
OpenFOAM勉強会 for beginner @関西の紹介

OpenFOAM勉強会 for beginner@関西
幹事 富原 大介

OpenFOAM勉強会 for beginner@関西

昨年の12月から、関西におけるOpenFOAM初心者ターゲットとした勉強会を開催しています。

ほぼ月1回、大阪大学の高木先生をアドバイザーにお招きして、大阪大学や大阪市内の会議室で開催。
次回は1月を予定（詳細は未定）。

超初心者から上級者、学生から社会人まで幅広い方々が参加。
依然として少人数ではありますが、
どんなことでも質問できる雰囲気の特徴です。

OpenFOAMってどうなんだ？
使ってみようとしたけれどよくわからない。
OpenFOAMの情報、知識を発信したい！
(受信もできるかも)

いろんな方のご参加をお待ちしています。

勉強会の日程、参加のお知らせは
Googleグループのフォーラムにて随時お知らせしています。

勉強会

<https://groups.google.com/forum/#!forum/openfoambeginner>

またはユーザー会

<https://groups.google.com/forum/#!forum/openfoam>

Q. 普段どのような感じでやっているのか？

- A. 初めての方の自己紹介、発表される方の発表とそれらに対する質疑応答といった内容です。
時間が余っていることが多いので、最後にみなさんから自由な質問の時間を設けています。

