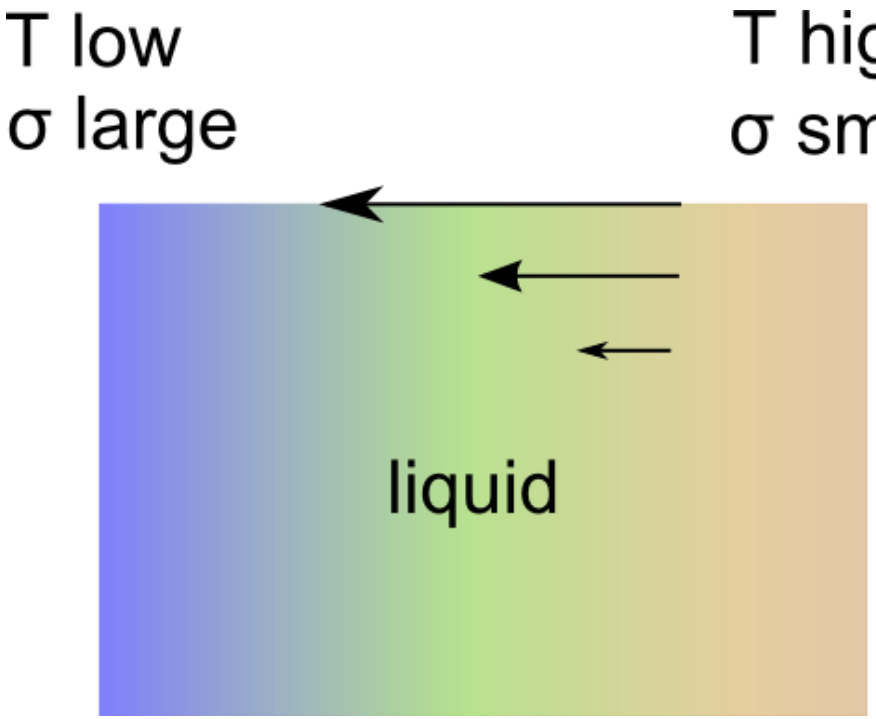
OpenFOAMによる thermocapillary 効果の検証

大阪大学基礎工学部
岡野研　研究生
山本　卓也

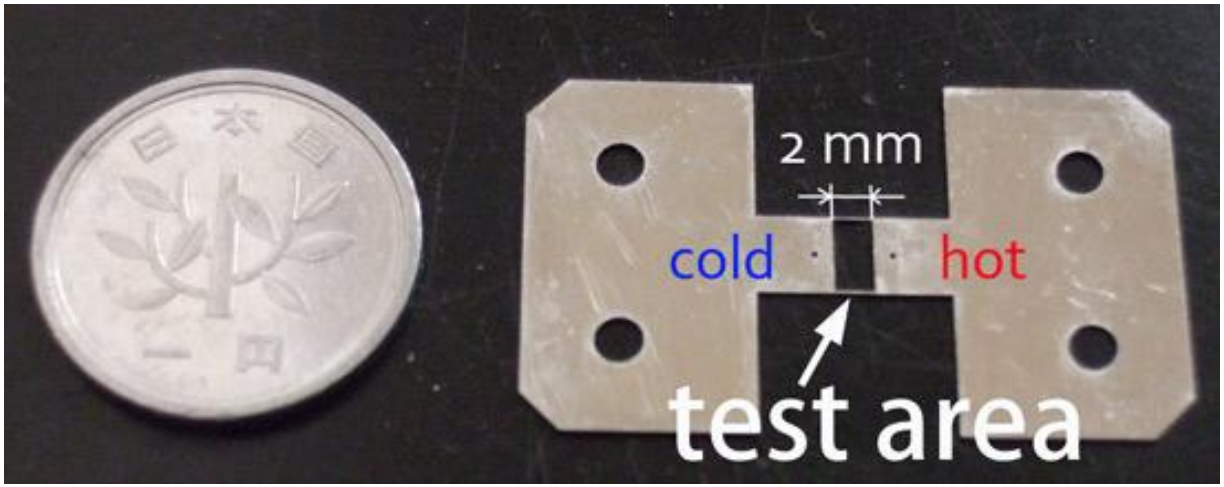
Thermocapillary効果とは

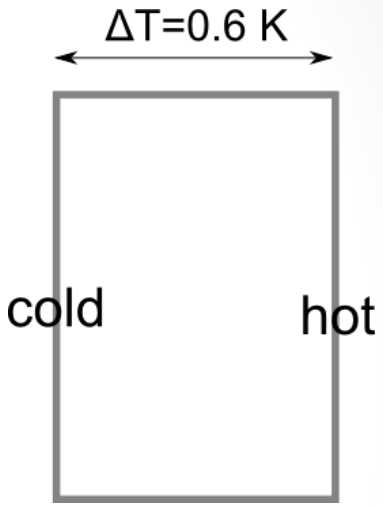


