

！ DRAFT ! 2013/11/11

OpenFOAM(R)ソースコード入門pt1 熱伝導方程式の解法から有限体積 法の実装について考える

後編: laplacianFoamでの実装

2013/11/17 オープンCAE勉強会@富山
富山県立大学 中川慎二

#ミニ講習会: OpenFOAM(R)ソースコード入門pt1 熱伝導方程式の解法(laplacianFoam)から有限体積法の実装について考える#

* OpenFOAM のソースコードでは, 基礎式を偏微分方程式の形で記述する. OpenFOAM内部では, 有限体積法を使ってこの微分方程式を解いている. どのようにして, 有限体積法に基づく離散化が実現されているのか, 最もシンプルな拡散方程式(熱伝導方程式)の場合を例として, 実装方法を考えていく.

* 2013年6月のオープンCAE勉強会@富山で, ソースコードの眺め方について講習を実施した. この時は, 使われている技術を断片的に見るに留まった. 今回は, 1つのソルバについて, じっくりとソースコードを眺めたい. まず, 簡単な形状の問題を手作業で解き, その際に出てきた変数などを再認識した上で, ソースコードと照らし合わせる.

* 1次元熱伝導方程式を有限体積法によって離散化し, 手作業で解く. この過程で必要な式変形などを確認する.

* 熱伝導方程式を解くソルバ "laplacianFoam" のソースコードを見ながら, 上記手作業で出てきた式が, どのように使われている(コーディングされている)かを読み解く.

* 与えた偏微分方程式から行列が作られる過程を確認する. しかし, 行列の解法には踏み込まない.

* 第11回 オープンCAE勉強会@富山 での講習資料(OpenFOAMソースコード解読へ向けて pt1_v1.pdf)をご一読ください.

前編のおさらいと後編のねらい

- 偏微分方程式 → 有限体積法で離散化 → セル間距離や物性値などで定まる係数列 (aE , aW , aP など) から行列式 → 解
- OpenFOAMでは、この部分がソルバ・レベルでは、巧妙に隠蔽されている。(知らなくても使える)
- OpenFOAMで作られる行列式は、どうなっているのか？ どうやって作られるのか？ Schemeを実行時に選べるのはどういう仕組みか？ を調べる

シミュレーションの流れ

- 偏微分方程式



- 離散方程式



- 行列式



- 解

プログラムの階層構造

- Level 0, Top:
 - 偏微分方程式 ソルバー
- Level 1:
 - 離散化・有限体積法 finiteVolume
- Level 2:
 - 行列 fvMatrices, lduMatrix
- Level 3:
 - 基礎的部品 OpenFOAM
 - 単位, 時間, リスト, メモリ, OS関連

laplacianFoam

Application laplacianFoam

Description Solves a simple Laplace equation, e.g. for thermal diffusion in a solid.

非定常 拡散方程式

$$\frac{\partial T}{\partial t} - \text{div}(\Gamma \text{ grad}T) = 0$$

$$\begin{aligned} \int_V \frac{\partial T}{\partial t} dV - \int_V \text{div}(\Gamma \text{ grad}T) dV \\ = \int_V \frac{\partial T}{\partial t} dV - \int_S (\Gamma \text{ grad}T) \cdot \mathbf{n} dS = 0 \end{aligned}$$

$$[A][T]=[b]$$

laplacianFoamで使うソースコード

- ソルバ ディレクトリ
 - OpenFOAM-2.2.2/application/solver/basic/laplacianFoam
- srcディレクトリ(ソルバディレクトリ/Make/optionsで指定)
 - OpenFOAM-2.2.2/src/finiteVolume
- 全ソルバ共通 src
 - OpenFOAM-2.2.2/src/OpenFOAM
 - OpenFOAM-2.2.2/src/OSspecific/POSIX

