

OpenFOAMで二次元円柱周りの流れを見る

参考: http://opencae.gifu-nct.ac.jp/pukiwiki/index.php?plugin=attach&refer=%E5%85%A5%E5%8A%A0%E5%8D%95H250810&openfile=2Dflow_around_cylinder_with_pisoFoam_rev.pdf

今日の目的: 円柱の C_D 値を計算してみる



本日の流れ

0. linuxコマンドの復習
1. 円柱.stlファイルの作成
2. メッシュ作成(blockMesh)
3. メッシュ作成(snappyHexMesh)
4. 境界条件, 計算条件
5. 計算実行(pisoFoam)
6. 計算結果の表示(paraFoam)

linuxコマンドの復習

最低限覚えておかないと太刀打ちできないシリーズ

```
# はコメントアウトです  
mkdir dir_name # ディレクトリを作成する  
ls dir_name # ディレクトリの中身を確認する  
cd dir_name # ディレクトリに移動する  
rm file_name # ファイルを削除する
```

知っておけば格段に作業効率があがるシリーズ

```
find # ファイルを見つけるやつ( コマンド次第で中身も見れる)  
grep # 標準出力等の結果の絞込  
touch # ファイルの作成
```

デフォルトでは入っていないが、あると便利なシリーズ

```
git # バージョン管理  
tmux # 画面分割  
fish, zsh # bash拡張系
```

bash(emacsも)のショートカット系

```
c-a: Home  
c-e: End  
c-k: 今いるカーソルから後ろを削除  
c-u: 今いるカーソルから前を削除  
c-pnfb: 方向キー
```

1. 円柱(.stl)ファイルの作成

やりました ファイルあげます

2. メッシュ作成(blockMesh)

baseとなるtutorialのコピー

```
$ cp -r ~/OpenFOAM/OpenFOAM-2.4.0/tutorials \
/incompressible/pisoFoam/ras/cavity .

$ mv cavity [dirname] # 好きな名前で e.g.) cylinder

# さっきの.stlファイルもここで入れちゃう
$ mkdir constant/triSurface
$ mv cylinder-s.stl constant/triSurface
```

constant/polyMesh/blockMeshDict 内 17行目以降

```
convertToMeters 0.01;

vertices
(
    (-2 -1.5 0)
    ( 4 -1.5 0)
    ( 4  1.5 0)
    (-2  1.5 0)
    (-2 -1.5 0.1)
    ( 4 -1.5 0.1)
    ( 4  1.5 0.1)
    (-2  1.5 0.1)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (60 30 1) simpleGrading (1 1 1)
);
```

blockMeshDict設定

```
boundary
(
    upstream
    {
        type patch;
        faces
        (
            (0 4 7 3)
        );
    }
    downstream
    {
        type patch;
        faces
        (
            (1 2 6 5)
        );
    }
}
```

まだ続く

blockMeshDict設定

```
upANDdown
{
    type patch;
    faces
    (
        (0 1 5 4)
        (3 7 6 2)
    );
}
frontANDback
{
    type empty;
    faces
    (
        (0 3 2 1)
        (4 5 6 7)
    );
}
);
```

blockMeshの実行

0/ とか constant/ とか system/ とかがあるディレクトリに cd

```
$ blockMesh #meshの作成
```

meshの確認をします

```
$ mv ./0 ./0.org # これをしないと表示のときにエラー  
$ paraFoam #結果の確認
```

薄めの直方体が出来るはず

3. メッシュ作成(snappyHexMesh)

1から作るのはめんどくさいので `tutorials` から持ってくる

```
$ cp ~/OpenFOAM/OpenFOAM-2.4.0/tutorials/ \
incompressible/pisoFoam/les/motorBike/ \
motorBike/system/snappyhexMeshDict .
```

```
29 geometry
30 {
31     cylinder-s.stl
32     {
33         type triSurfaceMesh;
34         name cylinder;
35     }
36 };
```

```
103     refinementSurfaces
104     {
105         cylinder
106         {
107             level (0 0);
108         }
109     }
```

```
146     locationInMesh (-0.01 0 0.0005);
```

snappyHexMeshの実行

```
$ snappyHexMesh # この時 0/が存在してはいけない
```

確認

```
$ paraFoam
```

snappyHexMeshのmesh情報をコピー

```
$ cp -r 0.01/polyMesh/* constant/polyMesh/
```

```
$ rm -r 0.005/ 0.01/ # こいつらはもう不要
```

物性値と解析モデル

constant/RASProperties ファイル

```
RASModel          laminar;  
turbulence       off;
```

動粘性係数 (多分変更なし)

constant/transportProperties

```
nu      nu[ 0 2 -1 0 0 0 0 ] 1e-5;
```

controlDict

system/controlDict の設定

```
application           pisoFoam;
startFrom            startTime;
startTime            0;
stopAt               endTime;
endTime              15;
deltaT               0.001;
writeControl         timeStep;
writeInterval        1000;
purgeWrite          0;
writeFormat          ascii;
writePrecision       6;
writeCompression     off;
timeFormat           general;
timePrecision        6;
runTimeModifiable   true;
```

O/の設定

p, U以外は不要なので削除

```
$ rm epsilon k nut nuTilda
```

0.org/U(流速の設定)

```
// 流速: 2.0E-2 [m]
internalField uniform (2.0E-2 0 0);
boundaryField
{
    upstream
    {
        type fixedValue;
        value uniform (2.0E-2 0 0);
    }
    downstream
    {
        type inletOutlet;
        inletValue uniform (2.0E-2 0 0);
        value $internalField;
    }
    upANDdown
    {
        type inletOutlet;
        inletValue uniform (2.0E-2 0 0);
        value $internalField;
    }
}
```

```
frontANDback
{
    type          empty;
}
cylinder
{
    type          fixedValue;
    value         uniform (0 0 0);
}
```