自由表面上で生じる
表面張力差による流れ

温度が高い ⟶ 表面張力が小さい
温度が低い ⟶ 表面張力が大きい

現在行われている研究



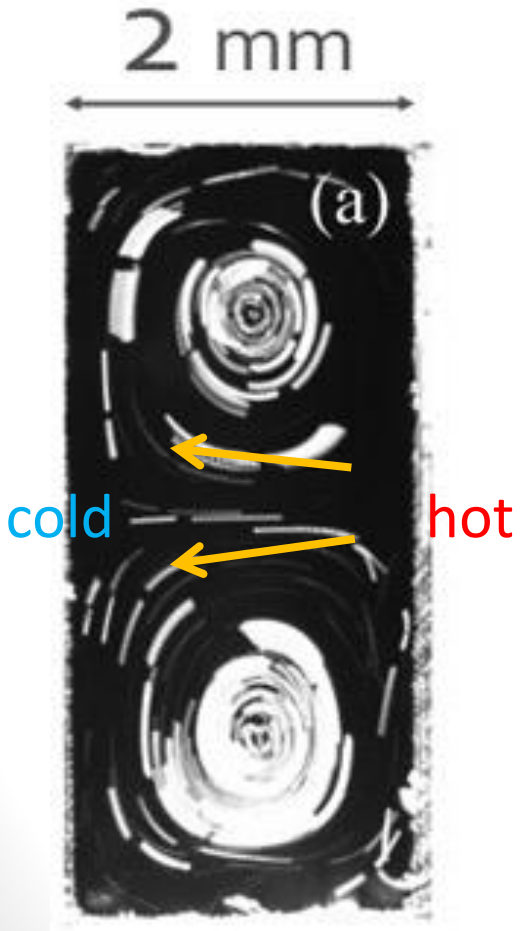


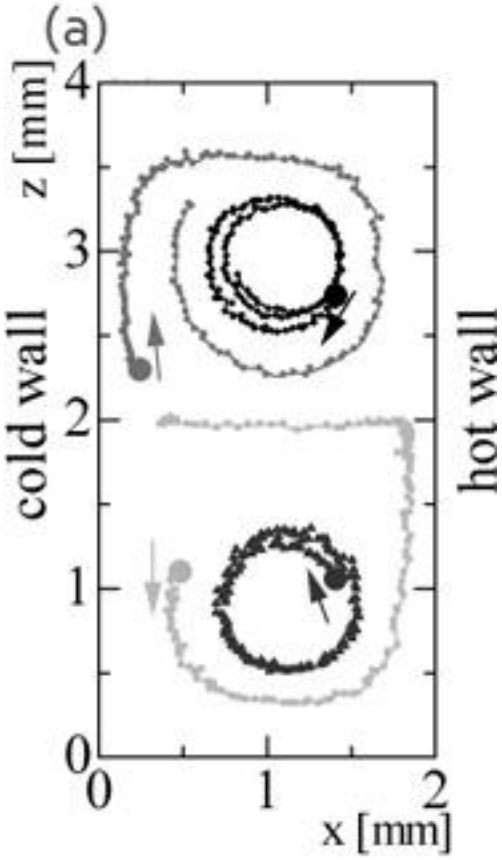
I. Ueno et al., Acta Astronautica, 66, pp.1017-1021 (2010)