Level 0: メイン部分の概説

```
// 多くのソルバーで共通するヘッダーファイルの読み込み。OpenFOAMの基
// 盤的な機能を有効にする。
#include "fvCFD.H"
// simple法の機能を使うためのヘッダーファイルを読み込む。
#include "simpleControl.H"

// ***** //

int main(int argc, char *argv[])
{
// 多くのソルバーで共通するヘッダーファイルの読み込み。OpenFOAMの基
// 盤的な機能を有効にする。
#include "setRootCase.H"
#include "createTime.H"
#include "createMesh.H"
// 変数 温度場T, 拡散係数DT, 設定ディクショナリ transportProperties, を作
// 成して, ファイルから読み込む。
#include "createFields.H"

// simple法のためのクラスから, simple法をコントロールするためのオブジェ
// クトsimpleを作成
simpleControl simple(mesh);

// ***** //

Info<< "\nCalculating temperature distribution\n" << endl;

// シンプル法のオブジェクトを使って, 繰り返し回数をコントロールする
while (simple.loop())
{
Info<< "Time = " << runTime.timeName() << nl << endl;

// 非直交性補正が有効な場合のみ実行。
while (simple.correctNonOrthogonal())
```

```
{
// 非定常拡散方程式から線形代数式を生成して, 解く。
// 非定常項と拡散項をimplicit(陰的)に解く。fvm
solve
(
fvm::ddt(T) - fvm::laplacian(DT, T)
);
}

// 結果の出力
#include "write.H"

Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
<< " ClockTime = " << runTime.elapsedClockTime() << " s"
<< nl << endl;
} // end of the simple loop

Info<< "End\n" << endl;

return 0;
}
```


行列

$$a_P T_P = a_E T_E + a_W T_W + S_u$$

$$[A][T]=[b]$$

$$\begin{bmatrix} a_{p1} & -a_{E1} & 0 & 0 \\ -a_{W2} & a_{P2} & -a_{E2} & 0 \\ 0 & -a_{W3} & a_{P3} & -a_{E3} \\ 0 & 0 & -a_{W4} & a_{P4} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = \begin{bmatrix} S_{u1} \\ S_{u2} \\ S_{u3} \\ S_{u4} \end{bmatrix}$$

- 偏微分方程式の各項から, これら行列に入る係数が出てくる。
- 項毎に行列を作り, まとめることで, 最終的に解きたい行列式を作成する。

行列

$$\begin{bmatrix} a_{p1} & -a_{E1} & 0 & 0 \\ -a_{W2} & a_{P2} & -a_{E2} & 0 \\ 0 & -a_{W3} & a_{P3} & -a_{E3} \\ 0 & 0 & -a_{W4} & a_{P4} \end{bmatrix} =$$

$$\begin{bmatrix} a_{P1_ddt} & 0 & 0 & 0 \\ 0 & a_{P2_ddt} & 0 & 0 \\ 0 & 0 & a_{P3_ddt} & 0 \\ 0 & 0 & 0 & a_{P4_ddt} \end{bmatrix} + \begin{bmatrix} a_{E1} & -a_{E1} & 0 & 0 \\ -a_{W2} & a_{W2} + a_{E2} & -a_{E2} & 0 \\ 0 & -a_{W3} & a_{W3} + a_{E3} & -a_{E3} \\ 0 & 0 & -a_{W4} & a_{W4} \end{bmatrix}$$

非定常項

拡散項

$$\begin{bmatrix} S_{u1} \\ S_{u2} \\ S_{u3} \\ S_{u4} \end{bmatrix} = \begin{bmatrix} S_{u1_ddt} \\ S_{u2_ddt} \\ S_{u3_ddt} \\ S_{u4_ddt} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

拡散項

非定常項

よく使うクラス

	非フィールド値 場所によらず一定	フィールド値 場所によって値が変わる	
		セル面での値	セル体積(中心)での値
スカラー	<code>dimensionedScalar</code> <code>dimensioned<scalar></code>	<code>surfaceScalarField</code> <code>GeometricField<scalar, fvsPatchField,</code> <code>surfaceMesh></code>	<code>volScalarField</code> <code>GeometricField<scalar, fvPatchField,</code> <code>volMesh></code>
ベクトル	<code>dimensionedVector</code> <code>dimensioned<vector></code>	<code>surfaceVectorField</code> <code>GeometricField<vector, fvsPatchField,</code> <code>surfaceMesh></code>	<code>volVectorField</code> <code>GeometricField<vector, fvPatchField,</code> <code>volMesh></code>

