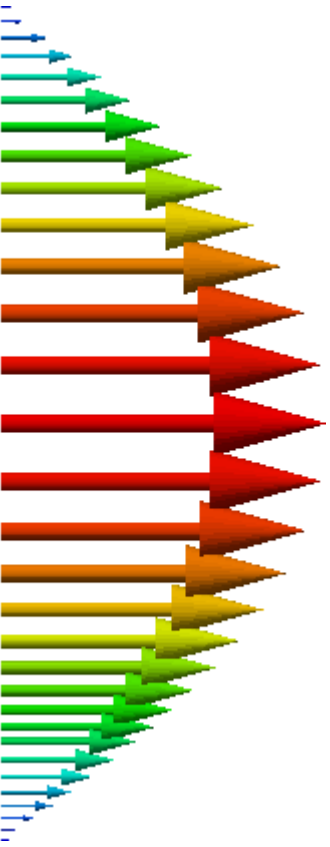


OpenFOAM の境界条件クラスの探索とカスタマイズ

Keywords：

- **mixed**
- **directionMixed**
- **Topology optimization**
- **adjointShapeOptimizationFoam**

Fumiya Nozaki

Last Updated: 6 March 2016

Disclaimer

"This offering is not approved or endorsed by OpenCFD Limited, the producer of the OpenFOAM software and owner of the OPENFOAM® and OpenCFD® trade marks."

資料の表記

- この資料では，下記のように色分け表記しています．

実行コマンド (Execution Command)

実行結果 (Execution Result)

ソースコードとファイル (Source Code)

補足説明 (Supplemental Explanation)

予備知識｜ディレクトリの移動コマンド

- OpenFOAM の代表的なディレクトリへの移動コマンドが登録されています.

etc/config/aliases.sh

```
# Change directory aliases
# ~~~~~~~~~~~~~~~~~~~~~~~
alias src='cd $FOAM_SRC'
alias lib='cd $FOAM_LIBBIN'
alias run='cd $FOAM_RUN'
alias foam='cd $WM_PROJECT_DIR'
alias foamsrc='cd $FOAM_SRC/$WM_PROJECT'
alias foamfv='cd $FOAM_SRC/finiteVolume'
alias app='cd $FOAM_APP'
alias util='cd $FOAM_UTILITIES'
alias sol='cd $FOAM_SOLVERS'
alias tut='cd $FOAM_TUTORIALS'

alias foamApps='cd $FOAM_APP'
alias foamSol='cd $FOAM_SOLVERS'
alias foamTuts='cd $FOAM_TUTORIALS'
alias foamUtils='cd $FOAM_UTILITIES'
alias foam3rdParty='cd $WM_THIRD_PARTY_DIR'
```

etc/config/settings.sh

```
# convenience
export FOAM_ETC=$WM_PROJECT_DIR/etc
export FOAM_APP=$WM_PROJECT_DIR/applications
export FOAM_SRC=$WM_PROJECT_DIR/src
export FOAM_TUTORIALS=$WM_PROJECT_DIR/tutorials
export FOAM_UTILITIES=$FOAM_APP/utilities
export FOAM_SOLVERS=$FOAM_APP/solvers
export FOAM_RUN=$WM_PROJECT_USER_DIR/run
```

ディレクトリに関する環境変数

使用例

```
$ app
$ pwd
/home/custom/OpenFOAM/OpenFOAM-2.4.x/applications
$ src
$ pwd
/home/custom/OpenFOAM/OpenFOAM-2.4.x/src
```

目次

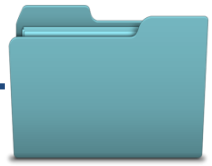
第１章　OpenFOAMの境界条件

この章では，OpenFOAMに実装された境界条件の基本をおさらいします.

境界条件のソースコードの設置ディレクトリ

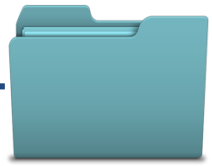
src/finiteVolume/fields/fvPatchFields

fvPatchField
全ての境界条件クラスの基底クラス



basic
基本的な境界条件 (ディリクレ条件，ノイマン条件など)

constraint
幾何学的な周期性や対称性などを考慮する境界条件



derived
基本的な境界条件から派生した具体的な境界条件

doc
ドキュメント

OpenFOAM の境界条件クラスの構成

- ***fvPatchField*** クラスが全ての境界条件クラスの**基底クラス**になっています.
- 境界条件クラスに共通して必要なメンバ関数が，***fvPatchField*** クラスに実装されています.
- ***fvPatchField*** クラスから ***basic***，***derived*** ディレクトリに格納されたクラスに継承を繰り返して，用途や使用できる変数の型を限定したより具体的な境界条件クラスを実装しています.

src/finiteVolume/fields/fvPatchFields/fvPatchField/fvPatchField.H

Class
    Foam::fvPatchField

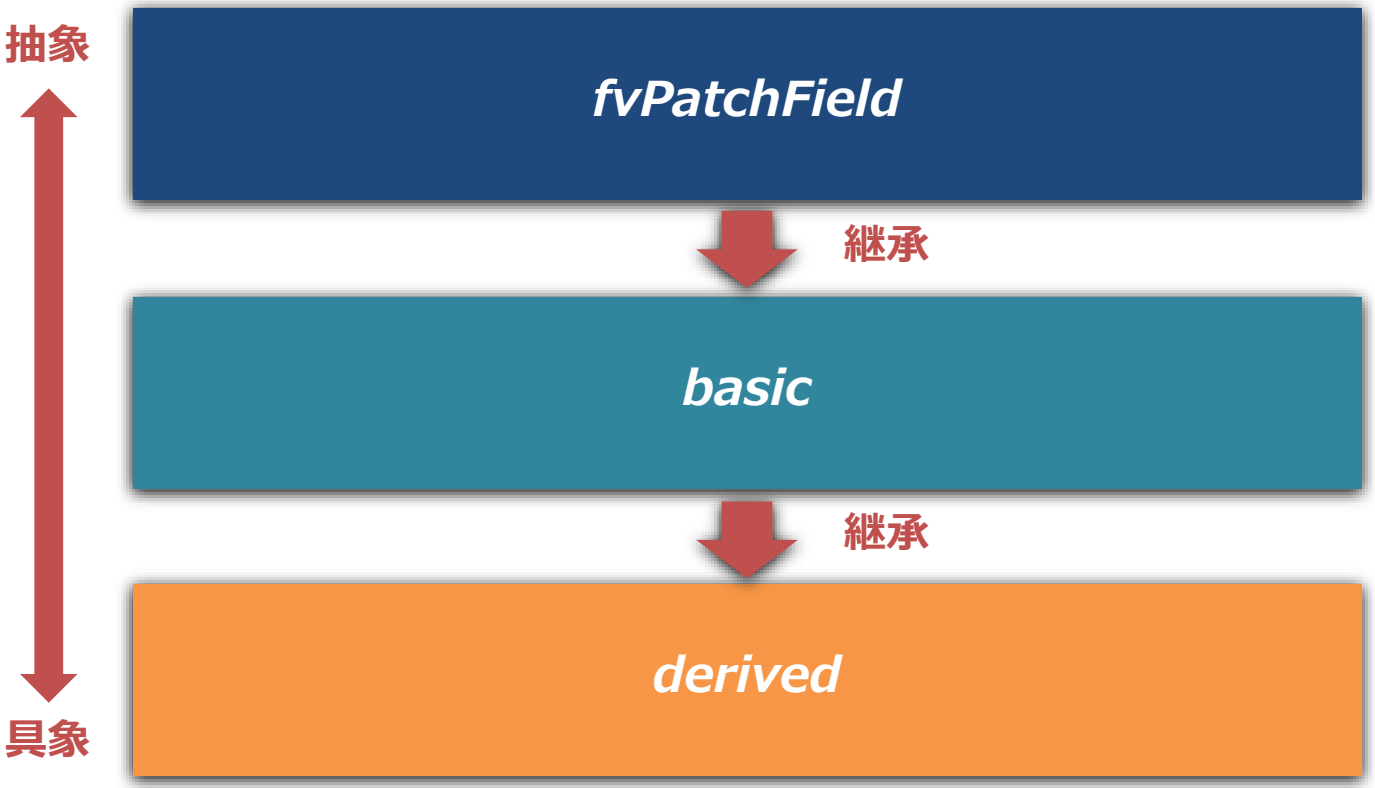
Description
    Abstract base class with a fat-interface to all derived classes
    covering all possible ways in which they might be used.

    The first level of derivation is to basic patchFields which cover
    zero-gradient, fixed-gradient, fixed-value and mixed conditions.

    The next level of derivation covers all the specialised types with
    specific evaluation proceedures, particularly with respect to specific
    fields.

OpenFOAM の境界条件クラスの構成

- ***fvPatchField*** クラスが全ての境界条件クラスの**基底クラス**になっています．
- 境界条件クラスに共通して必要なメンバ関数が，***fvPatchField*** クラスに実装されています．
- ***fvPatchField*** クラスから ***basic***，***derived*** ディレクトリに格納されたクラスに継承を繰り返して，用途や使用できる変数の型を限定したより具体的な境界条件クラスを実装しています．



fvPatchField クラス

- ***fvPatchField*** クラスの複数のメンバ関数は具体的な実装を持ちません.
  ⇒ 継承して使うことを前提にした基底クラスです.

src/finiteVolume/fields/fvPatchFields/fvPatchField/fvPatchField.H

```
//- Return the matrix diagonal coefficients corresponding to the
//  evaluation of the value of this patchField with given weights
virtual tmp<Field<Type> > valueInternalCoeffs
(
    const tmp<Field<scalar> >&
) const
{
    notImplemented
    (
        type()
      + "::valueInternalCoeffs(const tmp<Field<scalar> >&)"
    );
    return *this;
}
```

- このメンバ関数を使用しようとすると，エラーを出力して終了します.
  ⇒ マクロ関数 ***notImplemented(Fn)*** については，次ページ.

fvPatchField クラス

- マクロ関数 ***notImplemented(Fn)***

src/OpenFOAM/db/error/error.H

```
/**
 * ¥def notImplemented(functionName)
 * Issue a FatalErrorIn for the functionName.
 * This is used for functions that are not currently implemented.
 * The functionName is printed and then abort is called.
 *
 * ¥note
 * This macro can be particularly useful when methods must be defined to
 * complete the interface of a derived class even if they should never be
 * called for this derived class.
*/
#define notImplemented(fn) ¥
    FatalErrorIn(fn) << "Not implemented" << ::Foam::abort(FatalError);
```

- 基底クラス ***fvPatchField*** で具体的に実装されていないメンバ関数に対して，派生クラスでは，それぞれの条件に応じてオーバーライド (再定義) を行っています．

basic と derived ディレクトリ

- この資料では，独自に次のような名前を使用しています．
  - ***basic*** ディレクトリに設置された境界条件(クラス): **基本タイプ**
  - ***derived*** ディレクトリに設置された境界条件(クラス): **具象タイプ**
- ***basic*** と ***derived*** ディレクトリには，多数の境界条件クラスのソースコードが設置されています．
- 下記の点に特に注目して見ていきましょう．
  - ***basic*** と ***derived*** ディレクトリが区別されている理由
  - ***basic*** ディレクトリに設置されたそれぞれの境界条件クラスが，どのような基準で別クラスとして定義されているのか？

基本タイプについて

➢ **境界条件の与え方が異なるタイプ**ごとにクラスが実装されています.

- 値を規定する条件
- 勾配を規定する条件
- これらを組み合わせた条件
  - 条件で切り替える
  - 空間方向で切り替える　など

境界条件名	説明
fixedValue	ディリクレ (**Dirichlet**) 境界条件 境界上の値を規定
fixedGradient	ノイマン (**Neumann**) 境界条件 境界上の法線方向勾配を規定
zeroGradient	***fixedGradient*** 条件において, 特に勾配の値が 0 の場合の条件
mixed	***fixedValue*** と ***fixedGradient*** の 2 つを 組み合わせた条件
directionMixed	***fixedValue*** と ***fixedGradient*** の 2 つを 空間の方向により切り替える条件
basicSymmetry	対称境界条件のベース
coupled	周期境界条件のベース

基本タイプ｜ラプラシアン項の離散化の比較

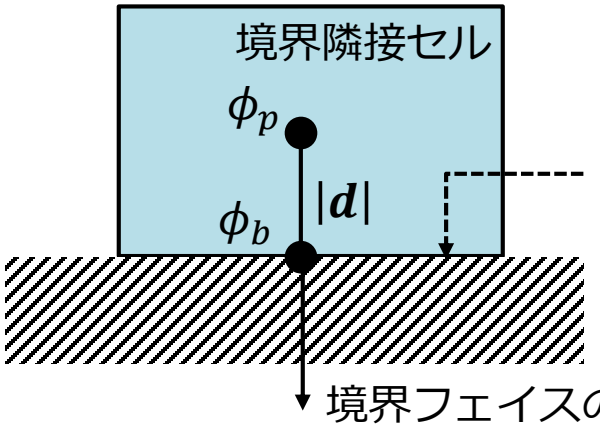
- 境界条件の与え方が異なると，**境界における離散化の処理が異なります**.
- まず，**ラプラシアン項**の離散化を考えます.

$$\int_V \nabla \cdot (\Gamma \nabla \phi)\, dV = \int_S d\boldsymbol{S} \cdot (\Gamma \nabla \phi) = \sum_f \boxed{\Gamma_f \ \boldsymbol{S}_f \cdot (\nabla \phi)_f}$$

ラプラシアン流束

- 境界フェイスにおけるラプラシアン流束の評価 (添え字 b は境界の意味)

$$\Gamma_f \boldsymbol{S}_f \cdot (\nabla \phi)_f = \Gamma_b |\boldsymbol{S}_b| \frac{\phi_b - \phi_p}{|\boldsymbol{d}|}$$



ϕ_p: 境界隣接セル中心値
ϕ_b: 境界フェイス中心値
$|\boldsymbol{d}|$: 境界隣接セル中心と
境界フェイス中心間の距離

- 境界において値 ϕ_b を規定した場合 (***fixedValue***)

$$\Gamma_f \boldsymbol{S}_f \cdot (\nabla\phi)_f = \Gamma_b |\boldsymbol{S}_b| \frac{\phi_b - \phi_p}{|\boldsymbol{d}|}$$

$$= \Gamma_b |\boldsymbol{S}_b| \left(\frac{-1}{|\boldsymbol{d}|}\right) \phi_p + \Gamma_b |\boldsymbol{S}_b| \frac{1}{|\boldsymbol{d}|} \phi_b$$

係数行列

右辺ベクトル

$$\begin{pmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{N1} & \cdots & a_{NN} \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_N \end{pmatrix} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_N \end{pmatrix}$$

係数行列の対角成分と右辺ベクトルの両方に対して寄与があります.

- 境界の法線方向の勾配の値 g_b を規定した場合 (***fixedGradient***)

$$\Gamma_f \boldsymbol{S}_f \cdot (\nabla\phi)_f = \Gamma_b |\boldsymbol{S}_b| \left.\frac{\partial\phi}{\partial \boldsymbol{n}}\right|_b = \boxed{\Gamma_b |\boldsymbol{S}_b| g_b}$$

$$\begin{pmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{N1} & \cdots & a_{NN} \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_N \end{pmatrix} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_N \end{pmatrix}$$

全てを陽的に処理できるため，係数行列の対角成分に対しての寄与はありません.

- 特に，法線方向の勾配の値を 0 に規定した場合 (***zeroGradient***)

$$\Gamma_f \boldsymbol{S}_f \cdot (\nabla\phi)_f = \Gamma_b |\boldsymbol{S}_b| \left.\frac{\partial\phi}{\partial \boldsymbol{n}}\right|_b = 0$$

$$\begin{pmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{N1} & \cdots & a_{NN} \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_N \end{pmatrix} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_N \end{pmatrix}$$

係数行列の対角成分と右辺ベクトルの両方に
対して寄与がありません.

基本タイプ｜対流項の離散化の比較

- 続いて，**対流項**の離散化を見てみましょう．
- 対流項はガウスの発散定理を使用して，次式のように離散化されます．

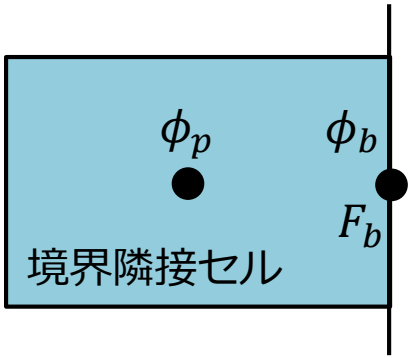
$$\int_V \nabla \cdot (\boldsymbol{U}\phi)\, dV = \int_S d\boldsymbol{S} \cdot (\boldsymbol{U}\phi) = \sum_f \boldsymbol{S}_f \cdot \boldsymbol{U}_f\, \phi_f = \sum_f F \phi_f$$

流束 (flux)

- 境界隣接セルでの離散化では，境界フェイスにおいて

$$F_b \phi_b$$

の評価が必要です．



ϕ_p：境界隣接セル中心値
ϕ_b：境界フェイス中心値
F_b：境界フェイス流束

- 境界において値 ϕ_b を規定した場合 (***fixedValue***)

$$F_b\phi_b$$

ϕ_b は境界条件から既知
流束は，陽的に処理される

$$\begin{pmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{N1} & \cdots & a_{NN} \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_N \end{pmatrix} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_N \end{pmatrix}$$

全て陽的に処理されるため，係数行列の対角成分に
対しての寄与はありません.

- 境界の法線方向の勾配の値 g_b を規定した場合 (***fixedGradient***)

$$\frac{\phi_b - \phi_p}{|\boldsymbol{d}|} = g_b \rightarrow \phi_b = \phi_p + g_b|\boldsymbol{d}|$$

$$\rightarrow F_b\phi_b = F_b\phi_p + F_b g_b|\boldsymbol{d}|$$

$$\begin{pmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{N1} & \cdots & a_{NN} \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_N \end{pmatrix} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_N \end{pmatrix}$$

係数行列の対角成分と右辺ベクトルの両方に
対して寄与があります.

- 特に，法線方向の勾配の値を 0 に規定した場合 (***zeroGradient***)

$$\frac{\phi_b - \phi_p}{|\boldsymbol{d}|} = 0 \quad \rightarrow \phi_b = \phi_p \quad \rightarrow F_b\phi_b = \boxed{F_b}\phi_p$$

$$\begin{pmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{N1} & \cdots & a_{NN} \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_N \end{pmatrix} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_N \end{pmatrix}$$

係数行列に対してのみ寄与があります.

基本タイプ｜mixed と directionMixed

- 少し分かりにくい ***mixed*** と ***directionMixed*** の2つの条件が境界値をどのように計算しているのかを詳しく見ていきます.

境界条件名	説明
fixedValue	ディリクレ (**Dirichlet**) 境界条件 境界上の値を規定
fixedGradient	ノイマン (**Neumann**) 境界条件 境界上の法線方向勾配を規定
zeroGradient	***fixedGradient*** 条件において, 特に勾配の値が 0 の場合の条件
mixed	***fixedValue*** と ***fixedGradient*** の２つを 組み合わせた条件
directionMixed	***fixedValue*** と ***fixedGradient*** の２つを 空間の方向により切り替える条件
basicSymmetry	対称境界条件のベース
coupled	周期境界条件のベース

基本タイプ｜mixed 条件

- ***mixed*** 境界条件は，***fixedValue*** と ***fixedGradient*** の組み合わせ
- 境界値の計算に使用される３つの変数

変数名	型
refValue_	変数と同じ型
refGrad_	変数と同じ型
valueFraction_	スカラー

mixedFvPatchField.H

```
//- Value field
Field<Type> refValue_;

//- Normal gradient field
Field<Type> refGrad_;

//- Fraction (0-1) of value used
                for boundary condition
scalarField valueFraction_;
```

- どちらの条件が課されるのかを，***valueFraction_*** がコントロール
  - ***valueFraction_*** = 1 の場合: ***fixedValue*** 条件
    境界値を，***refValue_*** で指定
  - ***valueFraction_*** = 0 の場合: ***fixedGradient*** 条件
    境界法線方向の勾配値を，***refGradient_*** で指定
  - 0 < ***valueFraction_*** < 1 の場合: ２つの条件の足し合わせ

基本タイプ｜mixed 条件

➢ 境界値の計算方法

src/finiteVolume/fields/fvPatchFields/basic/mixed/mixedFvPatchField.C

```
template<class Type>
void mixedFvPatchField<Type>::evaluate(const Pstream::commsTypes)
{
    if (!this->updated())
    {
        this->updateCoeffs();
    }

    Field<Type>::operator=
    (
        valueFraction_*refValue_
      +
        (1.0 - valueFraction_)*
        (
            this->patchInternalField()
          + refGrad_/this->patch().deltaCoeffs()
        )
    );

    fvPatchField<Type>::evaluate();
}
```

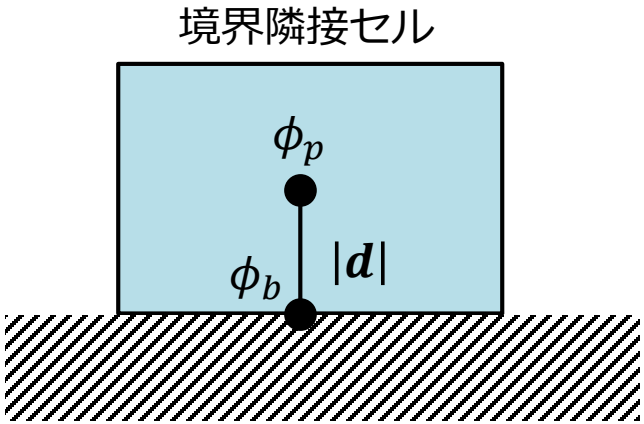
3つの変数を使用して，
境界値を計算

- ***this->patchInternalField()***: 境界隣接セル中心での変数値
- ***this->patch().deltaCoeffs()***: 隣接セル中心と境界フェイス中心間の距離の逆数

- 境界値の計算式

$$\phi_b = \mathbf{valueFraction} \times \mathbf{refValue} + (1 - \mathbf{valueFraction}) \times (\phi_p + \mathbf{refGradient} \times |\boldsymbol{d}|)$$

$$\phi_b = \begin{cases} \mathbf{refValue} & (\mathbf{valueFraction} = 1) \\ \phi_p + \mathbf{refGradient} \times |\boldsymbol{d}| & (\mathbf{valueFraction} = 0) \end{cases}$$



設定書式 (変数がベクトルの場合)

```
myPatch
{
    type            mixed;
    refValue        uniform (10 0 0);
    refGradient     uniform (0 0 0);
    valueFraction   uniform 1;
    value           uniform (10 0 0);
}
```

基本タイプ｜directionMixed 条件

- ***directionMixed*** 条件は，***fixedValue*** と ***fixedGradient*** の2つの条件を空間の方向で切り替える条件です.

- 境界値の計算に使用される３つの変数

パラメータ名	型
refValue_	変数と同じ型
refGrad_	変数と同じ型
valueFraction_	対称テンソル

directionMixedFvPatchField.H

```
//- Value field
Field<Type> refValue_;

//- Normal gradient field
Field<Type> refGrad_;

//- Fraction (0-1) of value used
        for boundary condition
symmTensorField valueFraction_;
```

- これらの3つの変数を使って，

  - 境界値がどのように計算されるのか
  - どのようにして方向により条件を切り替えているのか

  次ページから詳しく見ていきます.

基本タイプ｜directionMixed 条件

- どの方向にどちらの条件が課されるのかを ***valueFraction_*** がコントロール

*valueFraction_*	境界の法線方向	境界の接線方向
$\boldsymbol{n} \otimes \boldsymbol{n}$	*fixedValue* ディリクレ条件	*fixedGradient* ノイマン条件
$\boldsymbol{I} - \boldsymbol{n} \otimes \boldsymbol{n}$	*fixedGradient* ノイマン条件	*fixedValue* ディリクレ条件
$\boldsymbol{I}$ (単位行列)	*fixedValue* ディリクレ条件	
$\boldsymbol{O}$ (零行列)	*fixedGradient* ノイマン条件	

- $\boldsymbol{n}$: 境界の単位法線ベクトル
- 演算子 ⊗: 外積 (outer product) [1]

基本タイプ｜directionMixed 条件

- 境界値の計算方法

src/finiteVolume/fields/fvPatchFields/basic/directionMixed/directionMixedFvPatchField.C

```
template<class Type>
void Foam::directionMixedFvPatchField<Type>::evaluate(const Pstream::commsTypes)
{
    if (!this->updated())
    {
        this->updateCoeffs();
    }

1   tmp<Field<Type> > normalValue = transform(valueFraction_, refValue_);

    tmp<Field<Type> > gradValue =
        this->patchInternalField() + refGrad_/this->patch().deltaCoeffs();

2   tmp<Field<Type> > transformGradValue =
        transform(I - valueFraction_, gradValue);

    Field<Type>::operator=(normalValue + transformGradValue);

    transformFvPatchField<Type>::evaluate();
}
```

境界値 = 1 + 2

- ***transform*** を使用して，各方向成分を計算

- ***transform*** について

src/OpenFOAM/primitives/transform/symmTransform.H

```
inline scalar transform(const symmTensor&, const scalar s)
{
    return s;
}


template<class Cmpt>
inline Vector<Cmpt> transform(const symmTensor& stt, const Vector<Cmpt>& v)
{
    return stt & v;
}
```

スカラー変数に対して

ベクトル変数に対して

- 境界値の計算式

$$\text{refValue\_} = \boldsymbol{U} = (U_1, U_2, U_3), \qquad \text{refGrad\_} = \boldsymbol{G} = (G_1, G_2, G_3)$$

単位法線ベクトル $\boldsymbol{n} = (n_1, n_2, n_3)$

ケース 1｜ valueFraction_ = $\boldsymbol{n} \otimes \boldsymbol{n}$ と設定した場合

$$\text{valueFraction\_} = \begin{pmatrix} n_1 n_1 & n_1 n_2 & n_1 n_3 \\ n_2 n_1 & n_2 n_2 & n_2 n_3 \\ n_3 n_1 & n_3 n_2 & n_3 n_3 \end{pmatrix}$$

$$\text{normalValue} = \begin{pmatrix} n_1 n_1 & n_1 n_2 & n_1 n_3 \\ n_2 n_1 & n_2 n_2 & n_2 n_3 \\ n_3 n_1 & n_3 n_2 & n_3 n_3 \end{pmatrix} \begin{pmatrix} U_1 \\ U_2 \\ U_3 \end{pmatrix}$$

$$= \begin{pmatrix} (U_1 n_1 + U_2 n_2 + U_3 n_3) n_1 \\ (U_1 n_1 + U_2 n_2 + U_3 n_3) n_2 \\ (U_1 n_1 + U_2 n_2 + U_3 n_3) n_3 \end{pmatrix}$$

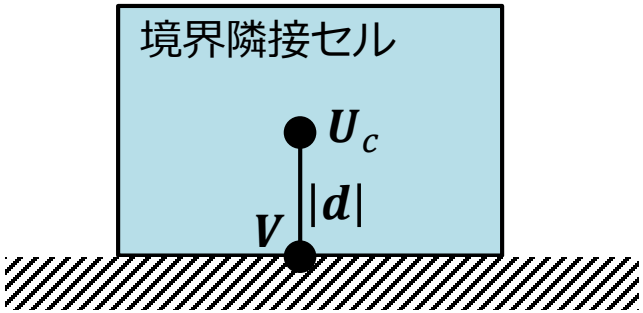
$$= (\boldsymbol{U} \cdot \boldsymbol{n})\boldsymbol{n} = \boldsymbol{U}_{\perp}$$

(***refValue_*** の法線方向成分)

次ページに続く

法線方向勾配の値が $\mathrm{refGrad\_} = \boldsymbol{G} = (G_1, G_2, G_3)$ になるような
境界の値 $\boldsymbol{V} = (V_1, V_2, V_3)(= \mathrm{gradValue})$ を，次式のように計算します．

$$\boldsymbol{V} = \boldsymbol{U}_c + \boldsymbol{G} \cdot |\boldsymbol{d}|$$



$$\mathrm{transformGradValue} = \left[\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} n_1 n_1 & n_1 n_2 & n_1 n_3 \\ n_2 n_1 & n_2 n_2 & n_2 n_3 \\ n_3 n_1 & n_3 n_2 & n_3 n_3 \end{pmatrix}\right] \begin{pmatrix} V_1 \\ V_2 \\ V_3 \end{pmatrix}$$
$$= \boldsymbol{V} - (\boldsymbol{V} \cdot \boldsymbol{n})\boldsymbol{n} = \boldsymbol{V}_{\parallel}$$

したがって，境界値は，次式のように計算されます．

$$\mathrm{normalValue} + \mathrm{transformGradValue} = \boldsymbol{U}_{\perp} + \boldsymbol{V}_{\parallel}$$

- **法線方向**成分: ***fixedValue*** (値は，***refValue_***)
- **接線方向**成分: ***fixedGradient*** (勾配は，***refGrad_***)

ケース 2｜valueFraction_ $= \boldsymbol{I} - \boldsymbol{n} \otimes \boldsymbol{n}$ と設定した場合

$$\text{normalValue} = \boldsymbol{U} - (\boldsymbol{U} \cdot \boldsymbol{n})\boldsymbol{n} = \boldsymbol{U}_{\parallel}$$

$$\text{transformGradValue} = (\boldsymbol{V} \cdot \boldsymbol{n})\boldsymbol{n} = \boldsymbol{V}_{\perp}$$

したがって，境界値は，次式のように計算されます．

$$\text{normalValue} + \text{transformGradValue} = \boldsymbol{U}_{\parallel} + \boldsymbol{V}_{\perp}$$

- **接線方向**成分: ***fixedValue*** (値は，***refValue_***)
- **法線方向**成分: ***fixedGradient*** (勾配は，***refGrad_***)

ケース 3｜valueFraction_ $= \boldsymbol{I}$ と設定した場合

全ての方向: ***fixedValue*** (値は，***refValue_***)

ケース 4｜valueFraction_ $= \boldsymbol{0}$ と設定した場合

全ての方向: ***fixedGradient*** (勾配は，***refGrad_***)

基本タイプ｜ポアソン方程式でのチェック

- 全ての境界において圧力の境界条件に ***zeroGradient*** 条件などを使用して，
  境界上で全く値を規定しないでソルバーを実行した場合に，
  次のエラーメッセージが表示されます．

  --> FOAM FATAL IO ERROR:
  Unable to set reference cell for field p
  Please supply either pRefCell or pRefPoint

- 圧力ポアソン方程式を解く際に，解が一意に決まるためには，
  計算領域内の少なくとも１点において圧力の値が規定されている必要があります．

- 境界において値が規定されているかどうかは，基本タイプにより異なります．

src/finiteVolume/fields/fvPatchFields/basic/fixedValue/fixedValueFvPatchField.H

```
//- Return true if this patch field fixes a value.
//  Needed to check if a level has to be specified while solving
//  Poissons equations.
virtual bool fixesValue() const
{
    return true;
}
```

fixedValue 条件は，
値を規定しているので
true を返します．

具象タイプについて

- 例えば，流速の値を規定するディリクレ境界条件を考えてみます．

- ユーザーが値を設定する方法にはいくつものバリエーションがあります．

  - ベクトルの (デカルト座標) 3 成分を指定する
  - 境界の法線方向成分を指定する
  - 体積 (質量) 流量を指定する
  - 円柱座標系で指定する　　など

- これらは全て値を規定しているので，基本タイプは ***fixedValue*** ですが，
  次の点が異なります．

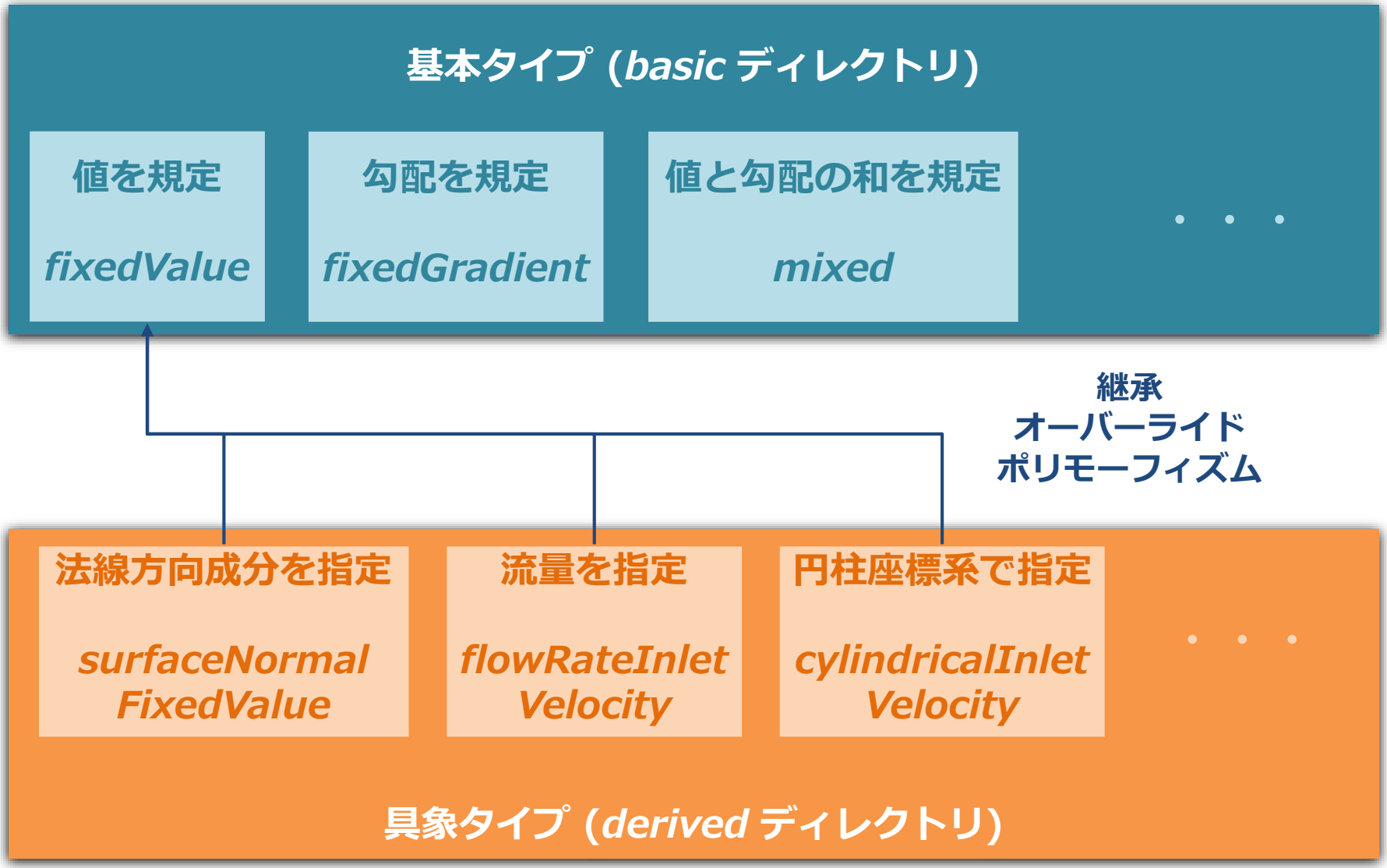
  - 規定する値の具体的な計算方法
  - ユーザーが設定するパラメータの種類・数