0.org/p

```
internalField    uniform 0;

boundaryField
{
    upstream
    {
        type          fixedValue;
        value         0;
    }
    downstream
    {
        type          fixedValue;
        value         0;
    }
    upANDdown
    {
        type          fixedValue;
        value         0;
    }
}
```

```
frontANDback
{
    type          empty;
}
cylinder
{
    type          zeroGradient;
}
```

実行

```
$ cp -r ./0.org ./0  
$ pisoFoam > piso.log
```

```
$ paraFoam
```

CD(Constant Drag, 空気抵抗力係数)値 とは

- 簡単に言うと, その形が受ける無次元化された力
- 無次元化されているので, 面積や流体の密度などは関係ない
- 空気抵抗 D は以下の式で求めることができる

$$D = C_D \times \rho \times U^2 \times S/2$$

ここで,

C_D : 抗力係数

ρ : 流体の密度

U : 速度

S : 前面投影面積

OpenFOAMでCD値を求める方法

- すっごく簡単

```
$ cp ~/OpenFOAM/OpenFOAM-2.4.0/tutorials/ \
incompressible/pisoFoam/les/ \
motorBike/motorBike/system/forceCoeffs system/
```

```
forces
{
    type          forceCoeffs;
    functionObjectLibs ( "libforces.so" );
    outputControl timeStep;
    outputInterval 1;

    patches      ( "cylinder" );
    pName        p;
    UName        U;
    rhoName      rhoInf;    // Indicates incompressible
    log          true;
    rhoInf       1000;     // Redundant for incompressible
    liftDir      (0 1 0);
    dragDir      (1 0 0);
    CofR         (0 0 0);  // Axle midpoint on ground
    pitchAxis    (0 1 0);
    magUInf     0.02;
    lRef         0.01;      // Wheelbase length
    Aref         1.0E-5;    // Estimated
}
```

system/controlDictの追加

```
functions
{
    #include "forceCoeffs"
}
```

実行

```
$ pisoFoam > piso.log
```

- どうせ一緒なのでparaFoamはいらないです

結果の確認

`postProcessing/forces/0/forceCoeffs.dat` を見てみよう

時間変化と C_D 値の値を適当にplotしてみよう
(matplotlib, gnuplot, etc.)

- C_D 値の値は安定? 定常? 一定?
- どの値を取るのが適切か → 自分で考える

レイノルズ数 Re

レイノルズ数 Re は、以下の式で求めることができる 今回は？

$$Re = \frac{\rho U L}{\mu} = \frac{U L}{\nu} [-]$$

U (代表速度) $= 2 \times 10^{-2} \text{ m/s}$
 L (代表長さ)(今回は円柱の直径) $= 0.01 \text{ m}$
 μ (粘性係数) $[kg/(m s)]$ ν (動粘性係数) $= 1 \times 10^{-5} \text{ m}^2/\text{s}$
 ρ (流体の密度) $= 1000 \text{ kg/m}^3$

ref) <http://www.cybernet.co.jp/ansys/case/lesson/004.html>

ref) <https://ja.wikipedia.org/wiki/レイノルズ数>

(visited 2016-10-19)

Re の利点

無次元化されているので, 大きさに関係せず, 相対的な速度で考えることができる.

たとえば, 円柱周りの流れでも, 大きくしたり, 直径を変えたり, 粘性係数(たとえば空気に入ったり)を変えても、

Re が同じならば似たような流れになる.

もちろん, 代表長さ L のとり方によって Re が変わってくるので, 慎重に選ぶ必要がある.

クーラン数(Courant Number)

クーラン数 C は、以下の式で求めることができる。

1ステップあたりの時間が経過したときに、流れの要素いくつ分進むか

$$C = \frac{u\Delta t}{\Delta L}$$

u :流速

Δt :時間間隔

ΔL :要素幅

- クーラン数は、OpenFOAMが勝手に求めてくれる
- `pisoFoam` ではクーラン数が大きくなり過ぎないように Δt を調整してくれる

ref) <http://www.cradle.co.jp/tec/column01/017.html>

(visited: 2016-10-19)

クーラン数を見てみよう

- pisoFoam 実行時の標準出力に出てくる.

```
cat piso.log | grep -e "Time =" -e Courant | \
grep -v Execution
```

- クーラン数は1未満でなければならない。(格子を飛び越していってしまうため)

クーラン数を大きくするということ

$$C = \frac{U\Delta t}{\Delta L} < 1$$

- 細かく計算したいとき...
meshを細かくする \Leftrightarrow ΔL を小さくする $\Leftrightarrow C$ が大きくなる
- より流速の早い流れ計算したい時...
 U が大きくなる $\Leftrightarrow C$ が大きくなる

クーラン数が大きくなると

Δt を小さくする必要がある ... 計算時間がかかる

meshを細かくしたら計算数も多くなり, より計算時間がかかる

残った時間で

Re と C_D 値の関係を見てみよう

自分で Re と C_D 値のグラフを探して比較する

*Re*を変えるにはどうするか?

$$Re = \frac{U \cdot L}{\nu}$$

- (*L*の考え方: surfaceTransformPoints) ← やったことない
- *U*: 0/を見る
- *ν*: constant/transportProperties の nu

meshの工夫

- blockMeshDict
- snappyHexMesh