T と DT の定義 : createFields.H

volScalarField T

```
(  
  IOObject  
  (  
    "T",  
    runtime.timeName(),  
    mesh,  
    IOObject::MUST_READ,  
    IOObject::AUTO_WRITE  
  ),  
  mesh  
);
```

dimensionedScalar DT

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

laplacianFoam.C メインループ

```
while (simple.loop())
```

```
{
```

```
Info<< "Time = " << runTime.timeName() << nl << endl,
```

```
while (simple.correctNonOrthogonal())
```

```
{
```

```
  solve
```

```
  (
```

```
    fvm::ddt(T) - fvm::laplacian(DT, T)
```

```
  );
```

```
}
```

```
#include "write.H"
```

```
Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
```

```
  << " ClockTime = " << runTime.elapsedClockTime() << " s"
```

```
  << nl << endl;
```

```
}
```

行列式を作成(各項ごとの行列を作成した後, 加減算)し, 解く

非定常項
引数: volScalarField
fvmDdt.C

拡散項
引数: dimensionedScalar と volScalarField
fvmLaplacian.C

どちらも戻り値
tmp<fvMatrix<Type>>

fvmDdt.C

```
template<class Type>  
tmp<fvMatrix<Type>>  
ddt  
(  
    const GeometricField<Type, fvPatchField, volMesh>& vf  
)  
{  
    return fv::ddtScheme<Type>::New  
    (  
        vf.mesh(),  
        vf.mesh().ddtScheme("ddt(" + vf.name() + ')')  
    )().fvmDdt(vf);  
}
```

scalar

参照渡し: vf は、読み出し側変数の別名となる。同一のものを操作することになる。これを変更すると、もとの変数の内容も変わる。

volScalarField T

fvSchemesファイルで設定した ddtSchemeの fvmDdtが実行される。

ddtScheme

- Euler, localEuler, CrankNicholson, backward, steadyState など (UserGuide 4.4.6)
- src/finiteVolume/finiteVolume/ddtSchemes にソースコードがある
- 今回は, 最もシンプルな Euler (implicit)を考える。

Euler implicit

$$\int_V \frac{\partial T}{\partial t} dV = \frac{(T_P V_P)^{new} - (T_P V_P)^{old}}{\Delta t}$$
$$= \frac{(V_P)^{new}}{\Delta t} T_P - \frac{(T_P V_P)^{old}}{\Delta t}$$

T_P の係数に追加

生成項に追加
(現時刻での温度とは
無関係)

EulerDdtScheme.C

```

template<class Type>
tmp<fvMatrix<Type> >
EulerDdtScheme<Type>::fvmDdt
(
    const GeometricField<Type, fvPatchField,
volMesh>& vf
)
{
    tmp<fvMatrix<Type> > tfvm
    (
        new fvMatrix<Type>
        (
            vf,
            vf.dimensions()*dimVol/dimTime
        )
    );

```

line 260

```

    if (mesh().moving())
    {
        fvm.source() =
rDeltaT*vf.oldTime().internalField()*mesh().VO();
    }
    else
    {
        fvm.source() =
rDeltaT*vf.oldTime().internalField()*mesh().V();
    }

    return tfvm;
}

