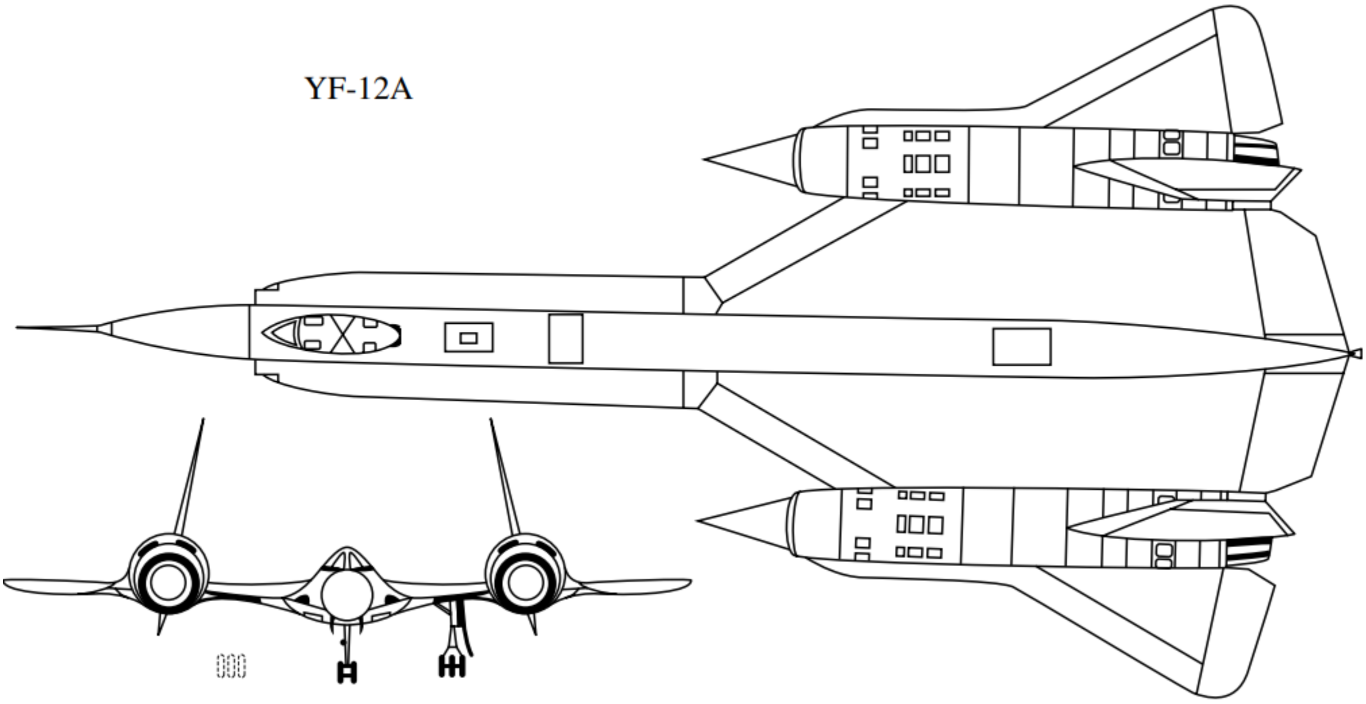
Adjoint Optimization

Fr13ndSDP 收录于 CFD
2020-12-03 约 3491 字 预计阅读 7 分钟



伴随优化基本方程的推导以及OpenFOAM中的实现

> 使用如下的符号记法：全导数 $d_x\ (\nabla_x)$，偏导 ∂_x，微分 d.

1 伴随方法

在一个偏微分方程系统中，假设存在变量 $x \in R^{n_x}, p \in R^{n_p}$，方程 $f(x,p) : R^{n_x} \times R^{n_p} \to R$以及关系 $g(x,p) = 0$，且 g_x 处处非奇异，$d_p f$怎么求？

1.1 动机

$g(x,p) = 0$ 的求解是CFD求解器的核心，给定参数 p，程序将计算出 x。例如 p 可以是边界条件或初始条件的参数，也可以是物性参数，而 x 是计算得到的场量，这种求解模式是*正问题*。如果引入 $f(x,p)$作为度量标准，例如用来计算 x 的光滑程度，那么通常需要最小化f，这种求解模式是*反问题*。

梯度 $d_p f$ 很有用，可以用来计算优化问题 $min_p f$ 中 f 对于参数 p 变化的敏感程度，并使用梯度下降方法求解最优化问题。

1.2 推导

1）

考虑一个简单的函数 $f(x)$，首先由链式法则有

$$d_p f = d_d f(x(p)) = \partial_x f d_p x \; (= f_x x_p)$$

其次因为 $g(x,p) = 0$ 处处成立，则 $d_p g = 0$， 也就是

$$g_x x_p + g_p = 0$$

那么得到

$$d_p f = -f_x g_x^{-1} g_p$$

从线性代数的观点看，$f_x g_x^{-1}$ 是一个行向量乘以 $n_x \times n_x$ 的矩阵，并且是以下方程的解

$$g_x^T \lambda = -f_x^T$$

称为伴随方程，λ称为伴随变量。得到 $d_p f = \lambda^T g_p$

2）

定义拉格朗日函数

$$L(x,p,\lambda) = f(x) + \lambda^T g(x,p)$$

这里λ是拉格朗日乘子组成的向量，由于 $g(x,p)$ 处处为零，λ可以随意选取。

$$\begin{aligned} d_p f(x) = d_p L &= f_x x_p + d_p \lambda^T g + \lambda^T (g_x x_p + g_p) \\ &= (f_x + \lambda^T g_x) x_p + \lambda^T g_p \end{aligned}$$

