海兮吾槊
仰望星空3763

OpenFOAM里的MULES::correct依据的原理是FCT通量修正的方法，把MULES的代码解析跟之前写的FCT合并一下，功德圆满～

以下先讲一下FCT的原理，再分析MULES的代码

一、FCT原理

无源相的对流方程：

$$\frac{\partial \phi}{\partial t} + \nabla \cdot (\phi U) = 0$$

设 $F = \phi U$，为标量传递通量，离散有：

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} V + \sum_f (F^n \cdot S) = 0$$

对一维问题有：

$$\phi_i^{n+1} = \phi_i^n - \frac{\Delta t}{V}\left(F^n_{i+\frac{1}{2}} - F^n_{i-\frac{1}{2}}\right)$$

FCT限制器需要把通量 F 分解成两个部分，一部分是由低阶方法计算的通量 F^L，可保证有界性，但数值耗散较大，精度不够；另一部分是由高阶方法计算的通量 F^H。

定义Anti-Diffusive Flux: $A = F^H - F^L$

设 $\lambda \in (0,1)$ 限制器，防止 A 改变局部极值。修正后的通量 F：

$$F^{Corr} = F^L + \lambda A$$

则原对流方程离散形式为：

$$
\begin{aligned}
\left(\phi_i^H\right)^{n+1} &= \phi_i^n - \frac{\Delta t}{V}\left(F_{i+\frac{1}{2}}^{Corr} - F_{i-\frac{1}{2}}^{Corr}\right) \\
&= \phi_i^n - \frac{\Delta t}{V}\left(F_{i+\frac{1}{2}}^{L} - F_{i-\frac{1}{2}}^{L}\right) - \frac{\Delta t}{V}\left(\lambda_{i+\frac{1}{2}} A_{i+\frac{1}{2}} - \lambda_{i-\frac{1}{2}} A_{i-\frac{1}{2}}\right) \\
&= \left(\phi_i^L\right)^{n+1} - \frac{\Delta t}{V}\left(\lambda_{i+\frac{1}{2}} A_{i+\frac{1}{2}} - \lambda_{i-\frac{1}{2}} A_{i-\frac{1}{2}}\right)
\end{aligned}
$$

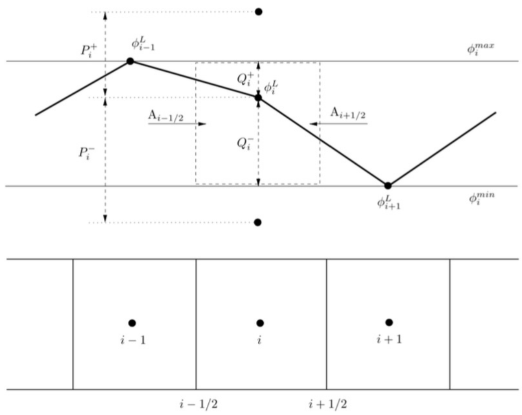
进一步，需要确定限制器 λ 的取值。其在边界处的通量有进有出：

设 $P_i^{\pm}$ 为 A 的入口/出口通量：

$$
P_i^{+} = -\sum_f \left(A_f^{-}\right), \quad P_i^{-} = \sum_f \left(A_f^{+}\right)
$$

设 $Q_i^{\pm}$ 为由局部极值所得 A 的总通量：

$$
Q_i^{+} = \frac{V}{\Delta t}\left(\phi_i^{max} - \phi_i^n\right) + \sum_f F_f^L, \quad Q_i^{-} = -\left[\frac{V}{\Delta t}\left(\phi_i^{min} - \phi_i^n\right) + \sum_f F_f^L\right]
$$



有如下情况：

case1:

如果 $P_i^{\pm} = 0$，此时有 $\phi_i^H = \phi_i^L$ ，并不需要修正，则 $\lambda_i^{\pm} = 0$。

case2:

若 $P_i^{\pm} > 0$，意味着 $\phi_i^H \neq \phi_i^L$，此时边界产生了anti-diffusive flux的净通量，所以需对低阶方法计算的结果进行修正。