- この例のように，基本タイプとしては ***fixedValue*** で一緒でも，
  規定する値の具体的な設定方法がいろいろある方がユーザーには便利です．

- プログラミングを行う人の視点から見れば，***fixedValue*** のクラスを継承した
  派生クラスを作成して，規定する値を計算するメンバ関数を
  オーバーライド (再定義) することで，このバリエーションを実現できます．
  ⇒ ポリモーフィズム (多相性，多態性，多様性)

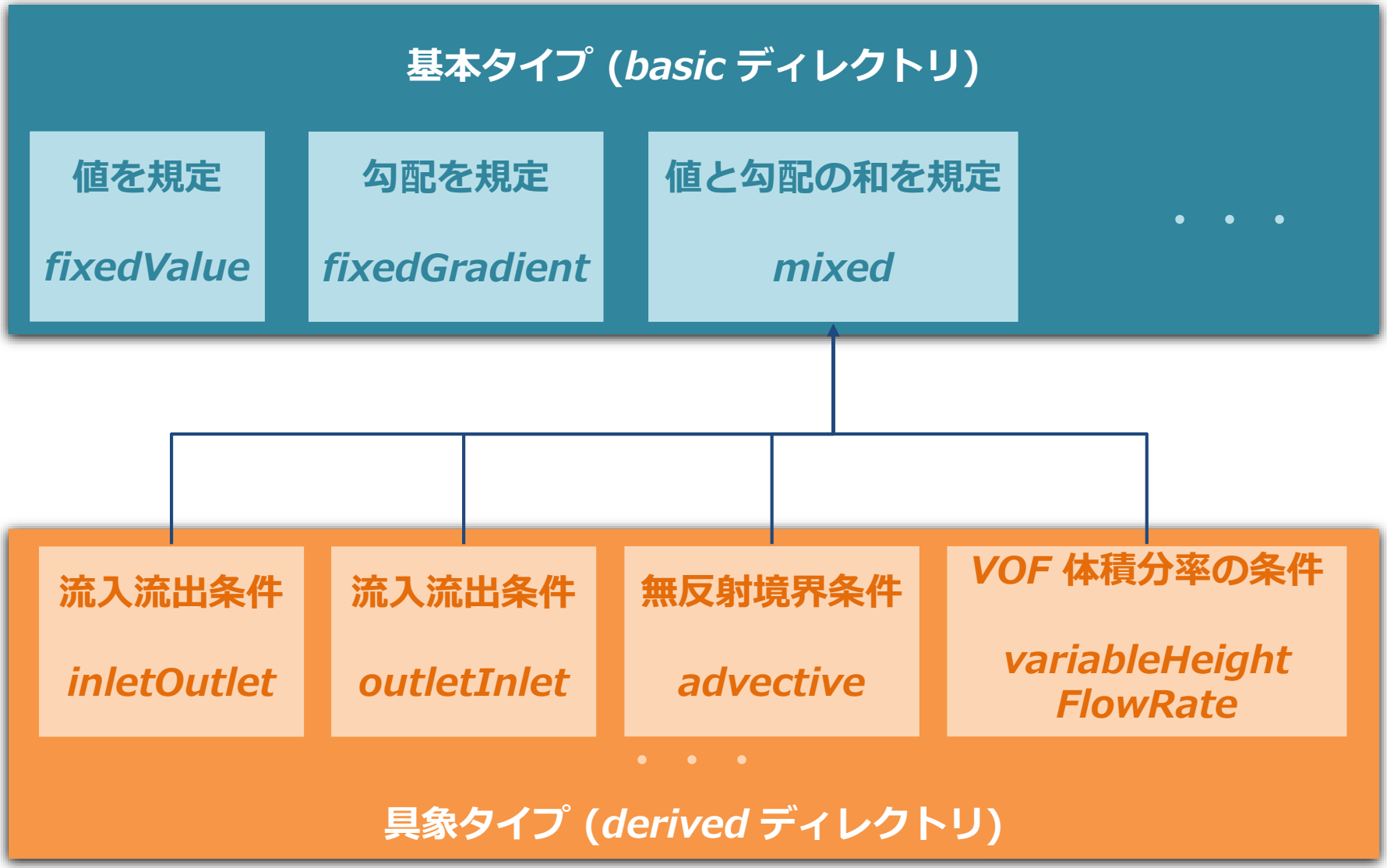
- 基本タイプの条件クラスの派生クラスが，***derived*** ディレクトリに設置されています．

fixedValue の具象タイプ



規定する値の具体的な計算方法・設定方法に多様性を持たせています

mixed の具象タイプ



第２章　TJunction チュートリアル

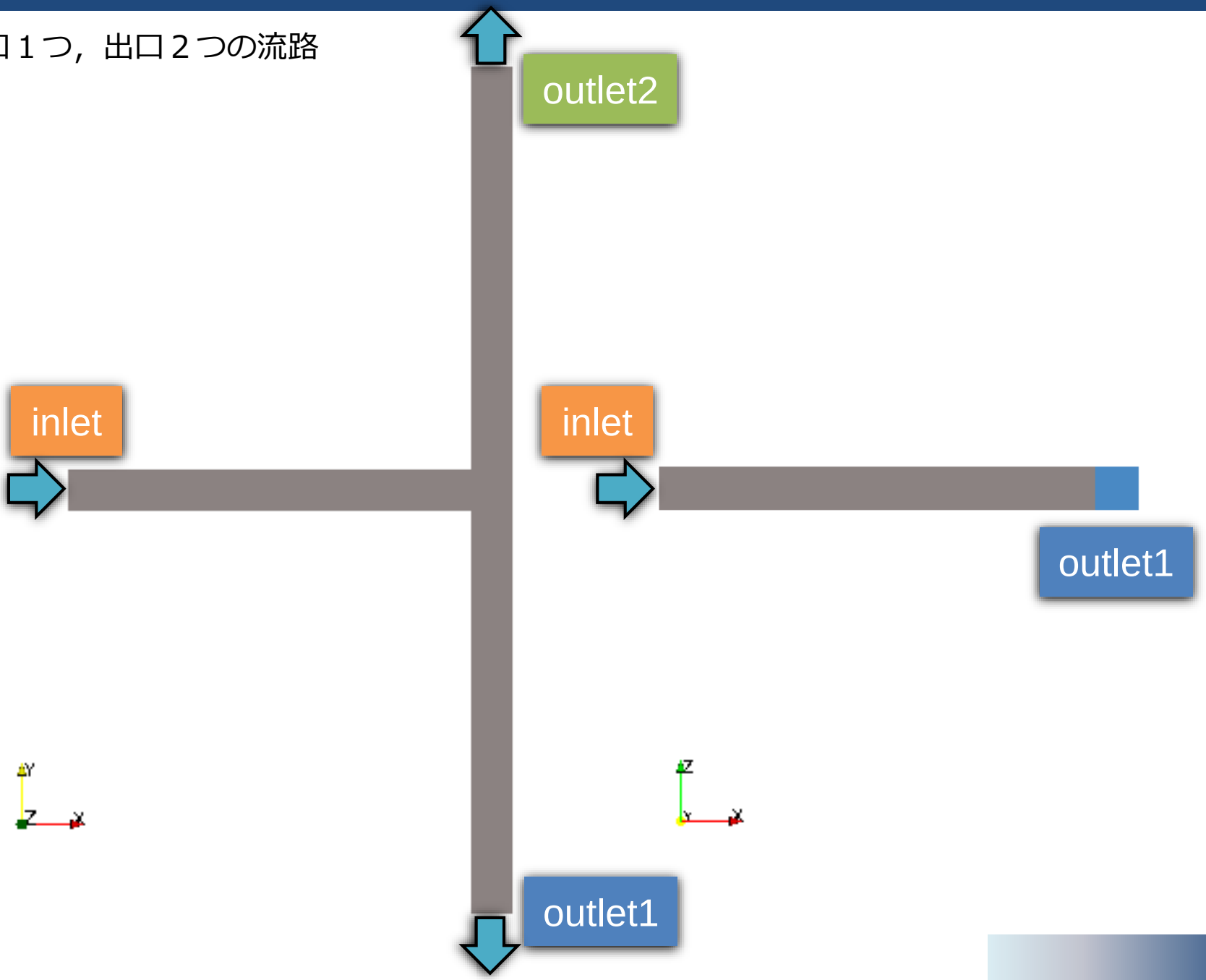
この章では，カスタマイズした *pimpleFoam* を使用し，チュートリアル *TJunction* に取り組みます．

この章のキーワードは，次の３つです．

- ***mixed*** 条件
- ***directionMixed*** 条件
- 非定常境界条件

計算形状

入口１つ，出口２つの流路



流速の境界条件

```
outlet2
{
    type          inletOutlet;
    inletValue    uniform (0 0 0);
    value         uniform (0 0 0);
}
```

```
defaultFaces
{
    type    fixedValue;
    value   uniform (0 0 0);
}
```

```
inlet
{
    type    pressureInletOutletVelocity;
    value   uniform (0 0 0);
}
```

チェックポイント２

```
outlet1
{
    type          inletOutlet;
    inletValue    uniform (0 0 0);
    value         uniform (0 0 0);
}
```

チェックポイント１

圧力の境界条件

```
inlet
{
    type        uniformTotalPressure;
    pressure    table
    (
        (0 10)
        (1 40)
    );
    p0          40;
    U           U;
    phi         phi;
    rho         none;
    psi         none;
    gamma       1;
    value       uniform 40;
}
```

チェックポイント3

非定常な境界条件

```
outlet2
{
    type    fixedValue;
    value   uniform 0;
}
```

```
defaultFaces
{
    type    zeroGradient;
}
```

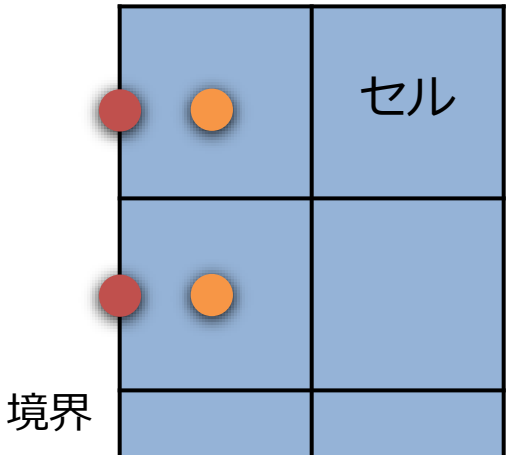
```
outlet1
{
    type    fixedValue;
    value   uniform 10
}
```

myPimpleFoam の作成

- ここでは，より境界条件に着目するために，
  計算に使用するソルバーをカスタマイズします．

- 具体的な変更点は，各時間ステップにおいて，次の情報をファイルに
  出力するように ***pimpleFoam*** をカスタマイズします．

  - 指定した境界フェイスの中心点 ● での変数 (流速と圧力) の値
  - その境界フェイスに隣接するセルの中心点 ● での変数の値



- 計算実行後に，ファイルに出力したこれらの値の間に成り立つ関係を
  確認することで，境界条件に対する理解を深めることを目指します．

myPimpleFoam の作成｜端末の起動

myPimpleFoam の作成

- ***myPimpleFoam*** のディレクトリを作成します.

実行コマンド

```
$ foam
$ mkdir -p $FOAM_RUN
$ cp -r --parents applications/solvers/incompressible/pimpleFoam ¥
> $WM_PROJECT_USER_DIR
$ cd $WM_PROJECT_USER_DIR/applications/solvers/incompressible
$ mv pimpleFoam myPimpleFoam
$ cd myPimpleFoam
$ mv pimpleFoam.C myPimpleFoam.C
$ wclean
```

このトレーニングでは不要なので,
- SRFPimpleFoam，pimpleDyMFoam の２つのディレクトリと
- Allwmake ファイル

は削除しても構いません.

```
$ rm -r pimpleDyMFoam SRFPimpleFoam Allwmake
```

補足説明

入力コマンドが長くなる場合には，バックスラッシュ『¥』で改行を行い，コマンド入力を複数行に分けることができます.

myPimpleFoam の作成

- ***Make/files*** ファイルの内容を確認します.

実行コマンド

```
$ cat Make/files
```

編集前の *Make/files*

```
pimpleFoam.C

EXE = $(FOAM_APPBIN)/pimpleFoam
```

- ***Make/files*** ファイルを置き換えコマンド ***sed*** を使用して編集します.

実行コマンド

```
$ sed -i -e "s/pimpleFoam/myPimpleFoam/g" Make/files
$ sed -i -e "s/FOAM_APPBIN/FOAM_USER_APPBIN/g" Make/files
$ cat Make/files
```

編集後の *Make/files*

```
myPimpleFoam.C

EXE = $(FOAM_USER_APPBIN)/myPimpleFoam
```

- 前ページで行ったソースファイルの名前変更に対応
  pimpleFoam.C ⇒ myPimpleFoam.C

myPimpleFoam の作成

- ***Make/files*** ファイルの書式

編集後の *Make/files*

myPimpleFoam.C　ソースファイル名

EXE = $(FOAM_USER_APPBIN)/myPimpleFoam

- **EXE =** 実行ファイルの設置ディレクトリ/実行ファイル名
- 上記の設定と DEXCS の環境では，/home/custom/OpenFOAM/custom-2.4.x/platforms/linux64GccDPOpt/bin/ 以下に myPimpleFoam が作成されます．

実行ファイルの設置ディレクトリ

- 実行ファイルの設置ディレクトリは，次の２種類用意されています．
  - ***$FOAM_APPBIN***
  - ***$FOAM_USER_APPBIN***

- ユーザーが独自に作成したアプリケーションの実行ファイルについては，***$FOAM_USER_APPBIN*** に設置することが推奨されています．

- これらの環境変数は，***etc/config/settings.sh*** ファイルに設定されています．

```
# user executables/libraries
export FOAM_USER_APPBIN=$WM_PROJECT_USER_DIR/platforms/$WM_OPTIONS/bin
export FOAM_USER_LIBBIN=$WM_PROJECT_USER_DIR/platforms/$WM_OPTIONS/lib
```

コーディングスタイルの指針

- The OpenFOAM Foundation のページ ***OpenFOAM Code Style Guide*** に OpenFOAM でコーディングする際のスタイルの指針が示されています.

Splitting long lines at an "="

Indent after split

```
variableName =
    longClassName.longFunctionName(longArgument);
```

OR (where necessary)

```
variableName =
    longClassName.longFunctionName
    (
        longArgument1,
        longArgument2
    );
```

not

```
variableName =
longClassName.longFunctionName(longArgument);
```

nor

```
variableName = longClassName.longFunctionName
(
    longArgument1,
    longArgument2
);
```

myPimpleFoam の作成

➢ ***myPimpleFoam.C*** ファイルを開いて，下記の２箇所を編集します．

- 編集箇所１： ２つの出力ファイル (Uboundary.dat と pboundary.dat) を設定

/home/custom/OpenFOAM/custom-2.4.x/applications/solvers/incompressible/myPimpleFoam/myPimpleFoam.C

```
57      pimpleControl pimple(mesh);
58
59      OFstream Ubc("Uboundary.dat");
60      OFstream pbc("pboundary.dat");
61
62      // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
63
64      Info<< "¥nStarting time loop¥n" << endl;
```

追加１

- 編集箇所２：新しいヘッダーファイルのインクルード

```
93          runTime.write();
94
95          #include "checkBoundaries.H"
96          #include "checkReverseFlow.H"
97
98          Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
```

追加２

これらのヘッダーファイルは，次ページ以降で作成します．

➢ 上記の変更が終了したら，***myPimpleFoam.C*** ファイルを上書き保存します．

myPimpleFoam の作成

- ***checkBoundaries.H*** ファイルを新規作成し，下記をコピーします(次頁に続く).

/home/custom/OpenFOAM/custom-2.4.x/applications/solvers/incompressible/myPimpleFoam/checkBoundaries.H

```
1     if (mesh.solutionDict().isDict("checkBC"))
2     {
3         const dictionary& bcDict = mesh.solutionDict().subDict("checkBC");
4         labelList localFaceId(bcDict.lookup("targetFaceId"));
5
6         // Output to Uboundary.dat
7         Ubc<< "Time = " << runTime.timeName() << endl;
8
9         forAll(U.boundaryField(), patchI)
10        {
11            label facei = localFaceId[patchI];
12
13            Ubc<< "-----------------------------------------------------" << nl
14               << "Patch Name: " << mesh.boundary()[patchI].patch().name() << nl
15               << "BC Type: " << U.boundaryField()[patchI].type() << nl
16               << "Face Checked: localID(" << facei << "), Centre"
17               << mesh.boundary()[patchI].Cf()[facei] << nl
18               << "Face Value: " << U.boundaryField()[patchI][facei] << nl
19               << "Cell Value: " << U[mesh.boundary()[patchI].faceCells()[facei]] << nl
20               << "Flux phi : " << phi.boundaryField()[patchI][facei] << endl;
21        }
22
23        Ubc<< endl;
```

次ページに続く

myPimpleFoam の作成

- 前ページに続けて下記をコピーし，保存します．

/home/custom/OpenFOAM/custom-2.4.x/applications/solvers/incompressible/myPimpleFoam/checkBoundaries.H

```
25          // Output to pboundary.dat
26          pbc<< "Time = " << runTime.timeName() << endl;
27
28          forAll(p.boundaryField(), patchI)
29          {
30              label facei = localFaceId[patchI];
31
32              pbc<< "Patch Name: " << mesh.boundary()[patchI].patch().name() << nl
33                 << "BC Type: " << p.boundaryField()[patchI].type() << nl
34                 << "Face Checked: localID(" << facei << "), Centre"
35                 << mesh.boundary()[patchI].Cf()[facei] << nl
36                 << "Face Value: " << p.boundaryField()[patchI][facei] << nl
37                 << "Cell Value: " << p[mesh.boundary()[patchI].faceCells()[facei]] << endl;
38          }
39
40          pbc<< endl;
41      }
```

myPimpleFoam の作成

- ***checkReverseFlow.H*** ファイルを新規作成し，下記をコピーします(次頁に続く).

/home/custom/OpenFOAM/custom-2.4.x/applications/solvers/incompressible/myPimpleFoam/checkReverseFlow.H

```
1   if (mesh.solutionDict().isDict("checkBC"))
2   {
3       const dictionary& bcDict = mesh.solutionDict().subDict("checkBC");
4
5       labelHashSet patchSet_ =
6           mesh.boundaryMesh().patchSet(wordReList(bcDict.lookup("checkFlux")));
7
8       forAllConstIter(labelHashSet, patchSet_, iter)
9       {
10          label patchI = iter.key();
11          scalar inFlux = 0.0;
12          scalar inFluxArea = 0.0;
13          scalar outFlux = 0.0;
14          scalar outFluxArea = 0.0;
15
16          forAll(mesh.boundary()[patchI], facej)
17          {
18              scalar volFlux = phi.boundaryField()[patchI][facej];
19              scalar area = mesh.boundary()[patchI].magSf()[facej];
20
21              inFlux += volFlux*Foam::neg(volFlux);
22              inFluxArea += area*Foam::neg(volFlux);
23              outFlux += volFlux*Foam::pos(volFlux);
24              outFluxArea += area*Foam::pos(volFlux);
25          }
```

次ページに続く

myPimpleFoam の作成

- 前ページに続けて下記をコピーし，保存します．

/home/custom/OpenFOAM/custom-2.4.x/applications/solvers/incompressible/myPimpleFoam/checkReverseFlow.H

```
27              Info<< mesh.boundary()[patchI].patch().name() << " "
28                  << " Positive flux [m^3/s] (Area [m^2]): " << outFlux << "(" << outFluxArea << "),"
29                  << " Negative flux [m^3/s] (Area [m^2]): " << inFlux << "(" << inFluxArea << ")"
30                  << endl;
31          }
32      }
```

myPimpleFoam のコンパイル

- 以上で必要な修正が完了したので，***myPimpleFoam*** をコンパイルします．

実行コマンド

```
$ wmake
```

- エラーが出ていないことを確認します．

- エラーなくコンパイルできていれば，実行ファイル ***myPimpleFoam*** が ***$FOAM_USER_APPBIN*** に生成されます．

確認コマンド

```
$ ls $FOAM_USER_APPBIN
```

TJunction チュートリアルの実行

➢ TJunction チュートリアルのケースディレクトリに移動して，計算格子を生成します．

実行コマンド

```
$ run
$ cp -r $FOAM_TUTORIALS $FOAM_RUN
$ cd tutorials/incompressible/pimpleFoam/TJunction
$ blockMesh
$ checkMesh
```

checkMesh の実行結果

```
                                   (省略)

Checking patch topology for multiply connected surfaces...
    Patch                 Faces    Points   Surface topology
    inlet                 25       36       ok (non-closed singly connected)
    outlet1               25       36       ok (non-closed singly connected)
    outlet2               25       36       ok (non-closed singly connected)
    defaultFaces          3075     3104     ok (non-closed singly connected)

Checking geometry...
                                   (省略)
```

- ここで，各境界上のフェイスの数を確認します．
- もちろん，***constant/polyMesh/boundary*** ファイルの ***nFaces*** の値でも確認が可能です．

TJunction チュートリアルの実行

- ***system/fvSolution*** ファイルの最後に下記の設定を追加します．
  これが，***pimpleFoam*** に追加した２つのヘッダーファイルのコードの設定項目です．

system/fvSolution

```
checkBC
{
    targetFaceId        4(12 12 12 1012);          1
    checkFlux           3(inlet outlet1 outlet2);  2
}
```

1. 各境界上において，それぞれ指定した番号のフェイスの値をチェックします．

   - 1つ目の境界 *inlet* 上では，12番目のフェイス
   - 2つ目の境界 *outlet1* 上では，12番目のフェイス
   - 3つ目の境界 *outlet2* 上では，12番目のフェイス
   - 4つ目の境界 *defaultFaces* 上では，1012番目のフェイス

2. 3つの境界 *inlet*，*outlet1*，*outlet2* において体積流量を計算し出力します．

計算実行

- 設定が完了したので，ソルバーを実行します．

実行コマンド

```
$ myPimpleFoam > log.myPimpleFoam &
```

- 計算は数秒で終了し，終了後のディレクトリ構成は下記のようになります．

計算終了後のディレクトリ構成

```
0    0.2  0.4  0.6  0.8  1    1.2  1.4  README.txt     constant          pboundary.dat   system
0.1  0.3  0.5  0.7  0.9  1.1  1.3  1.5  Uboundary.dat  log.myPimpleFoam  postProcessing
```

- *Uboundary.dat* と *pboundary.dat* の２つのファイルが，前ページの設定 (1) に対する出力ファイルです．
- 設定 (2) に対する出力は，ログファイル *log.myPimpleFoam* に出力されます．

計算結果の確認

➢ ログファイルの確認

log.myPimpleFoam

```
Courant Number mean: 1.4954 max: 4.23791
deltaT = 0.0029344
Time = 0.191197

PIMPLE: iteration 1
smoothSolver:  Solving for Ux, Initial residual = 0.00634859, Final residual = 4.00613e-06, No Iterations 6
smoothSolver:  Solving for Uy, Initial residual = 0.0064305, Final residual = 3.56779e-06, No Iterations 6
smoothSolver:  Solving for Uz, Initial residual = 0.0407448, Final residual = 6.30377e-06, No Iterations 7
GAMG:  Solving for p, Initial residual = 0.0287092, Final residual = 0.000139891, No Iterations 3
time step continuity errors : sum local = 1.42255e-05, global = -3.17691e-06, cumulative = -9.34917e-05
GAMG:  Solving for p, Initial residual = 0.00640188, Final residual = 7.28446e-07, No Iterations 8
time step continuity errors : sum local = 7.41307e-08, global = -4.86505e-08, cumulative = -9.35404e-05
smoothSolver:  Solving for epsilon, Initial residual = 0.0101615, Final residual = 4.44932e-06, No Iterations 6
smoothSolver:  Solving for k, Initial residual = 0.0228015, Final residual = 3.84138e-06, No Iterations 7
inlet  Positive flux [m^3/s] (Area [m^2]): 0(0), Negative flux [m^3/s] (Area [m^2]): -0.000980878(0.0004)
outlet1  Positive flux [m^3/s] (Area [m^2]): 0(0), Negative flux [m^3/s] (Area [m^2]): -0.000205702(0.0004)
outlet2  Positive flux [m^3/s] (Area [m^2]): 0.00118658(0.0004), Negative flux [m^3/s] (Area [m^2]): 0(0)
ExecutionTime = 1.14 s  ClockTime = 1 s
```

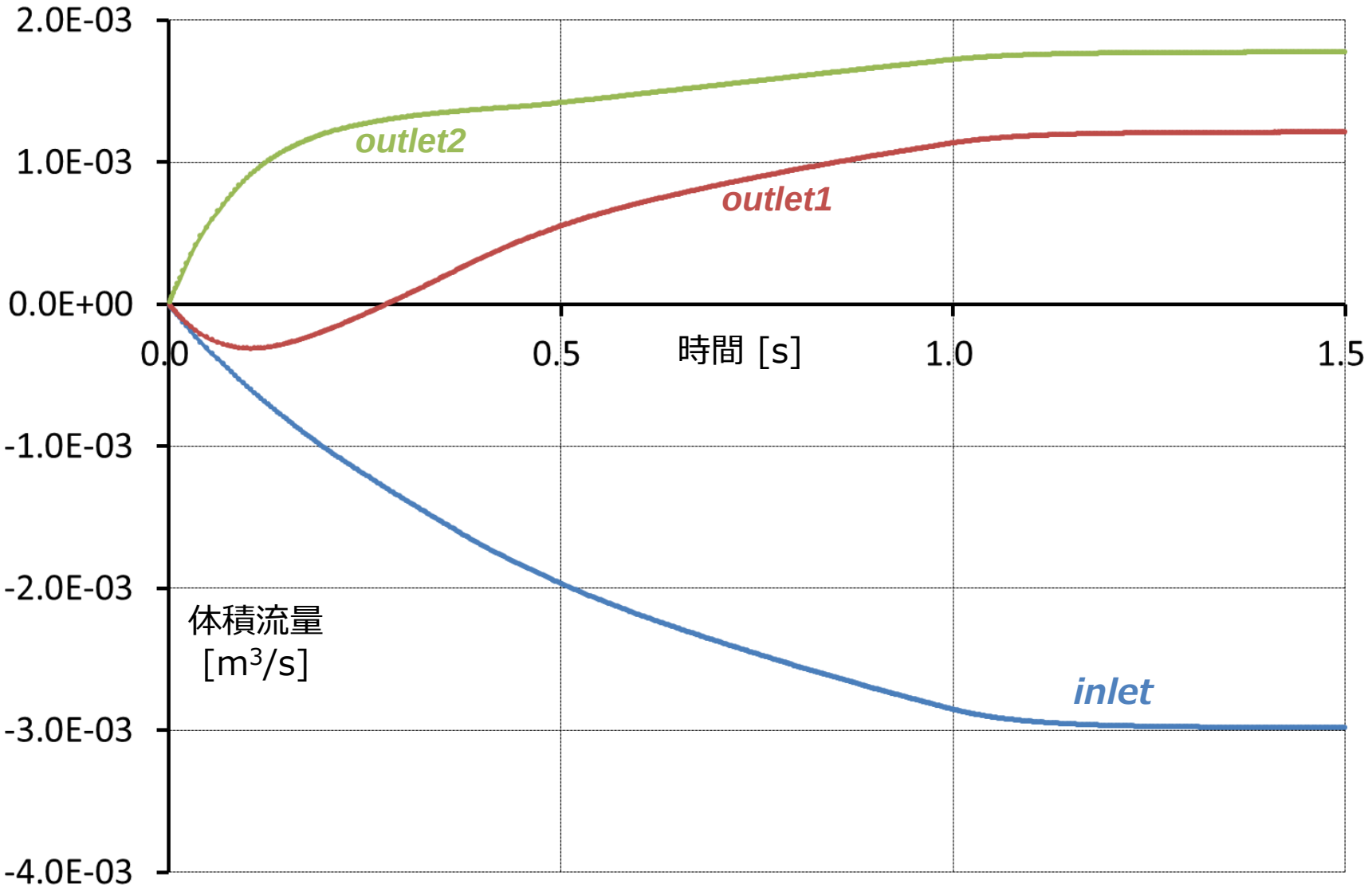
…

設定 2 の出力

- *Positive flux* が計算領域から流出する体積流量，*Negative flux* が計算領域に流入する体積流量を表しています．
- 計算初期 (Time = 0.191197 [s]) には，*outlet1* において完全な逆流が生じています．

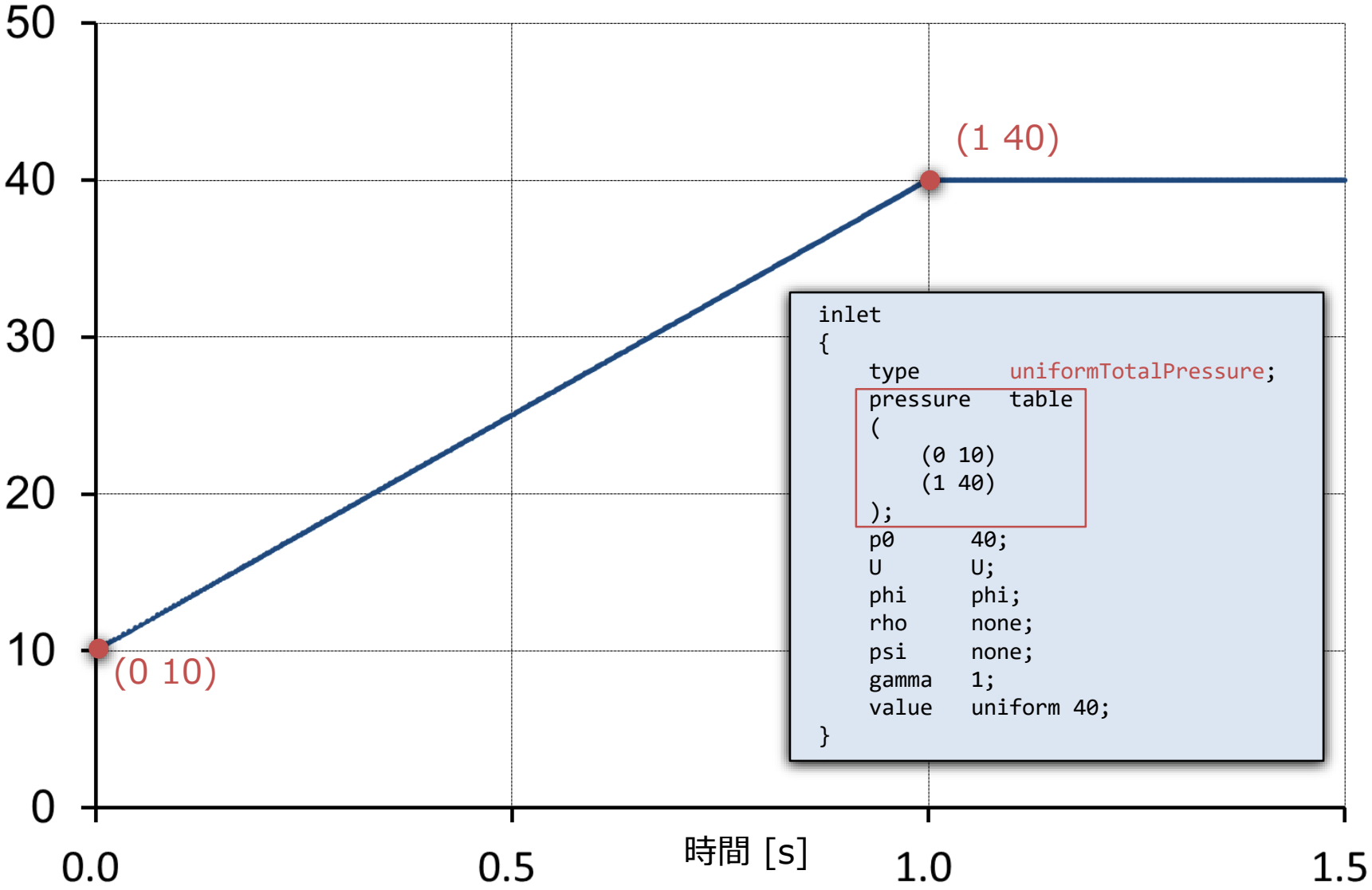
計算結果の確認

- 体積流量の時間変化



計算結果の確認

- *inlet* の全圧の時間変化




```
inlet
{
    type        uniformTotalPressure;
    pressure    table
    (
        (0 10)
        (1 40)
    );
    p0          40;
    U           U;
    phi         phi;
    rho         none;
    psi         none;
    gamma       1;
    value       uniform 40;
}
```

計算結果の確認

- 出力ファイル *Uboundary.dat* の確認

Uboundary.dat

```
Time = 0.1
----------------------------------------------------
Patch Name: inlet
BC Type: pressureInletOutletVelocity
Face Checked: localID(12), Centre(0 0 0.01)
Face Value: (1.45069 0 0)
Cell Value: (1.45069 5.78097e-06 -0.000256649)
Flux phi : -2.34818e-05
----------------------------------------------------
Patch Name: outlet1
BC Type: inletOutlet
Face Checked: localID(12), Centre(0.21 -0.21 0.01)
Face Value: (0 0 0)
Cell Value: (-5.97909e-08 0.689182 8.5515e-06)
Flux phi : -1.28301e-05
----------------------------------------------------
Patch Name: outlet2
BC Type: inletOutlet
Face Checked: localID(12), Centre(0.21 0.21 0.01)
Face Value: (0.00114319 2.35528 0.000702555)
Cell Value: (0.00114319 2.35528 0.000702555)
Flux phi : 3.75937e-05
----------------------------------------------------
Patch Name: defaultFaces
BC Type: fixedValue
Face Checked: localID(1012), Centre(0.22 0 0.01)
Face Value: (0 0 0)
Cell Value: (0.136467 1.42974 0.00129461)
Flux phi : 0
```

各境界に対して

- 境界名
- 流速 ***U*** の境界条件名
- 注目しているフェイス番号と
  その中心点座標
- 上記のフェイス中心点における ***U*** の値
- 隣接セル中心点での ***U*** の値
- フェイスの体積流量

が出力されています.