発表の内容は学生の研究発表から幹事の実験まで様々。

勉強会@関西の様子をイメージしていただくために
今日は次ような発表を用意いたしました

初心者によるOpenFOAM有限体積法入門

OpenFOAM勉強会@関西
幹事 富原 大介

当然ながら、
OpenFOAMの計算には「有限体積法」が用いられている。

……らしい。

「有限体積法」って聞いたことはあるけれど、
学生時代にプログラムを組んだことがあるわけでもないし…

「オープンソース」→「中身が見れる」とは言いながら見たことない。

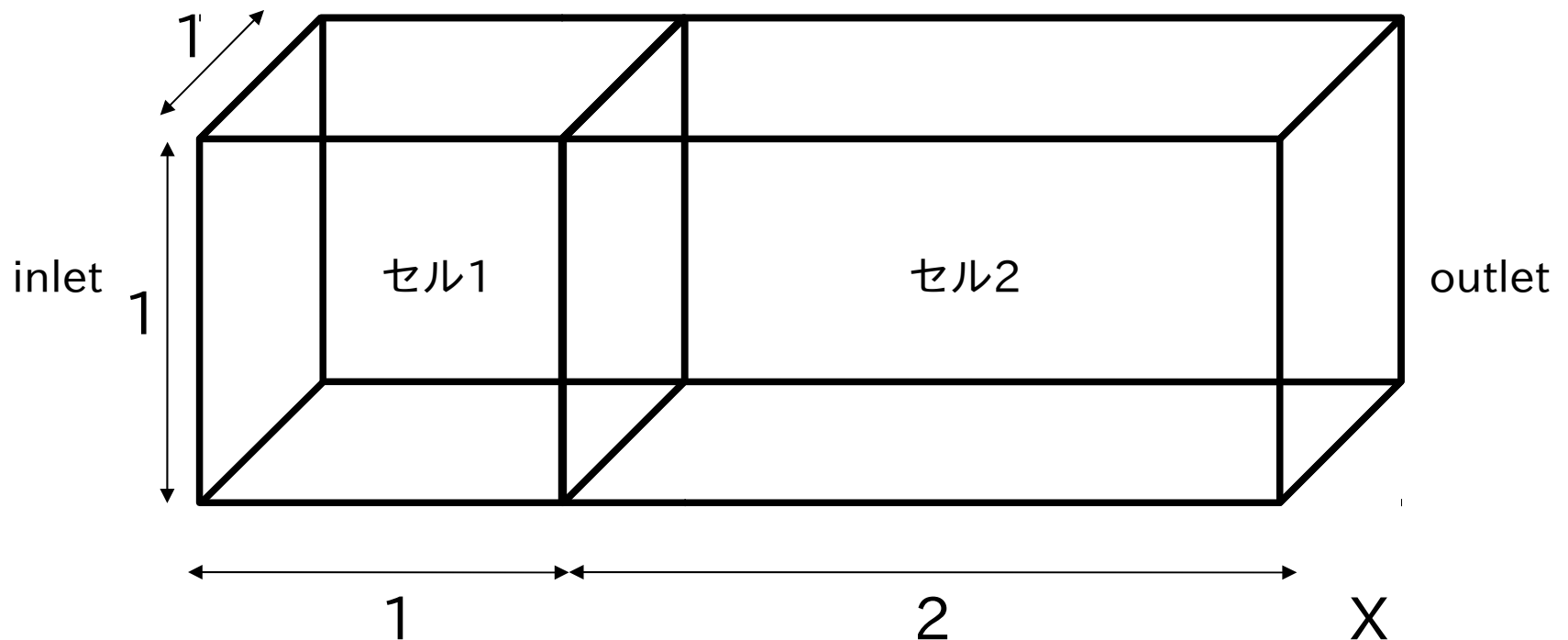
と、ずっと思っていたので、
実際に見てみました。

まず、簡単そうなソルバを探してきて、
中身を変更していきます。

選んだのは「scalarTransportFoam」
/opt/openfoam171/applications/solvers/scalarTransportFoam
をフォルダごと適当なディレクトリにコピー。

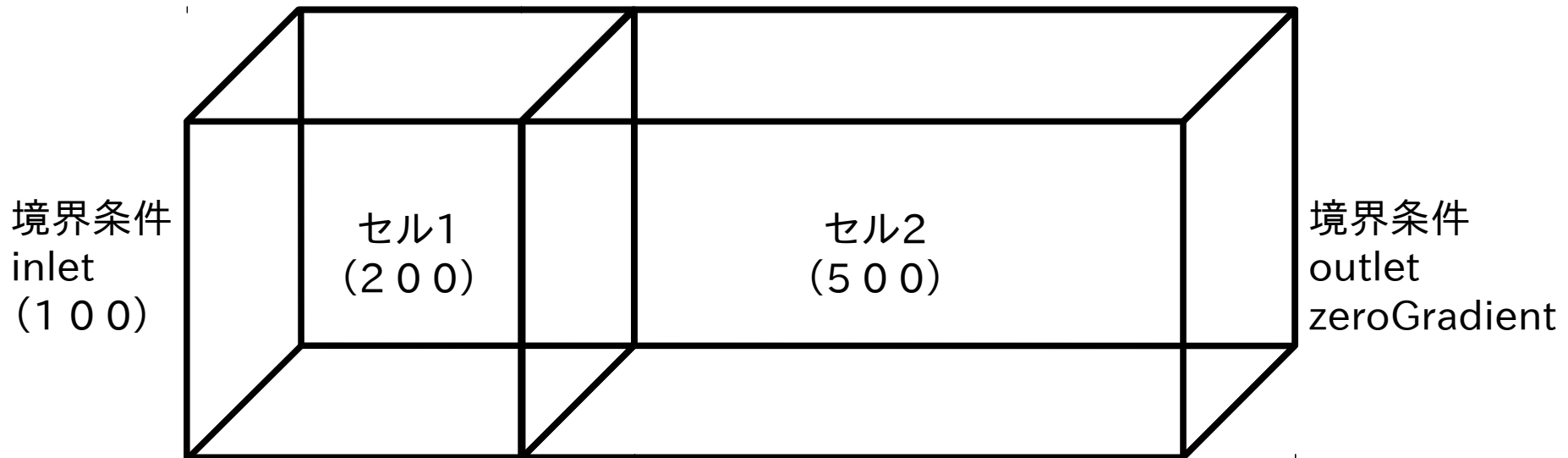
scalarTransportFoam.C→mathFoam.Cとして作業していきます。
(makeの方法等の説明は割愛)

モデルを用意。
X方向についてのみ考えるために、
「inlet」および「outlet」以外はデフォルトでemptyとしてしまう。



0/Uの設定

※あくまで数学的に見るのが目的なので
物理的な意味は正しくありません



- ①まずはモデルの情報を出力してみる。
math.Cの中身を以下のようにしてmake→実行してみる。

```
(...)  
int main(int argc, char *argv[])  
{  
(...)  
# include "CourantNo.H"  
(ここまではscalarTransportFoam.Cそのまま)  
  
Info<< endl;  
Info<< "-----" << nl << endl;  
Info<< "Cell volume V() = " << mesh.V() << nl << endl;  
Info<< "-----"  
  
Info<< "End\n" << endl;  
return 0;  
}
```

セルの体積のオブジェクト
mesh.V()

- ①まずはモデルの情報を出力してみる。
math.Cの中身を以下のようにしてmake→実行してみる。

(...)