如果选取 $g^T \lambda = -f_x^T$，第一项为零，可以避免计算 x_p 并且得到 $d_p f = \lambda^T g_p$，与第一种推导方式相同。

或者，由于$df = f_x dx + f_p dp = (f_x + \lambda^T g_x) dx + \lambda^T g_p dp$，可以推至同样的结果。

2 PDE约束的连续伴随问题

2.1 一般伴随方程

给出一个特定的优化问题

$$\begin{aligned} min \quad & J = J(\mathbf{v}, p, \alpha) \\ s.t. \quad & R(\mathbf{v}, p, \alpha) = 0 \end{aligned}$$

考虑由连续介质力学给出的约束，$\mathbf{v}$ 与 p 作为变量，即等价为前文中的 x; α 作为 设计参数，即等价于前文中的p，给出问题

$$\begin{aligned} min \quad & J = J(\mathbf{v}, p, \alpha) \\ s.t. \quad & R^u = \nabla \cdot (\mathbf{v}\mathbf{v}) + \nabla p - \nabla \cdot (2\nu D(\mathbf{v})) + \alpha \mathbf{v} = 0 \\ & R^p = \nabla \cdot \mathbf{v} = 0 \end{aligned}$$

上式由达西定律引入了渗透率作为源项，假设某一区域使得目标函数增大，可以通过减小这一区域的渗透率作为惩罚。$D(\mathbf{v})=\frac{1}{2}(\nabla\mathbf{v}+\nabla\mathbf{v}^T)$ 代表应变率张量。

使用拉格朗日乘数法将其转变为无约束优化问题

$$
\begin{aligned}
min \quad L &= J+\int_\Omega (\mathbf{u},q)Rd\Omega \\
&= J+\int_\Omega qR^p d\Omega+\int_\Omega \mathbf{u}\cdot R^u d\Omega
\end{aligned}
$$

其中$(\mathbf{u},q)$为拉格朗日乘子，称其为伴随速度和伴随压力（后面会看到为什么），但是实际上并没有物理含义。

因为变分

$$\delta L=\delta_\alpha L+\delta_\mathbf{v} L+\delta_p L$$

注意这里没有将粘度作为微分变量，是一种近似的做法，被称为“冻结湍流”。

由于 $(\mathbf{u},q)$可以自由选取，取 适当的值使得 $\delta_\mathbf{v} L+\delta_p L=0$ 成立，便得到

$$\delta_\alpha L=\delta_\alpha J+\int_\Omega \mathbf{u}\cdot\mathbf{v}d\Omega$$

当考虑网格中的目标函数关于 α的梯度，可以得到

$$\frac{\partial L}{\partial \alpha_i}=\frac{\partial J}{\partial \alpha_i}+\mathbf{u_i}\cdot\mathbf{v_i}V_i$$

其中 V_i 为网格体积。

如果 $\delta_\mathbf{v} L+\delta_p L=0$ 成立，那么伴随方程

$$
\begin{aligned}
\delta_\mathbf{v} J+\delta_p J+\int_\Omega \mathbf{u}\cdot[\nabla\cdot(\mathbf{v}\delta\mathbf{v})+\nabla\cdot(\delta\mathbf{v}\mathbf{v})-\nabla\cdot(2\nu D(\delta\mathbf{v}))+\alpha\delta\mathbf{v}]d\Omega \\
-\int_\Omega q\nabla\cdot\delta\mathbf{v}d\Omega+\int_\Omega \mathbf{u}\cdot\nabla\delta p d\Omega=0
\end{aligned}
$$

积分使用散度公式与高斯散度定理(以及一些张量运算的推导)：

$$
\begin{gathered}
\nabla\cdot(q\mathbf{v})=\nabla q\cdot\mathbf{v}+q\nabla\cdot\mathbf{v} \\
\int_\Omega \nabla\cdot(q\mathbf{v})\ d\Omega=\int_\Gamma q\mathbf{v}\cdot\mathbf{n}d\Gamma
\end{gathered}
$$

并且把 J 拆分为

$$J=\int_\Gamma J_\Gamma\ d\Gamma+\int_\Omega J_\Omega\ d\Omega$$

得到 (*注意* $(\mathbf{v}\cdot\nabla)\mathbf{u}=\nabla(\mathbf{v}\mathbf{u})$)