計算結果の確認

- 出力ファイル *Uboundary.dat* の確認

Uboundary.dat

```
Time = 0.1
----------------------------------------------------
Patch Name: inlet
BC Type: pressureInletOutletVelocity
Face Checked: localID(12), Centre(0 0 0.01)
Face Value: (1.45069 0 0)
Cell Value: (1.45069 5.78097e-06 -0.000256649)
Flux phi : -2.34818e-05
----------------------------------------------------
Patch Name: outlet1
BC Type: inletOutlet
Face Checked: localID(12), Centre(0.21 -0.21 0.01)
Face Value: (0 0 0)
Cell Value: (-5.97909e-08 0.689182 8.5515e-06)
Flux phi : -1.28301e-05
----------------------------------------------------
Patch Name: outlet2
BC Type: inletOutlet
Face Checked: localID(12), Centre(0.21 0.21 0.01)
Face Value: (0.00114319 2.35528 0.000702555)
Cell Value: (0.00114319 2.35528 0.000702555)
Flux phi : 3.75937e-05
----------------------------------------------------
Patch Name: defaultFaces
BC Type: fixedValue
Face Checked: localID(1012), Centre(0.22 0 0.01)
Face Value: (0 0 0)
Cell Value: (0.136467 1.42974 0.00129461)
Flux phi : 0
```

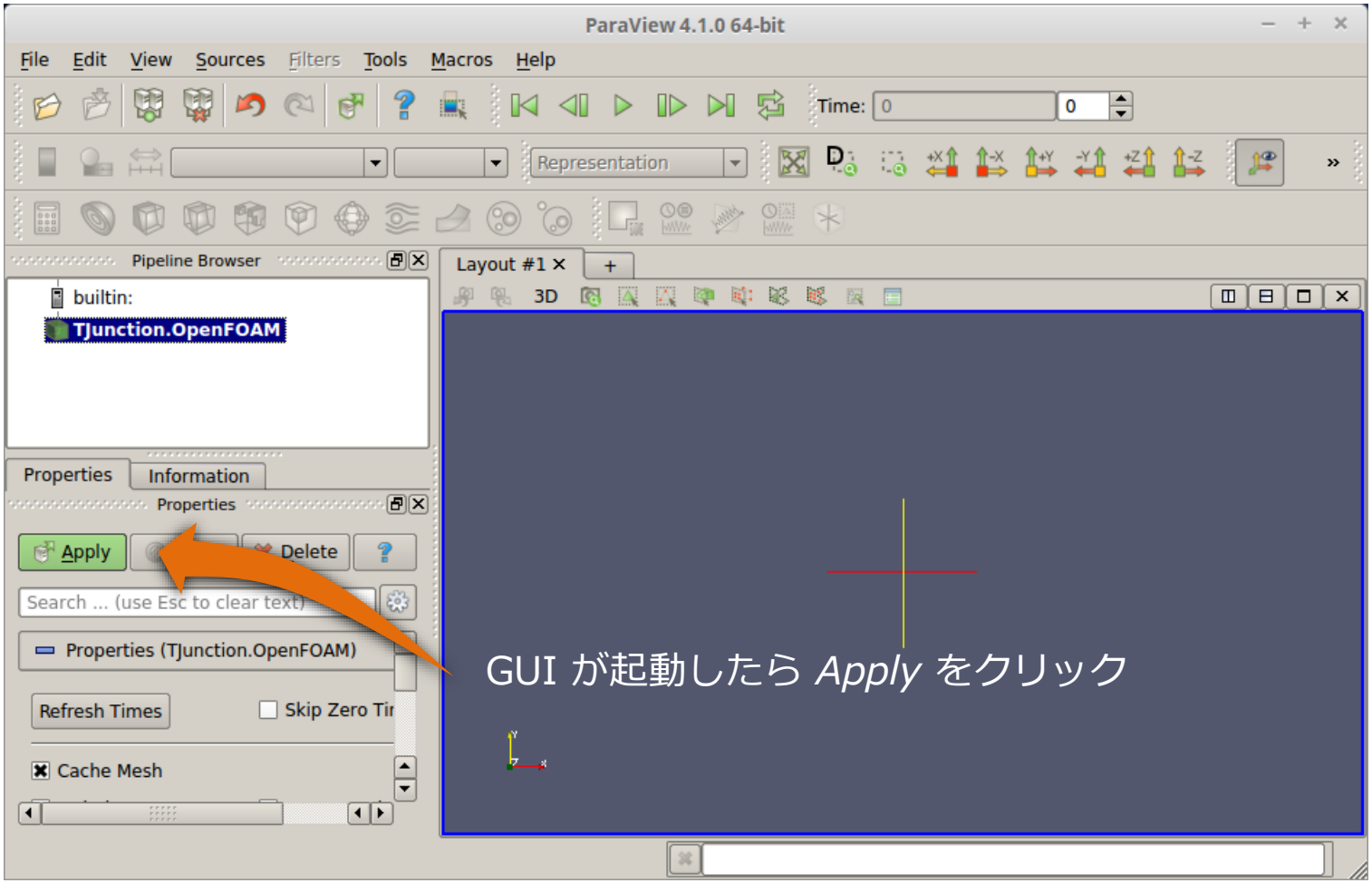
気づくことは？

計算結果の可視化

- ***ParaView*** を起動します.

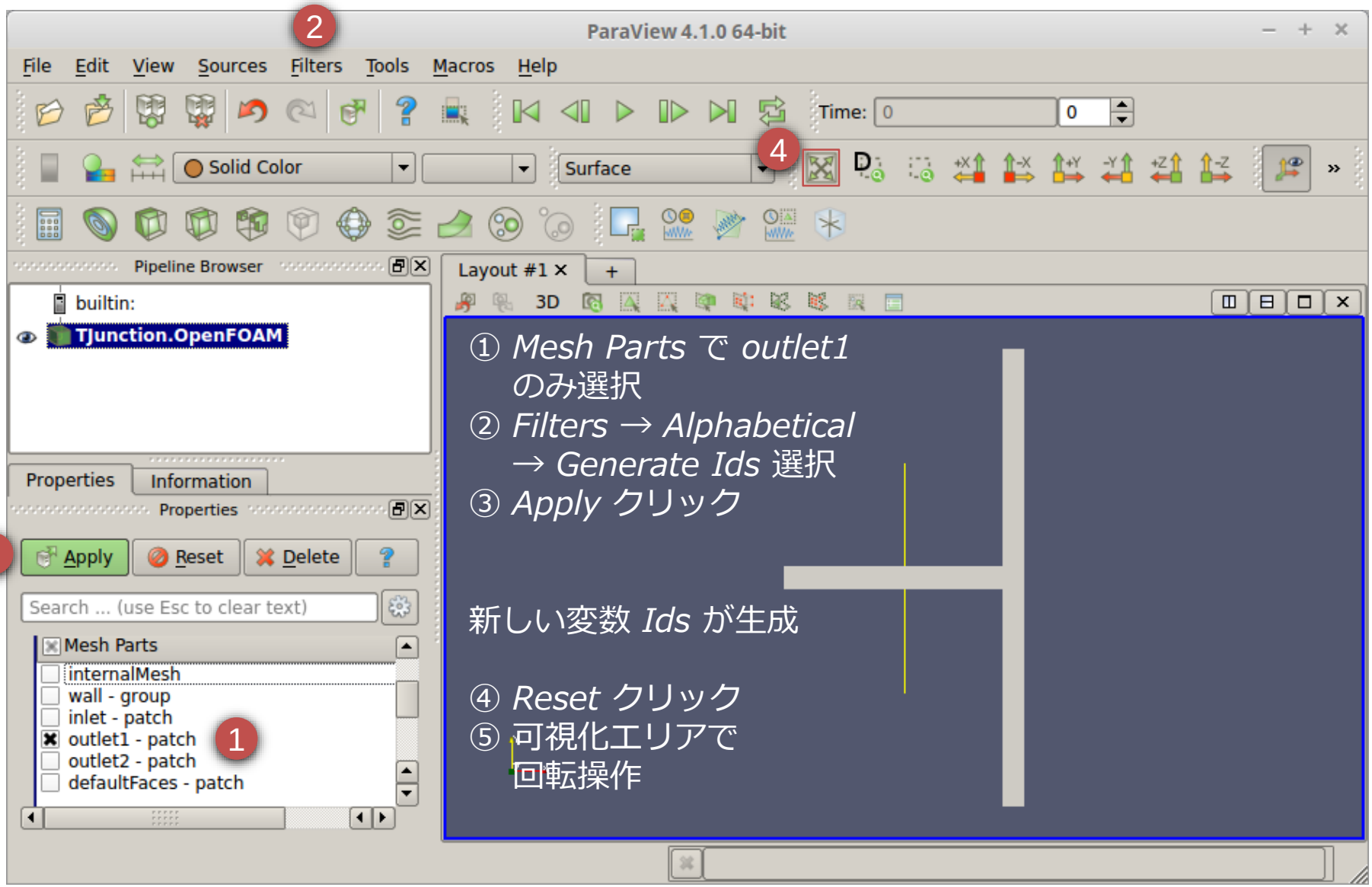
実行コマンド

```
$ paraFoam
```



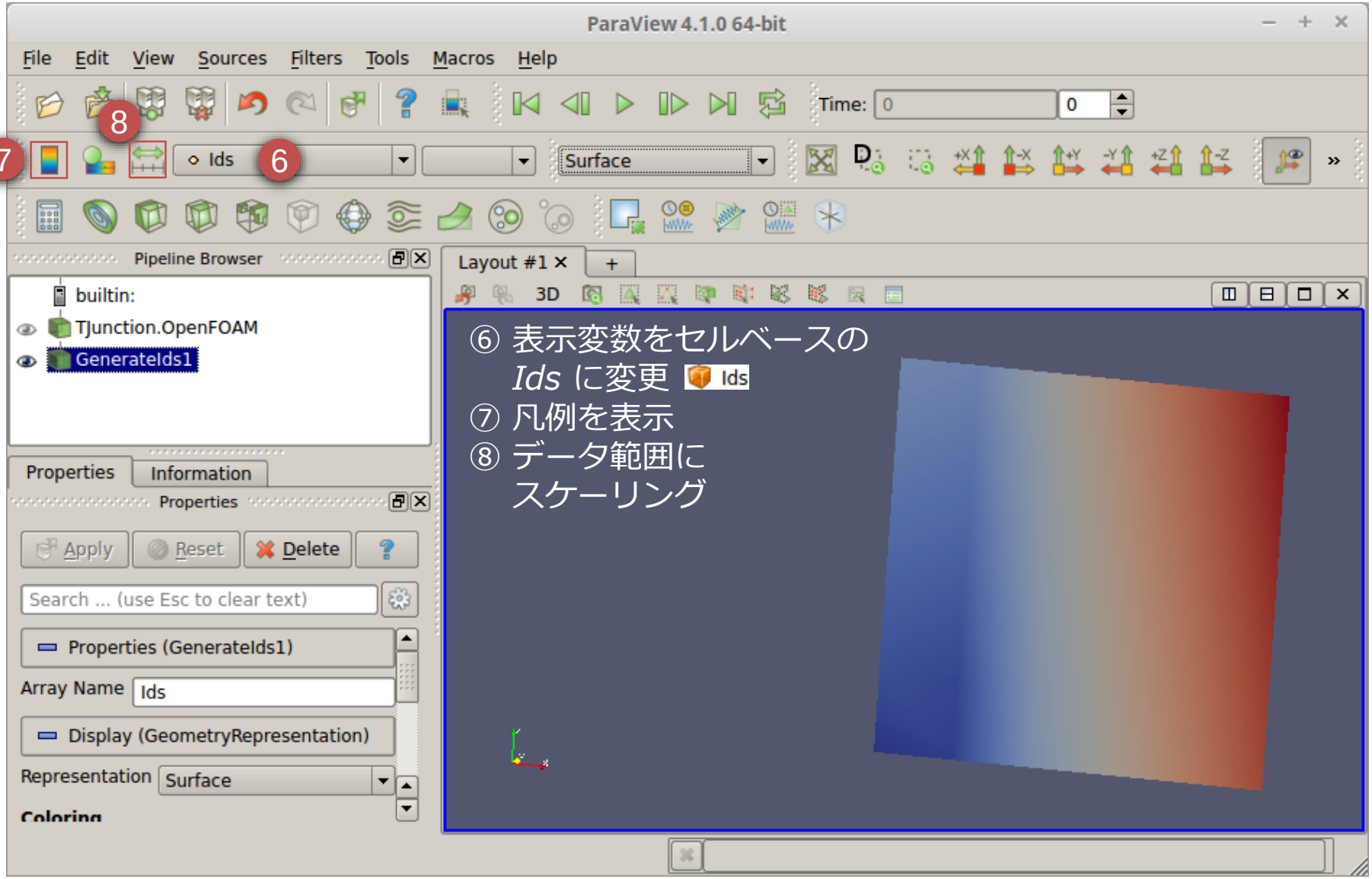
計算結果の可視化

- 境界上のフェイスの番号の確認



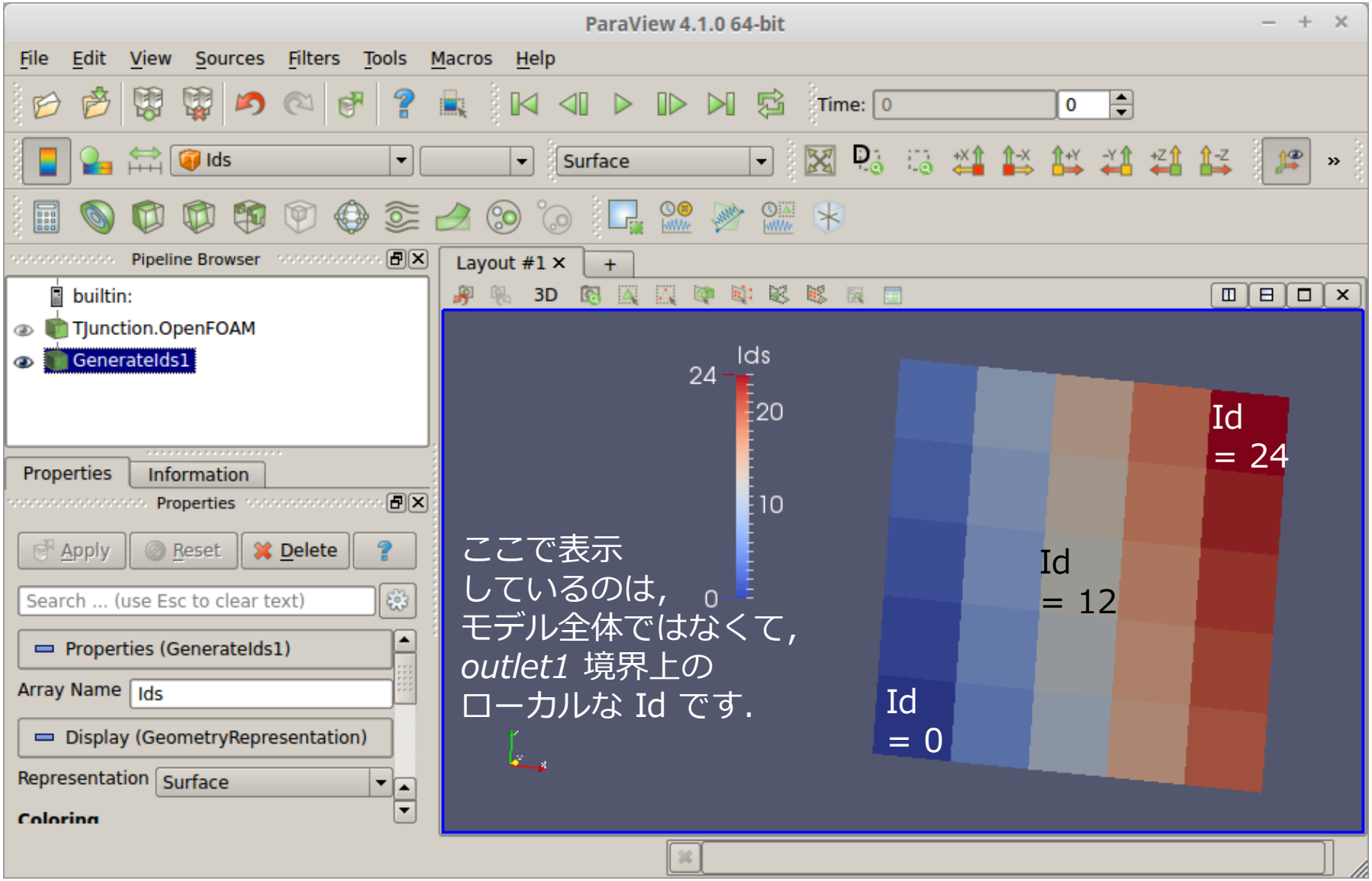
計算結果の可視化

- 境界上のフェイスの番号の確認



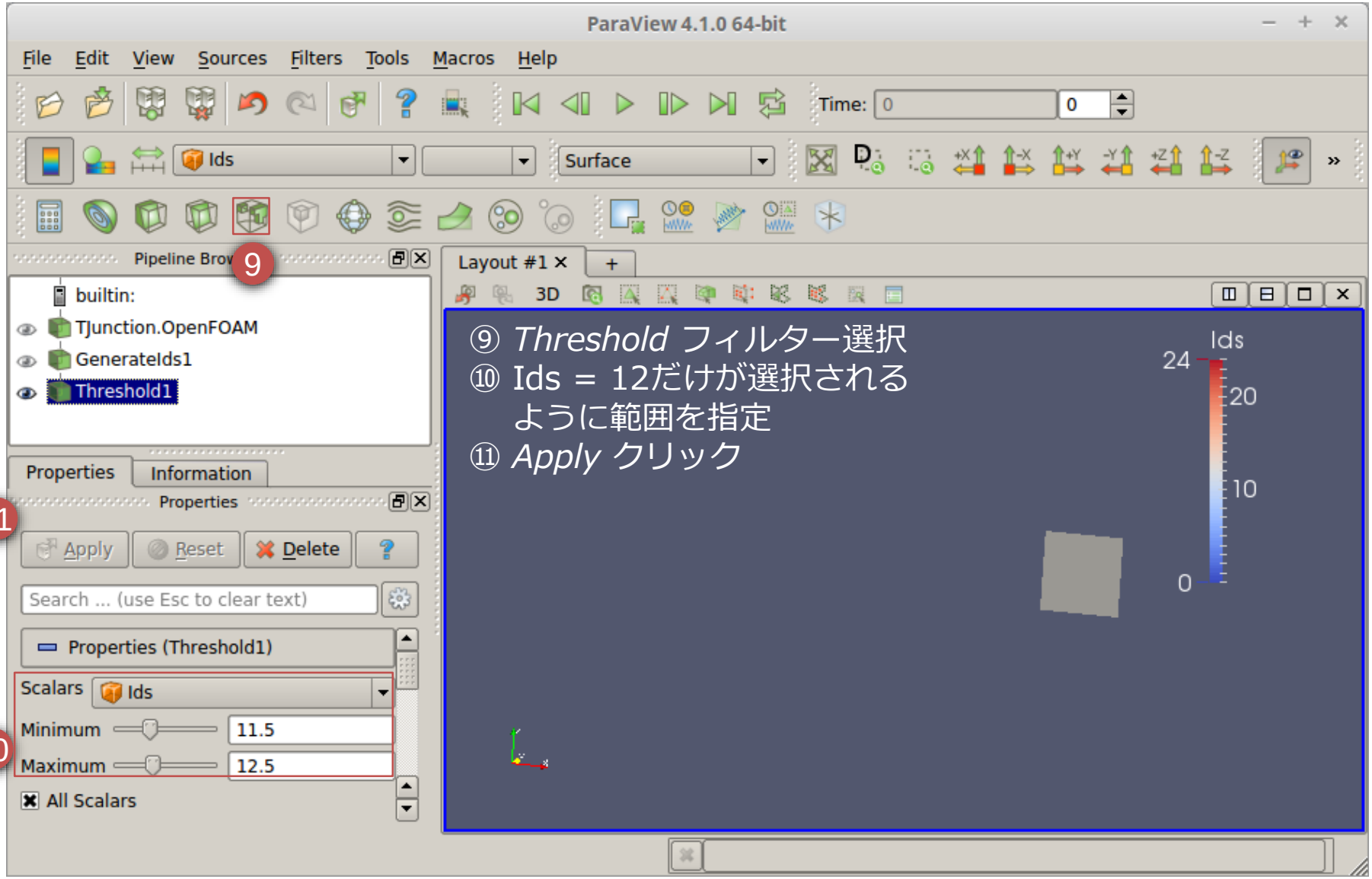
計算結果の可視化

- 境界上のフェイスの番号の確認



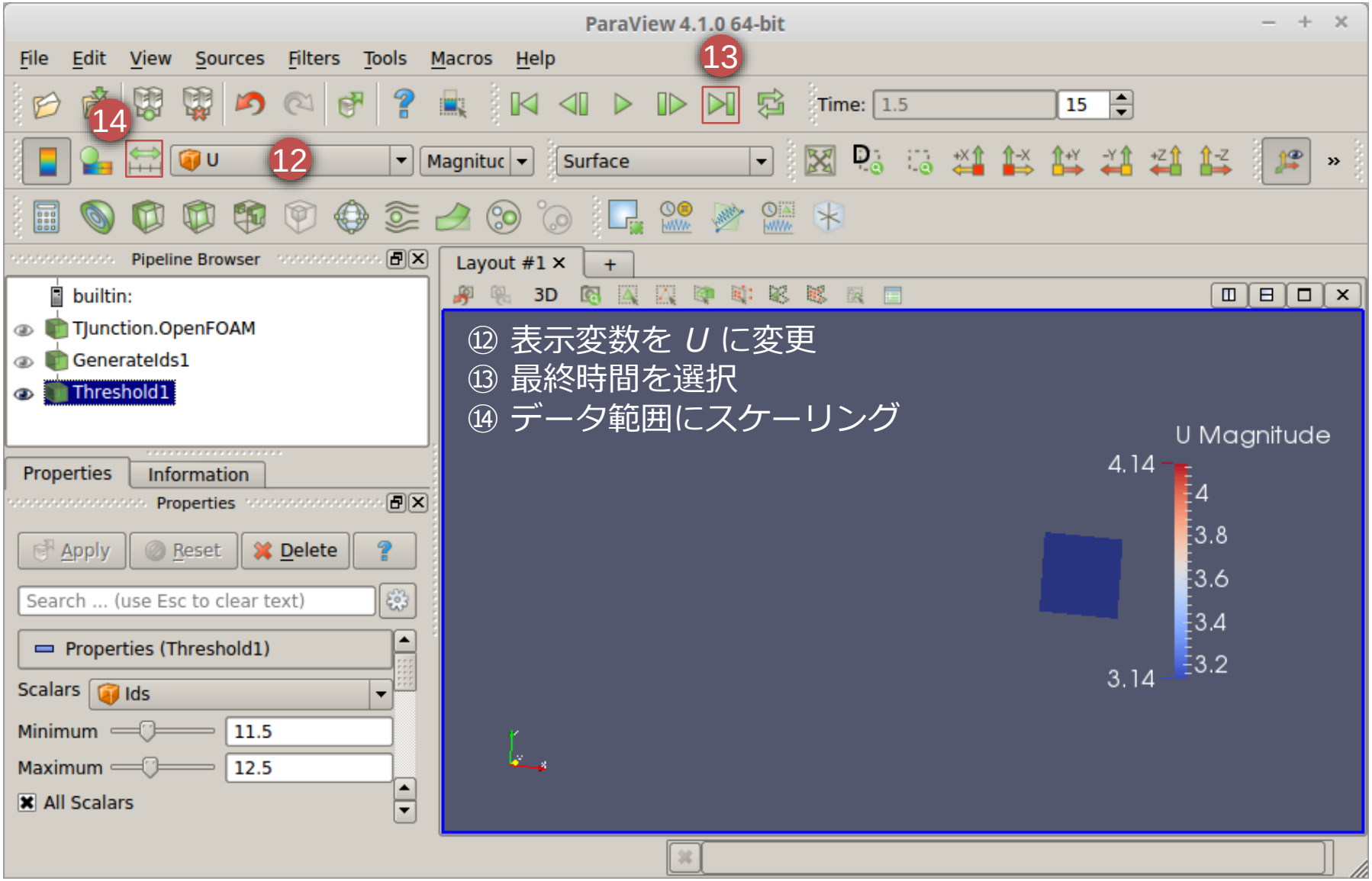
計算結果の可視化

- *outlet1* 境界上の12番目のフェイスのみ表示



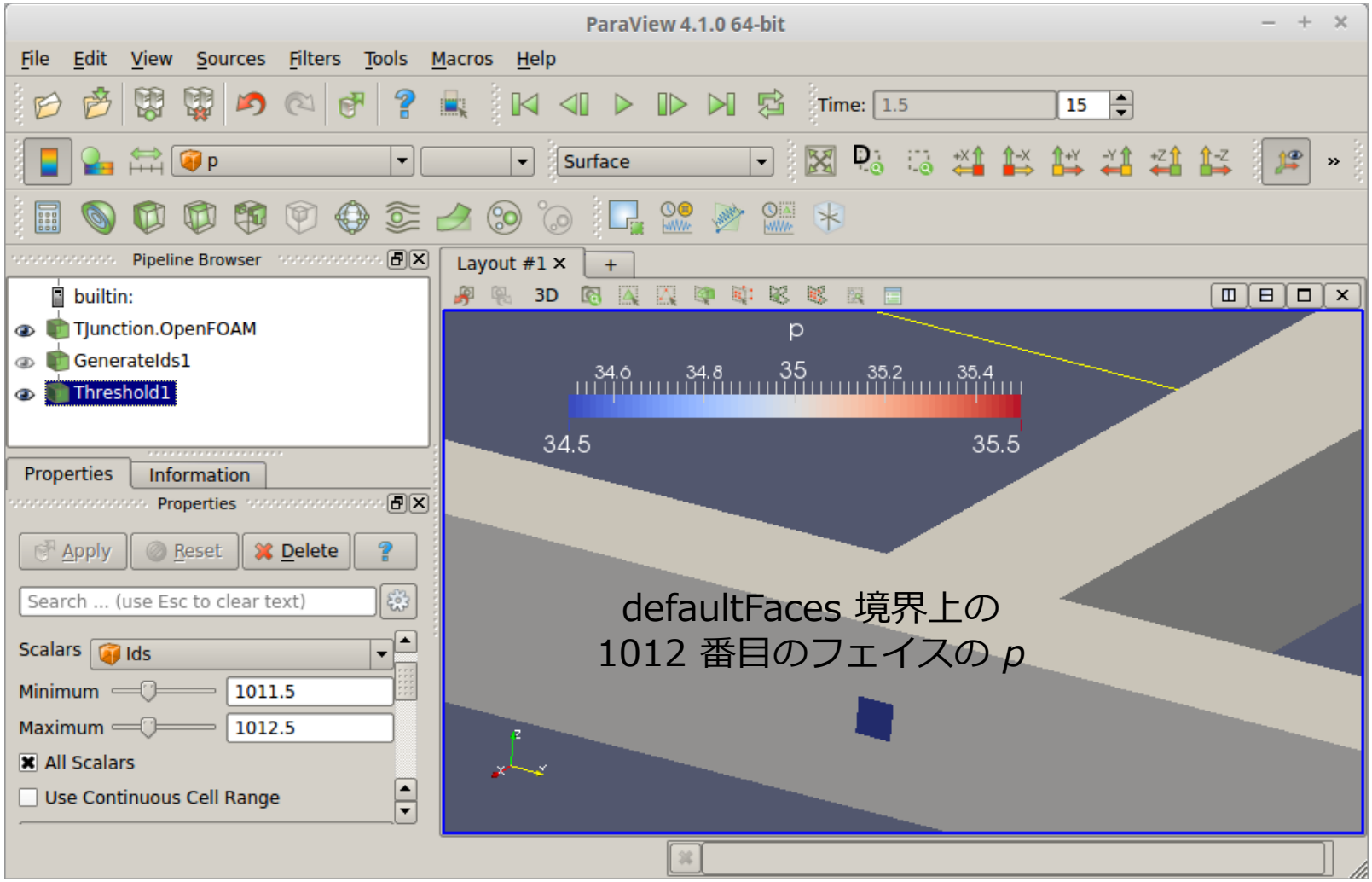
計算結果の可視化

- *outlet1* 境界上の12番目のフェイス (中心点) における流速 ***U*** の値



計算結果の可視化

- その他の境界について同様の操作を行う場合，次の２つを実施します．
  - 手順① Mesh Parts において，別の境界を選択
  - 手順⑩ 表示範囲の最小・最大値を変更



inletOutlet 条件｜ソースコードの場所

➢ ***inletOutlet*** 条件のソースコードのディレクトリを検索します.

実行コマンド

```
$ src
$ find . -name "inletOutlet*"
```

実行結果

```
(省略)
./finiteVolume/fields/fvPatchFields/derived/inletOutlet
./finiteVolume/fields/fvPatchFields/derived/inletOutlet/inletOutletFvPatchField.C
./finiteVolume/fields/fvPatchFields/derived/inletOutlet/inletOutletFvPatchField.H
./finiteVolume/fields/fvPatchFields/derived/inletOutlet/inletOutletFvPatchFields.C
./finiteVolume/fields/fvPatchFields/derived/inletOutlet/inletOutletFvPatchFields.H
./finiteVolume/fields/fvPatchFields/derived/inletOutlet/inletOutletFvPatchFields.dep
./finiteVolume/fields/fvPatchFields/derived/inletOutlet/inletOutletFvPatchFieldsFwd.H
(省略)
```

- ***inletOutlet*** 条件のソースコードの設置ディレクトリ:
  src/finiteVolume/fields/fvPatchFields/derived/inletOutlet
- 5つのファイル
  - inletOutletFvPatchField.C
  - inletOutletFvPatchField.H
  - inletOutletFvPatchFields.C
  - inletOutletFvPatchFields.H
  - inletOutletFvPatchFieldsFwd.H

inletOutlet 条件｜ヘッダーファイル

➢ ヘッダーファイル ***inletOutletFvPatchField.H*** の内容を表示して確認します.

実行コマンド

```
$ cat finiteVolume/fields/fvPatchFields/derived/inletOutlet/inletOutletFvPatchField.H
```

実行結果

```
/*---------------------------------------------------------------------------*¥
  =========                 |
  ¥¥      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   ¥¥    /   O peration     |
    ¥¥  /    A nd           | Copyright (C) 2011-2015 OpenFOAM Foundation
     ¥¥/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Class
    Foam::inletOutletFvPatchField

Group
    grpOutletBoundaryConditions
                                                    （以下省略）
```

inletOutlet 条件｜クラスの説明 Description & Note

- 冒頭の ***Description*** と ***Note*** の項に，境界条件の説明が記述されています．

inletOutletFvPatchField.H

Description

```
This boundary condition provides a generic outflow condition, with
specified inflow for the case of return flow.

¥heading Patch usage
```

説明文

```
¥table
    Property     | Description              | Required    | Default value
    phi          | Flux field name          | no          | phi
    inletValue   | Inlet value for reverse flow | yes     |
¥endtable
```

設定パラメータ

```
Example of the boundary condition specification:
¥verbatim
myPatch
{
    type            inletOutlet;
    phi             phi;
    inletValue      uniform 0;
    value           uniform 0;
}
¥endverbatim
```

設定書式

```
The mode of operation is determined by the sign of the flux across the
patch faces.
```

Note

```
Sign conventions:
- Positive flux (out of domain): apply zero-gradient condition
- Negative flux (into of domain): apply the "inletValue" fixed-value
```

inletOutlet 条件｜基本クラスの確認

- どのタイプの条件から派生しているのかを確認します．

inletOutletFvPatchField.H

```
#ifndef inletOutletFvPatchField_H
#define inletOutletFvPatchField_H

#include "mixedFvPatchField.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{

/*---------------------------------------------------------------------------*¥
                    Class inletOutletFvPatchField Declaration
¥*---------------------------------------------------------------------------*/

template<class Type>
class inletOutletFvPatchField
:
    public mixedFvPatchField<Type>
{

protected:

    // Protected data

        //- Name of flux field
        word phiName_;
```

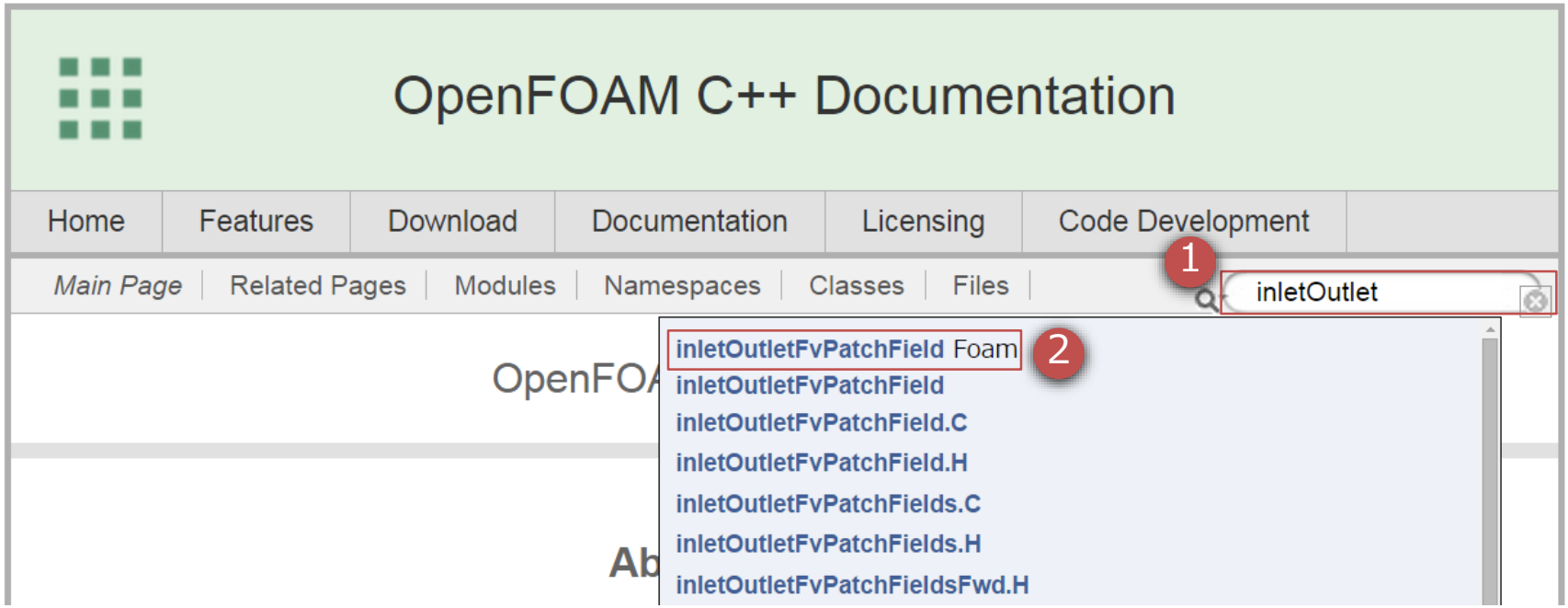
- *inletOutlet* 条件は，*mixed* 条件から派生
- テンプレートクラスを使用
  スカラーやベクトル等，複数の型に対して
  使用可能な境界条件クラスを実装

親クラスのタイプを調べることで，子のタイプを判別できます．

値を指定するタイプ，勾配を指定するタイプ，等など

Doxygen ドキュメントの閲覧

- これと同様の情報がブラウザ上でも閲覧できます.
  URL: http://www.openfoam.org/docs/cpp/



1. 検索窓に，境界条件 (クラス) 名 ***inletOutlet*** を入力します.