溶液
シリコンオイル
ρ $9.12 \times 10^{2} \mathrm{kg/m}$
ν $5.0 \times 10^{-6} \mathrm{m^2/s}$
σ_T $6.37 \times 10^{-5} \mathrm{N/m \cdot K}$

研究のターゲット
どのような流れが生じるか

実験結果 (I.Ueno *et al.*(2010))





2次元的な流れが生じた

OpenFOAMで再現してみる

数値計算方法

Solver :: exforthmarangoniboussinesq (1.6-ext版boussinesqBoyantFoamの改造)

元々のboussinesqBoyantFoamは非定常のブジネスク近似を含む
熱と運動量輸送方程式を解く

支配方程式

$$\rho \frac{Du}{Dt} = -\nabla p + \mu \nabla^2 u - \rho_0 g_i \beta (T - T_0)$$

$$\nabla \cdot u = 0$$

$$\frac{DT}{Dt} = \alpha \nabla^2 T$$

外力項に重力項
（ブジネスク近似の項）
をN-S式に加えている

ブジネスク近似とは
熱輸送を伴う流れでは流体は密度などの物性が変わる。
密度変化が小さい場合は式中の密度は一定であるとみなし、
重力項でのみ密度変化を考慮する近似のこと。

数値計算方法

exforcemarangoniboussinesq

支配方程式（２次元）

$$\rho \frac{Du}{Dt} = -\nabla p + \mu \nabla^2 u - \frac{2\sigma_T}{\delta_z} \nabla T$$

$$\nabla \cdot u = 0$$

$$\frac{DT}{Dt} = \alpha \nabla^2 T$$

N-S式の外力項に
thermocapillaryの項を加える

$\sigma_T = \frac{\partial \sigma}{\partial T}$(定数)
σ :: 表面張力
δ_z :: 厚み

ソルバーに外力項を一つ加えるだけ

boussinesqBoyantFoam

solverの中身

boussinesq BoyantFoam	Make
	boussinesqBoyantFoam.C
	boussinesqBoyantFoam.dep
	createFields.H
	readTransportProperties.H

Make　::　コンパイル用フォルダ
.Cファイル　::　ソースコードの内容
.depファイル　::　依存関係のファイル（コンパイル後に生成されるもの）
.H　::　ヘッダファイル（.Cファイルに読み込まれる）

今回関係あるのは.Cファイルとと readTransportProprties.Hのみ
他に詳しく知りたければOpenCAE学会HPに
（http://www.opencae.jp/wiki）

boussinesqBoyantFoam

一部抜粋

```
Info<< "¥nStarting time loop¥n" << endl;

    for (runTime++; !runTime.end(); runTime++)
    {
        Info<< "Time = " << runTime.timeName()
<< nl << endl;

#       include "readPISOControls.H"
#       include "CourantNo.H"
```

この部分がN-S式だと分かる

```
        fvVectorMatrix UEqn
        (
            fvm::ddt(U)
          + fvm::div(phi, U)
          - fvm::laplacian(nu, U)
         ==
           -beta*(T - T0)*g
        );

        solve(UEqn == -fvc::grad(p));
```

marangoniboussinesqBoyantFoam

一部抜粋

```
fvVectorMatrix UEqn
    (
      fvm::ddt(U)
     + fvm::div(phi, U)
     - fvm::laplacian(nu, U)
     ==
      -beta*(T - T0)*g
    );

    solve
    (
     UEqn == -fvc::grad(p) + fvc::grad(T)*marangCoeff
    );
```

fvc::陽的に微分を計算
fvm::陰的に微分を計算

ここでmarangCoeffは
$-\frac{2\sigma_T}{\delta_z}$のこと

この項を加えてやる
marangCoeffは
readTransportProperties.H中で指定

readTransportProperties.H

一部抜粋

定数の指定

```
dimensionedScalar T0
(
    transportProperties.lookup("T0")
);

dimensionedScalar nu
(
    dimensionedScalar(transportProperties.lookup("mu"))/rho0
);
```