$$
\begin{aligned}
&\int_\Gamma \mathrm{d}\Gamma\left(\mathbf{u}\cdot\mathbf{n}+\frac{\partial J_\Gamma}{\partial p}\right)\delta p+\int_\Omega \mathrm{d}\Omega\left(-\nabla\cdot\mathbf{u}+\frac{\partial J_\Omega}{\partial p}\right)\delta p\\
&+\int_\Gamma \mathrm{d}\Gamma\left(\mathbf{n}(\mathbf{u}\cdot\mathbf{v})+\mathbf{u}(\mathbf{v}\cdot\mathbf{n})+2v\mathbf{n}\cdot\mathbf{D}(\mathbf{u})-q\mathbf{n}+\frac{\partial J_\Gamma}{\partial \mathbf{v}}\right)\cdot\delta\mathbf{v}-\int_\Gamma \mathrm{d}\Gamma 2v\mathbf{n}\cdot\mathbf{D}(\delta\mathbf{v})\cdot\mathbf{u}\\
&+\int_\Omega \mathrm{d}\Omega\left(-\nabla\mathbf{u}\cdot\mathbf{v}-(\mathbf{v}\cdot\nabla)\mathbf{u}-\nabla\cdot(2v\mathbf{D}(\mathbf{u}))+\alpha\mathbf{u}+\nabla q+\frac{\partial J_\Omega}{\partial \mathbf{v}}\right)\cdot\delta\mathbf{v}=0
\end{aligned}
$$

观察上面的式子，他应该对任意的 δp和 $\delta\mathbf{v}$ 成立，因此各个积分分别等于0，当在 Ω 内积分时，边界积分为零，因此得到 Ω 内的伴随方程

$$
\begin{aligned}
-\nabla(\mathbf{v}\mathbf{u})-\nabla\mathbf{u}\cdot\mathbf{v}&=-\nabla q+\nabla\cdot(2\nu D(\mathbf{u}))-\alpha\mathbf{u}-\frac{\partial J_\Omega}{\partial \mathbf{v}}\\
\nabla\cdot\mathbf{u}&=\frac{\partial J_\Omega}{\partial p}
\end{aligned}
$$

可以看到，这组方程与 N-S 方程非常相似，因此变量 $(\mathbf{u},q)$ 被看作是非物理的速度和压力。

2.2 边界条件

在边界处，可以得到边界条件为

$$
\begin{aligned}
\int_\Gamma \mathrm{d}\Gamma\left(\mathbf{n}(\mathbf{u}\cdot\mathbf{v})+\mathbf{u}(\mathbf{v}\cdot\mathbf{n})+2v\mathbf{n}\cdot\mathbf{D}(\mathbf{u})-q\mathbf{n}+\frac{\partial J_\Gamma}{\partial \mathbf{v}}\right)\cdot\delta\mathbf{v}-\int_\Gamma \mathrm{d}\Gamma 2v\mathbf{n}\cdot\mathbf{D}(\delta\mathbf{v})\cdot\mathbf{u}&=0\\
\int_\Gamma \mathrm{d}\Gamma\left(\mathbf{u}\cdot\mathbf{n}+\frac{\partial J_\Gamma}{\partial p}\right)\delta p&=0
\end{aligned}
$$

或者

$$
\begin{aligned}
\int_\Gamma \mathrm{d}\Gamma\left(\mathbf{n}(\mathbf{u}\cdot\mathbf{v})+\mathbf{u}(\mathbf{v}\cdot\mathbf{n})+2v\mathbf{n}\cdot\mathbf{D}(\mathbf{u})-q\mathbf{n}+\frac{\partial J_\Gamma}{\partial \mathbf{v}}\right)\cdot\delta\mathbf{v}&=0\\
\int_\Gamma \mathrm{d}\Gamma\left(\mathbf{u}\cdot\mathbf{n}+\frac{\partial J_\Gamma}{\partial p}\right)\delta p+\int_\Gamma \mathrm{d}\Gamma 2v\mathbf{n}\cdot\mathbf{D}(\delta\mathbf{v})\cdot\mathbf{u}&=0
\end{aligned}
$$

1） 在入口边界以及壁面处，速度值为给定值或者0，因此 $\delta\mathbf{v}=0$，因此部分积分可以视为0，为了避免无解，显然应该选取第一种边界条件的定义。

因此

$$
\begin{aligned}
\int_\Gamma \mathrm{d}\Gamma 2v\mathbf{n}\cdot\mathbf{D}(\delta\mathbf{v})\cdot\mathbf{u}&=0\\
u_n&=-\frac{\partial J_\Gamma}{\partial p}
\end{aligned}
$$

其中 $u_n=\mathbf{u}\cdot\mathbf{n}$ 为伴随速度在边界法向的分量。现在需要定义u_t 在边界处的取值，这里需要用到一些近似，具体推导参看参考文献1，这里直接给出入口及壁面处应该满足的边界条件：

$$
u_t=0
$$

$$
u_n = -\frac{\partial J_\Gamma}{\partial p}
$$
$$
\mathbf{n} \cdot \nabla q = 0
$$

2) 出口边界常见设为压力为零和速度零梯度，因此除了自动满足的积分，边界条件剩下

$$
\mathbf{n}(\mathbf{u}\cdot\mathbf{v}) + \mathbf{u}(\mathbf{v}\cdot\mathbf{n}) + v(\mathbf{n}\cdot\nabla)\mathbf{u} - q\mathbf{n} + \frac{\partial J_\Gamma}{\partial \mathbf{v}} = 0
$$