2. 表示された検索結果から，***inletOutletFvPatchField Foam*** を選択します.

- ***Description*** と ***Note***

Detailed Description

template<class Type>
class Foam::inletOutletFvPatchField< Type >
This boundary condition provides a generic outflow condition, with specified inflow for the case of return flow.

Patch usage

Property	Description	Required	Default value
phi	Flux field name	no	phi
inletValue	Inlet value for reverse flow	yes	

Example of the boundary condition specification:

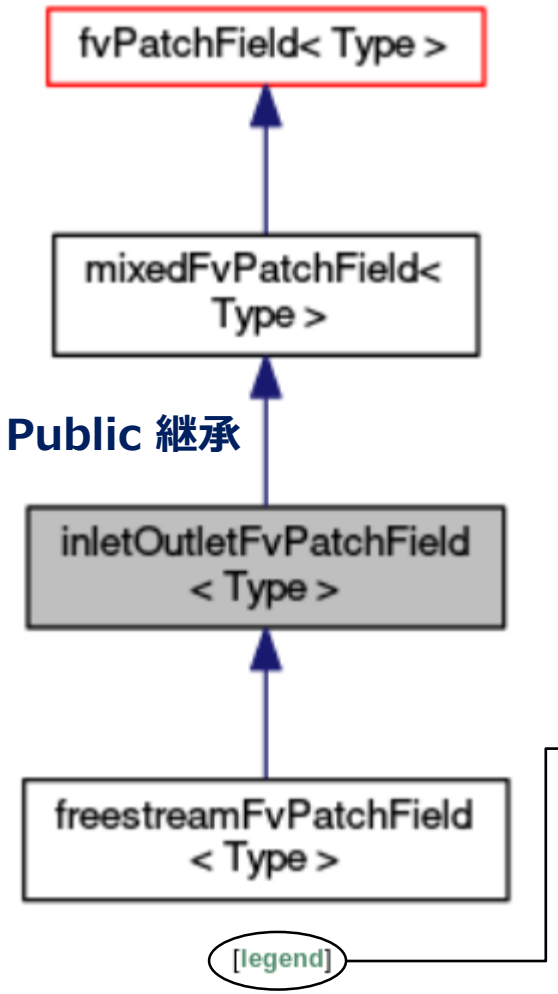
```
myPatch
{
    type            inletOutlet;
    phi             phi;
    inletValue      uniform 0;
    value           uniform 0;
}
```

The mode of operation is determined by the sign of the flux across the patch faces.
Note
Sign conventions:

- Positive flux (out of domain): apply zero-gradient condition
- Negative flux (into of domain): apply the "inletValue" fixed-value

Doxygen ドキュメントの閲覧

- クラスの継承関係



- 着目しているクラス: 灰色の四角形
- Public な継承: 濃紺の矢印

図の見方の詳細な説明

The boxes in the above graph have the following meaning:

- A filled gray box represents the struct or class for which the graph is generated.
- A box with a black border denotes a documented struct or class.
- A box with a grey border denotes an undocumented struct or class.
- A box with a red border denotes a documented struct or class forwhich not all inheritance/containment relations are shown. A graph is truncated if it does not fit within the specified boundaries.

The arrows have the following meaning:

- A dark blue arrow is used to visualize a public inheritance relation between two classes.
- A dark green arrow is used for protected inheritance.
- A dark red arrow is used for private inheritance.
- A purple dashed arrow is used if a class is contained or used by another class. The arrow is labeled with the variable(s) through which the pointed class or struct is accessible.
- A yellow dashed arrow denotes a relation between a template instance and the template class it was instantiated from. The arrow is labeled with the template parameters of the instance.

Doxygen ドキュメントの閲覧

- メンバ関数とメンバ変数のリストとその説明

Member Function Documentation

TypeName ("inletOutlet")

Runtime type information.

virtual tmp<fvPatchField<Type> > clone ( ) const inlinevirtual

Construct and return a clone.
Reimplemented from mixedFvPatchField< Type >.
Reimplemented in freestreamFvPatchField< Type >.
Definition at line 152 of file inletOutletFvPatchField.H.

virtual tmp<fvPatchField<Type> > clone (const DimensionedField< Type, volMesh > & iF) const inlinevirtual

Construct and return a clone setting internal field reference.
Reimplemented from mixedFvPatchField< Type >.
Reimplemented in freestreamFvPatchField< Type >.
Definition at line 169 of file inletOutletFvPatchField.H.

Member Data Documentation

word phiName_ protected

Name of flux field.
Definition at line 110 of file inletOutletFvPatchField.H.

inletOutlet 条件｜コンストラクタ

- 次にソースファイルを開いて，コンストラクタを確認します．

src/finiteVolume/fields/fvPatchFields/derived/inletOutlet/inletOutletFvPatchField.C

```
template<class Type>
Foam::inletOutletFvPatchField<Type>::inletOutletFvPatchField
(
    const fvPatch& p,
    const DimensionedField<Type, volMesh>& iF,
    const dictionary& dict
)
:
    mixedFvPatchField<Type>(p, iF),
    phiName_(dict.lookupOrDefault<word>("phi", "phi"))
{
    this->refValue() = Field<Type>("inletValue", dict, p.size());

    if (dict.found("value"))
    {
        fvPatchField<Type>::operator=
        (
            Field<Type>("value", dict, p.size())
        );
    }
    else
    {
        fvPatchField<Type>::operator=(this->refValue());
    }

    this->refGrad() = pTraits<Type>::zero;
    this->valueFraction() = 0.0;
}
```

コンストラクタでは，各変数の値を
- 読み込んだユーザー指定値に設定
- デフォルト値に設定

するなどして，初期化します．

親クラス ***mixed*** 条件の境界値の計算に使用される３つの変数の値を初期化

- ***refValue_*** = *inletValue* (指定値)
- ***refGrad_*** = pTraits<Type>::zero
- ***valueFraction_*** = 0.0

inletOutlet 条件｜コンストラクタ

- デフォルト値の設定

src/finiteVolume/fields/fvPatchFields/derived/inletOutlet/inletOutletFvPatchField.C

```
template<class Type>
Foam::inletOutletFvPatchField<Type>::inletOutletFvPatchField
(
    const fvPatch& p,
    const DimensionedField<Type, volMesh>& iF,
    const dictionary& dict
)
:
    mixedFvPatchField<Type>(p, iF),
    phiName_(dict.lookupOrDefault<word>("phi", "phi"))
{
    this->refValue() = Field<Type>("inletValue", dict, p.size());
```

- ***lookupOrDefault*** を使用すると，ユーザーが値を指定していない場合には，デフォルト値を使用するため，設定が必須 (***Required***) ではなくなります．
- ２つ目の引数に，デフォルト値を設定します．

```
    ¥table
        Property     | Description             | Required    | Default value
        phi          | Flux field name         | no          | phi
        inletValue   | Inlet value for reverse flow | yes    |
    ¥endtable
```

inletOutlet 条件｜境界値の計算

- メンバ関数 ***updateCoeffs*** は，***inletOutlet*** 条件の基本タイプである ***mixed*** 条件の境界値の計算に使用される変数の値を更新します．

inletOutletFvPatchField.C

```
// * * * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * * * * //

template<class Type>
void Foam::inletOutletFvPatchField<Type>::updateCoeffs()
{
    if (this->updated())
    {
        return;
    }

    const Field<scalar>& phip =
        this->patch().template lookupPatchField<surfaceScalarField, scalar>
        (
            phiName_
        );

    this->valueFraction() = 1.0 - pos(phip);

    mixedFvPatchField<Type>::updateCoeffs();
}
```

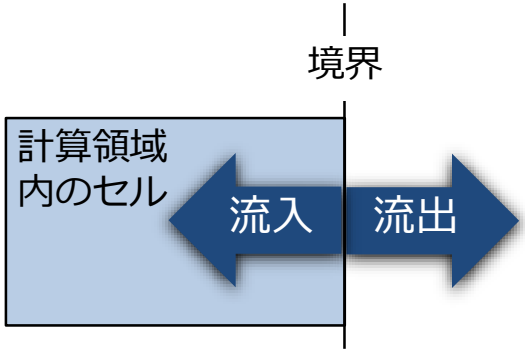
流束場 *phi* の符号に応じて ***valueFraction_*** の値を計算

- 境界値の計算は，***mixed*** 条件クラスの ***evaluate*** 関数が行います (詳細は4章).

inletOutlet 条件｜境界値の計算

- 流束 ***phi*** の符号

流れの向き	流束の符号
流入	phi < 0
流出	phi > 0



境界における法線ベクトルの向きが，計算領域の外向きに定義されているために，このような対応関係になります．

- ***pos*** 関数

src/OpenFOAM/primitives/Scalar/Scalar.H

```
inline Scalar pos(const Scalar s)
{
    return (s >= 0)? 1: 0;
}
```

$$\mathrm{pos}(s) = \begin{cases} 1 & \text{if } s \geq 0 \\ 0 & \text{if } s < 0 \end{cases}$$

以上から

- 変数 ***valueFraction_*** の値は，流れの向き (流束 *phi* の符号) に応じて変化

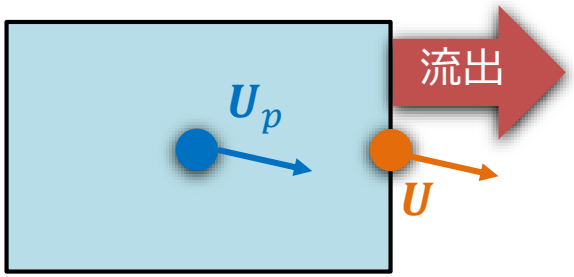
$$\boldsymbol{valueFraction\_} = \begin{cases} 0 & \text{(流出の場合)} \\ 1 & \text{(流入の場合)} \end{cases}$$

inletOutlet 条件｜境界値の計算

- ***inletOutlet*** 条件は，流束場 ***phi*** の符号から判定した流れの向きにより，下記のように場合分け処理を行います．

  - 流れが **流出** (phi > 0) の場合:

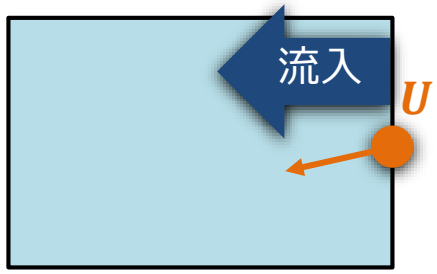
    ノイマン0 (***zeroGradient***) 条件
    境界値 $\boldsymbol{U} = \boldsymbol{U}_p$



境界隣接セル

  - 流れが **流入** (phi < 0) の場合:

    ディリクレ (***fixedValue***) 条件
    境界値 $\boldsymbol{U} = \boldsymbol{inletValue}$ (ユーザー指定値)



設定書式

```
myPatch
{
    type            inletOutlet;
    phi             phi;
    inletValue      uniform (0 0 0);
    value           uniform (0 0 0);
}
```

pressureInletOutletVelocity 条件｜ソースコードの場所

➢ ***pressureInletOutletVelocity*** 条件のソースコードの場所を検索します．

実行コマンド

```
$ src
$ find . -name "pressureInletOutletVelocity*"
```

実行結果

（一部省略）

```
./finiteVolume/fields/fvPatchFields/derived/pressureInletOutletVelocity
./finiteVolume/fields/fvPatchFields/derived/pressureInletOutletVelocity/pressureInletOutletVelocityFvPatchVectorField.C
./finiteVolume/fields/fvPatchFields/derived/pressureInletOutletVelocity/pressureInletOutletVelocityFvPatchVectorField.H
./finiteVolume/fields/fvPatchFields/derived/pressureInletOutletVelocity/pressureInletOutletVelocityFvPatchVectorField.dep
```

- ***pressureInletOutletVelocity*** 条件のソースコードの設置ディレクトリ
  src/finiteVolume/fields/fvPatchFields/derived/pressureInletOutletVelocity
- ２つのファイル
  - pressureInletOutletVelocityFvPatchVectorField.C
  - pressureInletOutletVelocityFvPatchVectorField.H

チェックポイント	***inletOutlet*** の場合とファイルの数が異なっています．

pressureInletOutletVelocity 条件｜ファイルの数の違い

- ２つの境界条件クラスの違い
  - ***inletOutletFvPatchField*** クラス:
    テンプレートクラスとして実装され，全ての変数のタイプ (スカラー，ベクトル，(対称) テンソル etc.) に対して使用可能なクラスとして設計されています．
  - ***pressureInletOutletVelocityFvPatchVectorField*** クラス:
    特定のタイプの変数 (この場合はベクトル) に対してのみ使用可能なクラスとして設計されています．
- OpenFOAM の境界条件クラスの命名規則
  - テンプレートクラス名

    境界条件名 + ***FvPatchField***

    inletOutletFvPatchField
  - 変数のタイプが特定されている境界条件クラス名

    境界条件名 + ***FvPatch*** + **変数の型** + ***Field***

    pressureInletOutletVelocityFvPatchVectorField
    inletOutletFvPatchVectorField

pressureInletOutletVelocity 条件

inletOutlet 条件の場合と同じように，ソースコードを確認していきます．

- まず，ヘッダーファイル ***pressureInletOutletVelocityFvPatchVectorField.H*** の内容を表示して確認します．

実行コマンド

```
$ cat finiteVolume/fields/fvPatchFields/derived/pressureInletOutletVelocity/¥
> pressureInletOutletVelocityFvPatchVectorField.H
```

pressureInletOutletVelocity 条件｜クラスの説明 Description & Note

- 冒頭の ***Description*** と ***Note*** の項に，境界条件の説明が記述されています．

pressureInletOutletVelocityFvPatchVectorField.H

Description

```
This velocity inlet/outlet boundary condition is applied to pressure
boundaries where the pressure is specified.  A zero-gradient condition is
applied for outflow (as defined by the flux); for inflow, the velocity is
obtained from the patch-face normal component of the internal-cell value.

The tangential patch velocity can be optionally specified.
```

説明文

```
¥heading Patch usage

¥table
    Property     | Description             | Required    | Default value
    phi          | flux field name         | no          | phi
    tangentialVelocity | tangential velocity field | no  |
¥endtable
```

設定パラメータ

```
Example of the boundary condition specification:
¥verbatim
myPatch
{
    type            pressureInletOutletVelocity;
    phi             phi;
    tangentialVelocity uniform (0 0 0);
    value           uniform 0;
}
¥endverbatim
```

設定書式

Note

```
Sign conventions:
- positive flux (out of domain): apply zero-gradient condition
- negative flux (into of domain): derive from the flux in the patch-normal
  direction
```

pressureInletOutletVelocity 条件｜基本クラスの確認

- どのタイプの条件から派生しているのかを確認します.

pressureInletOutletVelocityFvPatchVectorField.H

```
#include "fvPatchFields.H"
#include "directionMixedFvPatchFields.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{

/*---------------------------------------------------------------------------*¥
       Class pressureInletOutletVelocityFvPatchVectorField Declaration
¥*---------------------------------------------------------------------------*/

class pressureInletOutletVelocityFvPatchVectorField
:
    public directionMixedFvPatchVectorField
{
```

- ***pressureInletOutletVelocity*** 条件は，***directionMixed*** 条件から派生
- クラス名 ***pressureInletOutletVelocityFvPatchVectorField*** に ***Vector*** と付いていることから，ベクトル変数に対してのみ使用可能であることが分かります.

pressureInletOutletVelocity 条件｜コンストラクタ

➢ 次にソースファイルを開いて，コンストラクタを確認します．

pressureInletOutletVelocityFvPatchVectorField.C

```
// * * * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * * * * //
Foam::pressureInletOutletVelocityFvPatchVectorField::
pressureInletOutletVelocityFvPatchVectorField
(
    const fvPatch& p,
    const DimensionedField<vector, volMesh>& iF,
    const dictionary& dict
)
:
    directionMixedFvPatchVectorField(p, iF),
    phiName_(dict.lookupOrDefault<word>("phi", "phi"))
{
    fvPatchVectorField::operator=(vectorField("value", dict, p.size()));

    if (dict.found("tangentialVelocity"))
    {
        setTangentialVelocity
        (
            vectorField("tangentialVelocity", dict, p.size())
        );
    }
    else
    {
        refValue() = vector::zero;
    }

    refGrad() = vector::zero;
    valueFraction() = symmTensor::zero;
}
```

３つのパラメータ値を初期化しています．

- ***refValue_*** = (0, 0, 0)
- ***refGrad_*** = (0, 0, 0)
- ***valueFraction_*** = 零行列

特に，***tangentialVelocity*** パラメータが設定されている場合には，この値を使って，***refValue_*** を計算 (次ページ)

pressureInletOutletVelocity 条件

➢ メンバ関数 ***setTangentialVelocity*** を確認します.

pressureInletOutletVelocityFvPatchVectorField.C

```
// * * * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * * * //

void Foam::pressureInletOutletVelocityFvPatchVectorField::
setTangentialVelocity(const vectorField& tangentialVelocity)
{
    tangentialVelocity_ = tangentialVelocity;
    const vectorField n(patch().nf());
    refValue() = tangentialVelocity_ - n*(n & tangentialVelocity_);
}
```

- ***tangentialVelocity*** キーワードが設定されている場合には,
  この設定値 (ベクトル) の境界接線成分を計算して, ***refValue_*** の値に設定

pressureInletOutletVelocity 条件｜境界値の計算

- 境界値の計算に使用される変数の値の更新

pressureInletOutletVelocityFvPatchVectorField.C

```
void Foam::pressureInletOutletVelocityFvPatchVectorField::updateCoeffs()
{
    if (updated())
    {
        return;
    }

    const fvsPatchField<scalar>& phip =
        patch().lookupPatchField<surfaceScalarField, scalar>(phiName_);

    valueFraction() = neg(phip)*(I - sqr(patch().nf()));

    directionMixedFvPatchVectorField::updateCoeffs();
    directionMixedFvPatchVectorField::evaluate();
}
```

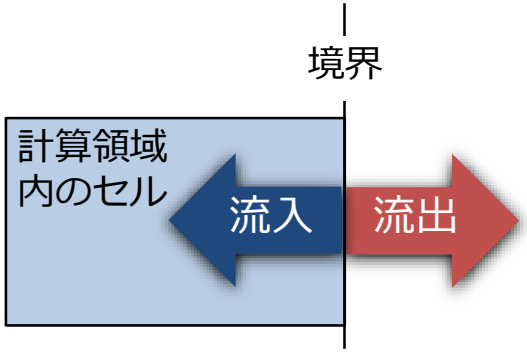
valueFraction の更新

	tangentialVelocity を設定した場合	*tangentialVelocity* を設定しない場合
refValue_	tangentialVelocity - n*(n & tangentialVelocity)	vector::zero
refGrad_	vector::zero	
valueFraction_	neg(phip)*(I - sqr(patch().nf()))	

pressureInletOutletVelocity 条件｜境界値の計算

- 流束 *phi* の符号

流れの向き	流束の符号
流入	phi < 0
流出	phi > 0



- ***neg*** 関数

src/OpenFOAM/primitives/Scalar/Scalar.H

```
inline Scalar neg(const Scalar s)
{
    return (s < 0)? 1: 0;
}
```

$$\mathrm{neg}(s) = \begin{cases} 0 & \text{if } s \geq 0 \\ 1 & \text{if } s < 0 \end{cases}$$

- 変数 ***valueFraction_*** の値は，流れの向き (流束 *phi* の符号) に応じて変化

valueFraction() = neg(phip)*(I - sqr(patch().nf()));

$$\boldsymbol{valueFraction\_} = \begin{cases} \boldsymbol{0} & \text{零行列 (流出の場合)} \\ \boldsymbol{I - n \otimes n} & \text{(流入の場合)} \end{cases}$$

directionMixed 条件｜valueFraction の役割

- どの方向にどちらの条件が課されるのかを ***valueFraction*** がコントロール

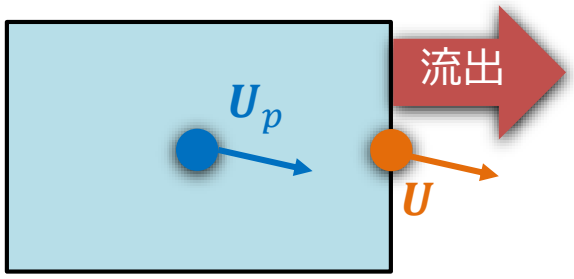
valueFraction	境界の法線方向	境界の接線方向
$\boldsymbol{n} \otimes \boldsymbol{n}$	fixedValue ディリクレ条件	fixedGradient ノイマン条件
流入の場合 $\boldsymbol{I} - \boldsymbol{n} \otimes \boldsymbol{n}$	fixedGradient ノイマン条件	fixedValue ディリクレ条件
$\boldsymbol{I}$ (単位行列)	fixedValue ディリクレ条件	
流出の場合 $\boldsymbol{O}$ (零行列)	fixedGradient ノイマン条件	

pressureInletOutletVelocity 条件｜境界値の計算

- ***pressureInletOutletVelocity*** 条件は，流束場 *phi* の符号から判定した流れの向きにより，下記のように場合分け処理を行います．

  - 流れが **流出** (*phi* > 0) の場合:

    全方向成分: ノイマン0 (***zeroGradient***) 条件
    境界値 $\boldsymbol{U} = \boldsymbol{U}_p$

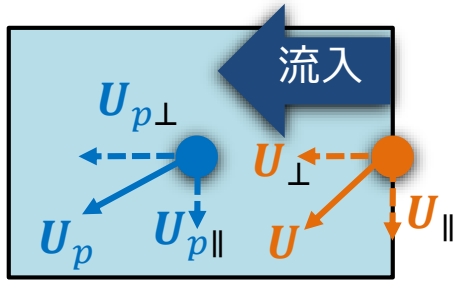


境界隣接セル

  - 流れが **流入** (*phi* < 0) の場合:

    - 法線成分: ***zeroGradient*** 条件
      $\boldsymbol{U}_\perp = \boldsymbol{U}_{p\perp}$

    - 接線成分: ディリクレ (***fixedValue***) 条件
      $\boldsymbol{U}_\parallel = \boldsymbol{tangentialVelocity}_\parallel$



境界値 $\boldsymbol{U} = \boldsymbol{U}_{p\perp} + \boldsymbol{tangentialVelocity}_\parallel$
(***tangentialVelocity*** を設定していない場合には，接線成分は零ベクトル)

同じタイプのその他の条件の検索

- ***pressureInletOutletVelocity*** 条件は，
  圧力を規定した境界において使用できる流速の境界条件

  Description
  This velocity inlet/outlet boundary condition is applied to pressure boundaries where the pressure is specified.

- この ***pressure is specified*** をキーワードにして，他に似たタイプの条件がないかどうか検索してみます．

実行コマンド

```
$ cd $FOAM_SRC/finiteVolume/fields/fvPatchFields/derived/
$ find . -name "*.H" | xargs grep -l "pressure is specified"
```

検索結果

```
./fluxCorrectedVelocity/fluxCorrectedVelocityFvPatchVectorField.H
./pressureDirectedInletOutletVelocity/pressureDirectedInletOutletVelocityFvPatchVectorField.H
./pressureDirectedInletVelocity/pressureDirectedInletVelocityFvPatchVectorField.H
./pressureInletOutletParSlipVelocity/pressureInletOutletParSlipVelocityFvPatchVectorField.H
./pressureInletOutletVelocity/pressureInletOutletVelocityFvPatchVectorField.H
./pressureInletUniformVelocity/pressureInletUniformVelocityFvPatchVectorField.H
./pressureInletVelocity/pressureInletVelocityFvPatchVectorField.H
./pressureNormalInletOutletVelocity/pressureNormalInletOutletVelocityFvPatchVectorField.H
./rotatingPressureInletOutletVelocity/rotatingPressureInletOutletVelocityFvPatchVectorField.H
```

uniformTotalPressure 条件｜｜クラスの説明 Description & Note

- 冒頭の ***Description*** と ***Note*** の項にある説明を確認します.

uniformTotalPressureFvPatchScalarField.H

```
Description
    This boundary condition provides a time-varying form of the uniform total
    pressure boundary condition.

    ¥heading Patch usage

    ¥table
        Property     | Description             | Required    | Default value
        U            | velocity field name     | no          | U
        phi          | flux field name         | no          | phi
        rho          | density field name      | no          | none
        psi          | compressibility field name | no       | none
        gamma        | ratio of specific heats (Cp/Cv) | yes |
        pressure     | total pressure as a function of time | yes |
    ¥endtable

    Example of the boundary condition specification:
    ¥verbatim
    myPatch
    {
        type            uniformTotalPressure;
        U               U;
        phi             phi;
        rho             rho;
        psi             psi;
        gamma           1.4;
        pressure        uniform 1e5;
    }
    ¥endverbatim

    The ¥c pressure entry is specified as a DataEntry type, able to describe
    time varying functions.

Note
    The default boundary behaviour is for subsonic, incompressible flow.
```

uniformTotalPressure 条件｜非定常条件のチェックポイント

- 時間変化に対応している条件かどうかの判定ポイント

uniformTotalPressureFvPatchScalarField.H

```
#ifndef uniformTotalPressureFvPatchScalarField_H
#define uniformTotalPressureFvPatchScalarField_H

#include "fixedValueFvPatchFields.H"
#include "DataEntry.H"

// * * * * * * * * * * *                                          * * //

namespace Foam
{

/*---------------------------------------------------------------------------*¥
              Class uniformTotalPressureFvPatchField Declaration
¥*---------------------------------------------------------------------------*/

class uniformTotalPressureFvPatchScalarField
:
    public fixedValueFvPatchScalarField
{
    // Private data
                    (中略)

        //- Heat capacity ratio
        scalar gamma_;

        //- Table of time vs total pressure, including the bounding treatment
        autoPtr<DataEntry<scalar> > pressure_;
```

非定常な境界条件は，
“DataEntry.H” がインクルードされている

時間的な変化を扱える変数の宣言

その他の非定常条件の検索

- ***DataEntry.H*** ファイルをインクルードしているファイルを探すことで，
  時間変化に対応したその他の条件を検索可能です．

実行コマンド

```
$ cd $FOAM_SRC/finiteVolume/fields/fvPatchFields/derived/
$ find . -name "*.H" | xargs grep -l "DataEntry.H"
```

検索結果

```
./cylindricalInletVelocity/cylindricalInletVelocityFvPatchVectorField.H
./fan/fanFvPatchField.H
./flowRateInletVelocity/flowRateInletVelocityFvPatchVectorField.H
./oscillatingFixedValue/oscillatingFixedValueFvPatchField.H
./rotatingPressureInletOutletVelocity/rotatingPressureInletOutletVelocityFvPatchVectorField.H
./rotatingTotalPressure/rotatingTotalPressureFvPatchScalarField.H
./rotatingWallVelocity/rotatingWallVelocityFvPatchVectorField.H
./swirlFlowRateInletVelocity/swirlFlowRateInletVelocityFvPatchVectorField.H
./timeVaryingMappedFixedValue/timeVaryingMappedFixedValueFvPatchField.H
./uniformFixedGradient/uniformFixedGradientFvPatchField.H
./uniformFixedValue/uniformFixedValueFvPatchField.H
./uniformInletOutlet/uniformInletOutletFvPatchField.H
./uniformJump/uniformJumpFvPatchField.H
./uniformJumpAMI/uniformJumpAMIFvPatchField.H
./uniformTotalPressure/uniformTotalPressureFvPatchScalarField.H
```

名前に ***uniform*** が付いている条件は，非定常に対応しています．

非定常条件の例

- ***uniformFixedValue*** 条件
  - 時間的に境界値 (***uniformValue***) が変化するディリクレ条件
  - ***fixedValue*** 条件の非定常版

- ***uniformFixedGradient*** 条件
  - 時間的に境界の法線方向勾配値 (***uniformGradient***) が変化するノイマン条件
  - ***fixedGradient*** 条件の非定常版

- ***rotatingWallVelocity***
  時間的に回転速度 (***omega***) が変化する回転壁速度条件

- ***flowRateInletVelocity***
  時間的に体積流量 (***volumetricFlowRate***) or 質量流量 (***massFlowRate***) が変化する流入速度条件

- ***uniformTotalPressure***
  時間的に全圧 (***pressure***) が変化する圧力境界条件

  など

第３章　pipeCyclic チュートリアル

この章では，チュートリアル *pipeCyclic* に取り組みます．
この章のキーワードは，次の２つです．

- ***codedFixed*** 条件
- 周期境界条件

pipeCyclic チュートリアルの実行

- ***pipeCyclic*** チュートリアルのディレクトリに移動して，計算格子を生成します．

実行コマンド

```
$ run
$ cd tutorials/incompressible/simpleFoam/pipeCyclic
$ ls
```

計算実行前のディレクトリ構成

```
0.org  Allclean  Allrun  constant  system
```

- ***Allrun*** スクリプトを実行して，計算格子の生成および計算を実行します．

実行コマンド

```
$ ./Allrun
```

- 計算が終了したら，ls コマンドでディレクトリの構成を調べます．

計算実行後のディレクトリ構成

```
0        239         dynamicCode           log.refineHexMesh  processor1  system
0.org    Allclean    log.blockMesh         log.simpleFoam     processor2
100      Allrun      log.decomposePar      log.topoSet        processor3
200      constant    log.reconstructPar    processor0         processor4
```

- ***dynamicCode*** という名前のディレクトリが作成されているのが確認できます．

patchSummary ユーティリティで境界条件の確認

- ***patchSummary*** ユーティリティを実行して，このチュートリアルに使用されている境界条件のリストを表示します．

実行コマンド

```
$ patchSummary
```

実行結果

```
Time = 239

Valid fields:
    volScalarField      nut
    volVectorField      U
    volScalarField      k
    volScalarField      p
    volScalarField      epsilon

patch       : inlet
    scalar              nut             calculated
    scalar              k               turbulentIntensityKineticEnergyInlet
    scalar              p               zeroGradient
    scalar              epsilon         turbulentMixingLengthDissipationRateInlet
    vector              U               codedFixedValue
```

チェックポイント 1
codedFixedValue 条件

次ページに続く

patchSummary ユーティリティで境界条件の確認

実行結果

```
patch       : outlet
    scalar              nut                 calculated
    scalar              k                   inletOutlet
    scalar              p                   fixedValue
    scalar              epsilon             inletOutlet
    vector              U                   inletOutlet

group       : cyclicAMI
    scalar              nut                 cyclicAMI
    scalar              k                   cyclicAMI
    scalar              p                   cyclicAMI
    scalar              epsilon             cyclicAMI
    vector              U                   cyclicAMI

wall        : walls
    scalar              nut                 nutkWallFunction
    scalar              k                   kqRWallFunction
    scalar              p                   zeroGradient
    scalar              epsilon             epsilonWallFunction
    vector              U                   fixedValue

End
```

チェックポイント2
周期境界条件: cyclicAMI

patchSummary ユーティリティ

各変数に対して使用されている境界条件のリストを境界ごとに表示します.

codedFixedValue 条件の設定

- まず，***codedFixedValue*** 条件から確認します．
- ***0/U*** ファイルを開いて，***codedFixedValue*** 条件の設定を確認

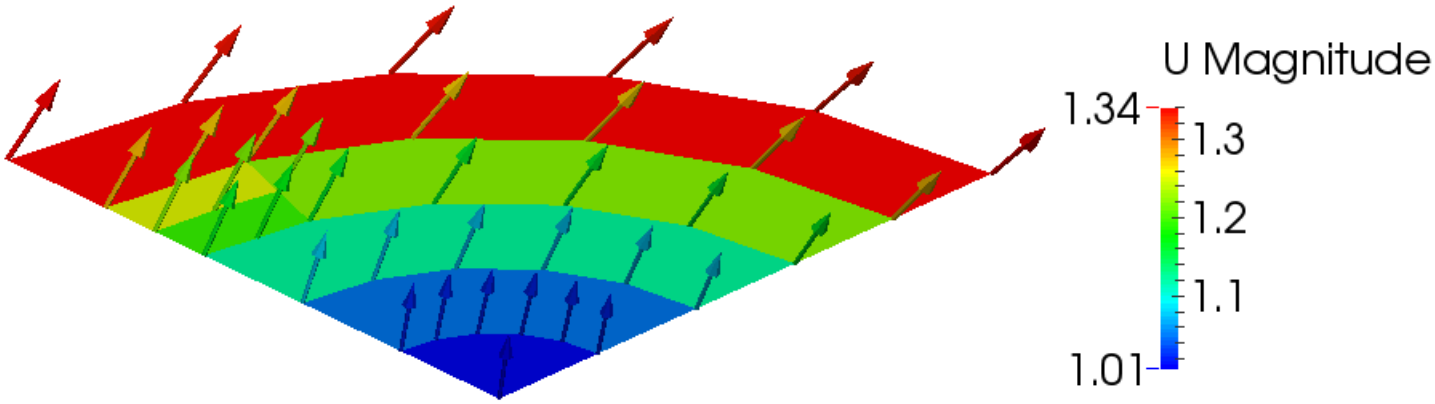
0/U

```
inlet
{
    type            codedFixedValue;
    redirectType    swirl;

    code
    #{
        const vector axis(1, 0, 0);

        vectorField v(2.0*this->patch().Cf() ^ axis);
        v.replace(vector::X, 1.0);
        operator==(v);
    #};
    value           $internalField;
}
```

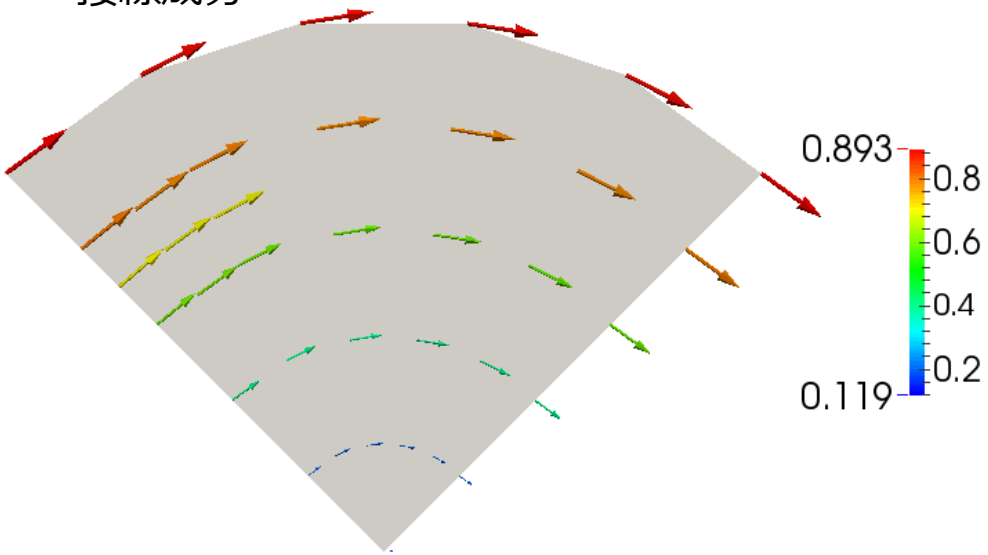
code セクション

- ソースファイルに記述されているようなコードが境界条件の設定に使用されているのが確認できます．

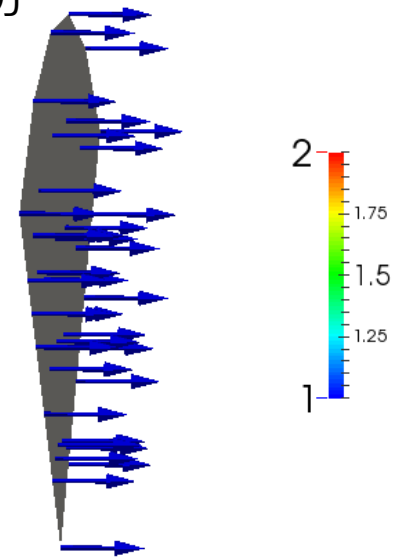
inlet の流速分布



接線成分



法線成分



codedFixedValue 条件について

- ***codedFixedValue*** 条件は，値を規定するディリクレ境界条件 (***fixedValue***) の場合には，その値の計算方法をコーディングすることが可能です．

- 計算終了後に存在を確認した ***dynamicCode*** という名前のディレクトリは，この ***codedFixedValue*** 条件を使用したことで作成されたものです．

dynamicCode ディレクトリの構成

```
├── _02cb04a62281a4e8f2d8aa84cec1115d4ee1650f
├── _2c1a7ee7b5369acae9da148f9823fdd8cde6256a
├── _91066c16201c6403a4b0f7d4a7edb347e7d99032
├── _9273df81ebe52476e105ce9128b702a57ad95417
├── platforms
│   └── linux64GccDPOpt
└── swirl
    ├── Make
    ├── fixedValueFvPatchFieldTemplate.C
    ├── fixedValueFvPatchFieldTemplate.H
    ├── fixedValueFvPatchFieldTemplate.dep
    └── lnInclude
```

- 赤枠で囲んだ部分が，***codedFixedValue*** 境界条件に関係しています．

codedFixedValue 条件について

- まず，ヘッダーファイルから確認します．

実行コマンド

```
$ cat dynamicCode/swirl/fixedValueFvPatchFieldTemplate.H
```

dynamicCode/swirl/fixedValueFvPatchFieldTemplate.H

```
/*---------------------------------------------------------------------------*¥
                        A templated FixedValueFvPatch
¥*---------------------------------------------------------------------------*/

class swirlFixedValueFvPatchVectorField
:
    public fixedValueFvPatchField<vector>
{
public:

    //- Information about the SHA1 of the code itself
    static const char* const SHA1sum;

    //- Runtime type information
    TypeName("swirl");
```

fixedValue 条件のクラスから派生

境界条件名は，***redirectType***

- 境界条件クラスの名前は，変数の型と ***redirectType*** に設定した名前から決まります．

redirectType + ***FixedValueFvPatch*** + **変数の型** + ***Field***

codedFixedValue 条件について

- 次にソースファイルを確認します.

