方程求解

2-D Euler Equation

将二维欧拉方程求解方程写成如下形式：

$$\frac{\partial}{\partial t}\int_{\Omega}\boldsymbol{W}d\Omega+\int_{\partial\Omega}(\boldsymbol{F}dy-\boldsymbol{G}dx)=\boldsymbol{0}$$

其中各参数意义如下：

$$\boldsymbol{W}=\begin{bmatrix}\rho\\ \rho U\\ \rho V\\ \rho E\end{bmatrix}\quad \boldsymbol{F}=\begin{bmatrix}\rho U\\ \rho U^2+P\\ \rho UV\\ \rho UH\end{bmatrix}\quad \boldsymbol{G}=\begin{bmatrix}\rho V\\ \rho UV\\ \rho V^2+P\\ \rho VH\end{bmatrix}$$

其中有：

$$\begin{cases}E=\dfrac{P}{\gamma-1}+\dfrac{U^2+V^2}{2}\\ H=\rho E+P\end{cases}$$

方程的空间离散化

考虑某一个单元Ω_k，将Ω_k的体积进行缩小，可以认为在该单元上的$\boldsymbol{W}$是均匀分布的，于是：

$$\frac{\partial}{\partial t}\int_{\Omega_k}\boldsymbol{W}d\Omega\approx\Omega_k\frac{\partial\boldsymbol{W}}{\partial t}$$

所以可以将前式写作：

$$\frac{\partial\boldsymbol{W}}{\partial t}=-\frac{1}{\Omega_k}\int_{\partial\Omega_k}(\boldsymbol{F}dy-\boldsymbol{G}dx)$$

现在假设Ω_k是由一个多边形构成的小单元，记一共有N_{edge}^k个边，在其第j号边上，可以计算其边界通量。

假设该边界是由(x_j^1,y_j^1)指向(x_j^2,y_j^2)的一条边，记作$j(1\le j\le N_{\text{edge}}^k)$，假设该顺序是绕着凸多边形$\Omega_k$的逆时针方向顺序，则其外法线方向为：

$$\vec{l}_j=\frac{1}{\sqrt{\Delta x_j^2+\Delta y_j^2}}(-\Delta y_j,\Delta x_j)$$

其中有：

$$\begin{aligned}\Delta x_j&=x_j^2-x_j^1\\\Delta y_j&=y_j^2-y_j^1\end{aligned}$$

则对于该边上的通量，可以近似认为$\boldsymbol{F}$与$\boldsymbol{G}$在小单元边界上保持不变，因此表示如下：

$$\boldsymbol{Q}_k^j=\boldsymbol{F}\Delta y_j-\boldsymbol{G}\Delta x_j$$

我们考虑这个值在单元k和单元p上的计算，上述的$\boldsymbol{F}$和$\boldsymbol{G}$可以近似用j边两侧单元平均值表示。

为了书写方便，令：

$$Z_j=U_j\Delta y_j-V_j\Delta x_j$$

则可以将这第k个单元的流动方程积分表达式写作：

$$\frac{d\boldsymbol{W}_k}{dt}=-\frac{1}{\Omega_k}\sum_{j=1}^{N_{\text{edge}}^k}\begin{bmatrix}Z\rho\\Z\rho U+P\Delta y\\Z\rho V-P\Delta x\\Z\rho H\end{bmatrix}_j$$

因此使用格心格式时，FVM迭代格式已经构造完成，方程计算如下：

$$\frac{d\boldsymbol{W}_k}{dt}=-\frac{1}{\Omega_k}\sum_{j=1}^{N_{\text{edge}}^k}\boldsymbol{Q}_k^j=-\frac{\boldsymbol{Q_k}}{\Omega_k}=\boldsymbol{R}_k$$

方程的时间离散化

理论上，我们在考虑时间步长Δt时，会有：

$$\frac{\boldsymbol{W}_k^{t+\Delta t}-\boldsymbol{W}_k^t}{\Delta t}=\boldsymbol{R}_k$$