3 特定问题–目标函数为能量损失

选取目标函数为流动的能量损失，即最小化能量损失

$$
J = -\int_\Gamma (p + \frac{1}{2}v^2)\mathbf{v}\cdot\mathbf{n}\, d\Gamma
$$

负号是因为考虑到通量的方向，即最小化进出口能量之差。

$J_\Omega = 0, J_\Gamma = -(p + \frac{1}{2}v^2)\mathbf{v}\cdot\mathbf{n}$，因此

$$
\frac{\partial J_\Gamma}{\partial p} = -\mathbf{v}\cdot\mathbf{n}, \quad \frac{\partial J_\Gamma}{\partial \mathbf{v}} = \frac{\partial(-p\mathbf{v}\cdot\mathbf{n} - \frac{1}{2}(\mathbf{v}\cdot\mathbf{v})\mathbf{v}\cdot\mathbf{n})}{\partial \mathbf{v}} = -p\mathbf{n} - (\mathbf{v}\cdot\mathbf{n})\cdot\mathbf{v} - \frac{1}{2}v^2\mathbf{n}
$$

代入伴随方程，得到在 Ω 内

$$
-\nabla(\mathbf{v}\mathbf{u}) - \nabla\mathbf{u}\cdot\mathbf{v} = -\nabla q + \nabla\cdot(2\nu D(\mathbf{u})) - \alpha\mathbf{u}
$$
$$
\nabla\cdot\mathbf{u} = 0
$$

入口边界条件：

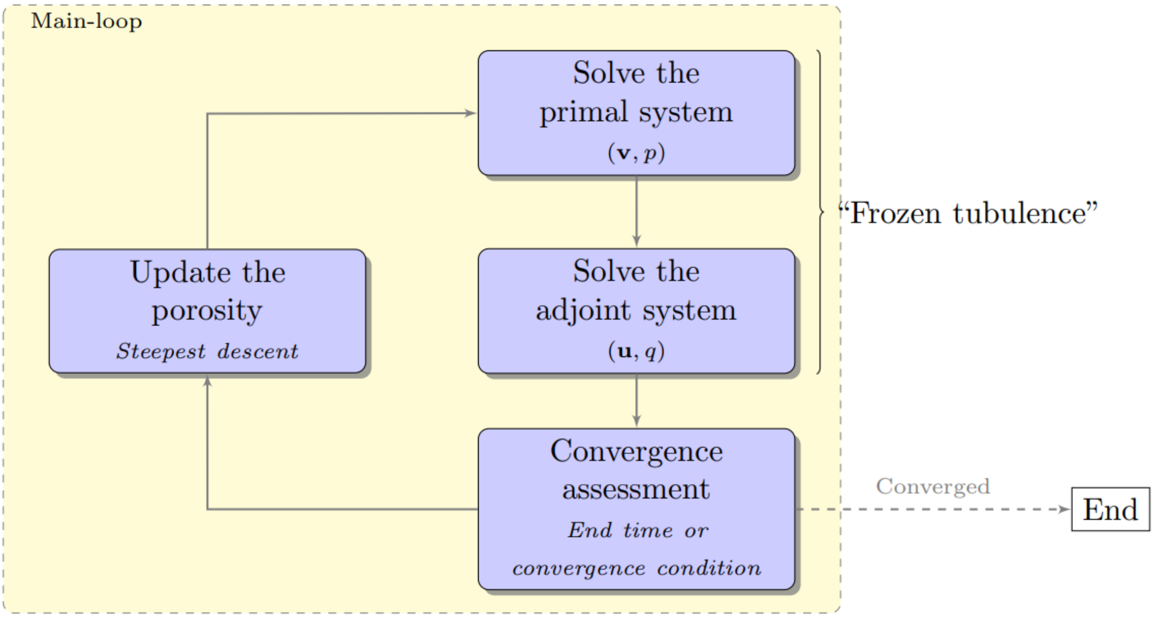
$$
u_t = 0
$$
$$
u_n = v_n
$$
$$
\mathbf{n}\cdot\nabla q = 0
$$

出口边界条件：

$$
\mathbf{n}(\mathbf{u}\cdot\mathbf{v}) + \mathbf{u}(\mathbf{v}\cdot\mathbf{n}) + v(\mathbf{n}\cdot\nabla)\mathbf{u} - q\mathbf{n} - p\mathbf{n} - (\mathbf{v}\cdot\mathbf{n})\cdot\mathbf{v} - \frac{1}{2}v^2\mathbf{n} = 0
$$

4. *OpenFOAM* 中的实现

首先给出算法流程图：



What's next

4.1 原始方程和伴随方程的求解

首先，使用SIMPLE算法求解原始变量

C++

```cpp
// Momentum predictor
            // @turbulence->divDevSigma: 湍流源项，应变率张量散度
            fvVectorMatrix UEqn
            (
                fvm::div(phi, U)
              + turbulence->divDevSigma(U)
              + fvm::Sp(alpha, U)
            );
            UEqn.relax();
            solve(UEqn == -fvc::grad(p));
// omit-----------------------------------------------------
//压力泊松方程
            volScalarField rAU(1.0/UEqn.A());
            volVectorField HbyA(constrainHbyA(rAU*UEqn.H(), U, p));
            surfaceScalarField phiHbyA("phiHbyA", fvc::flux(HbyA));
            fvScalarMatrix pEqn
            (
                fvm::laplacian(rAU, p) == fvc::div(phiHbyA)
            );

            pEqn.setReference(pRefCell, pRefValue);
            pEqn.solve();
// Explicitly relax pressure for momentum corrector
            p.relax();
// Momentum corrector
            U = HbyA - rAU*fvc::grad(p);
```