由于通量修正的基本原则是要保证 $\phi_i \in (\phi_{i-1}, \phi_{i+1})$ 有界性，$Q_i^{\pm}$ 代表了可修正的最大/最小值范围，则有：

$Q_i^{\pm} > P_i^{\pm}$，即anti-diffusive flux的修正值在**最值范围内**，用 $P_i^{\pm}$ 修正；
$Q_i^{\pm} < P_i^{\pm}$，即anti-diffusive flux的修正值**超过最值范围**，用 $Q_i^{\pm}$ 修正。

综上有：

$$\lambda_i^{\pm} = \begin{cases} \min\left(1, \frac{Q_i^{\pm}}{P_i^{\pm}}\right), & \text{if } P_i^{\pm} > 0 \\ 0, & \text{if } P_i^{\pm} = 0 \end{cases}$$

由cell向face插值时，取值如下：

$$\lambda_{i+\frac{1}{2}} = \begin{cases} \min\left(\lambda_{i+1}^{+}, \lambda_i^{-}\right), & \text{if } A_{i+\frac{1}{2}} > 0 \\ \min\left(\lambda_{i+1}^{-}, \lambda_i^{+}\right), & \text{if } A_{i+\frac{1}{2}} < 0 \end{cases}$$

以上，FCT修正的原理简述完毕

二、MULES算法

MULESsolver，位置在$FOAM_SRC/finiteVolume/fvMatrices/solvers/MULES文件夹内。内部包含三种类型的求解方法，对应如下

CMULES: 基于FCT的**通量修正**的求解器
MULES: 显式**直接**求解（explicitSolve）
IMULES: 隐式**直接**求解（implicitSolve）

在interPhaseChangeFoam求解器的alphaEqn.H内，提供了两个选择，通量修正CMULES和直接求解MULES，开关是bool MULESCorr。本文主要分析CMULES算法。

CMULES算法包含三个文件

CMUELS.H
CMULES.C
CMULESTemplates.C

其中，limiter计算、anti-diffusive flux的计算和返回修正场的函数都在CMULESTemplates.C中定义

计算limiter的函数:

```
template<class RdeltaTType, class RhoType, class SpType, class SuType>
void limiterCorr
(
    scalarField& allLambda,
    const RdeltaTType& rDeltaT,
    const RhoType& rho,
    const volScalarField& psi,
    const surfaceScalarField& phi,
    const surfaceScalarField& phiCorr,
    const SpType& Sp,
    const SuType& Su,
    const scalar psiMax,
    const scalar psiMin
);
```

计算带限制器的anti-diffusive flux的函数:

```
template<class RdeltaTType, class RhoType, class SpType, class SuType>
void limitCorr
(
    const RdeltaTType& rDeltaT,
    const RhoType& rho,
    const volScalarField& psi,
    const surfaceScalarField& phi,
    surfaceScalarField& phiCorr,
    const SpType& Sp,
    const SuType& Su,
    const scalar psiMax,
    const scalar psiMin
);
```

返回修正场:

```
template<class RdeltaTType, class RhoType, class SpType, class SuType>
void correct
(
    const RdeltaTType& rDeltaT,
```

```
    const RhoType& rho,
    volScalarField& psi,
    const surfaceScalarField& phi,
    const surfaceScalarField& phiCorr,
    const SpType& Sp,
    const SuType& Su
);
```

alphaEqn.H中correct的调用，和CMULES中的接口函数：

```
//调用
MULES::correct
(
        geometricOneField(),
        alpha1,
        talphaPhi(),
        talphaPhiCorr.ref(),
        vDotvmcAlphal,
        (
            divU*(alpha10 - alpha100)
          - vDotvmcAlphal*alpha10
        )(),
        1,
        0
);

//接口函数
template<class RhoType, class SpType, class SuType>
void correct
(
    const RhoType& rho,
    volScalarField& psi,
    const surfaceScalarField& phi,
    surfaceScalarField& phiCorr,
    const SpType& Sp,
    const SuType& Su,
    const scalar psiMax,
    const scalar psiMin
);
```