```

$$\frac{(T_P V_P)^{old}}{\Delta t}$$

行列の生成項に(旧時刻での値 × セル体積 / タイムステップ)を入れる。Told * Vcell / Δt

```
scalar rDeltaT = 1.0/mesh().time().deltaTValue();
```

```
fvm.diag() = rDeltaT*mesh().V();
```

$$\frac{(V_P)^{new}}{\Delta t} T_P$$

行列の対角成分に(セル体積 / タイムステップ)を入れる。Vcell / Δt

行列

$$\begin{bmatrix} a_{p1} & -a_{E1} & 0 & 0 \\ -a_{W2} & a_{P2} & -a_{E2} & 0 \\ 0 & -a_{W3} & a_{P3} & -a_{E3} \\ 0 & 0 & -a_{W4} & a_{P4} \end{bmatrix} =$$

$$\begin{bmatrix} a_{P1_ddt} & 0 & 0 & 0 \\ 0 & a_{P2_ddt} & 0 & 0 \\ 0 & 0 & a_{P3_ddt} & 0 \\ 0 & 0 & 0 & a_{P4_ddt} \end{bmatrix} + \begin{bmatrix} a_{E1} & -a_{E1} & 0 & 0 \\ -a_{W2} & a_{W2} + a_{E2} & -a_{E2} & 0 \\ 0 & -a_{W3} & a_{W3} + a_{E3} & -a_{E3} \\ 0 & 0 & -a_{W4} & a_{W4} \end{bmatrix}$$

ここまでで設定済み
非定常項

ここから設定
拡散項

$$\begin{bmatrix} S_{u1} \\ S_{u2} \\ S_{u3} \\ S_{u4} \end{bmatrix} = \begin{bmatrix} S_{u1_ddt} \\ S_{u2_ddt} \\ S_{u3_ddt} \\ S_{u4_ddt} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

設定済み
非定常項

今回は考えない: 拡散項
非直交性補正では値が
入る

Laplacian

- schemeの選択肢は, Gauss だけ (UserGuide 4.4.4) ただし, 下記補間方法の選択肢がある
 - Gauss <interpolationScheme> <snGradScheme>
- 拡散係数の補間(interpolation) 方法の選択肢は, linear, upwind, limitedLinear, vanLeer, MUSCL など (UserGuide 4.4.1.2, Table 4.6)
 - 隣接するセル中心での値から, 面での値を補間する
- 面垂直方向勾配(snGrad)の補間(interpolation) 方法の選択肢は, corrected, uncorrected, limited ψ , bounded, fourth など (UserGuide 4.4.2, 4.4.4, Table 4.9)
- src/finiteVolume/finiteVolume/laplacianSchemes にソースコードがある
- 今回は, 最もシンプルな場合を考える。補間の詳細については触れない。

拡散項

ベクトルの内積

$$\int_S (\Gamma \text{grad}T) \cdot \mathbf{n} dS = \sum_f [(\Gamma \text{grad}T) \cdot \mathbf{S}_f] = (\Gamma \text{grad}T)_e \cdot \mathbf{S}_{f_e} + (\Gamma \text{grad}T)_w \cdot \mathbf{S}_{f_w}$$

$$= \left(\Gamma \frac{dT}{dx} \right)_e S_{f_e} - \left(\Gamma \frac{dT}{dx} \right)_w S_{f_w} = \Gamma_e \left(\frac{dT}{dx} \right)_e S_{f_e} - \Gamma_w \left(\frac{dT}{dx} \right)_w S_{f_w}$$