同様にしてmarangCoeffも作成

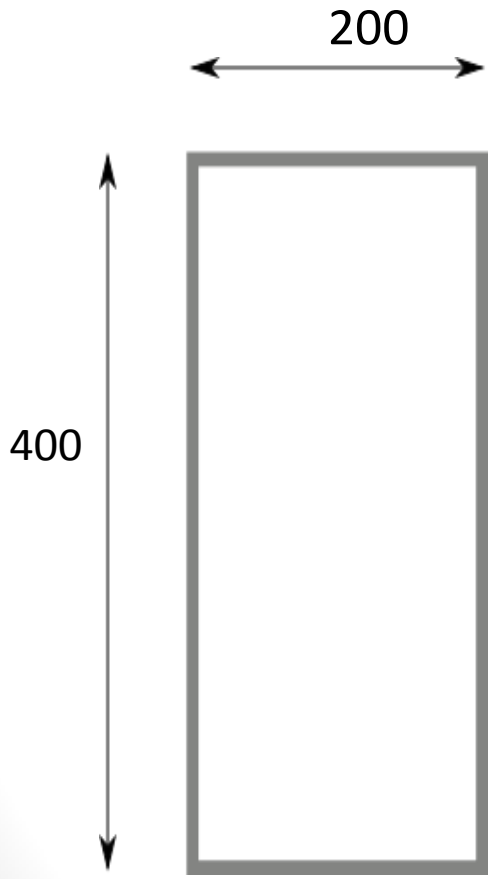
```
dimensionedScalar marangCoeff
    (

transportProperties.lookup("marangCoeff")
    );
```