Cell volume V() = dimensions [0 3 0 0 0 0 0];

value nonuniform List<scalar> 2(1 2);

難しいコードは何ひとつ書いていないにもかかわらず、
きちんとセルの体積が出力される。

面の面積ベクトルSf <mesh.Sf()>

Face area vectors Sf() = dimensions [0 2 0 0 0 0 0];

```
internalField uniform (1 0 0);
```

```
boundaryField
```

```
{  
  inlet  
  {  
    type      sliced;  
    value     uniform (-1 0 0);  
  }  
  outlet  
  {  
    type      sliced;  
    value     uniform (1 0 0);  
  }  
  defaultFaces  
  {  
    type      sliced;  
    value     nonuniform 0();  
  }  
}
```

面の面積の絶対値|Sf| <mesh.magSf()>

Face area magnitudes |Sf|() = dimensions [0 2 0 0 0 0 0];

```
internalField uniform 1;
```

```
boundaryField
```

```
{  
  inlet  
  {  
    type      calculated;  
    value     uniform 1;  
  }  
  outlet  
  {  
    type      calculated;  
    value     uniform 1;  
  }  
  defaultFaces  
  {  
    type      empty;  
  }  
}
```

セルの中心C <mesh.C()>

Cell centers C() = dimensions [0 1 0 0 0 0 0];

```
internalField nonuniform List<vector> 2((0.5 0.5 0.5) (2 0.5 0.5));
```

```
boundaryField
```

```
{
  inlet
  {
    type    sliced;
    value   uniform (0 0.5 0.5);
  }
  outlet
  {
    type    sliced;
    value   uniform (3 0.5 0.5);
  }
  defaultFaces
  {
    type    sliced;
    value   nonuniform 0();
  }
}
```

面中心Cf <mesh.Cf()>

Face centers Cf() = dimensions [0 1 0 0 0 0 0];

internalField uniform (1 0.5 0.5);

boundaryField

```
{
  inlet
  {
    type    sliced;
    value   uniform (0 0.5 0.5);
  }
  outlet
  {
    type    sliced;
    value   uniform (3 0.5 0.5);
  }
  defaultFaces
  {
    type    sliced;
    value   nonuniform 0();
  }
}
```

ちなみに、
面に関しては「face.C」内の
Foam::face::centreや
Foam::face::normal、
セルに関しては「cellModel.C」内の
Foam::cellModel::centreや
Foam::cellModel::mag等によって
中心や面積を求めている(と思われる)。
数学的にとことん追いかけてたい人は
そちらを参照してください。
(20~30行程度のコード。)

②少しずつ計算してみる。

scalarTransportFoam.Cの計算していそうな部分のコード

```
(...)  
    for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)  
    {  
        solve  
        (  
            fvm::ddt(T)  
            + fvm::div(phi, T)  
            - fvm::laplacian(DT, T)  
        );  
    }  
(...)
```

いきなりこれを見ても、正直よくわからない
→プログラマーズガイドをよく見てみる

②少しずつ計算してみる。

プログラマーズガイドによると、

`fvm::ddt(T)` → 陰的な $\partial T / \partial t$

`fvm::div(phi,T)` → 陰的な $\nabla \cdot (\phi T)$

…らしい。

「`fvm::`」を「`fv::`」にすることで陽的な値を求めることが出来る。
とあるが、違いがよくわからないので一度出力してみる。

② 少しずつ計算してみる。

math.Cの中身を以下のようにしてmake→実行してみる。

```
(...)  
int main(int argc, char *argv[])  
{  
  (...)  
  # include "CourantNo.H"  
  (ここまではscalarTransportFoam.Cそのまま)  
  
  Info<< endl;  
  Info<< "-----" << nl << endl;  
  Info<< "fvm::div(U) = " << fvm::div(U) << nl << endl;  
  Info<< "fvc::div(U) = " << fvc::div(U) << nl << endl;  
  Info<< "-----"  
  
  Info<< "End\n" << endl;  
  return 0;  
}
```

②少しずつ計算してみる。

math.Cの中身を以下のようにしてmake→実行してみる。

```
mathFoam.C:79: error: no matching function for call to  
'div(Foam::volVectorField&)'
```

fvm::div(U)ではエラーとなってmake出来ないことが判明。

fvc::div(U)のみにして再make実行すると成功する。

(温度Tだとfvm::(T)としてもmake可。ベクトルだとダメということらしい)