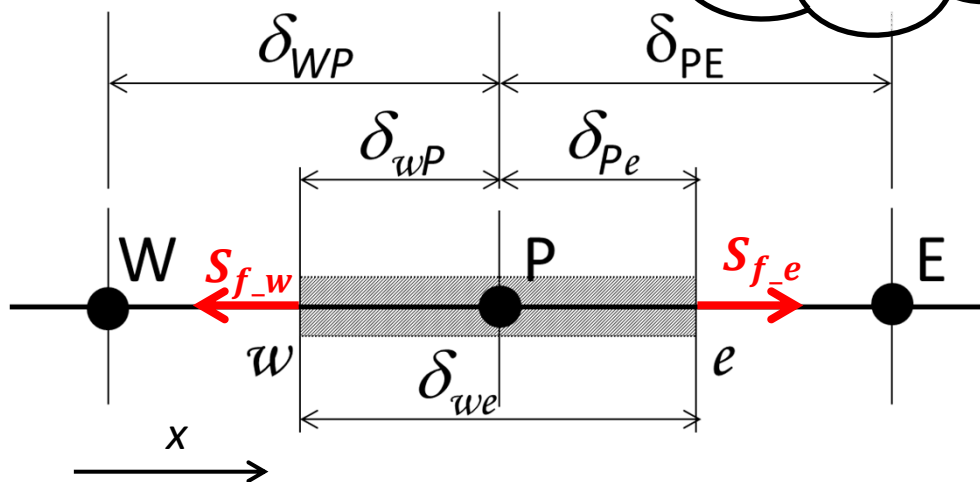
スカラー値の積

S_{f_w} が逆向きなので, 負

前編同様の1次元・直交を想定

snGradScheme

interpolationScheme



\mathbf{n} control volume界面での単位面積ベクトル
大きさ:1, 方向:面に垂直

\sum_f control volumeの全ての界面での和

$\mathbf{S}_f = S_f \mathbf{n}$
control volume界面での面積ベクトル
大きさ:面積 S_f , 方向:面に垂直

拡散項

$$\int_S (\Gamma \text{grad}T) \cdot \mathbf{n} dS = \sum_f \Gamma_f [(\text{grad}T)_f \cdot \mathbf{n}] S_f$$

面fにおけるΓの値(scalar) . 面を挟むセル中心の値から補間する. interpolationScheme

$$= \sum_f [\Gamma_f (\text{grad}T)_f \cdot (S_f \mathbf{n})]$$

面fの面積

$$= \sum_f [\Gamma_f (\text{grad}T)_f \cdot \mathbf{S}_f]$$

面fにおける(gradT)の値(vector)の, 面に垂直な成分 snGradScheme

$$\left(\frac{dT}{dx}\right)_f = \frac{T_N - T_O}{\delta_f} = \frac{T_N - T_O}{|C_N - C_O|}$$

T_N temperature of neighbor cell

T_O temperature of owner cell

C_N position vector: center of neighbor cell

C_O position vector: center of owner cell

↑ 基準: 直交の場合. 非直交の場合には, 補正項を追加

\sum_f control volumeの全ての界面での和

\mathbf{n} control volume界面での単位面積ベクトル
大きさ: 1, 方向: 面に垂直

$$\mathbf{n} = \frac{\mathbf{S}_f}{S_f}$$

$$\mathbf{S}_f = S_f \mathbf{n}$$

control volume界面での面積ベクトル
大きさ: 面積 S_f , 方向: 面に垂直

laplacian(DT, T)からlaplacianSchemeまで

line 179 スカラ値 (dimensiondScalar) DT と スカラ一場 (volScalarField) T とが引数 ; スカラ場 gamma を作成して, すべてにDTの値を入れる.

```
tmp<fvMatrix<Type> > laplacian  
( const dimensioned<GType>& gamma,  
  const GeometricField<Type, fvPatchField, volMesh>& vf  
)
```

line303 DTをスカラ一場 (surfaceScalarField)にした gamma と スカラ一場 (volScalarField) T とが引数 ; laplacianの名前を付ける

```
tmp<fvMatrix<Type> >  
laplacian  
( const GeometricField<GType, fvsPatchField, surfaceMesh>& gamma,  
  const GeometricField<Type, fvPatchField, volMesh>& vf  
)
```