実行コマンド

```
$ cat dynamicCode/swirl/fixedValueFvPatchFieldTemplate.C
```

dynamicCode/swirl/fixedValueFvPatchFieldTemplate.C

```
void swirlFixedValueFvPatchVectorField::updateCoeffs()
{
    if (this->updated())
    {
        return;
    }

    if (false)
    {
        Info<<"updateCoeffs swirl sha1: 3ff7f7c0f3ae27d7889404869a7b49e22adab2a7¥n";
    }

//{{{ begin code
    #line 33 "/opt/OpenFOAM/custom-
2.4.x/run/tutorials/incompressible/simpleFoam/pipeCyclic/0/U.boundaryField.inlet"
const vector axis(1, 0, 0);

            vectorField v(2.0*this->patch().Cf() ^ axis);
            v.replace(vector::X, 1.0);
            operator==(v);
//}}} end code

    this->fixedValueFvPatchField<vector>::updateCoeffs();
}
```

code セクションに記述した
コードがここに使用されます.

codedFixedValue 条件について

- このソースファイルのコンパイルの設定ファイル

dynamicCode/swirl/Make/files

```
/* dynamicCode:
 * SHA1 = 3ff7f7c0f3ae27d7889404869a7b49e22adab2a7
 */
fixedValueFvPatchFieldTemplate.C

LIB = $(PWD)/../platforms/$(WM_OPTIONS)/lib/libswirl_3ff7f7c0f3ae27d7889404869a7b49e22adab2a7
```

dynamicCode/swirl/Make/options

```
/* dynamicCode:
 * SHA1 = 3ff7f7c0f3ae27d7889404869a7b49e22adab2a7
 */
EXE_INC = -g ¥
-I$(LIB_SRC)/finiteVolume/lnInclude ¥


LIB_LIBS = ¥
    -lOpenFOAM ¥
    -lfiniteVolume ¥
```

- 計算実行時に自動でコンパイルして作成されたライブラリは，***Make/files*** に指定されたディレクトリ dynamicCode/platforms/linux64GccDPOpt/lib/ に作成されます．

- ライブラリの名前は，重複しないように自動で命名されます．

codedFixedValue 条件まとめ

- ***codedFixedValue*** 条件を使用すると,

  - ***code*** セクションに記述したコードを ***updateCoeffs***() に持ち
  - ***fixedValueFvPatchField<type>*** クラスから派生した

  境界条件クラスのライブラリがソルバー実行時に自動で作成されます.

  - ソースコードの格納ディレクトリ: ***dynamicCode/redirectType***
  - ライブラリの格納ディレクトリ: ***dynamicCode/platforms***

- ここで，***type*** は，この境界条件を使用した変数の型になります.
  この例では，流速 ***U*** というベクトル変数に使用したので，***type*** は ***vector***

- 境界条件クラスの名前は，変数の型と ***redirectType*** に設定した名前から決まります.

- 全ての基本タイプ (***fixedValue***，***fixedGradient***，***mixed*** など) について,
  実行時コードコンパイルを使用できる境界条件が実装されてはいません.

- 現状では，***fixedValue*** と ***mixed*** 条件について対応する境界条件が利用でき,
  境界条件の名前は，それぞれ，***codedFixedValue***，***codedMixed*** です.

dynamicCodeディレクトリの補足説明

- ***dynamicCode*** ディレクトリの中を見ると，***dynamicCode*** と ***platforms*** 以外にもディレクトリが作成されているのが確認できます．

dynamicCode ディレクトリの構成

```
├── _02cb04a62281a4e8f2d8aa84cec1115d4ee1650f
├── _2c1a7ee7b5369acae9da148f9823fdd8cde6256a
├── _91066c16201c6403a4b0f7d4a7edb347e7d99032
├── _9273df81ebe52476e105ce9128b702a57ad95417
├── platforms
│   └── linux64GccDPOpt
└── swirl
    ├── Make
    ├── fixedValueFvPatchFieldTemplate.C
    ├── fixedValueFvPatchFieldTemplate.H
    ├── fixedValueFvPatchFieldTemplate.dep
    └── lnInclude
```

- これらは，***blockMeshDict*** でも実行時コードコンパイルの機能（***#codeStream***）を使用しているためです．

constant/polyMesh/blockMeshDict

```
radHalfAngle    #calc "degToRad($halfAngle)";
y               #calc "$radius*sin($radHalfAngle)";
minY            #calc "-1.0*$y";
z               #calc "$radius*cos($radHalfAngle)";
minZ            #calc "-1.0*$z";
```

codedFixedValue 条件を使用する際の注意点

- 実行時コードコンパイルの機能を使用する場合には，***$WM_PROJECT_DIR/etc/controlDict*** ファイル内の ***allowSystemOperations*** の項目の値を 1 に設定する必要があります．

$WM_PROJECT_DIR/etc/controlDict

```
InfoSwitches
{
    writePrecision  6;
    writeJobInfo    0;
    writeDictionaries 0;
    writeOptionalEntries 0;

    // Allow case-supplied C++ code (#codeStream, codedFixedValue)
    allowSystemOperations   1;
}
```

- DEXCS OpenFOAM では，この値がデフォルトで 1 に設定してあります．

ディリクレタイプの流速境界条件

- ***codedFixedValue*** の設定で参考にできるコードを探索します．

- 流速を規定する境界条件は，***fixedValueFvPatchVectorField*** クラスを継承しているので，次のコマンドで検索可能です．

実行コマンド

```
$ cd $FOAM_SRC/finiteVolume/fields/fvPatchFields/derived/
$ find . -name "*.H" | xargs grep -l 'public fixedValueFvPatchVectorField'
```

検索結果

```
./activeBaffleVelocity/activeBaffleVelocityFvPatchVectorField.H
./activePressureForceBaffleVelocity/activePressureForceBaffleVelocityFvPatchVectorField.H
./cylindricalInletVelocity/cylindricalInletVelocityFvPatchVectorField.H
./flowRateInletVelocity/flowRateInletVelocityFvPatchVectorField.H
./interstitialInletVelocity/interstitialInletVelocityFvPatchVectorField.H
./mappedFlowRate/mappedFlowRateFvPatchVectorField.H
./mappedVelocityFluxFixedValue/mappedVelocityFluxFixedValueFvPatchField.H
./movingWallVelocity/movingWallVelocityFvPatchVectorField.H
./pressureDirectedInletVelocity/pressureDirectedInletVelocityFvPatchVectorField.H
./pressureInletVelocity/pressureInletVelocityFvPatchVectorField.H
./rotatingWallVelocity/rotatingWallVelocityFvPatchVectorField.H
./surfaceNormalFixedValue/surfaceNormalFixedValueFvPatchVectorField.H
./swirlFlowRateInletVelocity/swirlFlowRateInletVelocityFvPatchVectorField.H
./translatingWallVelocity/translatingWallVelocityFvPatchVectorField.H
./variableHeightFlowRateInletVelocity/variableHeightFlowRateInletVelocityFvPatchVectorField.H
```

- これらのソースファイルに記述されている ***updateCoeffs***() を参考にできます．

cylindricalInletVelocity 条件クラスの uppdateCoeffs

derived/cylindricalInletVelocity/cylindricalInletVelocityFvPatchVectorField.C

```
void Foam::cylindricalInletVelocityFvPatchVectorField::updateCoeffs()
{
    if (updated())
    {
        return;
    }

    const scalar t = this->db().time().timeOutputValue();
    const scalar axialVelocity = axialVelocity_->value(t);
    const scalar radialVelocity = radialVelocity_->value(t);
    const scalar rpm = rpm_->value(t);

    vector hatAxis = axis_/mag(axis_);

    const vectorField r(patch().Cf() - centre_);
    const vectorField d(r - (hatAxis & r)*hatAxis);

    tmp<vectorField> tangVel
    (
        (rpm*constant::mathematical::pi/30.0)*(hatAxis) ^ d
    );

    operator==(tangVel + hatAxis*axialVelocity + radialVelocity*d/mag(d));

    fixedValueFvPatchField<vector>::updateCoeffs();
}
```

時間依存のパラメータ値の更新

patch().Cf()
境界フェイスの中心座標

円周率πの使用

codedFixedValue 条件の補足情報

- ***codedFixedValue*** 条件は，パラメータの値をハードコーディングします．
- 下記のサイトに ***codedFixedValue*** 条件により作成されたソースコードを編集して，パラメータ (メンバ変数) を追加する手順が紹介されています．

境界条件の作成

2014年9月20日

はじめに

境界条件の作成について。

使用バージョン

OpenFOAM 2.3.0

ステップ

ここでは、次のステップで境界条件の作成方法の習得を進める。

1. 実行時コードコンパイルによる実装
2. 実行時コードコンパイルにより作成されたコードを利用した実装
3. 既存のコードをベースにした実装

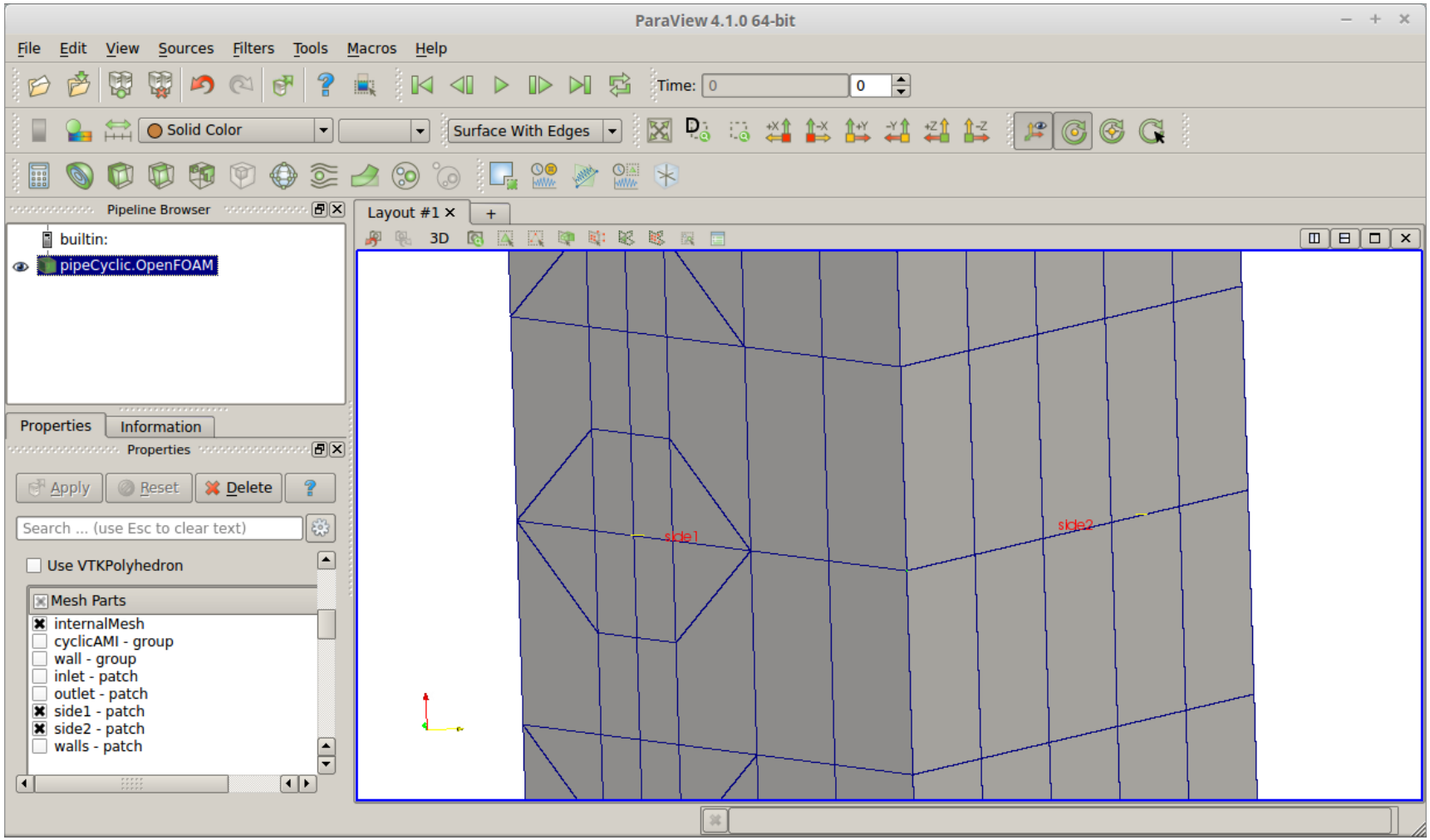
http://www.geocities.jp/penguinitis2002/study/OpenFOAM/create_bc.html

チュートリアルのモデルの確認

- ***ParaView*** を起動します.

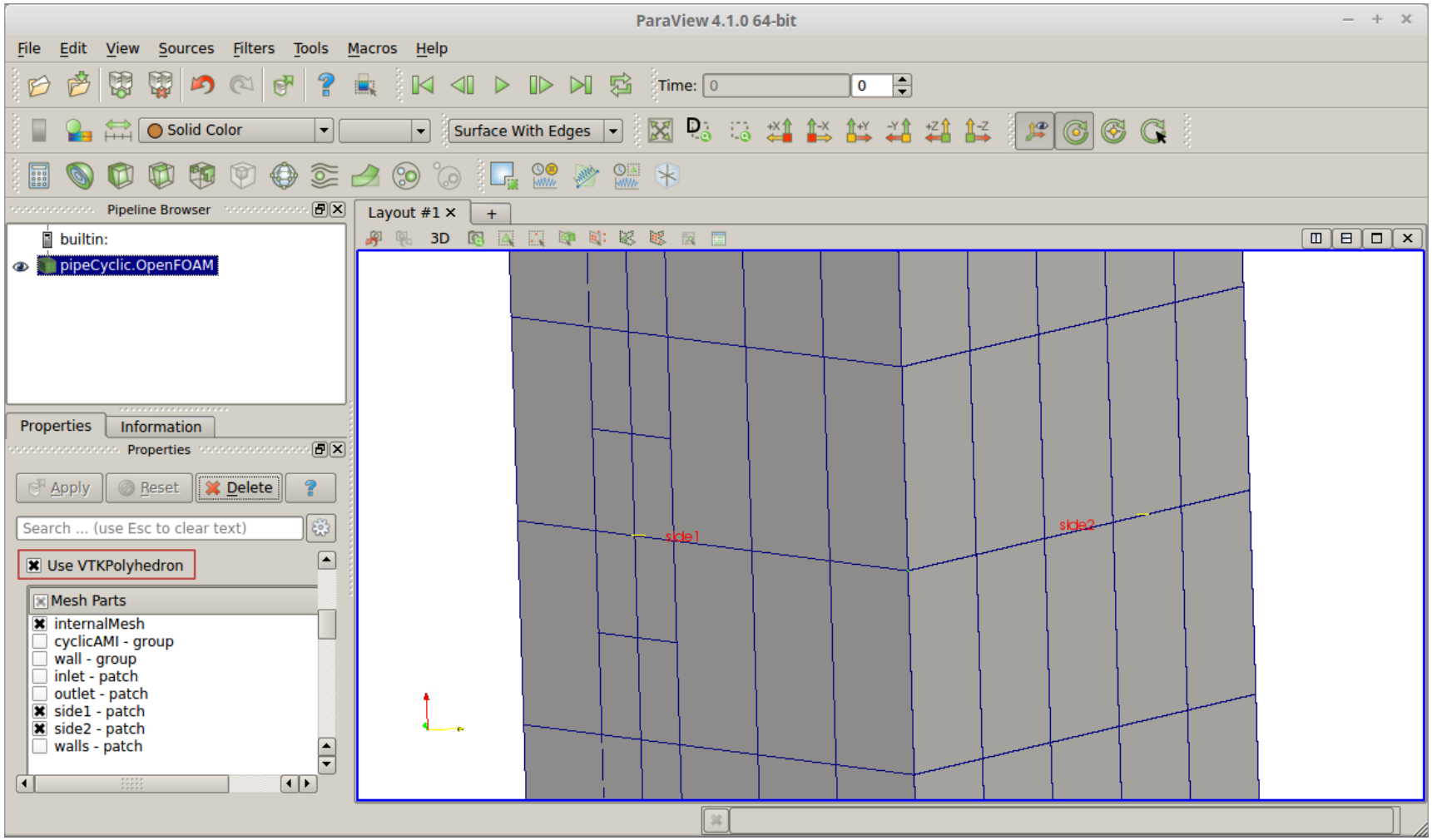
実行コマンド

```
$ paraFoam
```



チュートリアルのモデルの確認

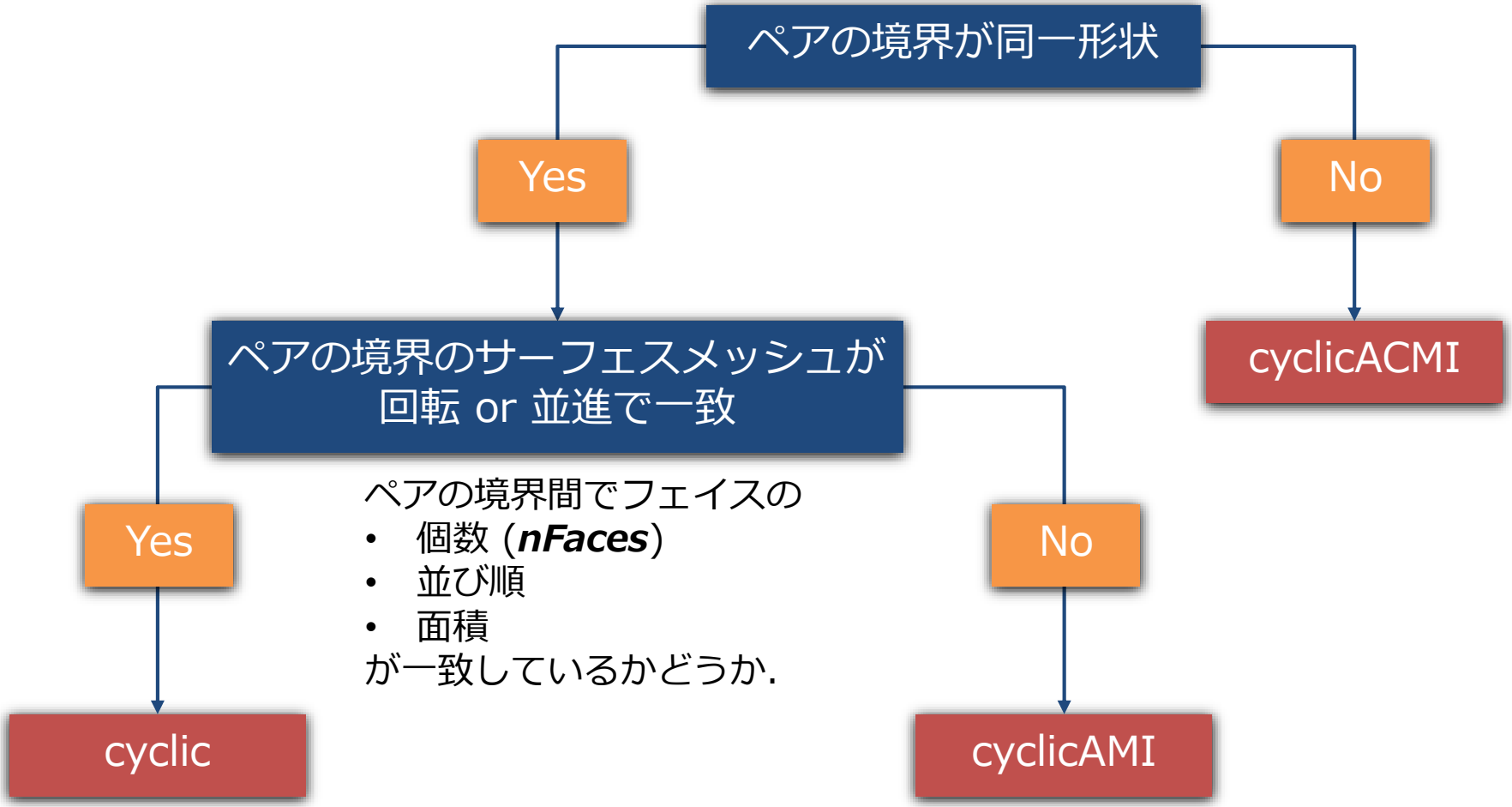
- ***Use VTKPolyhedron*** のオプションを有効にすると，計算格子が正しく表示されます．



- このモデルでは，周期境界 (***side1*** と ***side2***) の形状は同じですが，
  フェイスの数が異なっているのがわかります．

３種類の周期境界条件の使い分け

- 周期境界条件 (**cyclic**，**cyclicAMI**，**cyclicACMI**) の使い分け



boundary ファイルの設定

- ***boundary*** ファイルを見ると，境界のタイプが ***cyclicAMI*** に設定されています．

constant/polyMesh/boundary

```
side1                                   side2
{                                       {
    type            cyclicAMI;              type            cyclicAMI;
    inGroups        1(cyclicAMI);           inGroups        1(cyclicAMI);
    nFaces          400;                    nFaces          250;
    startFace       4478;                   startFace       4878;
    matchTolerance  0.0001;                 matchTolerance  0.0001;
    transform       rotational;             transform       rotational;
    neighbourPatch  side2;                  neighbourPatch  side1;
    rotationAxis    (1 0 0);                rotationAxis    (1 0 0);
    rotationCentre  (0 0 0);                rotationCentre  (0 0 0);
}                                       }
```

- これを ***cyclic*** に書き直してソルバーを実行しようとすると，
  ２つの境界でフェイス数が異なるので，下記のエラーメッセージが出力されて
  計算が実行できません．

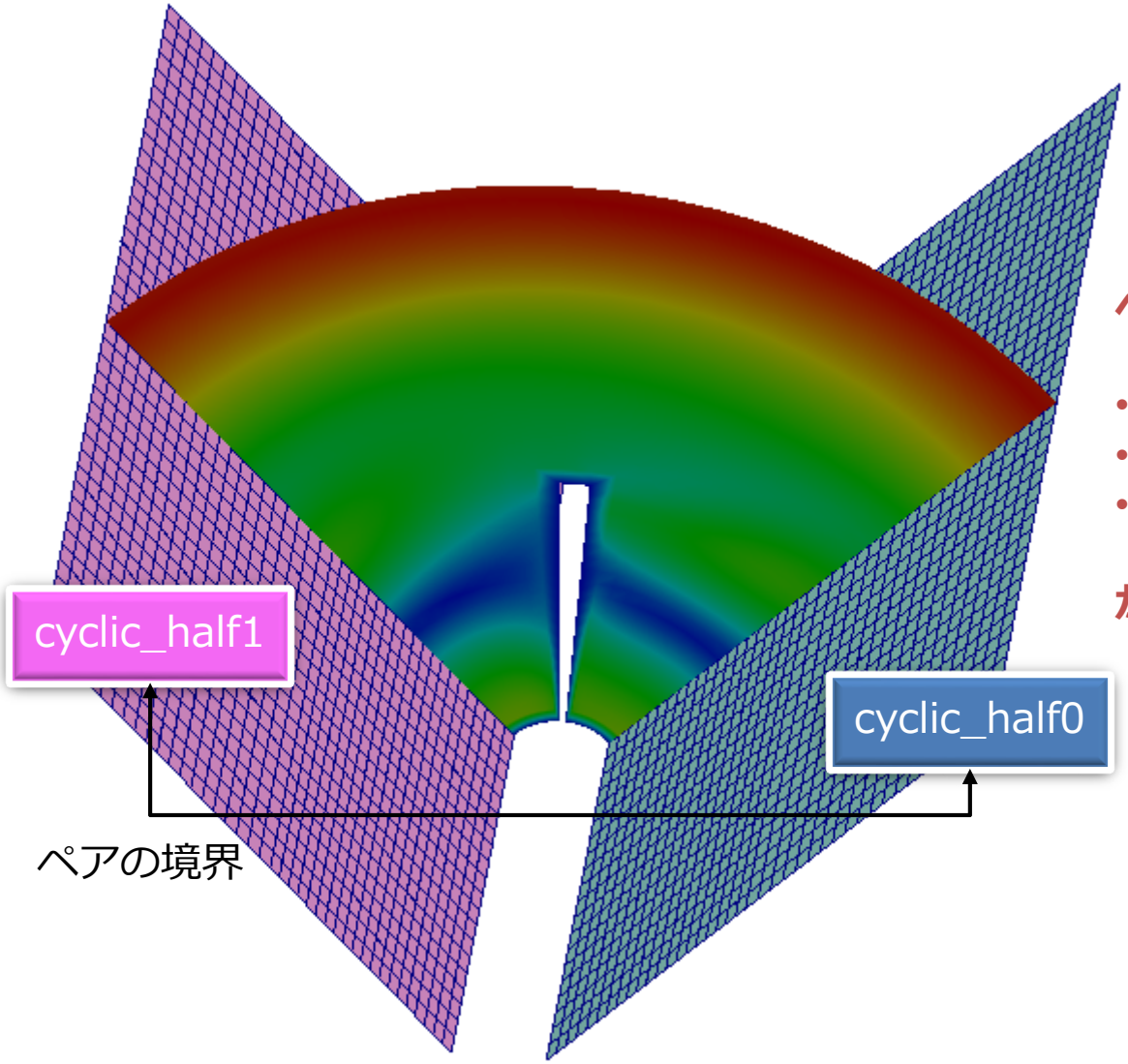
--> FOAM FATAL ERROR:
For patch side1 there are 400 face centres, for the neighbour patch side2 there are 250

From function cyclicPolyPatch::calcTransforms()
in file meshes/polyMesh/polyPatches/constraint/cyclic/cyclicPolyPatch.C at line 161.

FOAM exiting

周期境界条件 cyclic

- ***cyclic*** の使用例
  チュートリアル：incompressible/SRFSimpleFoam/mixer



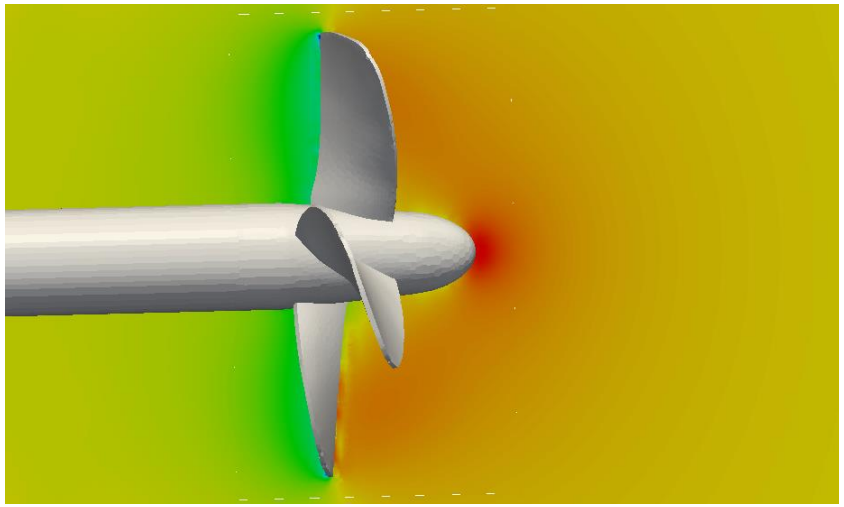
ペアの境界間で

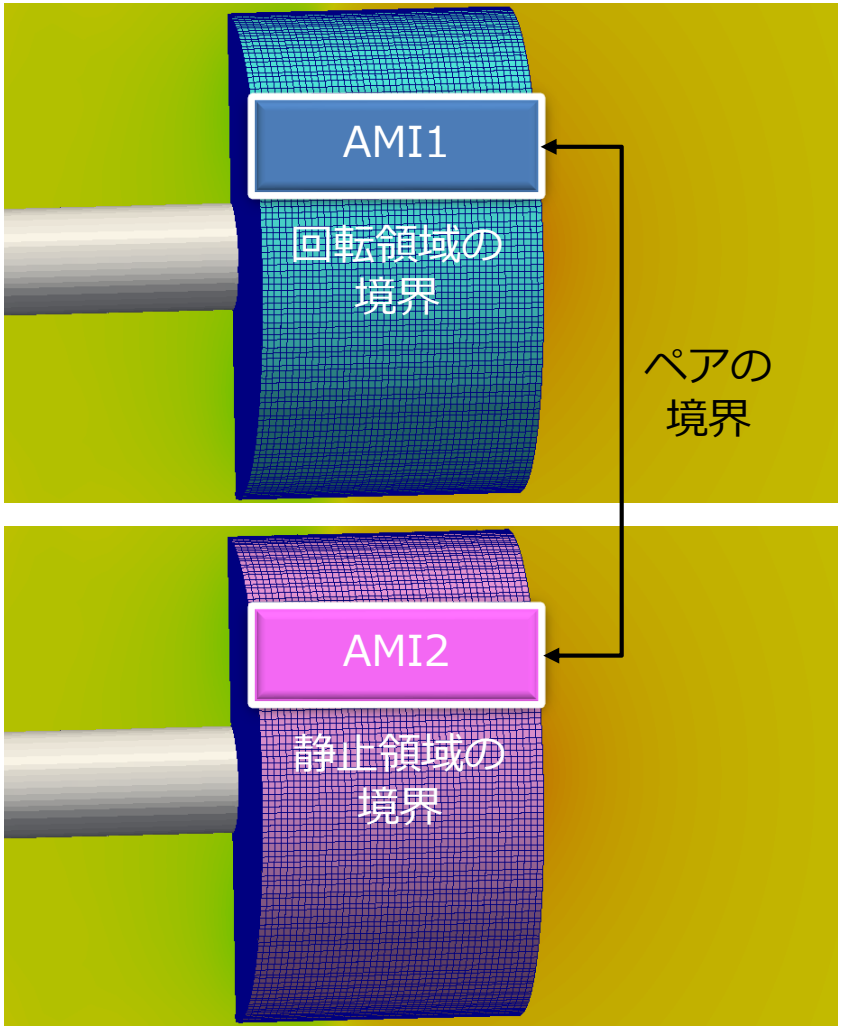
- **境界の形状**
- **フェイスの数**
- **対応するフェイスの面積**

が一致しています.

