh.V()*sp.field();

    return tfvm;
}
```

源项自动处理

对于复杂的源项，在不同位置的正负可能不同。为了使整个线性方程组的求解更稳定，需要对正负源项分开处理。具体做法如下：

- 若 $S_\psi < 0$，则对其采用隐式离散，即 $S_u = 0$，$S_p = S_\psi / \psi^o$，将其贡献计入 $[\mathbf{A}]$ 对角。
- 若 $S_\psi > 0$，则对其采用显式离散，即 $S_u = S_\psi$，$S_p = 0$，将其贡献计入 $[\mathbf{b}]$ 中。

以上过程通过 `fvm::SuSp` 这个函数实现：

C++

```
// src/finiteVolume/finiteVolume/fvm/fvmSup.C
template<class Type>
Foam::tmp<Foam::fvMatrix<Type>>
Foam::fvm::SuSp
(
    const volScalarField::Internal& susp,
    const GeometricField<Type, fvPatchField, volMesh>& vf
)
{
    const fvMesh& mesh = vf.mesh();

    tmp<fvMatrix<Type>> tfvm
    (
        new fvMatrix<Type>
        (
            vf,
            dimVol*susp.dimensions()*vf.dimensions()
        )
```

```
    );
    fvMatrix<Type>& fvm = tfvm.ref();

    fvm.diag() += mesh.V()*max(susp.field(), scalar(0));

    fvm.source() -= mesh.V()*min(susp.field(), scalar(0))
        *vf.primitiveField();

    return tfvm;
}
```

这个函数可以将自动处理正负源项，对 fvmB 而言：

- 若 `susp` 大于 0，则执行 `fvm.diag() += mesh.V()*susp.field()`，这对应 $S_\psi < 0$。
- 若 `susp` 小于 0，则执行 `fvm.source() -= mesh.V()*susp.field()*vf.primitiveField()`，这对应 $S_\psi > 0$。

注意事项

`fvm::Sp` 在使用时，必须保证得到的对角元素为负，这样才能增强对角占优。若传递的sp为正，则需要在前面加上负号，形式大致如下：

C++
```
fvmA == - fvm::Sp(sp, psi)
```

同理，`fvm::SuSp` 写在 `==` 右边时，按照约定，必须在前面加负号，形式大致如下：

C++
```
fvmA == - fvm::SuSp(susp, psi)
```

应用示例

以 Spalart-Allmaras 这个一方程湍流模型为例。原始文献 [3] 中需要求解的输运方程形式如下：

$$\frac{\partial(\rho\tilde{\nu})}{\partial t}+\underbrace{\nabla\cdot(\rho\tilde{\nu}\mathbf{u})}_{\text{advection}}=\underbrace{\left[\nabla\cdot(\rho D_{\tilde{\nu}}\nabla\tilde{\nu})+\frac{1}{\sigma}C_{b2}\rho(\nabla\tilde{\nu})^2\right]}_{\text{diffusion}}+\underbrace{C_{b1}(1-f_{t2})\rho\tilde{S}\tilde{\nu}}_{\text{procduction}}-\underbrace{\left(C_{w1}f_w-\frac{C_{b1}}{\kappa^2}f_{t2}\right)\rho\left(\frac{\tilde{\nu}}{d}\right.}_{\text{destruction}}$$

OpenFOAM 的代码没有实现 f_{t2} 项，这在 *SpalartAllaras.H* 文件中已经说明：

> The model is implemented without the trip-term and hence the ft2 term is not needed.

于是公式简化成

$$\frac{\partial(\rho\tilde{\nu})}{\partial t}+\underbrace{\nabla\cdot(\rho\tilde{\nu}\mathbf{u})}_{\text{advection}}=\underbrace{\left[\nabla\cdot(\rho D_{\tilde{\nu}}\nabla\tilde{\nu})+\frac{1}{\sigma}C_{b2}\rho(\nabla\tilde{\nu})^2\right]}_{\text{diffusion}}+\underbrace{C_{b1}\rho\tilde{S}\tilde{\nu}}_{\text{procduction}}-\underbrace{(C_{w1}f_w)\rho\left(\frac{\tilde{\nu}}{d}\right)^2}_{\text{destruction}}\quad(10)$$

代码实现如下：

C++
```
// src/TurbulenceModels/turbulenceModels/RAS/SpalartAllmaras/SpalartAllmaras.C
    tmp<fvScalarMatrix> nuTildaEqn
    (
```