line 271 スカラ一場 (surfaceScalarField) gamma と スカラ一場 (volScalarField) T と 名前 laplacian(DT,T) が引数 ; laplacianSchemeを実行する

```
tmp<fvMatrix<Type> >  
laplacian  
(  
  const GeometricField<GType, fvPatchField, volMesh>& gamma,  
  const GeometricField<Type, fvPatchField, volMesh>& vf,  
  const word& name  
)
```

```
template<class Type >  
tmp< fvMatrix< Type > > laplacian (const GeometricField< Type, fvPatchField, volMesh > &vf, const word &name)
```

```
template<class Type >  
tmp< fvMatrix< Type > > laplacian (const GeometricField< Type, fvPatchField, volMesh > &vf)
```

```
template<class Type >  
tmp< fvMatrix< Type > > laplacian (const zero &, const GeometricField< Type, fvPatchField, volMesh > &vf, const word &name)
```

```
template<class Type >  
tmp< fvMatrix< Type > > laplacian (const zero &, const GeometricField< Type, fvPatchField, volMesh > &vf)
```

```
template<class Type >  
tmp< fvMatrix< Type > > laplacian (const one &, const GeometricField< Type, fvPatchField, volMesh > &vf, const word &name)
```

```
template<class Type >  
tmp< fvMatrix< Type > > laplacian (const one &, const GeometricField< Type, fvPatchField, volMesh > &vf)
```

```
template<class Type , class GType >  
tmp< fvMatrix< Type > > laplacian (const dimensioned< GType > &gamma, const GeometricField< Type, fvPatchField, volMesh > &vf, const word &name)
```

```
template<class Type , class GType >  
tmp< fvMatrix< Type > > laplacian (const dimensioned< GType > &gamma, const GeometricField< Type, fvPatchField, volMesh > &vf)
```

```
template<class Type , class GType >  
tmp< fvMatrix< Type > > laplacian (const GeometricField< GType, fvPatchField, volMesh > &gamma, const GeometricField< Type, fvPatchField, volMesh > &vf, const word &name)
```

```
template<class Type , class GType >  
tmp< fvMatrix< Type > > laplacian (const tmp< GeometricField< GType, fvPatchField, volMesh > > &gamma, const GeometricField< Type, fvPatchField, volMesh > &vf, const word &name)
```

```
template<class Type , class GType >  
tmp< fvMatrix< Type > > laplacian (const GeometricField< GType, fvPatchField, volMesh > &gamma, const GeometricField< Type, fvPatchField, volMesh > &vf)
```

```
template<class Type , class GType >  
tmp< fvMatrix< Type > > laplacian (const tmp< GeometricField< GType, fvPatchField, volMesh > > &gamma, const GeometricField< Type, fvPatchField, volMesh > &vf)
```

```
template<class Type , class GType >  
tmp< fvMatrix< Type > > laplacian (const GeometricField< GType, fvsPatchField, surfaceMesh > &gamma, const GeometricField< Type, fvPatchField, volMesh > &vf, const word &name)
```

fvmLaplacian.C

```
template<class Type, class GType>
tmp<fvMatrix<Type> >
laplacian
(
    const dimensioned<GType>& gamma,
    const GeometricField<Type, fvPatchField, volMesh>& vf
)
{
    const GeometricField<GType, fvsPatchField, surfaceMesh> Gamma
    (
        IOobject
        (
            gamma.name(),
            vf.instance(),
            vf.mesh(),
            IOobject::NO_READ
        ),
        vf.mesh(),
        gamma
    );

    return fvm::laplacian(Gamma, vf);
}
```

line 179

引数を (GeometricField , GeometricField)
とするlaplacianを実行。

fvmLaplacian.C

```
template<class Type, class GType>  
tmp<fvMatrix<Type> >  
laplacian  
(  
    const GeometricField<GType, fvsPatchField, surfaceMesh>& gamma,  
    const GeometricField<Type, fvPatchField, volMesh>& vf  
)  
{  
    return fvm::laplacian  
    (  
        gamma,  
        vf,  
        "laplacian(" + gamma.name() + ',' + vf.name() + ')'  
    );  
}
```

line 303

引数を (GeometricField , GeometricField,
word) とする laplacian を再実行。

fvmLaplacian.C

```
template<class Type, class GType>
tmp<fvMatrix<Type> >
laplacian
(
    const GeometricField<GType, fvsPatchField, surfaceMesh>& gamma,
    const GeometricField<Type, fvPatchField, volMesh>& vf,
    const word& name
)
{
    return fv::laplacianScheme<Type, GType>::New
    (
        vf.mesh(),
        vf.mesh().laplacianScheme(name)
    )().fvmLaplacian(gamma, vf);
}
```