コードのコンパイル等

今回はコンパイル、ケースの作成等のことは省略します。
先程のHPに今野先生の資料があります。
メインに書き換えることはこれぐらいです。

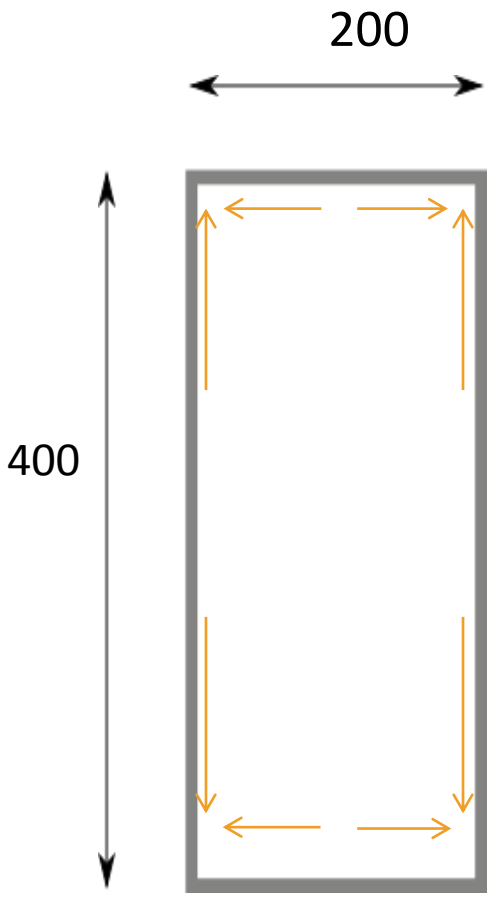
計算用格子

Grid 1　等間隔格子



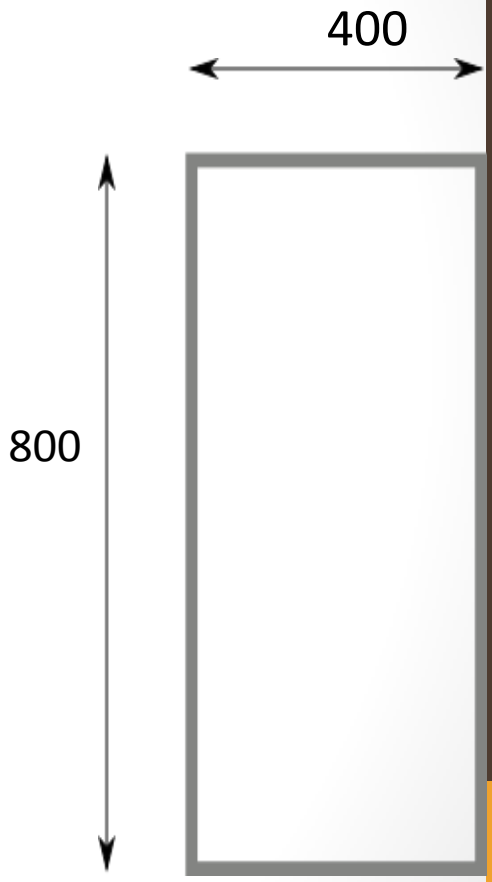
each direction
$\Delta x_{min}=1.0\times10^{-4}$

Grid 2　不等間隔格子



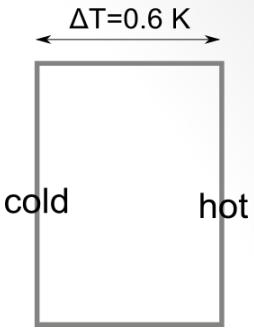
each direction
$\Delta x_{min}=5.0\times10^{-5}$

Grid 3　等間隔格子

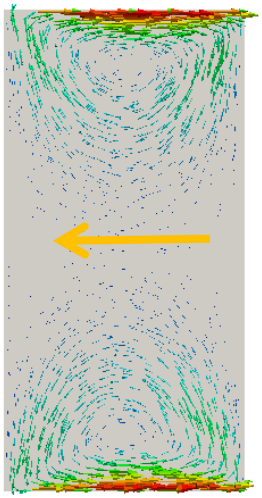


each direction
$\Delta x_{min}=5.0\times10^{-5}$

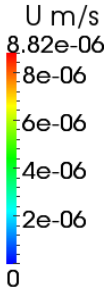
計算結果



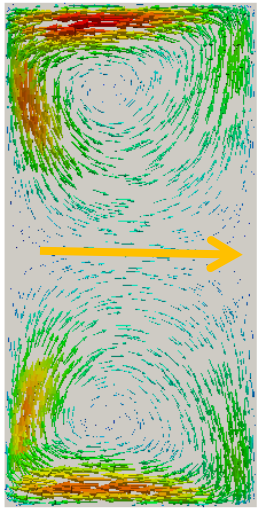
Grid 1



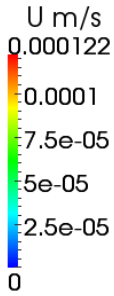
hot



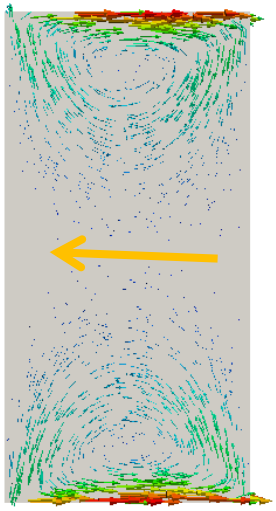
Grid 2



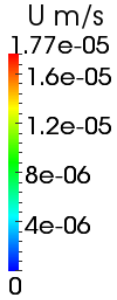
hot



Grid 3



hot



格子に依存して結果の流れが反転
速度のオーダーも全然違う

考察

- 流れが反転してしまったのは陽的に解いたため計算精度が低いためか？？
- 熱と流体の練成は？？
- ソースコードの改造方法が間違えている？？

何かアドバイスがいただけるのであれば
ご教授の方よろしくお願いします。