也就是求解

$$
\begin{gathered}
\nabla \cdot (\mathbf{v}\mathbf{v}) + \nabla p - \nabla \cdot (2\nu D(\mathbf{v})) + \alpha \mathbf{v} = 0 \\
\nabla \cdot \mathbf{v} = 0
\end{gathered}
$$

然后求解伴随方程（与原始变量方程非常相似）

C++

```cpp
// Adjoint Momentum predictor
            volVectorField adjointTransposeConvection((fvc::grad(Ua) & U));
            zeroCells(adjointTransposeConvection, inletCells);
            fvVectorMatrix UaEqn
            (
                fvm::div(-phi, Ua)
              - adjointTransposeConvection
              + turbulence->divDevSigma(Ua)
              + fvm::Sp(alpha, Ua)
            );
            UaEqn.relax()
            solve(UaEqn == -fvc::grad(pa));
```

$$
\begin{gathered}
-\nabla(\mathbf{v}\mathbf{u}) - \nabla \mathbf{u} \cdot \mathbf{v} = -\nabla q + \nabla \cdot (2\nu D(\mathbf{u})) - \alpha \mathbf{u} \\
\nabla \cdot \mathbf{u} = 0
\end{gathered}
$$

注意到在这里，单独对 $\nabla \mathbf{u} \cdot \mathbf{v}$ 在入口边界处置为0，并且处理为显式的源项。我认为这样处理是出于问题适定性的需要，正如前文处理入口边界将伴随压力设置为零梯度一样，将这项消去使得伴随方程和原始方程在入口边界完全一致。

4.2 边界条件的处理

对于入口边界以及壁面，满足以下的边界条件

$$
\begin{gathered}
u_t = 0 \\
u_n = v_n \\
\mathbf{n} \cdot \nabla q = 0
\end{gathered}
$$

即对于速度，壁面使用无滑移条件，入口使用固定值并与原始变量保持一致；在入口和壁面处压力施加零梯度条件。

对于出口边界，则比较复杂。前面推导的边界条件

$$
\mathbf{n}(\mathbf{u} \cdot \mathbf{v}) + \mathbf{u}(\mathbf{v} \cdot \mathbf{n}) + v(\mathbf{n} \cdot \nabla)\mathbf{u} - q\mathbf{n} - p\mathbf{n} - (\mathbf{v} \cdot \mathbf{n}) \cdot \mathbf{v} - \frac{1}{2} v^2 \mathbf{n} = 0
$$

注意原始变量取为零压力，零速度梯度，将其按照法向和切向分解，得到

$$
\begin{gathered}
q = \mathbf{u} \cdot \mathbf{v} + u_n v_n + \nu(\mathbf{n} \cdot \nabla) u_n - \frac{1}{2} v^2 - v_n^2 \\
0 = v_n(\mathbf{u_t} - \mathbf{v_t}) + \nu(\mathbf{n} \cdot \nabla)\mathbf{u_t}
\end{gathered}
$$

分别代表伴随压力和伴随速度满足的边界条件。在计算法向梯度时，使用近似

$$\nu(\mathbf{n}\cdot\boldsymbol{\nabla})u_n = \nu\frac{u_n - u_{n,neighbor}}{\boldsymbol{\Delta}}$$

$$\nu(\mathbf{n}\cdot\boldsymbol{\nabla})\mathbf{u_t} = \nu\frac{\mathbf{u_t} - \mathbf{u_{t,neighbor}}}{\boldsymbol{\Delta}}$$

因此出口处的压力边界表示为

C++

```cpp
const fvsPatchField<scalar>& phip =
    patch().lookupPatchField<surfaceScalarField, scalar>("phi");
const fvsPatchField<scalar>& phiap =
    patch().lookupPatchField<surfaceScalarField, scalar>("phia");
const fvPatchField<vector>& Up =
    patch().lookupPatchField<volVectorField, vector>("U");
const fvPatchField<vector>& Uap =
    patch().lookupPatchField<volVectorField, vector>("Ua");

const incompressible::RASModel& rasModel =
    db().lookupObject<incompressible::RASModel>("momentumTransport");
scalarField nueff = rasModel.nuEff()().boundaryField()[patch().index()];
const scalarField& deltainv = patch().deltaCoeffs(); // m^-1
operator==(phip*phiap/sqr(patch().magSf()) +
    (Uap&Up) +
    nueff*deltainv*(phiap/patch().magSf() -
    (Uap.patchInternalField()&patch().nf())) -
    0.5*sqr(mag(Up)) -
    magSqr(Up&patch().Sf()/patch().magSf()));

fixedValueFvPatchScalarField::updateCoeffs();
```