line 271

fvSchemesファイルで設定した
laplacianSchemeのfvmLaplacianが実行される。
laplacian(Dt,T) Gauss linear corrected;

gaussLaplacianScheme.C

```

template<class Type, class GType>
tmp<fvMatrix<Type> >
gaussLaplacianScheme<Type, GType>::fvmLaplacian
(
    const GeometricField<GType, fvsPatchField, surfaceMesh>& tfaceFluxCorrection
    gamma,
    const GeometricField<Type, fvPatchField, volMesh>& vf
)
{
    const fvMesh& mesh = this->mesh();

    const surfaceVectorField Sn(mesh.Sf()/mesh.magSf());

    const surfaceVectorField SfGamma(mesh.Sf() & gamma);
    const GeometricField<scalar, fvsPatchField, surfaceMesh>
    SfGammaSn
    (
        SfGamma & Sn
    );
    const surfaceVectorField SfGammaCorr(SfGamma -
    SfGammaSn*Sn);

    tmp<fvMatrix<Type> > tfvm = fvmLaplacianUncorrected
    (
        SfGammaSn,
        this->tsnGradScheme_().deltaCoeffs(vf),
        vf
    );
    fvMatrix<Type>& fvm = tfvm();

    tmp<GeometricField<Type, fvsPatchField, surfaceMesh> >
    tfaceFluxCorrection
    = gammaSnGradCorr(SfGammaCorr, vf);

    if (this->tsnGradScheme_().corrected())
    {
        tfaceFluxCorrection() +=
            SfGammaSn*this->tsnGradScheme_().correction(vf);
    }

    fvm.source() -=
    mesh.V()*fvc::div(tfaceFluxCorrection(), internalField());

    if (mesh.fluxRequired(vf.name()))
    {
        fvm.faceFluxCorrectionPtr() = tfaceFluxCorrection.ptr();
    }

    return tfvm;
}

```

line 154

SfGamma の面に垂直な成分(大きさ)

非直交性の補正

fvmLaplacianUncorrected を呼んで tfvm を作成。ここで行列を操作。

gaussLaplacianScheme.C

```
template<class Type, class GType>
tmp<fvMatrix<Type>>
gaussLaplacianScheme<Type,
GType>::fvmLaplacianUncorrected
(
    const surfaceScalarField& gammaMagSf,
    const surfaceScalarField& deltaCoeffs,
    const GeometricField<Type, fvPatchField,
volMesh>& vf
)
{
    tmp<fvMatrix<Type>> tfvm
    (
        new fvMatrix<Type>
        (
            vf,

            deltaCoeffs.dimensions()*gammaMagSf.dimensions()*
            vf.dimensions()
        )
    );
    fvMatrix<Type>& fvm = tfvm();
}
```

line 44

```
fvm.upper() =
deltaCoeffs.internalField()*gammaMagSf.internalField
();
fvm.negSumDiag();
境界での処理:省略
return tfvm;
}
```

行列の上三角に(セル間距離の逆数×ガンマ Γ ×セル面の面積)を入れる。 $\Gamma S/\delta$
対称性から, upperのみに代入。
自動的にlowerにも入る。

行列の対角成分に 周辺の係数を加える。(次のスライド)
LduMatrix::negSumDiag

IduMatrixOperations.C

```
void Foam::lduMatrix::negSumDiag()
```

line 50

```
{  
    const scalarField& Lower = const_cast<const lduMatrix&>(*this).lower();  
    const scalarField& Upper = const_cast<const lduMatrix&>(*this).upper();  
    scalarField& Diag = diag();
```

```
    const labelUList& l = lduAddr().lowerAddr();  
    const labelUList& u = lduAddr().upperAddr();
```

```
    for (register label face=0; face<l.size(); face++)
```

```
    {
```

```
        Diag[l[face]] -= Lower[face];
```

```
        Diag[u[face]] -= Upper[face];
```

```
    }
```

```
}
```