周期境界条件 cyclicAMI

- ***cyclicAMI*** の使用例
  チュートリアル：incompressible/pimpleDyMFoam/propeller



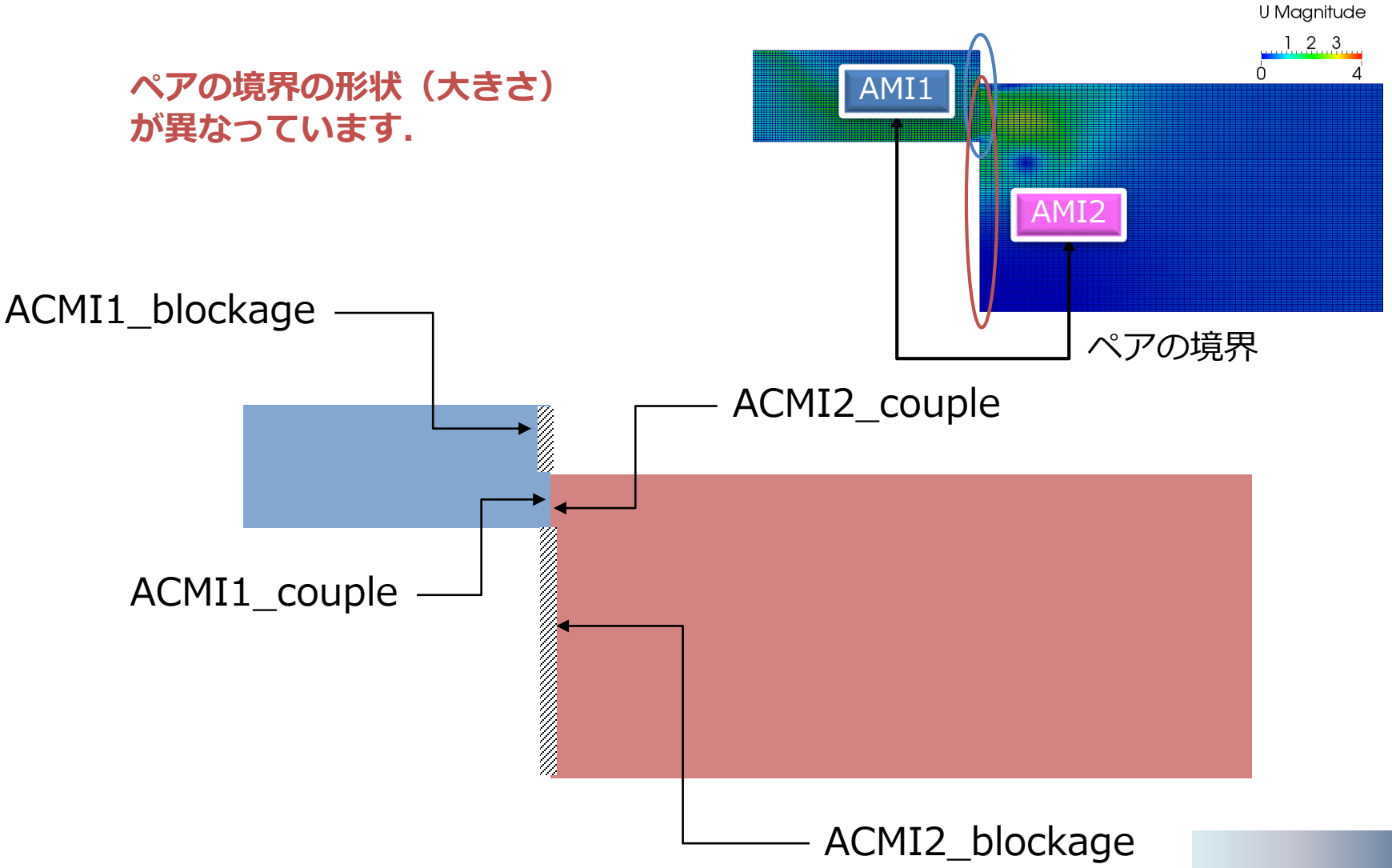


**境界の形状は一致していますが，
フェイスの数がペアの境界間で
異なっています．**

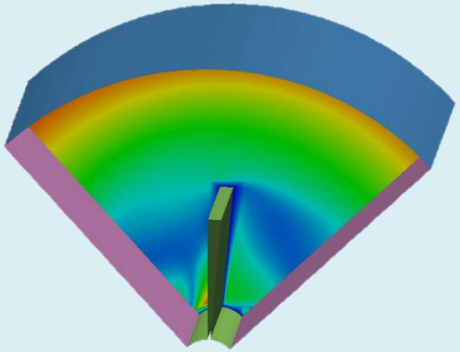
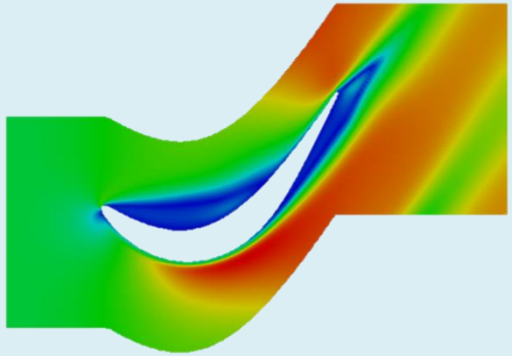
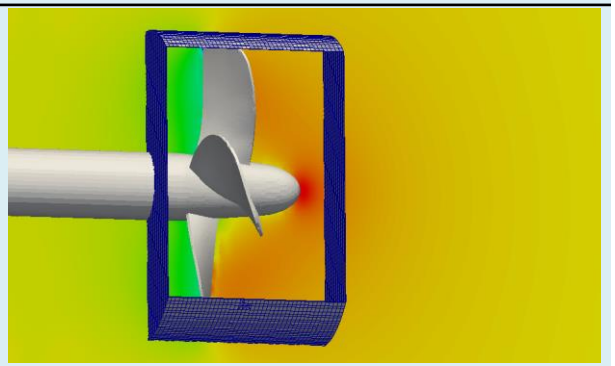
周期境界条件 cyclicACMI

- ***cyclicACMI*** の使用例
  チュートリアル：incompressible/pimpleDyMFoam/oscillatingInletACMI2D

ペアの境界の形状（大きさ）が異なっています.



周期性 transform の主な３つのタイプ

タイプ名	説明	使用例
rotational	• 回転移動で一致する周期性 • 回転軸の指定が必要(回転軸上の点 ***rotationCentre*** と軸の向き ***rotationAxis*** を指定)	
translational	• 平行移動で一致する周期性 • ペアを構成するもう一方の境界の位置をベクトルで指定 ***separationVector***	
noOrdering	• 隣接する２つの境界に使用 • ***cyclicAMI*** と使用して，**スライディングメッシュ**計算が可能	

第４章　境界条件クラスの探索

この章では，クラスのカスタマイズを行う際に
どの関数を編集すれば良いのか判断できるように，
境界条件クラスのメンバ関数への理解を深めます．

境界条件のカスタマイズ

- OpenFOAM の境界条件をカスタマイズすることは，
  新しい境界条件クラスを作成することを意味します．

- カスタマイズ作業の流れ

  1. 既存の境界条件クラスの中から，自分の作成したい条件に近いものを
     探します．

  2. 選択した境界条件クラスのソースコードをコピー，編集し，
     所望の機能を実現するようにクラスを作り直します．

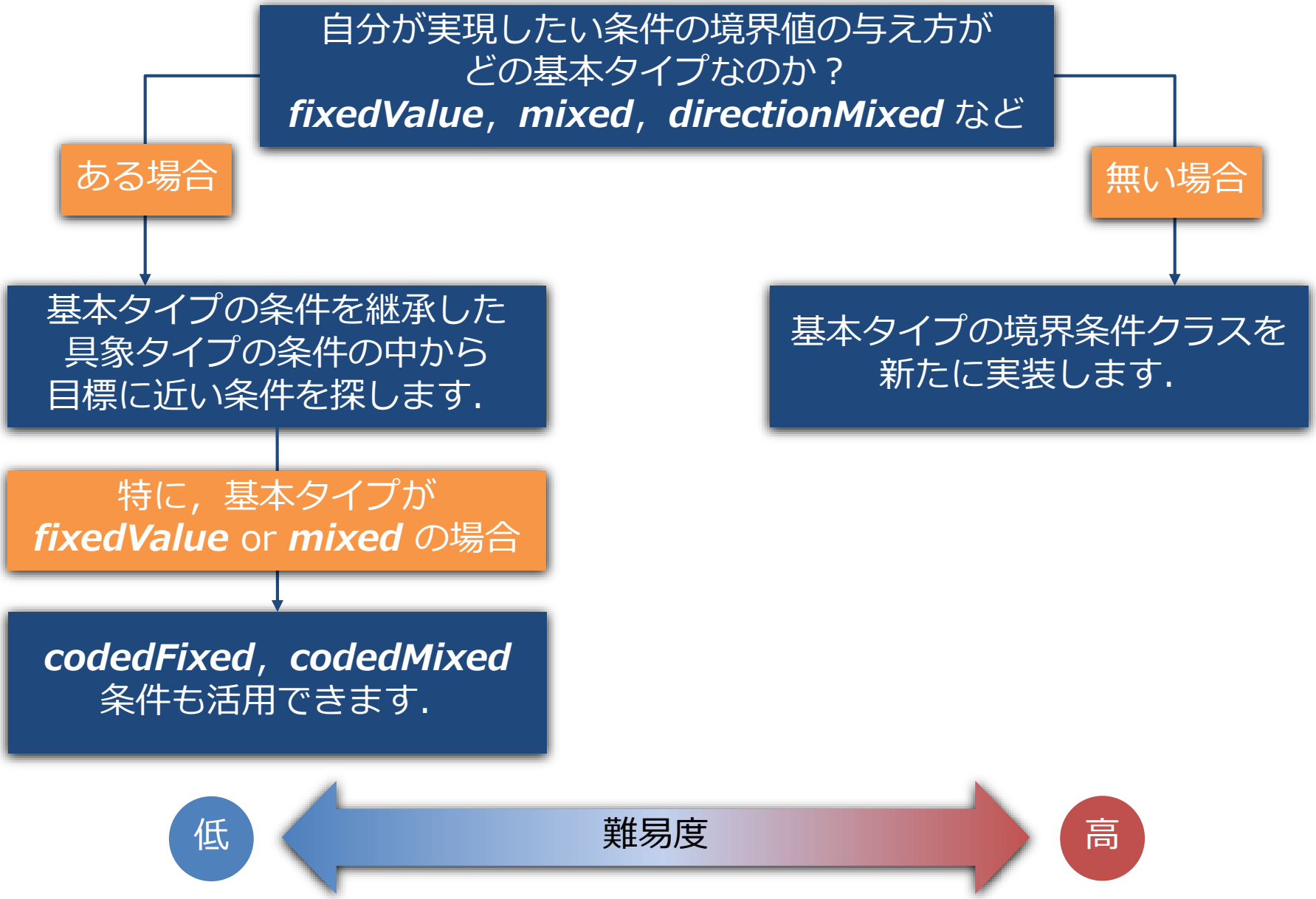
     - クラス名

     - メンバ関数・変数

- 手順１において，実現したい条件に近いクラスを選択することにより，
  手順２の編集作業が行いやすくなります．

- どの階層 (***basic*** or ***derived***) の境界条件を作成するのかにより，
  変更が必要なメンバ関数が異なります．

参考とする境界条件クラスの選択



編集が必要なメンバ

自分が実現したい条件の境界値の与え方が
どの基本タイプなのか？
fixedValue，***mixed***，***directionMixed*** など

ある場合

基本タイプの条件を継承した
具象タイプの条件の中から
目標に近い条件を探します．

要変更

- メンバ変数の追加や削除
- コンストラクタ (メンバ変数に対応して)
- メンバ関数： ***updateCoeffs***，***write*** など

編集が必要なメンバ

自分が実現したい条件の境界値の与え方が
どの基本タイプなのか？
fixedValue，***mixed***，***directionMixed*** など

無い場合

基本タイプの境界条件クラスを
新たに実装します．

要変更

- メンバ変数の追加や削除
- コンストラクタ (メンバ変数に対応して)
- メンバ関数： ***evaluate***，***write*** に加えて
  - ***valueInternalCoeffs***
  - ***valueBoundaryCoeffs***
  - ***gradientInternalCoeffs***
  - ***gradientBoundaryCoeffs***
  - ***snGrad***
  - ***fixesValue*** など

valueInternalCoeffs と valueBoundaryCoeffs

- 第１章で確認したように，**対流項**を境界隣接セルにおいて
  離散化する際に，境界フェイスからの

  - **係数行列の対角成分**
  - **右辺ベクトル**

  への寄与は基本タイプ (***fixedValue***，***mixed***，***directionMixed*** など)
  ごとに異なっていました.

- OpenFOAM の境界条件クラスではこの違いを取り扱うために，
  基本タイプのクラスごとに次の２つの関数がオーバーライドされています.

  - ***valueInternalCoeffs***()
    係数行列の対角成分への寄与

  - ***valueBoundaryCoeffs***()
    右辺ベクトルへの寄与

valueInternalCoeffs と valueBoundaryCoeffs

- 対流項の離散化

$$\int_V \nabla \cdot (\boldsymbol{U}\phi)\, dV = \int_S d\boldsymbol{S} \cdot (\boldsymbol{U}\phi) = \sum_f \boldsymbol{S}_f \cdot \boldsymbol{U}_f\, \phi_f = \sum_f F\phi_f$$

境界フェイスにおいて

$$F_b\phi_b = F_b\left(c_{vI}\phi_p + c_{vB}\right)$$

valueInternalCoeffs ***valueBoundaryCoeffs***

src/finiteVolume/finiteVolume/convectionSchemes/gaussConvectionScheme/gaussConvectionScheme.C

```
forAll(vf.boundaryField(), patchI)
{
    const fvPatchField<Type>& psf = vf.boundaryField()[patchI];
    const fvsPatchScalarField& patchFlux = faceFlux.boundaryField()[patchI];
    const fvsPatchScalarField& pw = weights.boundaryField()[patchI];

    fvm.internalCoeffs()[patchI] = patchFlux*psf.valueInternalCoeffs(pw);
    fvm.boundaryCoeffs()[patchI] = -patchFlux*psf.valueBoundaryCoeffs(pw);
}
```

fixedValue 条件

src/finiteVolume/fields/fvPatchFields/basic/fixedValue/fixedValueFvPatchField.C

```
template<class Type>
tmp<Field<Type> > fixedValueFvPatchField<Type>::valueInternalCoeffs
(
    const tmp<scalarField>&
) const
{
    return tmp<Field<Type> >
    (
        new Field<Type>(this->size(), pTraits<Type>::zero)
    );
}
```

係数行列の対角成分への寄与なし

$$F_b \phi_b = F_b \left(0 \cdot \phi_p + \phi_b \right)$$

```
template<class Type>
tmp<Field<Type> > fixedValueFvPatchField<Type>::valueBoundaryCoeffs
(
    const tmp<scalarField>&
) const
{
    return *this;
}
```

右辺ベクトルへの寄与あり

fixedGradient 条件

src/finiteVolume/fields/fvPatchFields/basic/fixedGradient/fixedGradientFvPatchField.C

```
template<class Type>
tmp<Field<Type> > fixedGradientFvPatchField<Type>::valueInternalCoeffs
(
    const tmp<scalarField>&
) const
{
    return tmp<Field<Type> >(new Field<Type>(this->size(), pTraits<Type>::one));
}
```

$$F_b \phi_b = F_b \left(1 \cdot \phi_p + g_b |\boldsymbol{d}| \right)$$

係数行列の対角成分と右辺ベクトルの両方に対して寄与あり

```
template<class Type>
tmp<Field<Type> > fixedGradientFvPatchField<Type>::valueBoundaryCoeffs
(
    const tmp<scalarField>&
) const
{
    return gradient()/this->patch().deltaCoeffs();
}
```

zeroGradient 条件

src/finiteVolume/fields/fvPatchFields/basic/zeroGradient/zeroGradientFvPatchField.C

```
template<class Type>
tmp<Field<Type> > zeroGradientFvPatchField<Type>::valueInternalCoeffs
(
    const tmp<scalarField>&
) const
{
    return tmp<Field<Type> >
    (
        new Field<Type>(this->size(), pTraits<Type>::one)
    );
}
```

係数行列の対角成分への寄与あり

$$F_b \phi_b = F_b \left(1 \cdot \phi_p + 0 \right)$$

```
template<class Type>
tmp<Field<Type> > zeroGradientFvPatchField<Type>::valueBoundaryCoeffs
(
    const tmp<scalarField>&
) const
{
    return tmp<Field<Type> >
    (
        new Field<Type>(this->size(), pTraits<Type>::zero)
    );
}
```

右辺ベクトルへの寄与なし

mixed 条件

src/finiteVolume/fields/fvPatchFields/basic/mixed/mixedFvPatchField.C

```
template<class Type>
tmp<Field<Type> > mixedFvPatchField<Type>::valueInternalCoeffs
(
    const tmp<scalarField>&
) const
{
    return Type(pTraits<Type>::one)*(1.0 - valueFraction_);
}
```

$$F_b\phi_b = F_b\Big((1-f)\cdot\phi_p + f\cdot refValue + (1-f)\cdot refGrad\cdot|d|\Big)$$

```
template<class Type>
tmp<Field<Type> > mixedFvPatchField<Type>::valueBoundaryCoeffs
(
    const tmp<scalarField>&
) const
{
    return
        valueFraction_*refValue_
      + (1.0 - valueFraction_)*refGrad_/this->patch().deltaCoeffs();
}
```

gradientInternalCoeffs と gradientBoundaryCoeffs

- 対流項の場合と同じように，**ラプラシアン項**を境界隣接セルにおいて離散化する際に，境界フェイスからの

  - **係数行列の対角成分**
  - **右辺ベクトル**

  への寄与は基本タイプ (***fixedValue***，***mixed***，***directionMixed*** など) ごとに異なっていました.

- OpenFOAM の境界条件クラスではこの違いを取り扱うために，基本タイプのクラスごとに次の２つの関数がオーバーライドされています.

  - ***gradientInternalCoeffs***()
    係数行列の対角成分への寄与

  - ***gradientBoundaryCoeffs***()
    右辺ベクトルへの寄与

- ラプラシアン項の離散化

$$\int_V \nabla \cdot (\Gamma \nabla \phi)\, dV = \int_S d\boldsymbol{S} \cdot (\Gamma \nabla \phi) = \sum_f \Gamma_f \; \boldsymbol{S}_f \cdot (\nabla \phi)_f$$

境界フェイスにおいて

$$\Gamma_b \boldsymbol{S}_b \cdot (\nabla \phi)_b = \Gamma_b |\boldsymbol{S}_b| \left(c_{gI} \phi_p + c_{gB} \right)$$

gradientInternalCoeffs

gradientBoundaryCoeffs

gradientInternalCoeffs と gradientBoundaryCoeffs

- ラプラシアン項の離散化

src/finiteVolume/finiteVolume/laplacianSchemes/gaussLaplacianScheme/gaussLaplacianScheme.C

```
forAll(vf.boundaryField(), patchi)
{
    const fvPatchField<Type>& pvf = vf.boundaryField()[patchi];
    const fvsPatchScalarField& pGamma = gammaMagSf.boundaryField()[patchi];
    const fvsPatchScalarField& pDeltaCoeffs =
        deltaCoeffs.boundaryField()[patchi];

    if (pvf.coupled())
    {
        fvm.internalCoeffs()[patchi] =
            pGamma*pvf.gradientInternalCoeffs(pDeltaCoeffs);
        fvm.boundaryCoeffs()[patchi] =
           -pGamma*pvf.gradientBoundaryCoeffs(pDeltaCoeffs);
    }
    else
    {
        fvm.internalCoeffs()[patchi] = pGamma*pvf.gradientInternalCoeffs();
        fvm.boundaryCoeffs()[patchi] = -pGamma*pvf.gradientBoundaryCoeffs();
    }
}
```

周期境界の処理

その他の境界の処理

fixedValue 条件

src/finiteVolume/fields/fvPatchFields/basic/fixedValue/fixedValueFvPatchField.C

```
template<class Type>
tmp<Field<Type> > fixedValueFvPatchField<Type>::gradientInternalCoeffs() const
{
    return -pTraits<Type>::one*this->patch().deltaCoeffs();
}
```

係数行列の対角成分と右辺ベクトルの両方に対して寄与あり

$$\Gamma_f \boldsymbol{S}_f \cdot (\nabla\phi)_f = \Gamma_b \, |\boldsymbol{S}_b| \left(\frac{-1}{|\boldsymbol{d}|}\right) \phi_P + \Gamma_b |\boldsymbol{S}_b| \frac{1}{|\boldsymbol{d}|} \phi_b$$

境界フェイスにおいて，
deltaCoeffs() は，フェイス中心点と隣接セル中心点間の**距離の逆数**を返します．

```
template<class Type>
tmp<Field<Type> > fixedValueFvPatchField<Type>::gradientBoundaryCoeffs() const
{
    return this->patch().deltaCoeffs()*(*this);
}
```

fixedGradient 条件

src/finiteVolume/fields/fvPatchFields/basic/fixedGradient/fixedGradientFvPatchField.C

```
template<class Type>
tmp<Field<Type> > fixedGradientFvPatchField<Type>::
gradientInternalCoeffs() const
{
    return tmp<Field<Type> >
    (
        new Field<Type>(this->size(), pTraits<Type>::zero)
    );
}
```

係数行列の対角成分への寄与なし

$$\Gamma_f \boldsymbol{S}_f \cdot (\nabla\phi)_f = \Gamma_b |\boldsymbol{S}_b| \left.\frac{\partial\phi}{\partial \boldsymbol{n}}\right|_b$$

$\frac{\partial\phi}{\partial \boldsymbol{n}}$: 変数 ϕ の法線方向 $\boldsymbol{n}$ 勾配
(変数 ϕ と同じ型)

右辺ベクトルへの寄与あり

```
template<class Type>
tmp<Field<Type> > fixedGradientFvPatchField<Type>::
gradientBoundaryCoeffs() const
{
    return gradient();
}
```

$\frac{\partial\phi}{\partial \boldsymbol{n}}$ の値を返すメンバ関数

zeroGradient 条件

src/finiteVolume/fields/fvPatchFields/basic/zeroGradient/zeroGradientFvPatchField.C

```
template<class Type>
tmp<Field<Type> > zeroGradientFvPatchField<Type>::gradientInternalCoeffs() const
{
    return tmp<Field<Type> >
    (
        new Field<Type>(this->size(), pTraits<Type>::zero)
    );
}
```

係数行列の対角成分と右辺ベクトルの両方に対して寄与なし

$$\Gamma_f \boldsymbol{S}_f \cdot (\nabla\phi)_f = \Gamma_b |\boldsymbol{S}_b| \left.\frac{\partial\phi}{\partial\boldsymbol{n}}\right|_b = 0$$ (変数 ϕ の型のゼロ)

```
template<class Type>
tmp<Field<Type> > zeroGradientFvPatchField<Type>::gradientBoundaryCoeffs() const
{
    return tmp<Field<Type> >
    (
        new Field<Type>(this->size(), pTraits<Type>::zero)
    );
}
```

mixed 条件

src/finiteVolume/fields/fvPatchFields/basic/mixed/mixedFvPatchField.C

```
template<class Type>
tmp<Field<Type> > mixedFvPatchField<Type>::gradientInternalCoeffs() const
{
    return -Type(pTraits<Type>::one)*valueFraction_*this->patch().deltaCoeffs();
}
```

係数行列の対角成分と右辺ベクトルの両方に対して寄与あり

$$\Gamma_f \boldsymbol{S}_f \cdot (\nabla\phi)_f = \Gamma_b |\boldsymbol{S}_b| \left(\frac{-f}{|\boldsymbol{d}|}\right) \phi_P + \Gamma_b |\boldsymbol{S}_b| \frac{f \cdot refValue + (1-f) \cdot refGrad \cdot |\boldsymbol{d}|}{|\boldsymbol{d}|}$$

```
template<class Type>
tmp<Field<Type> > mixedFvPatchField<Type>::gradientBoundaryCoeffs() const
{
    return
      valueFraction_*this->patch().deltaCoeffs()*refValue_
    + (1.0 - valueFraction_)*refGrad_;
}
```

pTraits について

- 前ページまでのコードでは，***pTraits*** という関数を使うことで，
  変数の型の違いを処理しています．

テストコード｜applications/test/pTraits/Test-pTraits.C

```
#include "IOstreams.H"
#include "pTraits.H"
#include "vector.H"
#include "tensor.H"

using namespace Foam;

// * * * * * * * * * * * * * * * * * * * * * * //
// Main program:
```

テンプレート関数の定義

```
template<class T>
void printTraits()
{
    Info<< pTraits<T>::typeName
        << ": zero=" << pTraits<T>::zero
        << " one=" << pTraits<T>::one << endl;
}


template<class T>
void printTraits(const pTraits<T>& p)
{
    Info<< p.typeName << " == " << p << endl;
}
```

テンプレート引数の型を
明示的に指定して実行

```
int main()
{
    printTraits<bool>();
    printTraits<label>();
    printTraits<scalar>();
    printTraits<vector>();
    printTraits<tensor>();

    {
        pTraits<bool> b(true);
        printTraits(b);
    }

    {
        pTraits<label> l(100);
        printTraits(l);
    }

    printTraits(pTraits<scalar>(3.14159));

    Info<< "End¥n" << endl;

    return 0;
}
```

- テストコードの実行結果を示します.

実行コマンド

```
$ app
$ cd test/pTraits
$ wmake
$ Test-pTraits
```

実行結果

```
bool: zero=0 one=1
label: zero=0 one=1
scalar: zero=0 one=1
vector: zero=(0 0 0) one=(1 1 1)
tensor: zero=(0 0 0 0 0 0 0 0 0) one=(1 1 1 1 1 1 1 1 1)
bool == 1
label == 100
scalar == 3.14159
End
```

```
printTraits<bool>();
printTraits<label>();
printTraits<scalar>();
printTraits<vector>();
printTraits<tensor>();
```

pTraits<型>::zero, pTraits<型>::one が, 指定した型の値を返しているのが分かります.

境界値の更新 correctBoundaryConditions()

- 次のコードを実行すると，流速場 ***U*** の境界値が更新されます．

U.correctBoundaryConditions();

- まず，***GeometricBoundaryField*** クラスのメンバ関数 ***evaluate()*** は，境界をループして，それぞれの境界において使用されている境界条件のクラスの ***evaluate*** を実行します．

src/OpenFOAM/fields/GeometricFields/GeometricField/GeometricBoundaryField.C

```
template<class Type, template<class> class PatchField, class GeoMesh>
void Foam::GeometricField<Type, PatchField, GeoMesh>::GeometricBoundaryField::
evaluate()
{
  (省略)
        forAll(*this, patchi)
        {
            this->operator[](patchi).evaluate(Pstream::defaultCommsType);
        }
  (省略)
}
```

- ここで，***U*** の境界条件に，***inletOutlet*** 条件を使用した場合に，どのように境界値が計算されるのかを詳しく見ていきます．

境界値の更新 correctBoundaryConditions()

- ***inletOutlet*** 条件のクラスでは，***evaluate*** をオーバーライドしていないので，その親クラスである ***mixed*** 条件クラスの ***evaluate*** が実行されます．

src/finiteVolume/fields/fvPatchFields/basic/mixed/mixedFvPatchField.C

```
template<class Type>
void mixedFvPatchField<Type>::evaluate(const Pstream::commsTypes)
{
    if (!this->updated())
    {
        this->updateCoeffs();
    }

    Field<Type>::operator=
    (
        valueFraction_*refValue_
      +
        (1.0 - valueFraction_)*
        (
            this->patchInternalField()
          + refGrad_/this->patch().deltaCoeffs()
        )
    );

    fvPatchField<Type>::evaluate();
}
```

1

valueFraction_の値の更新
詳細は次ページ．

2

3

境界値の更新 correctBoundaryConditions()

1. ***inletOutlet*** 条件のクラスのメンバ関数 ***updateCoeffs***() を実行し，次の処理を行います．

src/finiteVolume/fields/fvPatchFields/derived/inletOutlet/inletOutletFvPatchField.C

```
template<class Type>
void Foam::inletOutletFvPatchField<Type>::updateCoeffs()
{
    if (this->updated())
    {
        return;
    }

    const Field<scalar>& phip =
        this->patch().template lookupPatchField<surfaceScalarField, scalar>
        (
            phiName_
        );

    this->valueFraction() = 1.0 - pos(phip);

    mixedFvPatchField<Type>::updateCoeffs();
}
```

メンバ変数 ***valueFraction_*** の値の更新

updated_の値を true に変更

- ***updated_***は，***updateCoeffs***() を既に呼んだかどうかのフラグ変数の役割を果たしています．

境界値の更新 correctBoundaryConditions()

- ***inletOutlet*** 条件のクラスでは，***evaluate*** をオーバーライドしていないので，その親クラスである ***mixed*** 条件クラスの ***evaluate*** が実行されます．

src/finiteVolume/fields/fvPatchFields/basic/mixed/mixedFvPatchField.C

```
template<class Type>
void mixedFvPatchField<Type>::evaluate(const Pstream::commsTypes)
{
    if (!this->updated())
    {
        this->updateCoeffs();
    }

    Field<Type>::operator=
    (
        valueFraction_*refValue_
      +
        (1.0 - valueFraction_)*
        (
            this->patchInternalField()
          + refGrad_/this->patch().deltaCoeffs()
        )
    );

    fvPatchField<Type>::evaluate();
}
```

1

2

3

更新した ***valueFraction_*** を使用して，境界値を計算

次に備えて，***updated_***の値を false に戻す

evaluate と updateCoeffs の関係

- ソースコードを見ると，次の関係になっていることが分かります．
  - 基本タイプ (***basic*** ディレクトリに設置) のクラスにおいて

    evaluate をオーバーライド

  - 具象タイプ (***derived*** ディレクトリに設置) のクラスにおいて

    updateCoeffs をオーバーライド

- 基本タイプごとに，***evaluate*** に定義された境界値の計算式と，
  その計算に含まれるメンバ変数が異なります．

例えば，基本タイプの1つである ***mixed*** 条件の場合，

- その境界値の計算式は，前ページの 2 で与えられ，
- その計算式には，次の3つのメンバ変数が使用されています．

refValue_，***refGrad_***，***valueFraction_***

evaluate と updateCoeffs の関係

- 具象タイプのクラスでは,
  - 境界値の計算式は基本タイプのものを使用し,
  - その計算式に使用されているメンバ変数の計算方法を再定義することで,

  境界条件のバリエーションを増やしています.

- このメンバ変数の計算方法が, ***updateCoeffs*** に定義されています.

  例えば, 同じ基本タイプ ***mixed*** 条件から派生している次の 2 種類の境界条件クラスでは, ***updateCoeffs*** が異なっています.