速度边界表示为

C++

```
const fvsPatchField<scalar>& phiap =
    patch().lookupPatchField<surfaceScalarField, scalar>("phia");
const fvsPatchField<scalar>& phip =
    patch().lookupPatchField<surfaceScalarField, scalar>("phi");
const fvPatchField<vector>& Up =
    patch().lookupPatchField<volVectorField, vector>("U");
const fvPatchField<vector>& Uap =
    patch().lookupPatchField<volVectorField, vector>("Ua");

const incompressible::RASModel& rasModel =
    db().lookupObject<incompressible::RASModel>("momentumTransport");
const scalarField deltainv = patch().deltaCoeffs();
scalarField nueff = rasModel.nuEff()().boundaryField()[patch().index()];
scalarField Un(mag(patch().nf() & Up));
vectorField Ut(Up - phip*patch().nf()/patch().magSf());
vectorField Uaneigh(Uap.patchInternalField());
vectorField Uaneigh_n((Uaneigh&patch().nf())*patch().nf());
vectorField Uaneigh_t(Uaneigh - Uaneigh_n);
// Ut = (V-Vn)/Vn
vectorField Uap_t((Un*Ut + nueff*deltainv*Uaneigh_t)/(Un + deltainv*nueff));
vectorField Uap_n(phiap*patch().nf()/patch().magSf());
// U = Un + Ut
// Un = phi*\vec{A}/(A^2)
vectorField::operator=(Uap_t + Uap_n);

fixedValueFvPatchVectorField::updateCoeffs();
```

4.3 梯度下降法 （Deepest descent method）

完成伴随方程计算后，需要更新孔隙率，如前文推导

$$\frac{\partial L}{\partial \alpha_i} = \mathbf{u_i} \cdot \mathbf{v_i} \mathbf{V_i}$$

按照梯度下降的方向寻找目标函数极值，定义步长为 λ。则

$$\alpha_{n+1} = \alpha_n - \mathbf{u_i} \cdot \mathbf{v_i} V_i \lambda$$

在 *OpenFOAM* 的实现中，我们需要注意给 α 施加限制 $\alpha < \alpha_{max}$，并施加松弛因子保证稳定性，因此给出以下代码

C++

```cpp
// @mesh.fielfRelaxationFactor : fvSolution定义的松弛因子
        alpha +=
            mesh.fieldRelaxationFactor("alpha")
          *(min(max(alpha - lambda*(Ua & U), zeroAlpha), alphaMax) - alpha);
```

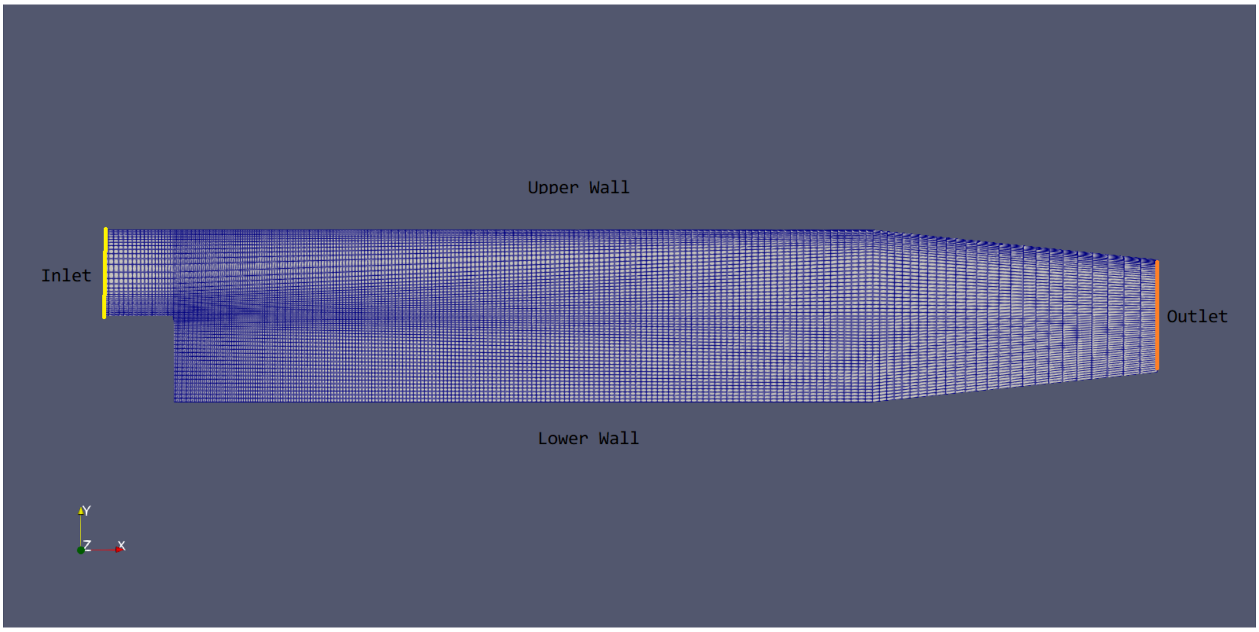
4.4 算例

测试算例为 *OpenFOAM* 自带算例 `pitzDaily`，管道内流动问题，使用 $k-\epsilon$ 湍流模型，采用的物性参数，湍流模型参数以及梯度下降算法参数如下：

粘度	入口湍动能	入口湍流耗散率	梯度下降步长	α_{max}
$\nu = 1 \times 10^{-5}\ m^2/s$	$k = 0.375\ m^2/s^2$	$\epsilon = 14.855\ m^2/s^3$	$\lambda = 1 \times 10^5\ s/m^2$	200