但这样做计算精度不高，我们可以考虑四阶龙格库塔时间离散方法：

$$\begin{cases} W^{(0)}=W^t \\ W^{(m)}=W^{(0)}+\alpha_m\Delta t\boldsymbol{R}^{(m-1)} \quad m=1\sim 4 \\ W^{t+\Delta t}=W^{(4)} \end{cases}$$

其中：

$$\begin{cases} \alpha_1=\frac{1}{4} \\ \alpha_2=\frac{1}{3} \\ \alpha_3=\frac{1}{2} \\ \alpha_4=1 \end{cases}$$

人工耗散项

前述式子在计算中不考虑耗散，因此离散误差、舍入误差等会随时间累计。
为了消除这个错误，人工耗散项由此加入，如下式所示：

$$\boldsymbol{R}_k=-\frac{\boldsymbol{Q}_k-\boldsymbol{D}_k}{\Omega_k}$$

由Jameson提出的二姐和四阶混合人工耗散项由下式所示：

$$\boldsymbol{D}_k=\sum_{j=1}^{N_{\mathrm{edge}}^k}\boldsymbol{d}_j^{(2)}+\sum_{j=1}^{N_{\mathrm{edge}}^k}\boldsymbol{d}_j^{(4)}$$

其中：

$$\begin{aligned} \boldsymbol{d}_j^{(2)}&=\alpha_j\varepsilon_j^{(2)}(\boldsymbol{W}_p-\boldsymbol{W}_k)_j \\ \boldsymbol{d}_j^{(4)}&=-\alpha_j\varepsilon_j^{(4)}\left(\nabla^2\boldsymbol{W}_p-\nabla^2\boldsymbol{W}_k\right)_j \end{aligned}$$

其中给定：

$$\nabla^2 \boldsymbol{W}_k = \sum_{j=1}^{N_{\text{edge}}^k} (\boldsymbol{W}_j - \boldsymbol{W}_k)$$

$$\varepsilon_j^{(2)} = k^{(2)} \nu_j$$

$$\varepsilon_j^{(4)} = \max(0, k^{(4)} - \varepsilon_j^{(2)})$$

(Scaling factor of shock sensor)

$$\nu_j = \frac{|P_p - P_k|}{|P_p + P_k|}$$

$$\alpha_j = \left(|U\Delta y - V\Delta x| + c\sqrt{\Delta x^2 + \Delta y^2} \right)_j$$

其中c_j表示当地声速，计算方式为：$c_j = \sqrt{\gamma \frac{P_j}{\rho_j}}$。

另外在论文中，$k^{(2)} \in [0.5, 1.0]$，$k^{(4)} \in [1/256, 1/32]$，这是由经验取值。

时间步长的选取

对于k号单元，采用如下的选取格式：

$$\Delta t_k = CFL \frac{\Omega_k}{\alpha_k}$$

于是在该时间层上的Δt取为$\min \Delta t_k$。

边界条件设置

因为在通量计算中，压力项P的存在，这导致在物面边界上存在通量。

对于远场边界条件，则存在入流与出流的不同，超音速出流的`edge`取值不依赖于下游的量，而亚音速则仍可以用格心格式进行平均。

example code

```
function laplace(grid, W_cur)
% 计算ΔW
end

function [Q, D2, D4, t] = each_step(case, grid, W_cur)
Q = zeros(grid.ncells, 4);
D2 = zeros(grid.ncells, 4);
D4 = zeros(grid.ncells, 4);
t = zeros(grid.ncells);
W_laplace = laplace(grid, W_cur)

% 需要计算粘性项D2、D4以及时间t

t = case.CFL * grid.vol / t;
end

function Q = each_step_non_viscosity(case, grid, W_cur)
% 不需要计算粘性项D2、D4及时间t
end

coeff = [1/4, 1/3, 1/2, 1];
function W_next = rk4(case, grid, W_cur)
Q, D2, D4, t = each_step(case, grid, W_cur);
dt = min(t);
W_next = W_cur - (Q - D2 - D4) / grid.vol * coeff(1) * dt;
for j = 2: 4
    Q = each_step(case, grid, W_next);
    W_next = W_cur - (Q - D2 - D4) / grid.vol * coeff(j) * dt;
end
end
```