为了方便说明，就采用这个带源项的调用做例子，对应一下传入函数的8个变量：

接口函数变量	调用传入变量
rho	$\mathrm{geometricOneField()}$
psi	$\mathrm{alpha1}$
phi	$\mathrm{talphaPhi()}$
$phiCorr$	$\mathrm{talphaPhiCorr.ref()}$
Sp	$\mathrm{vDotvmcAlphal}$
Su	$\mathrm{divU} * (\mathrm{alpha10} - \mathrm{alpha100}) - \mathrm{vDotvmcAlphal} * \mathrm{alpha10}$
$psiMax$	1
$psiMin$	0

limiterCorr函数：尽管该函数代码很长，但思路很简单。分为三步，通过迭代计算 $\lambda_{i\pm\frac{1}{2}}$ ：

Step1: 求得每个cell附近的最大值，并记录每个cell的 $P_i^{\pm}$ ，代码如下：

```
forAll(phiCorrIf, facei)
    {
        label own = owner[facei];
        label nei = neighb[facei];

        psiMaxn[own] = max(psiMaxn[own], psiIf[nei]);
        psiMinn[own] = min(psiMinn[own], psiIf[nei]);

        psiMaxn[nei] = max(psiMaxn[nei], psiIf[own]);
        psiMinn[nei] = min(psiMinn[nei], psiIf[own]);

        scalar phiCorrf = phiCorrIf[facei];

        if (phiCorrf > 0.0)
        {
            sumPhip[own] += phiCorrf;
            mSumPhim[nei] += phiCorrf;
        }
        else
        {
            mSumPhim[own] -= phiCorrf;
            sumPhip[nei] -= phiCorrf;
        }
    }
```

Step2: 基于 $psiMax$ 和 $psiMin$ 计算 $Q_i^{\pm}$

```
psiMaxn =
    V
   *(
       (rho.field()*rDeltaT - Sp.field())*psiMaxn
     - Su.field()
     - rho.field()*psi.internalField()*rDeltaT
    );

psiMinn =
    V
   *(
       Su.field()
     - (rho.field()*rDeltaT - Sp.field())*psiMinn
     + rho.field()*psi.internalField()*rDeltaT
    );
```

Step3: 迭代求 $\lambda_{i\pm\frac{1}{2}}$ ，开始时 λ 被初始化为1的场，在fvSolution文件中指定迭代次数nLimiterIter，迭代公式和对应的代码部分如下：

$$\lambda_i^{\mp,n+1} = \max\left[\min\left(\frac{\pm\sum_f \lambda_f^n A_f^{\pm} + Q_i^{\pm}}{P_i^{\pm}}, 1\right), 0\right]$$

```
forAll(sumlPhip, celli)
{
    sumlPhip[celli] =
        max(min
        (
            (sumlPhip[celli] + psiMaxn[celli])
           /(mSumPhim[celli] - SMALL),
            1.0), 0.0
        );

    mSumlPhim[celli] =
        max(min
        (
            (mSumlPhim[celli] + psiMinn[celli])
           /(sumPhip[celli] + SMALL),
            1.0), 0.0
        );
}
```

根据 A_f 通量方向，更新面上的 λ_f^{n+1} 。代码部分如下，公式见上方FCT原理：

```
forAll(lambdaIf, facei)
{
    if (phiCorrIf[facei] > 0.0)
    {
            lambdaIf[facei] = min
                (
                    lambdaIf[facei],
                        min(lambdap[owner[facei]], lambdam[neighb[facei]])
                );
    }
    else
        {
            lambdaIf[facei] = min
                (
                    lambdaIf[facei],
                        min(lambdam[owner[facei]], lambdap[neighb[facei]])
                );
        }
}
```