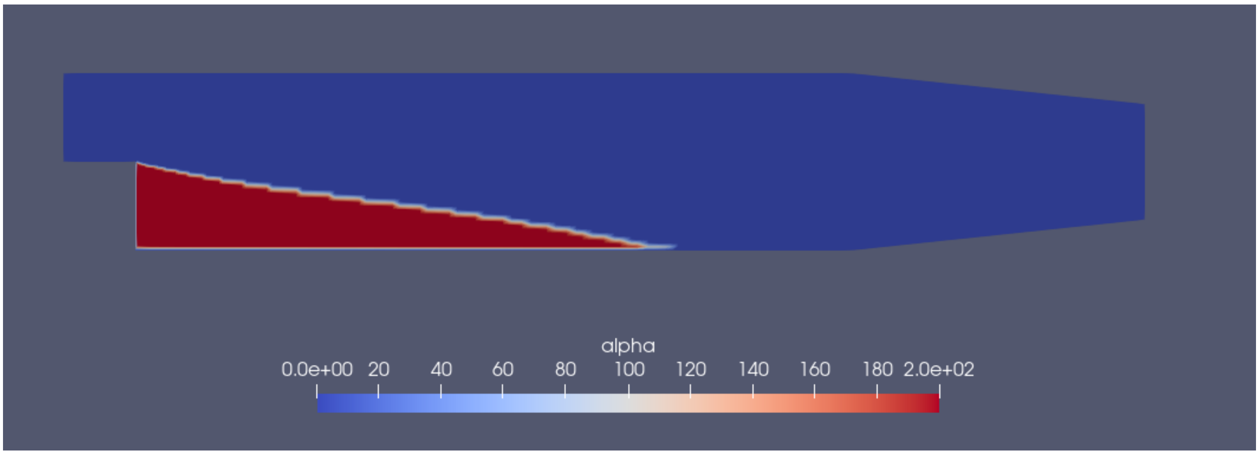
边界条件和使用的网格如下图所示，壁面为无滑移条件。

变量	Inlet	Outlet
$\mathbf{v}$	固定值 $10\ m/s$	零梯度
$\mathbf{u}$	固定值 $10\ m/s$	伴随速度条件
p	零梯度	固定值 0 pa
q	零梯度	伴随压力条件



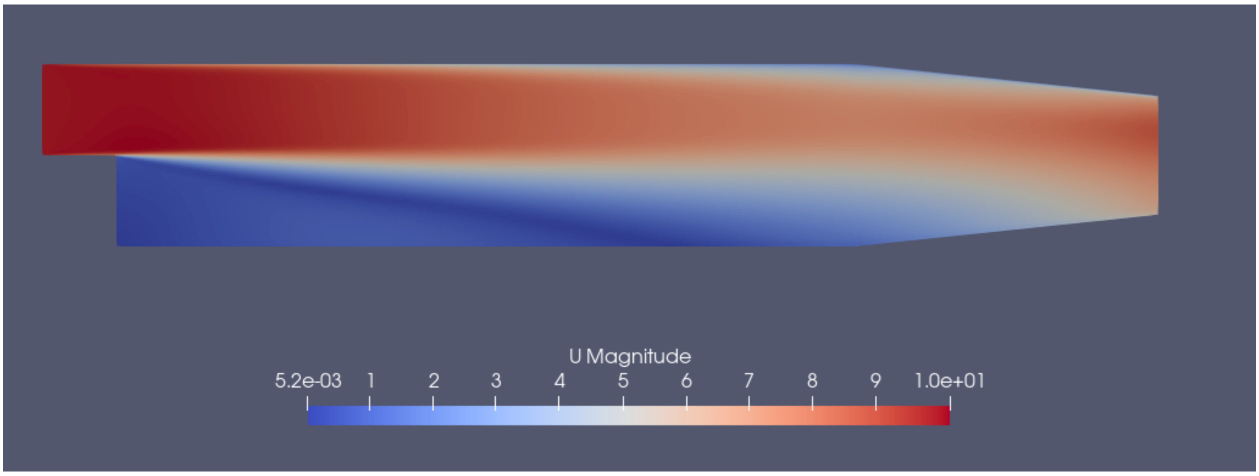
Mesh

N-S方程和伴随方程的求解采用SIMPLE算法，α 松弛因子设为0.1。每步计算收敛条件为相对残差小于0.1或绝对残差小于1e-8。为保证稳定性，N-S方程对流项使用二阶迎风格式，伴随方程及湍流模型有关项使用一阶迎风格式。最终得到 α 的分布情况：