Diag 対角成分
Lower 下三角成分
Upper 上三角成分
l 下三角での場所を指す
u 上三角での場所を指す

$a_P = -(-a_E) + -(-a_W)$
下三角, 上三角には, 負号がついた係数が格納されていることに注意。

surfaceInterpolation.C

```

void Foam::surfaceInterpolation::makeDeltaCoeffs() const
{
    if (debug)
    {
        Pout<< "surfaceInterpolation::makeDeltaCoeffs() : "
            << "Constructing differencing factors array for face
gradient"
            << endl;
    }

    // Force the construction of the weighting factors
    // needed to make sure deltaCoeffs are calculated for
    parallel runs.
    weights();

    deltaCoeffs_ = new surfaceScalarField
    (
        IOobject
        (
            "deltaCoeffs",
            mesh_.pointsInstance(),
            mesh_,
            IOobject::NO_READ,
            IOobject::NO_WRITE,
            false // Do not register
        ),
        mesh_,
        dimless/dimLength
    );
    surfaceScalarField& DeltaCoeffs = *deltaCoeffs_;

    // Set local references to mesh data
    const volVectorField& C = mesh_.C();
    const labelUList& owner = mesh_.owner();
    const labelUList& neighbour = mesh_.neighbour();

    forAll(owner, facei)
    {
        DeltaCoeffs[facei] = 1.0/mag(C[neighbour[facei]] -
C[owner[facei]]);
    }

    forAll(DeltaCoeffs.boundaryField(), patchi)
    {
        DeltaCoeffs.boundaryField()[patchi] =
            1.0/mag(mesh_.boundary()[patchi].delta());
    }
}

```

最も単純な場合の例：
セル中心間の距離を δ とする

行列

$$\begin{bmatrix} a_{p1} & -a_{E1} & 0 & 0 \\ -a_{W2} & a_{P2} & -a_{E2} & 0 \\ 0 & -a_{W3} & a_{P3} & -a_{E3} \\ 0 & 0 & -a_{W4} & a_{P4} \end{bmatrix} =$$

$$\begin{bmatrix} a_{P1_ddt} & 0 & 0 & 0 \\ 0 & a_{P2_ddt} & 0 & 0 \\ 0 & 0 & a_{P3_ddt} & 0 \\ 0 & 0 & 0 & a_{P4_ddt} \end{bmatrix} + \begin{bmatrix} a_{E1} & -a_{E1} & 0 & 0 \\ -a_{W2} & a_{W2} + a_{E2} & -a_{E2} & 0 \\ 0 & -a_{W3} & a_{W3} + a_{E3} & -a_{E3} \\ 0 & 0 & -a_{W4} & a_{W4} \end{bmatrix}$$

ここまでで設定済み
非定常項

ここまでで設定済み
拡散項

$$\begin{bmatrix} S_{u1} \\ S_{u2} \\ S_{u3} \\ S_{u4} \end{bmatrix} = \begin{bmatrix} S_{u1_ddt} \\ S_{u2_ddt} \\ S_{u3_ddt} \\ S_{u4_ddt} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

設定済み
非定常項

今回は考えない:
* 境界条件の項はsolve()の中で追加される。
* 拡散項の非直交性補正でも値が入る

さらに詳しく・・・

- 今回は，手作業計算と同じように，セル中心間を結ぶベクトルが面と直交する状態を想定して説明した。
- 実際には，これらが直交しない場合もある。
- その時には，面での勾配，面での拡散係数の値の算出時に，非直交性の影響を考慮する必要がある。
- 考慮の方法は，補間スキームに依存する。

ソースコードを読み解くために

- 変数のタイプ(クラス)を意識
 - volScalarField, dimensionedScalar ? など
- Slow and steady wins the race
 - 少しずつ, 理解を深める
 - 小さな部分の積み重ね
 - 繰り返す
- 基礎を学習
 - 理論とソースの両方を学ぶ

- pTraits が primitive の基本
- scalar, vector などに応じて、pTraits の個別版が作成されている。
 - OpenFOAM/primitives/pTraits.H
 - OpenFOAM/primitives/Scalar.H