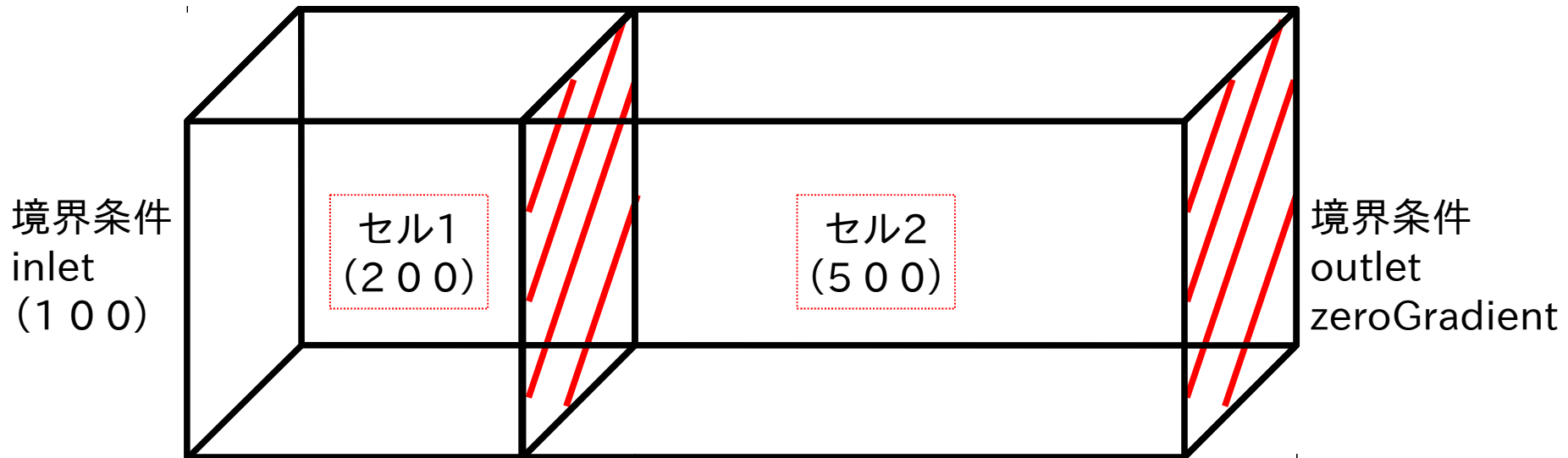
makeは成功。

しかし、実行すると「スキームが設定されていない」と
エラーメッセージが出力される。

いよいよスキームの話が登場することになる。

有限体積法による発散の求め方(教科書より)

- ①もともと値はセル(の中心)で持っている
- ②セルの中心の値を用いて面の中心の値を求める→値の内挿
- ③面の中心の値から面全体の値を求める
- ④面の値をすべて足し合わせ(体積で割ったもの)が発散



②少しずつ計算してみる。

math.Cの中身を以下のようにしてmake→実行してみる。

<面への内挿>

fvC::interpolate(U)で確認することが出来る。

math.CにfvC::interpolate(U)を出力するように設定し、makeする。
/system/fvSchemesファイルにスキームを指定して実行。

```
interpolationSchemes
```

```
{
```

```
  default    none;
```

```
  interpolate(U) linear; (線形の場合)
```

```
  //interpolate(U) upwind differenceFactors_; (風上の場合)
```

```
}
```

Uの内挿 <fv::interpolate(U)>

```
interpolate(U) = dimensions [0 1 -1 0 0 0 0];
```

```
internalField uniform (3 0 0);
```