```
          fvm::ddt(alpha, rho, nuTilda_)
        + fvm::div(alphaRhoPhi, nuTilda_)
        - fvm::laplacian(alpha*rho*DnuTildaEff(), nuTilda_)
        - Cb2_/sigmaNut_*alpha*rho*magSqr(fvc::grad(nuTilda_))
       ==
          Cb1_*alpha*rho*Stilda*nuTilda_
        - fvm::Sp(Cw1_*alpha*rho*fw(Stilda)*nuTilda_/sqr(y_), nuTilda_)
        + fvOptions(alpha, rho, nuTilda_)
      );
```

我们重点关注 production 和 destruction 项。其中，production 项大于0，用显式处理：

C++

```
          Cb1_*alpha*rho*Stilda*nuTilda_
```

这里也可以使用 `fvm::Su`

C++

```
          fvm::Su(Cb1_*alpha*rho*Stilda*nuTilda_, nuTilda_)
```

destruction 项小于0，用隐式处理：

C++

```
        - fvm::Sp(Cw1_*alpha*rho*fw(Stilda)*nuTilda_/sqr(y_), nuTilda_)
```

参考资料

1. https://www.cfd-online.com/Forums/openfoam-solving/60454-solver-details.html [return]
2. https://www.cfd-online.com/Forums/openfoam-programming-development/143281-negative-source-term-issues-fvscalarmatrix.html#post515436 [return]
3. Spalart, P. R., & Allmaras, S. R. (1994). A one-equation turbulence model for aerodynamic flows. *Recherche Aerospatiale*, *1*, 5–21. [return]

Author : wwzhao

LastMod : 2019-10-12

#source term #linearization #su #sp #susp

‹ OpenFOAM 中的对象注册机制

PISO 算法 ›

在 MARINECFD 上还有

OpenFOAM 中的对象注册机制

9 个月前 • 4条评论

对象注册（object registry）机制是 OpenFOAM …

OpenFOAM 中的 tmp 类

2 年前 • 1条评论

tmp 类是 OpenFOAM 中用来封装对象的一个类，这里将介绍 tmp

不可压缩流体 Navier-Stokes 方程的几种形式

2 年前 • 3条评论

在文献中会经常看到各种不同形式的 Navier-Stokes …

用 C++11 实现迷(

2 年前 • 1条评论

本文用 C++11 标准实个类似 OpenFOAM

汪洋 • 1 年前
真好。难怪是华科的大神和上交的大神
1 • 回复 • 分享 ›

Cr. Guan • 2 个月前
请问博主知不知道 使用 DarcyForchheimer Equation的动量源项在openFOAM中，为什么以这样的形式加入

```
forAll(cells, i)
{
const label celli = cells[i];
const label j = this->fieldIndex(i);
const tensor Cd =
mu[celli]*dZones[j] + (rho[celli]*mag(U[celli]))*fZones[j];

const scalar isoCd = tr(Cd);

Udiag[celli] += V[celli]*isoCd;
Usource[celli] -= V[celli]*((Cd - I*isoCd) & U[celli]);
}
```

这段代码，我跟openFOAMwiki里面给的公式，对应不上https://openfoamwiki.net/in...

$$S_m = -(\mu \mathbf{D} + \frac{1}{2}\rho \mathbf{tr}(\mathbf{U}\cdot\mathbf{I})\mathbf{F})\mathbf{U}$$

恳请指点，万分感谢
• 回复 • 分享 ›

Weiwen Zhao 管理员 ➚ Cr. Guan • 2 个月前
Hi，我不了解 Darcy Forchheimer 模型，不过可以尝试解答一下。

1. `fZones` 为 0.5 f，这在 `DarcyForchheimer.C` 里面赋值。因此 Cd 即对应你给的计算公式。

2. Cd 是一个 tensor，为了数值稳定性，将其分为 implicit isotropic (isoCd) 和 explicit deviatoric (Cd-isoCd)，可参考这个链接。

• 回复 • 分享 ›

Cr. Guan ➚ Weiwen Zhao • 2 个月前
感谢博主及时回复，从你的博客学到了许多东西，感谢。
• 回复 • 分享 ›

realRabbita • 1 年前

灰常感谢大佬分享～

回复 • 分享 ›

Cp Zhao • 1 年前

膜拜大佬，学习了

回复 • 分享 ›

订阅 在您的网站上使用 Disqus添加 Disqus添加 Do Not Sell My Data