  - ***inletOutlet*** 条件のクラス
    - ***valueFraction_*** の値を流束 ***phi*** の符号から計算

```
this->valueFraction() = 1.0 - pos(phip);
```

  - ***variableHeightFlowRate*** 条件のクラス (次ページ)
    - 流束 ***phi*** の符号と体積分率 ***alpha*** の値から, 境界フェイスごとに ***refValue_*** と ***valueFraction_*** を計算

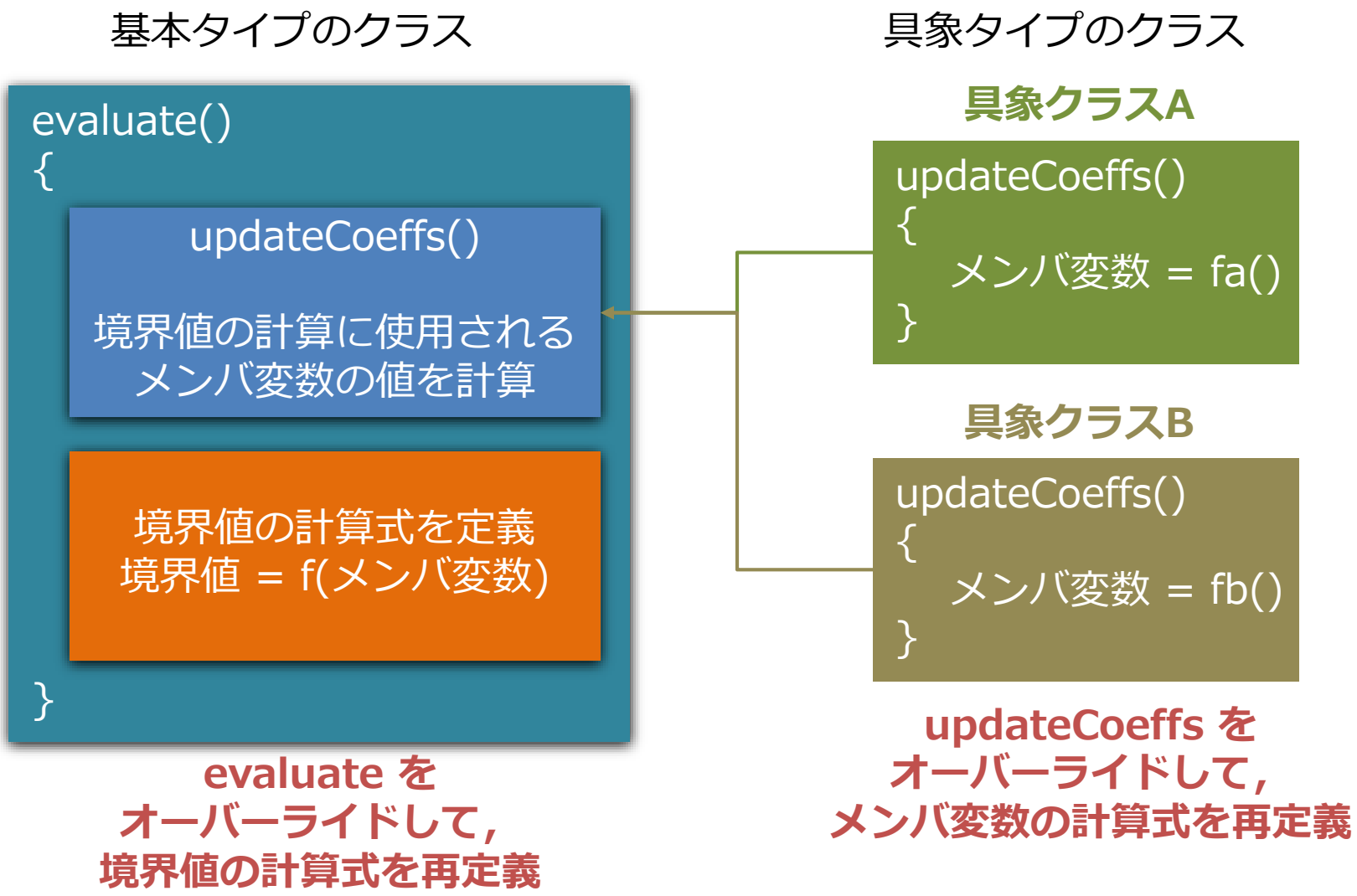
```
this->refValue()[i] = alphap[i];
this->valueFraction()[i] = 1.0;
```

variableHeightFlowRate 条件クラスの updateCoeffs

src/finiteVolume/fields/fvPatchFields/derived/variableHeightFlowRate/variableHeightFlowRateFvPatchField.C

```
void Foam::variableHeightFlowRateFvPatchScalarField::updateCoeffs()
{
    if (this->updated())
    {
        return;
    }

    const fvsPatchField<scalar>& phip =
        patch().lookupPatchField<surfaceScalarField, scalar>(phiName_);

    scalarField alphap(this->patchInternalField());


    forAll(phip, i)
    {
        if (phip[i] < -SMALL)
        {
            if (alphap[i] < lowerBound_)
            {
                this->refValue()[i] = 0.0;
            }
            else if (alphap[i] > upperBound_)
            {
                this->refValue()[i] = 1.0;
            }
            else
            {
                this->refValue()[i] = alphap[i];
            }

            this->valueFraction()[i] = 1.0;
        }
        else
        {
            this->refValue()[i] = 0.0;
            this->valueFraction()[i] = 0.0;
        }
    }

    mixedFvPatchScalarField::updateCoeffs();
}
```

evaluate と updateCoeffs の関係



境界条件のバリエーションが豊富になる

複数の型に対応した境界条件の実装

- スカラー，ベクトル，テンソルなど複数の型に対して使用される境界条件は，テンプレートクラスを使用して実装されています．
  - scalar
  - vector
  - tensor

- この場合，次のように境界条件クラス名を付けています．

  境界条件クラス名 = 境界条件名 + FvPatchField

  例) ***inletOutletFvPatchField***，***fixedMeanFvPatchField*** など

- 実装に必要なファイル

  - 境界条件クラス名.C
  - 境界条件クラス名.H
  - 境界条件クラス名s.C
  - 境界条件クラス名s.H
  - 境界条件クラス名sFwd.H

例) ***inletOutletFvPatchField*** の場合
- inletOutletFvPatchField.C
- inletOutletFvPatchField.H
- inletOutletFvPatchFields.C
- inletOutletFvPatchFields.H
- inletOutletFvPatchFieldsFwd.H

それぞれの型に対応したクラス名の定義が後半の３つのファイルに記述されています．

複数の型に対応した境界条件の実装

- それぞれの型に対するクラスの名前を定義

inletOutletFvPatchFields.H

```
makePatchTypeFieldTypedefs(inletOutlet);
```

- ここで，makePatchTypeFieldTypedefs マクロは，次のように定義されています．

src/finiteVolume/fields/fvPatchFields/fvPatchField/fvPatchField.H

```
#define makePatchTypeFieldTypedefs(type)                                    ¥
    typedef type##FvPatchField<scalar> type##FvPatchScalarField;            ¥
    typedef type##FvPatchField<vector> type##FvPatchVectorField;            ¥
    typedef type##FvPatchField<sphericalTensor>                             ¥
        type##FvPatchSphericalTensorField;                                  ¥
    typedef type##FvPatchField<symmTensor> type##FvPatchSymmTensorField;    ¥
    typedef type##FvPatchField<tensor> type##FvPatchTensorField;
```

inletOutletFvPatchField の場合には，
下記のコマンドが実行されます．

```
typedef inletOutletFvPatchField<scalar> inletOutletFvPatchScalarField;
typedef inletOutletFvPatchField<vector> inletOutletFvPatchVectorField;
typedef inletOutletFvPatchField<sphericalTensor> inletOutletFvPatchSphericalTensorField;
typedef inletOutletFvPatchField<symmTensor> inletOutletFvPatchSymmTensorField;
typedef inletOutletFvPatchField<tensor> inletOutletFvPatchTensorField;
```

複数の型に対応した境界条件の実装

- マクロ関数 makePatchFields を呼んで，RTS に追加

inletOutletFvPatchFields.C

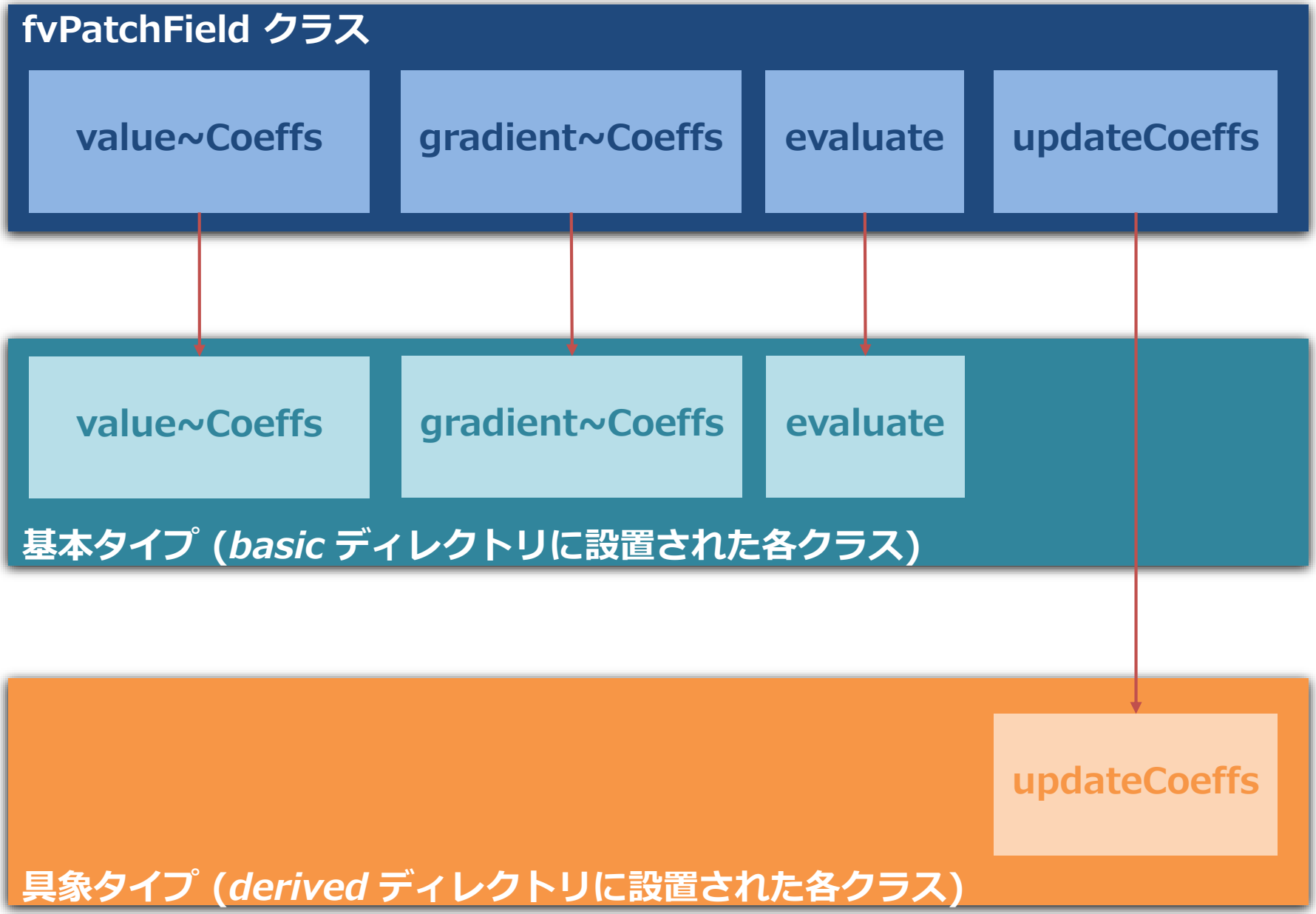
```
makePatchFields(inletOutlet);
```

- ここで，makePatchFields は，次のように定義されています.

src/finiteVolume/fields/fvPatchFields/fvPatchField/fvPatchField.H

```
#define makePatchFields(type)                                   ¥
    makeTemplatePatchTypeField                                  ¥
    (                                                           ¥
        fvPatchScalarField,                                     ¥
        type##FvPatchScalarField                                ¥
    );                                                          ¥
    makeTemplatePatchTypeField                                  ¥
    (                                                           ¥
        fvPatchVectorField,                                     ¥
        type##FvPatchVectorField                                ¥
    );                                                          ¥
    makeTemplatePatchTypeField                                  ¥
    (                                                           ¥
        fvPatchSphericalTensorField,                            ¥
        type##FvPatchSphericalTensorField                       ¥
    );                                                          ¥
    makeTemplatePatchTypeField                                  ¥
    (                                                           ¥
        fvPatchSymmTensorField,                                 ¥
        type##FvPatchSymmTensorField                            ¥
    );                                                          ¥
    makeTemplatePatchTypeField                                  ¥
    (                                                           ¥
        fvPatchTensorField,                                     ¥
        type##FvPatchTensorField                                ¥
    );
```

各メンバ関数のオーバーライドのタイミング



第６章　カスタマイズ演習２

OpenFOAM に実装されているトポロジー最適化ソルバー ***adjointShapeOptimizationFoam*** の境界条件のカスタマイズを行います.

具体的には，出口のターゲット流速分布との差を目的関数とした場合の最適化計算を行えるように，新たな境界条件を実装します.

CFD におけるトポロジー最適化

- 米倉・寒野 (2015) の緒言からの引用

特に形状のトポロジー変化を許容して最適形状を求めるトポロジー最適化手法は，その自由度の高さから注目を集めている．トポロジー最適化においては，設計空間が多孔質媒体で充たされていると仮定し，その空隙率を設計変数として最適化を行う空隙率法 (Borrval and Peterson, 2003) が広く用いられている．

トポロジー変化を許容

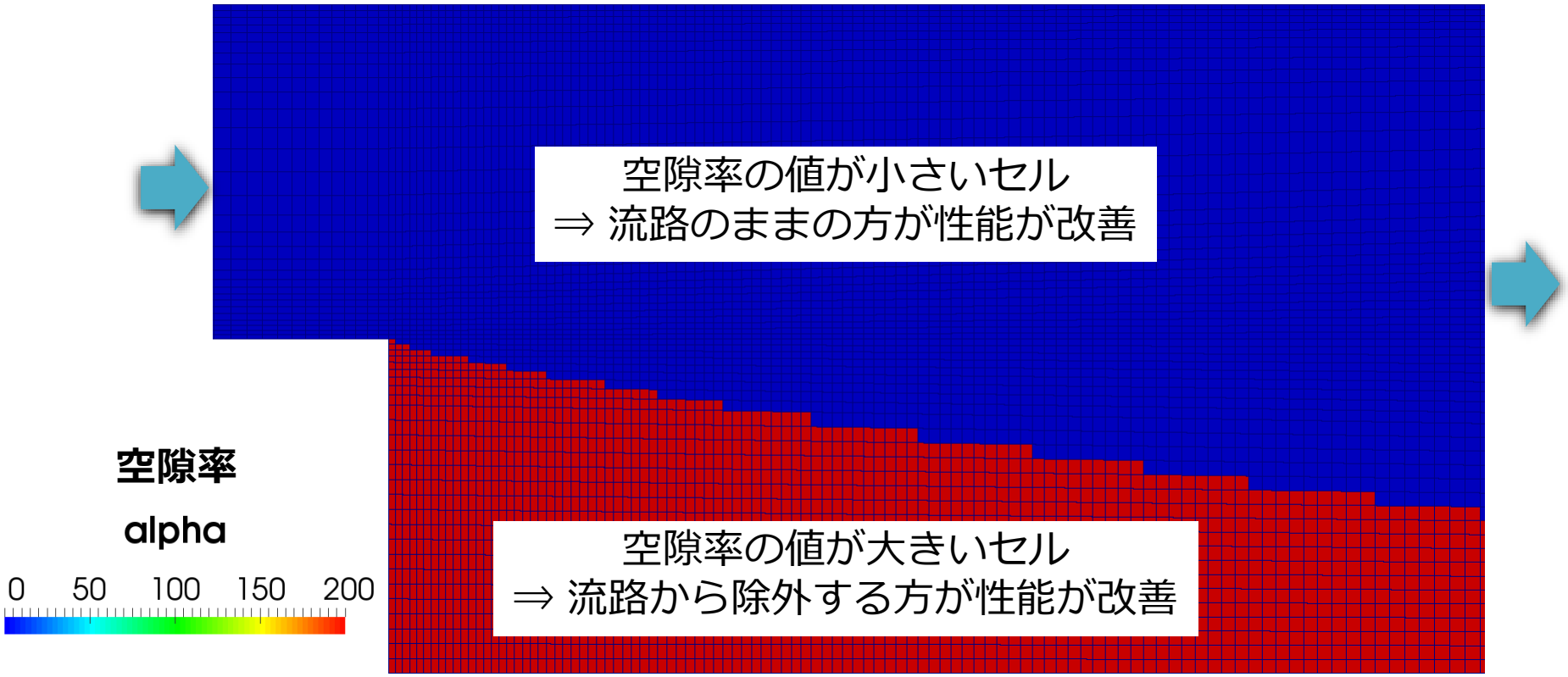
自由度の高さ

設計空間が多孔質媒体で充たされていると仮定

空隙率を設計変数として最適化を行う空隙率法

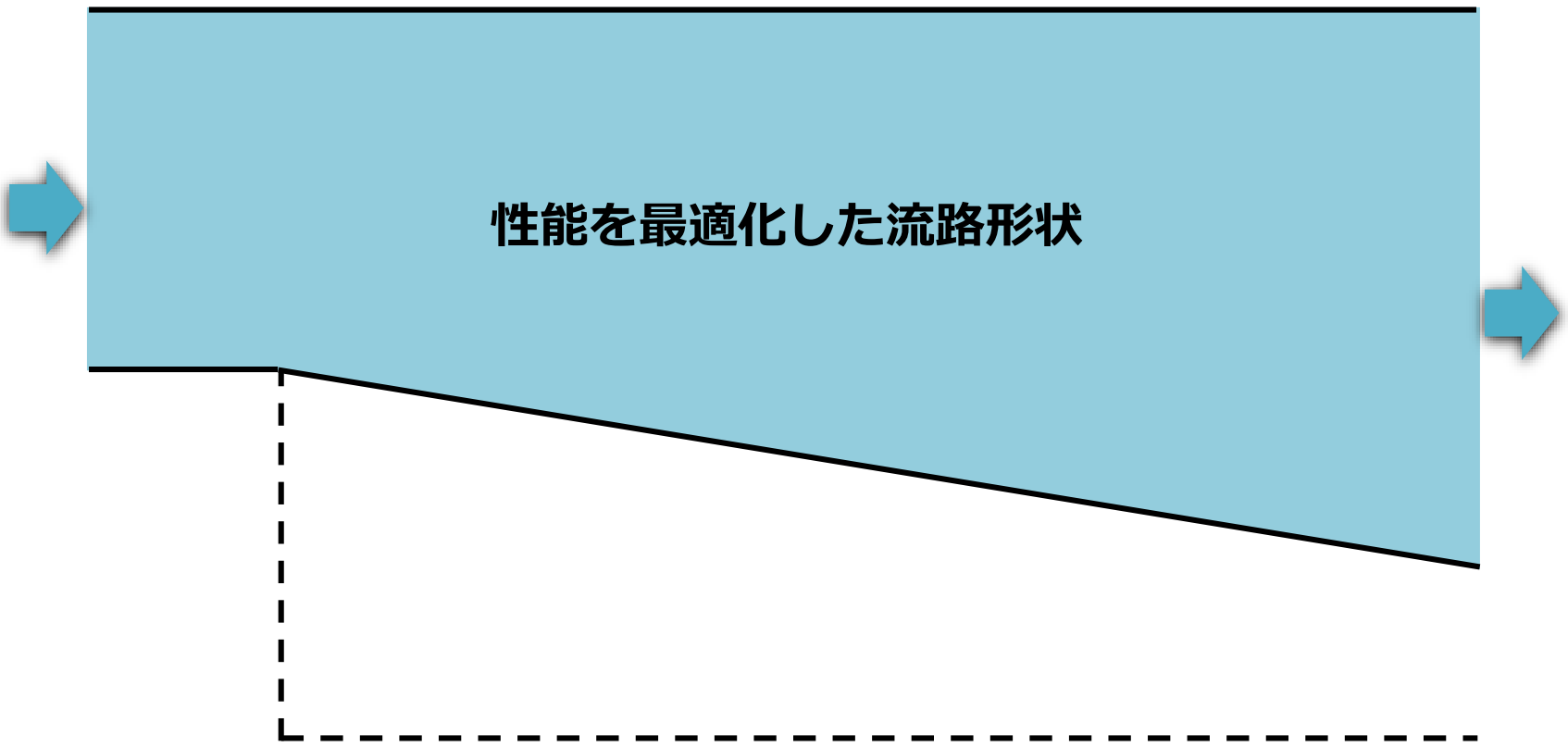
CFD におけるトポロジー最適化

- 各セルにおいて定義された空隙率 (**porosity**) の値を設計変数として，
  目的関数を最小化 (または最大化) するような空隙率の分布を求めることで，
  最適な流路形状を計算します．



CFD におけるトポロジー最適化

- 各セルにおいて定義された空隙率 (**porosity**) の値を設計変数として，
  目的関数を最小化 (または最大化) するような空隙率の分布を求めることで，
  最適な流路形状を計算します．



- OpenFOAM には，トポロジー最適化計算用のソルバーとして ***adjointShapeOptimizationFoam*** が実装されています．

1. ***adjointShapeOptimizationFoam*** は，
   - 次式を支配方程式とする**定常非圧縮性**流れ計算において，

$$\begin{cases} \nabla \cdot (\boldsymbol{u}\,\boldsymbol{u}) + \alpha \boldsymbol{u} = -\nabla p + \nabla \cdot \left[\nu_{eff} \left(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T \right) \right] \\ \nabla \cdot \boldsymbol{u} = 0 \end{cases}$$

空隙率 (OpenFOAM では，***alpha*** と表現) と流速に比例する抵抗力をソース項として付加

   - 入口と出口の全圧差 I を目的関数とした場合に，

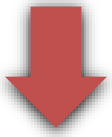
$$I = \int_{inlet} \left(p + \frac{1}{2} |\boldsymbol{u}|^2 \right) d\Gamma - \int_{outlet} \left(p + \frac{1}{2} |\boldsymbol{u}|^2 \right) d\Gamma$$

トポロジー最適化計算が可能です．

- OpenFOAM には，トポロジー最適化計算用のソルバーとして ***adjointShapeOptimizationFoam*** が実装されています．

2 ***adjointShapeOptimizationFoam*** は，勾配法の１つである最急降下法 (**Steepest decent method**) を最適化アルゴリズムに使用しています．

- 各設計変数 (設計変数の数 = セル数) に対する感度 $\partial I/\partial \alpha_i$ (i: セル番号) の計算が必要．



- 連続型の **Adjoint** (**Continuous adjoint**) 法を使用することにより，設計変数の数 (この場合はセル数) によらない計算コストで，感度計算が可能．

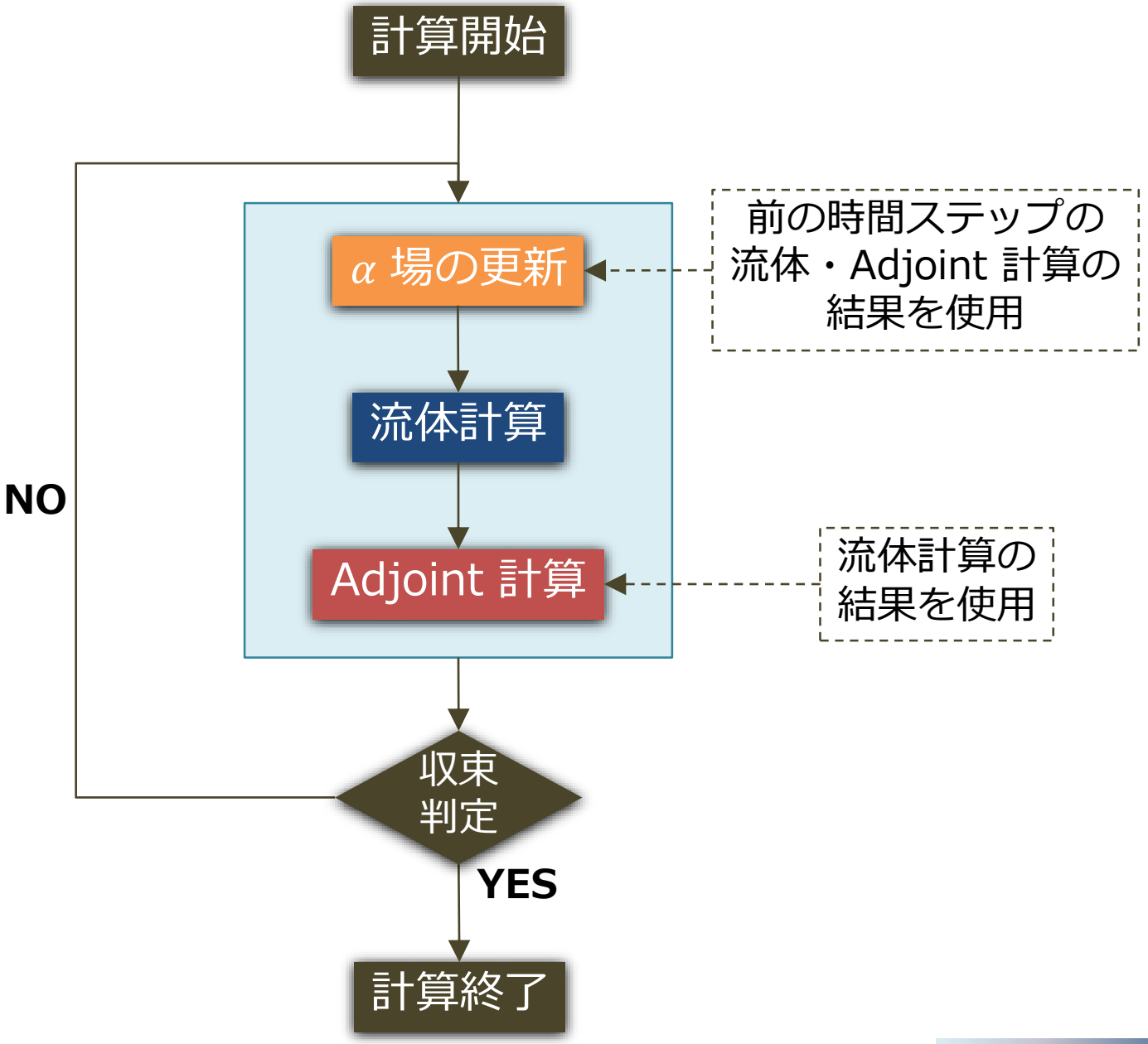
➢ ソースファイルの冒頭の説明

adjointShapeOptimizationFoam.C

```
Application
    adjointShapeOptimizationFoam

Description
    Steady-state solver for incompressible, turbulent flow of non-Newtonian
    fluids with optimisation of duct shape by applying "blockage" in regions
    causing pressure loss as estimated using an adjoint formulation.

    References:
    ¥verbatim
        "Implementation of a continuous adjoint for topology optimization of
         ducted flows"
        C. Othmer,
        E. de Villiers,
        H.G. Weller
        AIAA-2007-3947
        http://pdf.aiaa.org/preview/CDReadyMCFD07_1379/PV2007_3947.pdf
    ¥endverbatim

    Note that this solver optimises for total pressure loss whereas the
    above paper describes the method for optimising power-loss.
```

参考文献

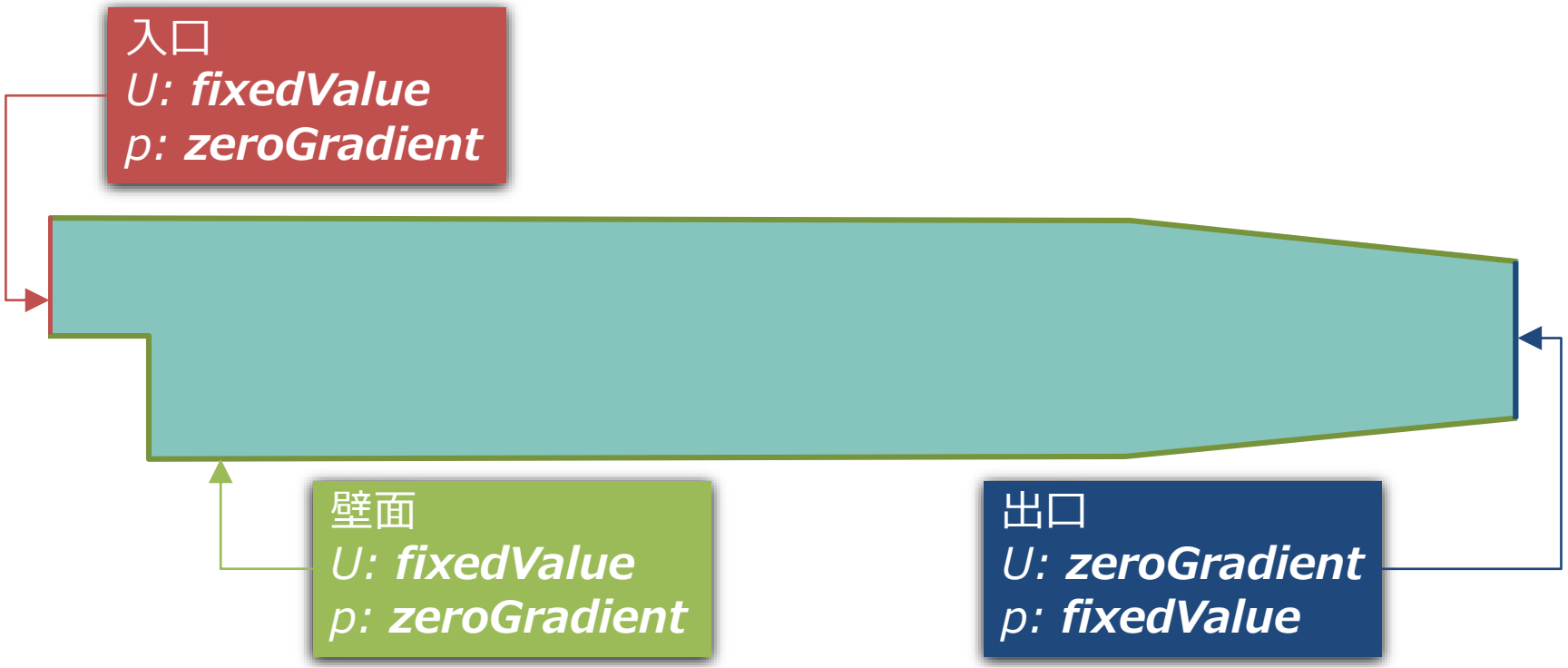
adjointShapeOptimizationFoam の計算の流れ



カスタマイズが必要なケース１

➢ 流体計算に使用する境界条件を変更する場合

- ***adjointShapeOptimizationFoam*** は，流体計算において，
  入口で流速規定，出口で圧力規定条件を使用することを想定しています．



- これ以外の境界条件を使用する場合には，Adjoint 方程式の境界条件の
  カスタマイズが必要です．

カスタマイズが必要なケース２

➢ 目的関数を変更する場合

- ***adjointShapeOptimizationFoam*** が対応している目的関数は，
  入口と出口の全圧差です．

$$I = \int_{inlet} \left(p + \frac{1}{2}|\boldsymbol{u}|^2 \right) d\Gamma - \int_{outlet} \left(p + \frac{1}{2}|\boldsymbol{u}|^2 \right) d\Gamma$$

- これ以外の目的関数を使用したトポロジー最適化計算を行う場合には，
  目的関数の評価式が，**境界積分** or **体積積分**で表されるのかにより，
  下記のようなカスタマイズが必要です．

目的関数の表現	**カスタマイズの必要性の有無**	
	Adjoint 方程式	境界条件
境界積分	不要	必要
体積積分	必要	場合による

カスタマイズが必要なケース２

➢ 目的関数を変更する場合

- ***adjointShapeOptimizationFoam*** が対応している目的関数は，
  入口と出口の全圧差です．

$$I = \int_{inlet} \left(p + \frac{1}{2}|\boldsymbol{u}|^2 \right) d\Gamma - \int_{outlet} \left(p + \frac{1}{2}|\boldsymbol{u}|^2 \right) d\Gamma$$

- これ以外の目的関数を使用したトポロジー最適化計算を行う場合には，
  目的関数の評価式が，**境界積分** or **体積積分**で表されるのかにより，
  下記のようなカスタマイズが必要です．

目的関数の表現	カスタマイズの必要性の有無	
	Adjoint 方程式	境界条件
境界積分	不要	必要
体積積分	必要	場合による

この講習では，このケースを扱います

カスタマイズが必要なケース３

- 流体計算の支配方程式を変更する場合

  - ***adjointShapeOptimizationFoam*** が対応しているのは,
    支配方程式が次式で与えられる定常非圧縮性流れ計算の場合です.

$$\begin{cases} \nabla \cdot (\boldsymbol{u}\,\boldsymbol{u}) + \alpha \boldsymbol{u} = -\nabla p + \nabla \cdot \left[\nu_{eff} \left(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T \right) \right] \\ \nabla \cdot \boldsymbol{u} = 0 \end{cases}$$

  - エネルギー保存則を考慮する場合や非定常計算の場合のように,
    流体計算の支配方程式が上記の式と異なる場合には,
    Adjoint 方程式およびその境界条件の両方のカスタマイズが必要です.

カスタマイズが必要なケース４

- 乱流モデルの渦粘性の変分も考慮する場合

  - ***adjointShapeOptimizationFoam*** は，乱流モデルにより計算される渦粘性の空隙率の変化に伴う変分を無視するという **Frozen turbulence** 近似を採用しています．

  - 渦粘性の変分を考慮する場合には，乱流変数 (k，ε，ω etc.) の輸送方程式に対応する Adjoint 方程式も解く必要があるので，Adjoint 乱流モデルの実装が必要です．

  - 乱流計算の場合には，Frozen turbulence の近似を使用せずに，乱流モデルの Adjoint 方程式も解く方が，感度の計算精度が向上します．

myAdjointShapeOptimizationFoam の作成

- チュートリアルを実行する前に，計算に使用するソルバーのカスタマイズを行います．
  ⇒ ***myAdjointShapeOptimizationFoam*** の作成

- ***adjointShapeOptimizationFoam*** からの変更点

  - 入口と出口の全圧差が減少することを確認するために，時間ステップ毎にこの値を計算してファイル出力します．

  - Adjoint 変数の境界条件の実装が (Othmer et al., 2008) と対応していない部分があるので，これを対応するように変更します．

myAdjointShapeOptimizationFoam の作成 (cont'd)

- OpenFOAM の環境設定を読み込んだターミナルを起動します.

- ***myAdjointShapeOptimizationFoam*** のディレクトリを作成し,
  カスタマイズに必要な準備を行います.

実行コマンド

```
$ foam

$ cp -r --parents applications/solvers/incompressible/adjointShapeOptimizationFoam ¥
> $WM_PROJECT_USER_DIR

$ cd $WM_PROJECT_USER_DIR/applications/solvers/incompressible

$ mv adjointShapeOptimizationFoam myAdjointShapeOptimizationFoam

$ cd myAdjointShapeOptimizationFoam

$ mv adjointShapeOptimizationFoam.C myAdjointShapeOptimizationFoam.C

$ wclean
```

myAdjointShapeOptimizationFoam の作成 (cont'd)

- ***Make/files*** ファイルを置き換えコマンド ***sed*** を使用して編集します.

実行コマンド

```
$ sed -i -e "s/adjointShape/myAdjointShape/g" Make/files

$ sed -i -e "s/FOAM_APPBIN/FOAM_USER_APPBIN/g" Make/files
```

編集後の *Make/files*

```
adjointOutletPressure/adjointOutletPressureFvPatchScalarField.C
adjointOutletVelocity/adjointOutletVelocityFvPatchVectorField.C
myAdjointShapeOptimizationFoam.C

EXE = $(FOAM_USER_APPBIN)/myAdjointShapeOptimizationFoam
```

myAdjointShapeOptimizationFoam の作成 (cont'd)

- ***myAdjointShapeOptimizationFoam.C*** ファイルを開いて，
  下記の２箇所を編集します．

- 編集箇所１：全圧差の値を出力するファイル Objective.dat の設定

myAdjointShapeOptimizationFoam.C

```
79      simpleControl simple(mesh);
80
81      OFstream out("Objective.dat");
82
83      // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
84
85      Info<< "¥nStarting time loop¥n" << endl;
```

追加１

- 編集箇所２：全圧差の計算を行うヘッダーファイルのインクルード

```
221         turbulence->correct();
222
223         #include "calcObjective.H"
224
225         runTime.write();
```

追加２

このヘッダーファイルは，次ページで作成します．

- 上記の変更が終了したら，***myAdjointShapeOptimizationFoam.C***
  ファイルを上書き保存します．

- ***calcObjective.H*** ファイルを新規作成し，下記を記述します(次頁に続く).

calcObjective.H

```
// Calculate total pressure difference
scalar inletTotP = 0.0;
scalar outletTotP = 0.0;

forAll(mesh.boundary(), patchI)
{
    const polyPatch& pp = mesh.boundary()[patchI].patch();

    if (pp.name() == "inlet")
    {
        inletTotP =
            gSum
            (
                (p.boundaryField()[patchI] + 0.5*magSqr(U.boundaryField()[patchI]))
               *mesh.magSf().boundaryField()[patchI]
            );

    }
    else if (pp.name() == "outlet")
    {
        outletTotP =
            gSum
            (
                (p.boundaryField()[patchI] + 0.5*magSqr(U.boundaryField()[patchI]))
               *mesh.magSf().boundaryField()[patchI]
            );
    }
}
```

➢ ***calcObjective.H*** ファイルを新規作成し，下記を記述します.

calcObjective.H

(前頁からの続き)

```
    scalar totPDiff = (inletTotP - outletTotP)/0.001;

    out<< runTime.timeName() << " " << totPDiff << endl;
```

- pitzDaily チュートリアルは，厚みが 0.001 m の 2 次元モデルなので，totPDiff の計算において 0.001 で割っています.