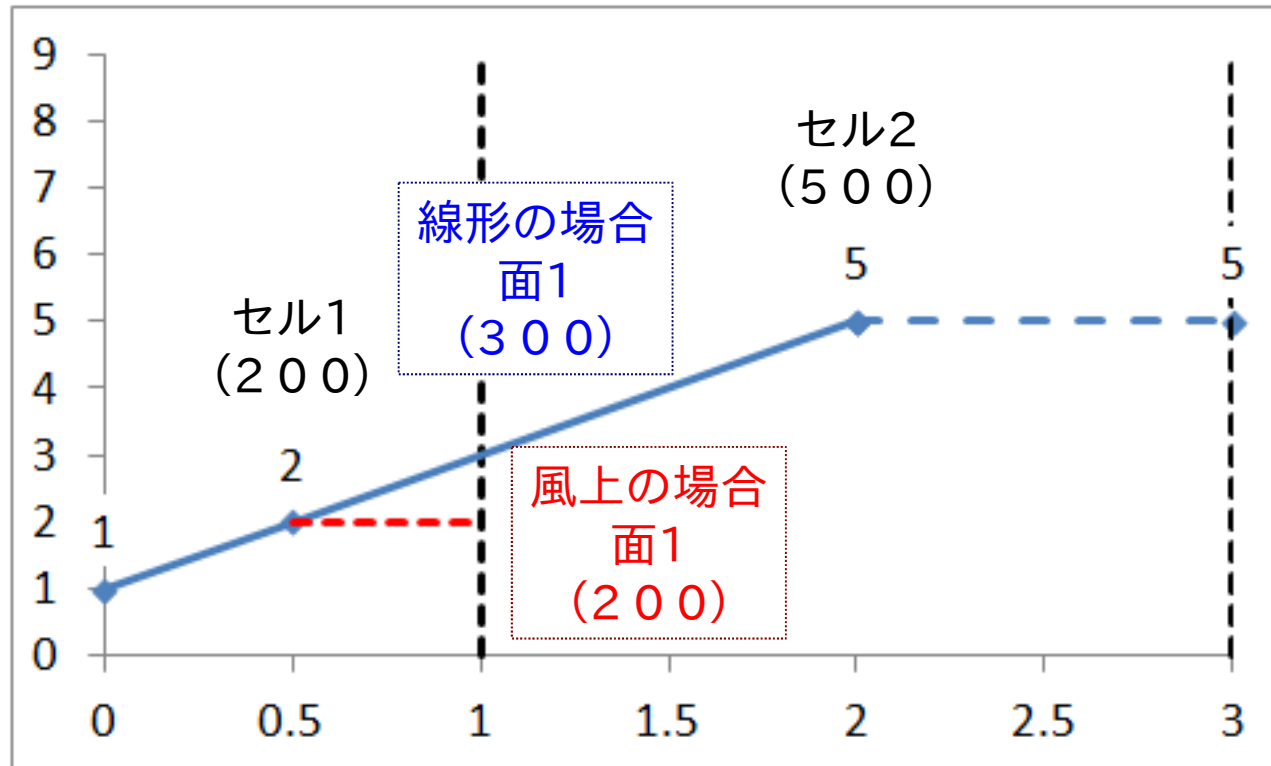
```
boundaryField
{
  inlet
  {
    type    calculated;
    value   uniform (1 0 0);
  }
  outlet
  {
    type    calculated;
    value   uniform (5 0 0);
  }
  defaultFaces
  {
    type    empty;
  }
}
```

```
interpolate(U) = dimensions [0 1 -1 0 0 0 0];
```

```
internalField uniform (1 0 0);
```

```
boundaryField
{
  inlet
  {
    type    calculated;
    value   uniform (1 0 0);
  }
  outlet
  {
    type    calculated;
    value   uniform (5 0 0);
  }
  defaultFaces
  {
    type    empty;
  }
}
```

0/Uの内挿 スキームによる違い



境界条件
inlet
(1 0 0)

境界条件
outlet
zeroGradient

②少しずつ計算してみる。

math.Cの中身を以下のようにしてmake→実行してみる。

＜面への内挿から面全体の値＞

`fv::interpolate(U)`と面の面積ベクトル`mesh.Sf()`の内積。

→`fv::interpolate(U)&mesh.Sf()`

＜面全体の値を足し合わせてセルの体積で割る＞

`fv::surfaceIntegrate()`という関数が用意されている。

→`fv::surfaceIntegrate(interpolate(U)&mesh.Sf())`

→これが、`fv::div(U)`と等しくなる。

スキームの設定を合わせて確かめてみる

Uの発散 <fvc::div(U)> ※fvSchemesの設定

```
divSchemes
{
    default      none;
    // div(phi,U)  Gauss linear;(線形)
    div(phi,U)   Gauss upwind differenceFactors_;(風上)
    // div(U)      Gauss linear;(線形)
    div(U)       Gauss upwind differenceFactors_;(風上)
}
```

```
interpolationSchemes
{
    default      none;
    // interpolate(U) linear;(線形)
    interpolate(U) upwind differenceFactors_;(風上)
}
```