limitCorr函数：主要代码段如下，先调用limiterCorr计算 $\lambda_{i\pm\frac{1}{2}}$ ，记录在类型为scalarField的变量allLambda中，而后将 $\lambda_{i\pm\frac{1}{2}}$ 与phiCorr(talphaPhiCorr)相乘，即为前述的修正通量 $\lambda_{i\pm\frac{1}{2}} A_{i\pm\frac{1}{2}}$

```
limiterCorr
    (
        allLambda,
        rDeltaT,
        rho,
        psi,
        phi,
        phiCorr,
        Sp,
        Su,
        psiMax,
        psiMin
    );
```

```
    phiCorr *= lambda;
```

correct函数：重点说明修正场 α_l^H 的计算，为什么源项Sp和Su传入的是那两个东西？以及代码中让人摸不到头脑的算法~

先放上更新 α_l 代码段：

```
{
    psi.internalField() =
    (
            rho.field()*psi.internalField()*rDeltaT
          + Su.field()
          - psiIf
        )/(rho.field()*rDeltaT - Sp.field());
}
```

翻译一下是这样的：

$$\alpha_l=\frac{\frac{\alpha_l}{\Delta t}+S_u-\nabla\cdot A}{\frac{1}{\Delta t}-S_p}$$

是不是很懵哔？详情这样滴，首先，要求解的相方程：

（出门左转，interPhaseChangeFoam求解器解析）

$$\frac{\partial\alpha_l}{\partial t}+\nabla\cdot(\alpha_l U)-\alpha_l\nabla\cdot U=\frac{\rho}{\rho_l\rho_v}\dot{m}$$

用“[]”代表数值离散，上标H代表高阶格式，L低阶格式。H和L的时间格式都用一阶Euler，则有：

$$\begin{cases}\frac{\left(\alpha_l^H\right)^{n+1}-\alpha_l^n}{\Delta t}+[\nabla\cdot(\alpha_l U)]^H-\left(\alpha_l^H\right)^{n+1}\nabla\cdot U=S_p\left(\alpha_l^H\right)^{n+1}+Su^m\\ \frac{\left(\alpha_l^L\right)^{n+1}-\alpha_l^n}{\Delta t}+[\nabla\cdot(\alpha_l U)]^L-\left(\alpha_l^L\right)^{n+1}\nabla\cdot U=S_p\left(\alpha_l^L\right)^{n+1}+Su^m\end{cases}$$

注意，这里的 S_u^m 是离散的 $\dot{m}$ 产生的，与 S_u 的不是一回事，将上两式相减 S_u^m 消失，有：

$$\frac{\left(\alpha_l^H\right)^{n+1}-\left(\alpha_l^L\right)^{n+1}}{\Delta t}+\nabla\cdot A-\left(\alpha_l^H-\alpha_l^L\right)^{n+1}\nabla\cdot U=S_p\left(\alpha_l^H-\alpha_l^L\right)^{n+1}$$

整理上式，有：

$$\left(\frac{1}{\Delta t}-S_p\right)\left(\alpha_l^H\right)^{n+1}=\frac{\left(\alpha_l^L\right)^{n+1}}{\Delta t}+\left[\left(\alpha_l^H\right)^{n+1}-\left(\alpha_l^L\right)^{n+1}\right]\nabla\cdot U-S_p\left(\alpha_l^L\right)^{n+1}-\nabla\cdot A$$

注意开启了MULESCorr后， $\left(\alpha_l^L\right)^{n+1}$ 是由预测步计算的已知量；为了保证算法鲁棒性，加入relaxFactor，则有

$$\left(\alpha_l^H\right)^{n+1}=\frac{\frac{\left(\alpha_l^L\right)^{n+1}}{\Delta t}+\left(\alpha_l^n-\alpha_l^{n-1}\right)\nabla\cdot U-S_p\alpha_l^n-\nabla\cdot A}{\frac{1}{\Delta t}-S_p}$$

其中， $S_u=\left(\alpha_l^n-\alpha_l^{n-1}\right)\nabla\cdot U-S_p\alpha_l^n$

以上