- 入口と出口境界の名前をそれぞれ “inlet” と “outlet” に固定しているので，これらをファイルで設定できるようにして，汎用性の向上にトライしてみてください.

myAdjointShapeOptimizationFoam の作成 (cont'd)

➢ Adjoint 圧力の境界条件のソースコードを論文と整合するように編集します.

- 編集箇所 1：***RASModel.H*** のインクルード

adjointOutletPressure/adjointOutletPressureFvPatchScalarField.C

```
#include "adjointOutletPressureFvPatchScalarField.H"
#include "addToRunTimeSelectionTable.H"
#include "fvPatchMapper.H"
#include "volFields.H"
#include "surfaceFields.H"
#include "RASModel.H"
```

追加

- ***updateCoeffs***() 関数の編集 (次頁に続く)

```
void Foam::adjointOutletPressureFvPatchScalarField::updateCoeffs()
{
    if (updated())
    {
        return;
    }

    const fvsPatchField<scalar>& phip =
        patch().lookupPatchField<surfaceScalarField, scalar>("phi");
```

myAdjointShapeOptimizationFoam の作成 (cont'd)

- ***updateCoeffs***() 関数の編集 (前頁の続き)

adjointOutletPressure/adjointOutletPressureFvPatchScalarField.C

```
    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");
```

追加と変更

```
    const incompressible::RASModel& rasModel =
        db().lookupObject<incompressible::RASModel>("RASProperties");

    scalarField nueff = rasModel.nuEff()().boundaryField()[patch().index()];
    scalarField U_n = phip/patch().magSf();
    scalarField Ua_n = phiap/patch().magSf();
    scalarField Uac_n = Uap.patchInternalField() & patch().nf();
    scalarField snGradUan = patch().deltaCoeffs()*(Ua_n - Uac_n);

    operator==((Up & Uap) + (Ua_n - 1.0)*U_n + nueff*snGradUan);

    fixedValueFvPatchScalarField::updateCoeffs();
}
```

myAdjointShapeOptimizationFoam の作成 (cont'd)

➢ Adjoint 流速の境界条件のソースコードを論文と整合するように編集します.

- 編集箇所 1 : ***RASModel.H*** のインクルード

adjointOutletVelocity/adjointOutletVelocityFvPatchVectorField.C

```
#include "adjointOutletPressureFvPatchScalarField.H"
#include "addToRunTimeSelectionTable.H"
#include "fvPatchMapper.H"
#include "volFields.H"
#include "surfaceFields.H"
#include "RASModel.H"
```

追加

- ***updateCoeffs***() 関数の編集 (次頁に続く)

```
void Foam::adjointOutletVelocityFvPatchVectorField::updateCoeffs()
{
    if (updated())
    {
        return;
    }

    const fvsPatchField<scalar>& phiap =
        patch().lookupPatchField<surfaceScalarField, scalar>("phia");
```

- ***updateCoeffs***() 関数の編集 (前頁の続き)

adjointOutletVelocity/adjointOutletVelocityFvPatchVectorField.C

```
 const fvPatchField<vector>& Up =
        patch().lookupPatchField<volVectorField, vector>("U");

    const incompressible::RASModel& rasModel =
        db().lookupObject<incompressible::RASModel>("RASProperties");

    scalarField nueff = rasModel.nuEff()().boundaryField()[patch().index()];
    scalarField nueDelta = nueff*patch().deltaCoeffs();

    scalarField U_n = Up & patch().nf();
    vectorField U_t = Up - patch().nf()*U_n;
    vectorField Uac_t = patchInternalField()
                    - (patchInternalField() & patch().nf())*patch().nf();

    vectorField Ua_t = (nueDelta*Uac_t + U_t)/(U_n + nueDelta);

    vectorField::operator=(phiap*patch().Sf()/sqr(patch().magSf()) + Ua_t);

    fixedValueFvPatchVectorField::updateCoeffs();
}
```

myAdjointShapeOptimizationFoam の作成 (cont'd)

- 必要な修正が完了したので，***myAdjointShapeOptimizationFoam*** をコンパイルします.

実行コマンド

```
$ wmake
```

- エラーなくコンパイルできていれば，
  実行ファイル ***myAdjointShapeOptimizationFoam*** が，
  $FOAM_USER_APPBIN に生成されます.

確認コマンド

```
$ ls $FOAM_USER_APPBIN
```

チュートリアルの実行

- ケースディレクトリへ移動

実行コマンド

```
$ run
$ cp -r $FOAM_TUTORIALS $FOAM_RUN
$ cd tutorials/incompressible/adjointShapeOptimizationFoam
$ cp -r pitzDaily mypitzDaily
$ cd mypitzDaily
```

- ***constant/transportProperties*** ファイルを開いて，***lambda*** の値を -1e5 に変更して，上書き保存します．

constant/transportProperties

```
transportModel  Newtonian;

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

lambda          lambda   [0 -2 1 0 0 0 0] -1e5;
alphaMax        alphaMax [0 0 -1 0 0 0 0] 200.0;
```

- **最小化**問題を考える場合： ***lambda < 0*** に設定
- **最大化**問題を考える場合： ***lambda > 0*** に設定
- 現在考えているのは，全圧差の最小化なので，負の値を設定しています．

チュートリアルの実行

- ***0/Ua*** ファイルを開いて，inlet の境界条件を下記のように変更し，上書き保存します．

0/Ua

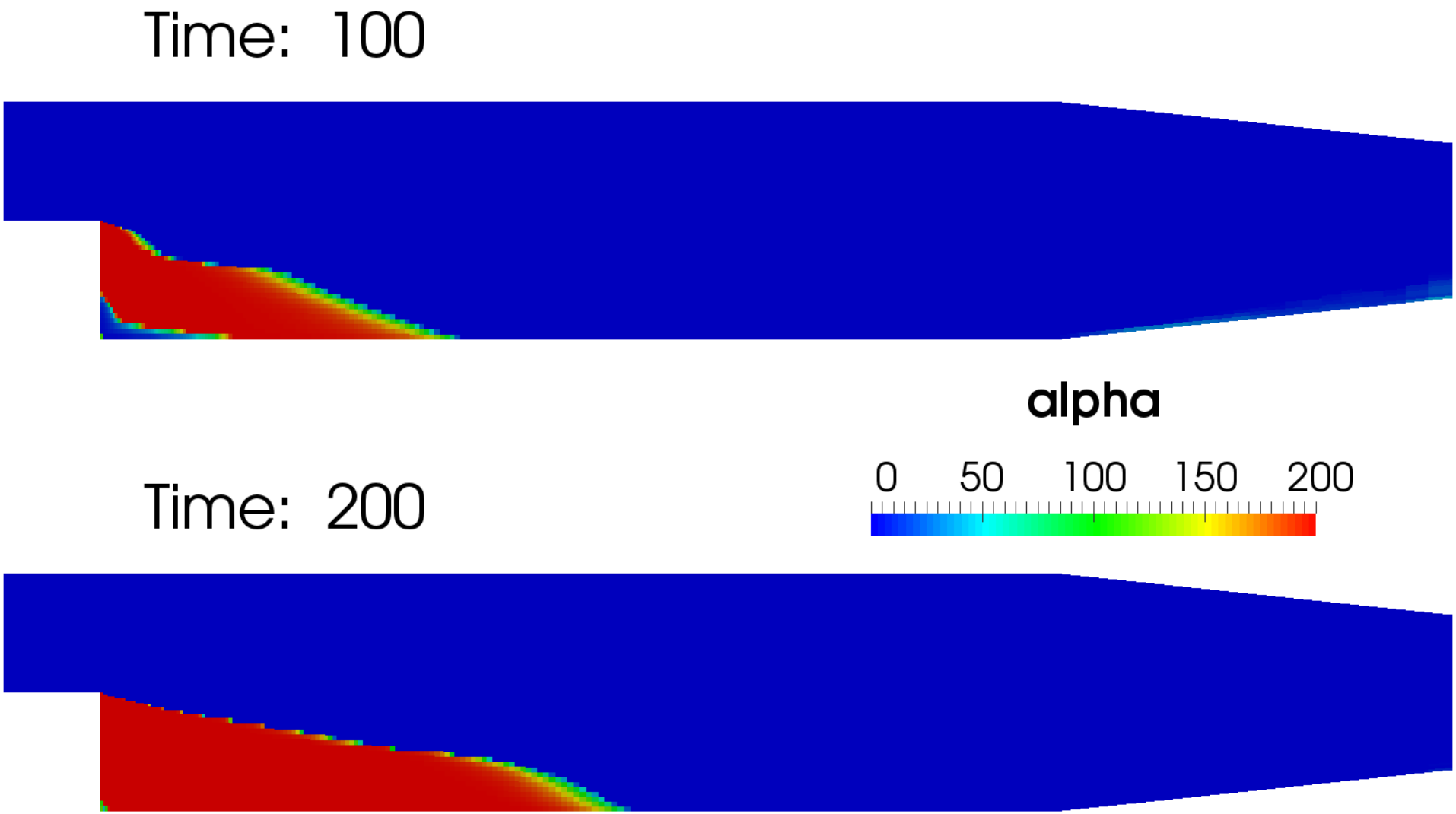
```
    inlet
    {
        type            fixedValue;
        value           uniform (1 0 0);
    }
```

- 計算格子を生成して，ソルバーを実行します．

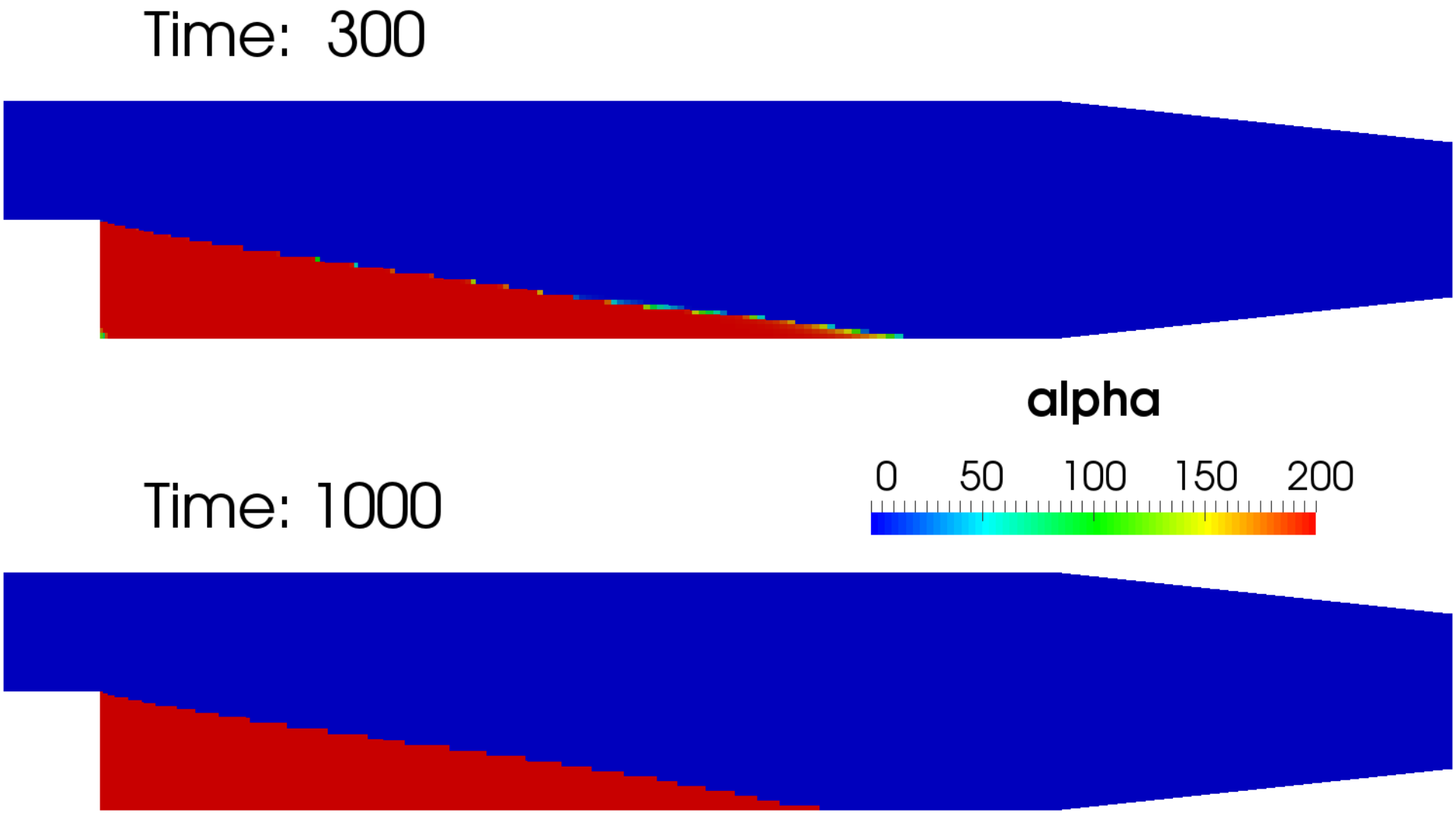
実行コマンド

```
$ blockMesh
$ myAdjointShapeOptimizationFoam
```

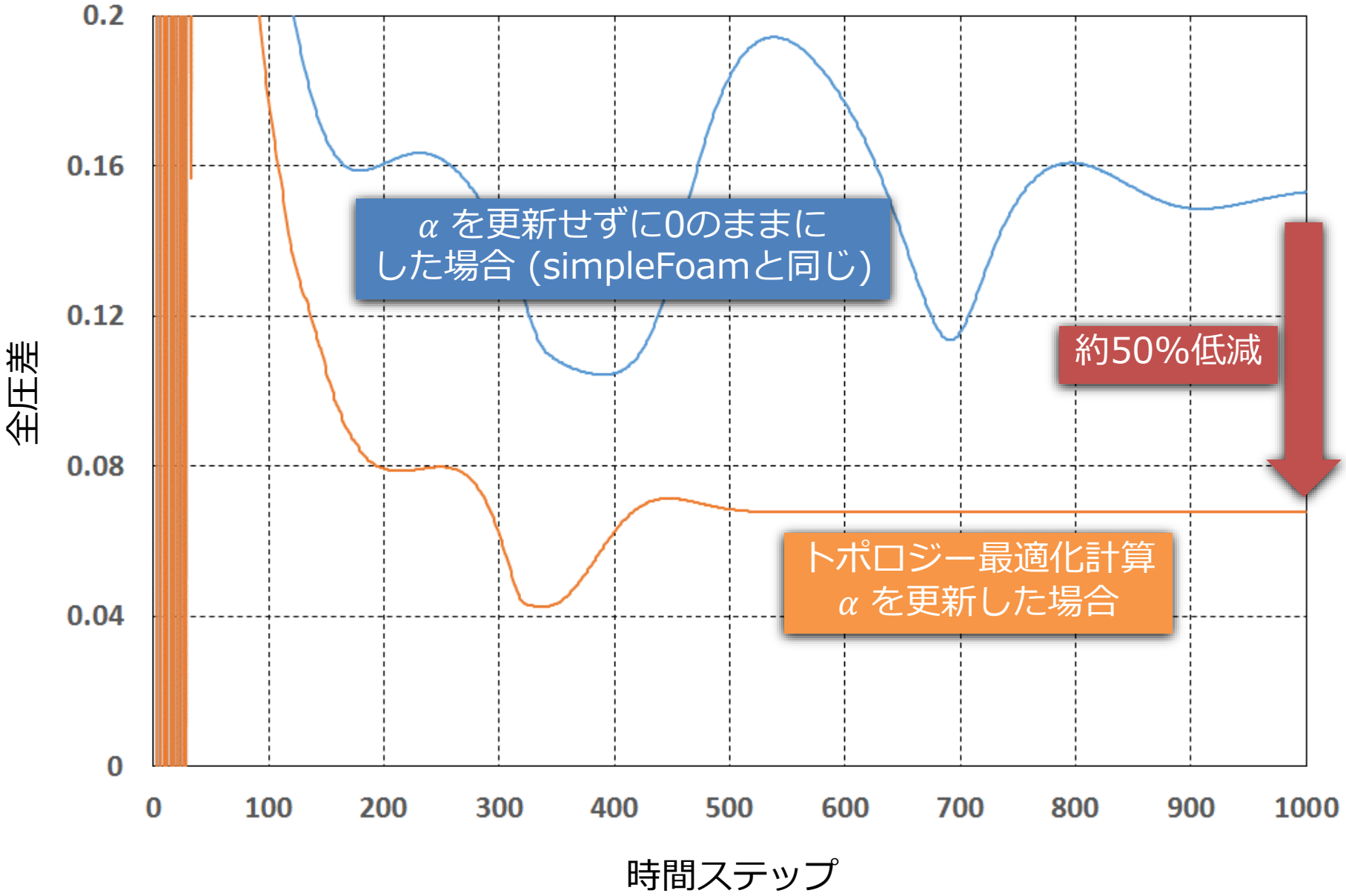
mypitzDaily チュートリアルの計算結果｜空隙率 *alpha* の分布

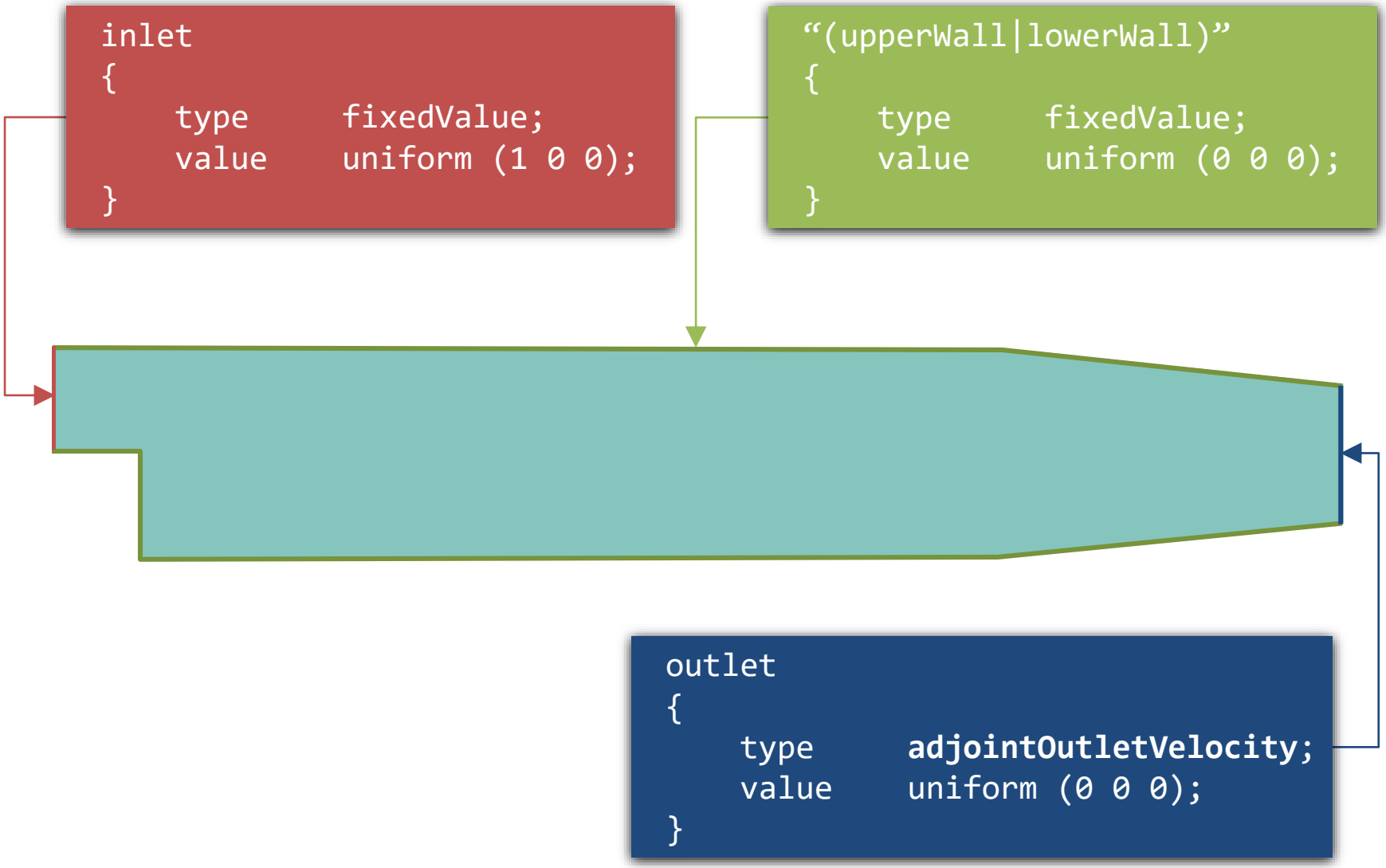


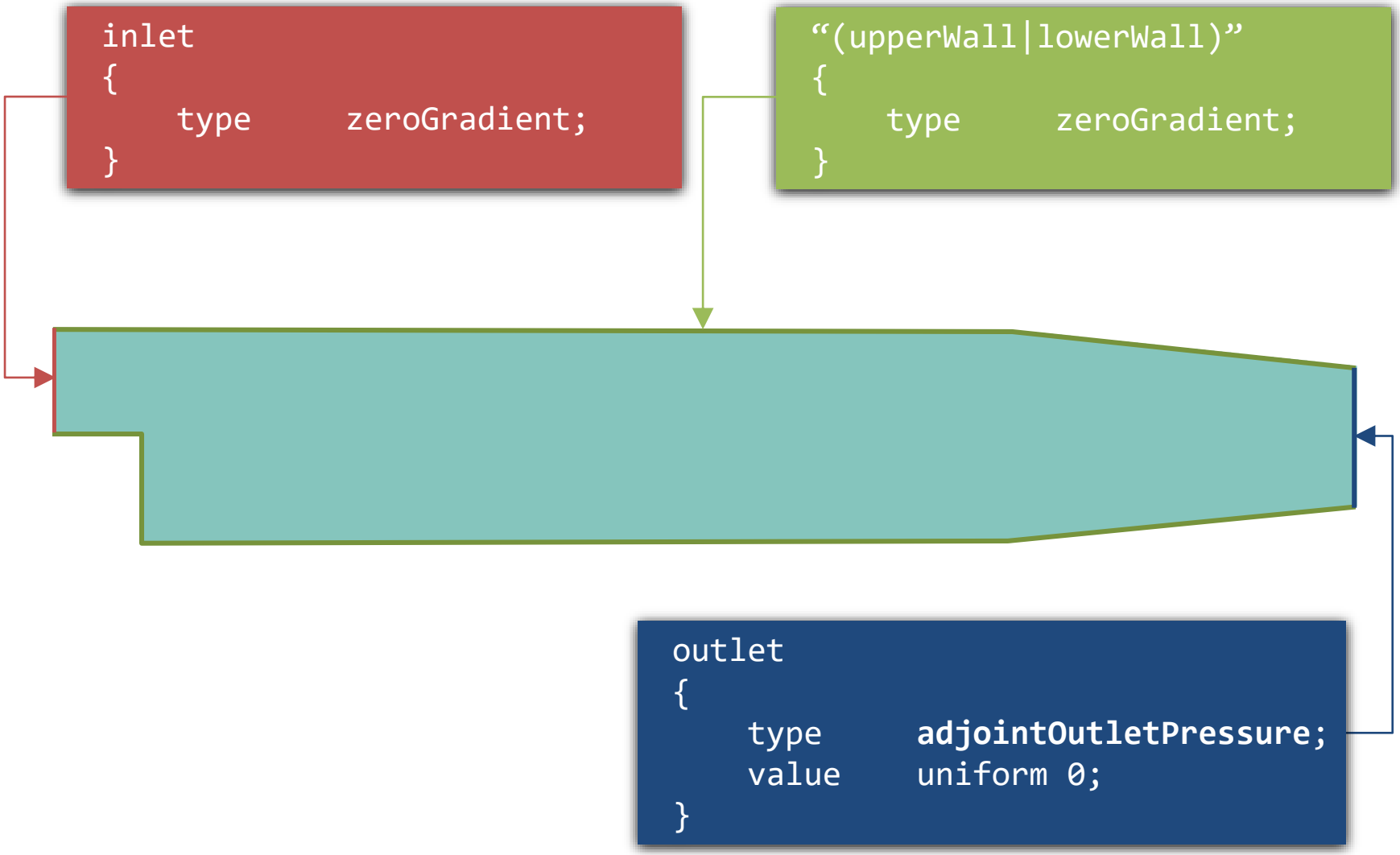
mypitzDaily チュートリアルの計算結果｜空隙率 *alpha* の分布



mypitzDaily チュートリアルの計算結果｜全圧差の履歴




inlet
{
type fixedValue;
value uniform (1 0 0);
}
“(upperWall|lowerWall)”
{
type fixedValue;
value uniform (0 0 0);
}
outlet
{
type adjointOutletVelocity;
value uniform (0 0 0);
}


inlet
{
type zeroGradient;
}
“(upperWall|lowerWall)”
{
type zeroGradient;
}
outlet
{
type adjointOutletPressure;
value uniform 0;
}

- 入口の境界条件 (Othmer et al., 2008)

$$\begin{cases} \hat{\boldsymbol{u}}_t = \boldsymbol{0} \\ \hat{u}_n = -\dfrac{\partial I_\Gamma}{\partial p} \\ \boldsymbol{n} \cdot \nabla q = 0 \end{cases}$$

ここで，現在考えている目的関数は，次式なので，

$$I = \int_{inlet} \left(p + \frac{1}{2} |\boldsymbol{u}|^2 \right) d\Gamma - \int_{outlet} \left(p + \frac{1}{2} |\boldsymbol{u}|^2 \right) d\Gamma$$

Adjoint 流速の境界法線方向成分は，次のようになります．

$$\hat{u}_n = -\frac{\partial}{\partial p} \left(p + \frac{1}{2} |\boldsymbol{u}|^2 \right) = -1$$

- 壁面の境界条件 (Othmer et al., 2008)

$$\begin{cases} \hat{\boldsymbol{u}}_t = \mathbf{0} \\ \hat{u}_n = -\dfrac{\partial I_\Gamma}{\partial p} \\ \boldsymbol{n} \cdot \nabla q = 0 \end{cases}$$

流体計算において，入口と壁面境界の両方とも，
- 流速規定
- 圧力の法線方向勾配0

条件の使用を想定しているため，Adjoint 変数が満たすべき条件も共通.

壁面では，$I_\Gamma = 0$ なので，法線方向成分も 0 になります.

$$\hat{u}_n = 0$$

- 出口の境界条件 (Othmer et al., 2008)

$$\begin{cases} \hat{p} = \boldsymbol{u} \cdot \hat{\boldsymbol{u}} + u_n \hat{u}_n + \nu(\boldsymbol{n} \cdot \nabla)\hat{u}_n + \dfrac{\partial I_\Gamma}{\partial u_n} \\ \mathbf{0} = u_n \hat{\boldsymbol{u}}_t + \nu(\boldsymbol{n} \cdot \nabla)\hat{\boldsymbol{u}}_t + \dfrac{\partial I_\Gamma}{\partial \boldsymbol{u}_t} \end{cases}$$

目的関数として全圧差を考えると，次式が得られます．

$$\begin{cases} \hat{p} = \boldsymbol{u} \cdot \hat{\boldsymbol{u}} + u_n \hat{u}_n + \nu(\boldsymbol{n} \cdot \nabla)\hat{u}_n - u_n \\ \mathbf{0} = u_n \hat{\boldsymbol{u}}_t + \nu(\boldsymbol{n} \cdot \nabla)\hat{\boldsymbol{u}}_t - \boldsymbol{u}_t \end{cases}$$

オリジナルのコードでは，赤字で表示した項が省略されています．

$$\begin{cases} \hat{p} = \boldsymbol{u} \cdot \hat{\boldsymbol{u}} + u_n \hat{u}_n + \nu(\boldsymbol{n} \cdot \nabla)\hat{u}_n - u_n \\ \boldsymbol{0} = u_n \hat{\boldsymbol{u}}_t + \nu(\boldsymbol{n} \cdot \nabla)\hat{\boldsymbol{u}}_t - \boldsymbol{u}_t \end{cases}$$

２つ目の式を，以下のように近似して，Adjoint 流速 Ua の境界接線成分の計算に使用しています．

$$\boldsymbol{0} = u_n \hat{\boldsymbol{u}}_t + \frac{\nu}{\Delta}(\hat{\boldsymbol{u}}_t - \hat{\boldsymbol{u}}_{tc}) - \boldsymbol{u}_t$$

$$\left(u_n + \frac{\nu}{\Delta}\right)\hat{\boldsymbol{u}}_t = \frac{\nu}{\Delta}\hat{\boldsymbol{u}}_{tc} + \boldsymbol{u}_t$$

$$\hat{\boldsymbol{u}}_t = \frac{\frac{\nu}{\Delta}\hat{\boldsymbol{u}}_{tc} + \boldsymbol{u}_t}{u_n + \frac{\nu}{\Delta}}$$

ここから先は受講者の方限定

完成までもうしばらく
お待ちください.

第７章　補足事項

value の設定の必要性の有無

➢ ***value*** の設定が必須な条件の例： ***pressureInletVelocity*** 境界条件

src/finiteVolume/fields/fvPatchFields/derived/pressureInletVelocity/pressureInletVelocityFvPatchVectorField.C

```
Foam::pressureInletVelocityFvPatchVectorField::
pressureInletVelocityFvPatchVectorField
(
    const fvPatch& p,
    const DimensionedField<vector, volMesh>& iF,
    const dictionary& dict
)
:
    fixedValueFvPatchVectorField(p, iF),
    phiName_(dict.lookupOrDefault<word>("phi", "phi")),
    rhoName_(dict.lookupOrDefault<word>("rho", "rho"))
{
    fvPatchVectorField::operator=(vectorField("value", dict, p.size()));
}
```

- この境界条件は，境界値の計算に流束 ***phi*** を使用します．
- 変数ファイルを読み込む順番は，流速 ***U*** の方が，流束 ***phi*** よりも先です (***createFields.H***)．

流速 ***U*** ファイルの読み込み時には，オブジェクト ***phi*** は定義されておらず，境界値の計算のために，***updateCoeffs***() を呼び出すことができません (次頁)．したがって，計算開始時の境界値 ***value*** の設定を必須にしています．

value の設定の必要性の有無

- 前頁のコンストラクタを下記のように変更すると，実行時にエラーが発生してしまいます．

  [変更前] fvPatchVectorField::operator=(vectorField("value", dict, p.size()));
  ⇒ [変更後] updateCoeffs();

src/finiteVolume/fields/fvPatchFields/derived/pressureInletVelocity/pressureInletVelocityFvPatchVectorField.C

```
void Foam::pressureInletVelocityFvPatchVectorField::updateCoeffs()
{
    if (updated())
    {
        return;
    }

    const surfaceScalarField& phi =
        db().lookupObject<surfaceScalarField>(phiName_);

    const fvsPatchField<scalar>& phip =
        patch().patchField<surfaceScalarField, scalar>(phi);
```

まだ，オブジェクト ***phi*** が定義されていないため，***lookupObject*** に失敗してしまいます．

value の設定の必要性の有無

- ***value*** の設定が不要な条件の例： ***surfaceNormalFixedValue*** 境界条件

src/finiteVolume/fields/fvPatchFields/derived/surfaceNormalFixedValue/surfaceNormalFixedValueFvPatchVectorField.C

```
Foam::surfaceNormalFixedValueFvPatchVectorField::
surfaceNormalFixedValueFvPatchVectorField
(
    const fvPatch& p,
    const DimensionedField<vector, volMesh>& iF,
    const dictionary& dict
)
:
    fixedValueFvPatchVectorField(p, iF),
    refValue_("refValue", dict, p.size())
{
    fvPatchVectorField::operator=(refValue_*patch().nf());
}
```

境界値 ***value*** を読み込まなくても，
計算開始時の境界値を正確に計算可能

0/U

```
    myPatch
    {
        type            surfaceNormalFixedValue;
        refValue        uniform -10; // 10 INTO the domain
    }
```

value の設定の必要性の有無

➢ ***value*** の設定が任意な条件の例： ***rotaingWallVelocity*** 境界条件

src/finiteVolume/fields/fvPatchFields/derived/rotatingWallVelocity/rotatingWallVelocityFvPatchVectorField.C

```
Foam::rotatingWallVelocityFvPatchVectorField::
rotatingWallVelocityFvPatchVectorField
(
    const fvPatch& p,
    const DimensionedField<vector, volMesh>& iF,
    const dictionary& dict
)
:
    fixedValueFvPatchField<vector>(p, iF),
    origin_(dict.lookup("origin")),
    axis_(dict.lookup("axis")),
    omega_(DataEntry<scalar>::New("omega", dict))
{
    if (dict.found("value"))
    {
        fvPatchField<vector>::operator=
        (
            vectorField("value", dict, p.size())
        );
    }
```

計算開始時において，境界値 ***value*** が設定されている場合 (リスタート時など) には，その値を読み込み，計算に使用

(次ページに続く)

value の設定の必要性の有無

- ***value*** の設定が任意な条件の例： ***rotaingWallVelocity*** 境界条件

src/finiteVolume/fields/fvPatchFields/derived/rotatingWallVelocity/rotatingWallVelocityFvPatchVectorField.C

```
    else
    {
        // Evaluate the wall velocity
        updateCoeffs();
    }
}
```

計算開始時において，境界値 ***value*** が設定されていない場合には，
メンバ関数 ***updateCoeffs***() を呼んで，
境界値を計算

0/U

```
    shaft
    {
        type            rotatingWallVelocity;
        origin          (0 0 0);
        axis            (0 0 1);
        omega           -5;
        // valueが設定されていれば，計算開始時には
        // その値が境界値として使用される．
    }
```

参考文献

米倉一男，寒野善博，格子ボルツマン法を用いた流れ場のトポロジー最適化において部分的に Newton 法を用いることによる収束速度の向上，日本機械学会論文集 (2015)

C. Othmer, E. de Villiers and H.G. Weller, Implementation of a continuous adjoint for topology optimization of ducted flows, AIAA-2007-3947 (2007).

Ulf Nilsson, Description of adjointShapeOptimizationFoam and how to implement new objective functions http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2013/UlfNilsson/reportAdjoint.pdf (accessed 2016-01-10)