Uの発散 <fvc::div(U)> ※風上スキームを指定した例

```
fvc::div(U) =  
dimensions [0 0 -1 0 0 0 0];
```

```
internalField nonuniform List<scalar> 2(1 1.5);
```

```
boundaryField  
{  
  inlet  
  {  
    type zeroGradient;  
  }  
  outlet  
  {  
    type zeroGradient;  
  }  
  defaultFaces  
  {  
    type empty;  
  }  
}
```

```
fvc::surfaceIntegrate(interpolate(U)&Sf) =  
dimensions [0 0 -1 0 0 0 0];
```

```
internalField nonuniform List<scalar> 2(1 1.5);
```

```
boundaryField  
{  
  inlet  
  {  
    type zeroGradient;  
  }  
  outlet  
  {  
    type zeroGradient;  
  }  
  defaultFaces  
  {  
    type empty;  
  }  
}
```

③方程式を考えてみる

対流項だけの運動方程式(?)を考える。

$$\partial U / \partial t = \int_V \nabla \cdot (\phi U) dV$$

$$\rightarrow \text{fvm}::\text{ddt}(U) = \text{fvc}::\text{div}(\text{phi}, U)$$

$\text{fvc}::\text{ddt}(U) = \text{fvc}::\text{div}(\text{phi}, U)$ とは出来ない。

fvcとは値がすでにはっきりしているもの？

→だからfvc=fvcは方程式ではない。

fvmとはddtで言えば $U(t + \Delta t)$ のように不明な変数を含む？

③方程式を考えてみる

流速phiの設定

ソルバによってphiの求め方が色々あるようだが、
ここではスキームの確認のため、自分でphiを設定してしまう。
(scalarTransportFoamではUの内挿に線形スキームを
使用してphiを求める設定になっている)

$\text{phi} = (\text{面に内挿したUの値}) \cdot (\text{面の面積ベクトル})$
コードにphiの生成を直接追加する。
→`phi = fvc::interpolate(U)&mesh.Sf();`

③方程式を考えてみる

$\text{div}(\text{phi}, \text{U})$ の確認

$\text{fvc}::\text{div}(\text{phi}, \text{U})$ は今までの div の考察から、

→ $\text{fvc}::\text{surfaceIntegrate}(\text{phi} * \text{fvc}::\text{interpolate}(\text{U}))$

となる。

③方程式を考えてみる

方程式のコード

```
solve (fvm::ddt(U) == fvc::div(phi,U))
```

とすることで、計算時間が Δt だけ進む。
Uを出力させると Δt だけ進んだ値になっている。

それとは別に、「runTime」というオブジェクトが時刻を持つ。

<時刻関係の出力>

runTime.timeName() : runTimeの時刻

runTime.deltaT().value : deltaT

runTime.write() : 結果ファイルを出力する

③方程式を考えてみる

以下のコードで対流項のみの方程式の計算が実行できる。

```
(...)  
phi = fvc::interpolate(U)&mesh.Sf();  
for(int n=1; n<=5; n++){  
    solve(fvm::ddt(U) == fvc::div(phi,U)); runTime++;  
    phi = fvc::interpolate(U)&mesh.Sf();  
    runTime.write()  
}  
Info<< "End\n" << endl;  
return 0;  
}
```

③方程式を考えてみる

以下のコードでも全く同じ計算が出来る

```
(...)  
for(int n=1; n<=5; n++){  
    solve(fvm::ddt(U) ==  
fvc::surfaceIntegrate((fvc::interpolate(U)&mesh.Sf())*fvc::interpolate(U)));  
    runTime++;  
    runTime.write()  
}  
Info<< "End\n" << endl;  
return 0;  
}
```

捕捉

今回は陽解法 (`solve (fvm==fvc)`) を用いているので、線形ソルバは用いていない。
(`fvSolution`ファイルは必要だが、中身は使用されていない。)

陰解法で解く場合は以下のようにする。

→ `solve (fvm::ddt(U) - fvm::div(phi,U))`

「==」ではなく、「-」になっている。

陰解法を使用する場合は別途`fvSolution`の設定が必要となる

以上、ご清聴ありがとうございました。

いろんな方の勉強会へのご参加をお待ちしています。

勉強会の日程、参加のお知らせは
Googleグループのフォーラムにて随時お知らせしています。

勉強会

<https://groups.google.com/forum/#!forum/openfoambeginner>

またはユーザー会

<https://groups.google.com/forum/#!forum/openfoam>