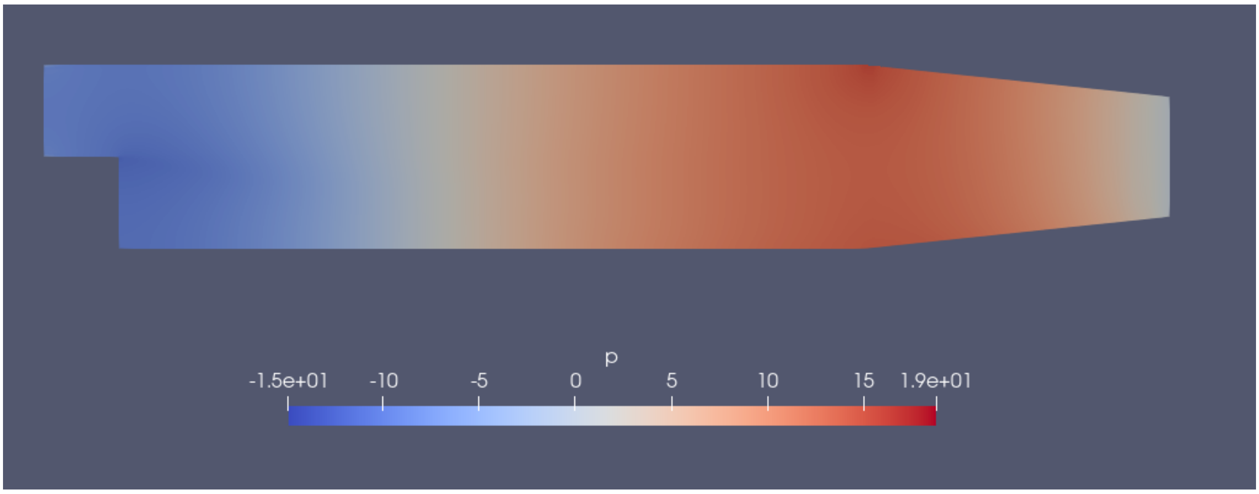


α

可以看到，α 值大的地方也就是被惩罚的区域，将这部分挖掉后流动将更加自然，台阶处的涡消失。

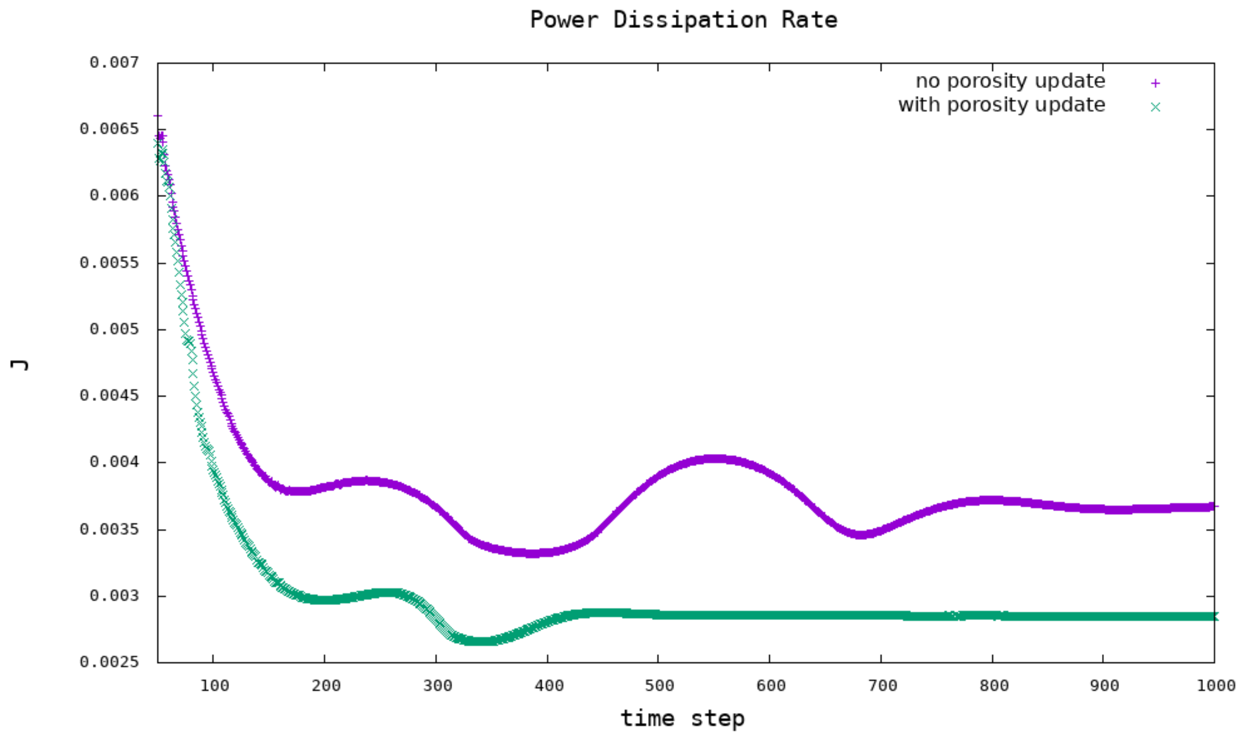


v



p

可以计算出损失函数 J 的变化情况，与不进行伴随优化的解对比，实现了流动能量损失的减小。



Dissipation Rate

5. 总结

伴随优化方法可以扩展到流动、传热、结构耦合问题的求解，也可以与基于梯度的其他优化算法相结合；不仅可以进行形状优化，也可以扩展到其他多参数，目前正得到越来越多的应用。

| 参考资料

- [1] C. Othmer, A continuous adjoint formulation for the computation of topological and surface sensitivities of ducted flows
- [2] Andrew M. Bradley, PDE-constrained optimization and the adjoint method
- [3] Luis Fernando Garcia Rodriguez, Topology Optimisation of Fluids Through the Continuous Adjoint Approach in OpenFOAM
- [4] Ulf Nilsson, Description of adjointShapeOptimizationFoam and how to implement new objective functions

本文于 2020-12-03 更新

阅读原始文档

Adjoint Method

返回 | 主页