

OpenFOAMソースコードの読み方入門 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

2019年3月29, 30日 オープンCAE講習会@富山
富山県立大学 中川慎二

Disclaimer: OPENFOAM® is a registered trade mark of OpenCFD Limited, the producer of the OpenFOAM software and owner of the OPENFOAM® and OpenCFD® trade marks. This offering is not approved or endorsed by OpenCFD Limited.

注意

- OpenFOAMユーザーガイド, プログラマーズガイド, OpenFOAM Wiki, CFD Online, その他多くの情報を参考にしています。開発者, 情報発信者の皆様に深い謝意を表します。
- この講習内容は, 講師の個人的な経験 (主に, 卒 研究生等とのコードリーディング) から得た知識を共有するものです。この内容の正確性を保証することはできません。この情報を使用したことによって問題が生じた場合, その責任は負いかねますので, 予めご了承ください。

概要

- OpenFOAM の利用者を対象とし、OpenFOAMのソースコードの読み方の基本を学びます。
- OpenFOAMの中で最も基本的なソルバの1つ“laplacianFoam”（熱伝導方程式を解くソルバ）を例にとります。
- 実際にOpenFOAMのソースコードを見ながら、OpenFOAMのソースコードの特徴、ソースコード解読初心者が躓きやすい点などについても、解説します。

概要

- 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 (a_E , a_W , a_P 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, どんなもの? どうやって作られる? Schemeを実行時に選べるのはどういう仕組みか? を調べる

前編では…

- 1次元熱伝導方程式を有限体積法によって離散化し,行列を作成する.
- この行列を手作業で解き,温度分布が求められることを確認する.
- この過程で必要な式変形などを確認する.

後編では…

- 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,どのように実装されている(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.

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 (a_E, a_W, a_P 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 $\frac{\partial T}{\partial t} = \text{div}(\Gamma \text{grad}T) + S_T$
有限体積法
- 離散方程式 $a_P T_P = a_E T_E + a_W T_W + S_u$
係数列の計算
- 行列 $[A][T]=[b]$
行列の解法
- 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 fvMatrix, lduMatrix
- Level 3:
 - 基礎的部品 OpenFOAM
 - 単位, 時間, リスト, メモリ, OS関連, 並列化

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

Linuxの基礎:\$付きの文字列は, 変数を意味する。その中身を知りたい時は, 端末にて次のコマンドを実行する。(echoコマンド)

```
echo $FOAM_APP
```

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"
// 任意のソース項を含めるためのfvOptionsを使うためのヘッダーファイル
// 読み込み
#include "fvOptions.H"
// simple法の機能を使うため。
#include "simpleControl.H"
// ***** //
int main(int argc, char *argv[])
{
// 多くのソルバーに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 "setRootCaseLists.H"
#include "createTime.H"
#include "createMesh.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);
// 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"
// ***** //
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(runTime))
{
Info<< "Time = " << runTime.timeName() << nl << endl;
// 非直行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に相当する行列を作成する。
// 非定常項と拡散項をimplicit(陰的)に解く。fvm
```

fvScalarMatrix TEqn

```
(
    fvm::ddt(T) - fvm::laplacian(DT, T)
    ==
    fvOptions(T)
);
```

```
// 行列から解を求める。
```

```
fvOptions.constrain(TEqn);
TEqn.solve();
fvOptions.correct(T);
```

```
// 結果の出力
```

```
#include "write.H"
Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
<< " ClockTime = " << runTime.elapsedClockTime() << " s"
<< nl << endl;
```

OpenFOAMでの偏微分方程式の離散化
PG-35 Table 2.2

$$\frac{\partial T}{\partial t} - \nabla^2 (\Gamma T) = 0$$

C++の基礎: #include “ファイル名” の部分には, 指定したファイルの中身がそのまま貼付けられる。そのファイルの存在する場所は, コンパイル用設定で指定している。

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

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

$$\begin{bmatrix} S_{P1} & 0 & 0 & 0 \\ 0 & S_{P2} & 0 & 0 \\ 0 & 0 & S_{P3} & 0 \\ 0 & 0 & 0 & S_{P4} \end{bmatrix} - \begin{bmatrix} -a_{E1} & a_{E1} & 0 & 0 \\ a_{W2} & -a_{W2} - a_{E2} & a_{E2} & 0 \\ 0 & a_{W3} & -a_{W3} - a_{E3} & a_{E3} \\ 0 & 0 & a_{W4} & -a_{W4} \end{bmatrix}$$

非定常項

拡散項

$$\begin{bmatrix} S_{BC0} + S_{u1} \\ S_{u2} \\ S_{u3} \\ S_{BC5} + S_{u4} \end{bmatrix} = \begin{bmatrix} S_{u1_ddt} \\ S_{u2_ddt} \\ S_{u3_ddt} \\ S_{u4_ddt} \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} S_{BC0} \\ 0 \\ 0 \\ S_{BC5} \end{bmatrix}$$

非定常項

拡散項

境界条件

よく使うクラス

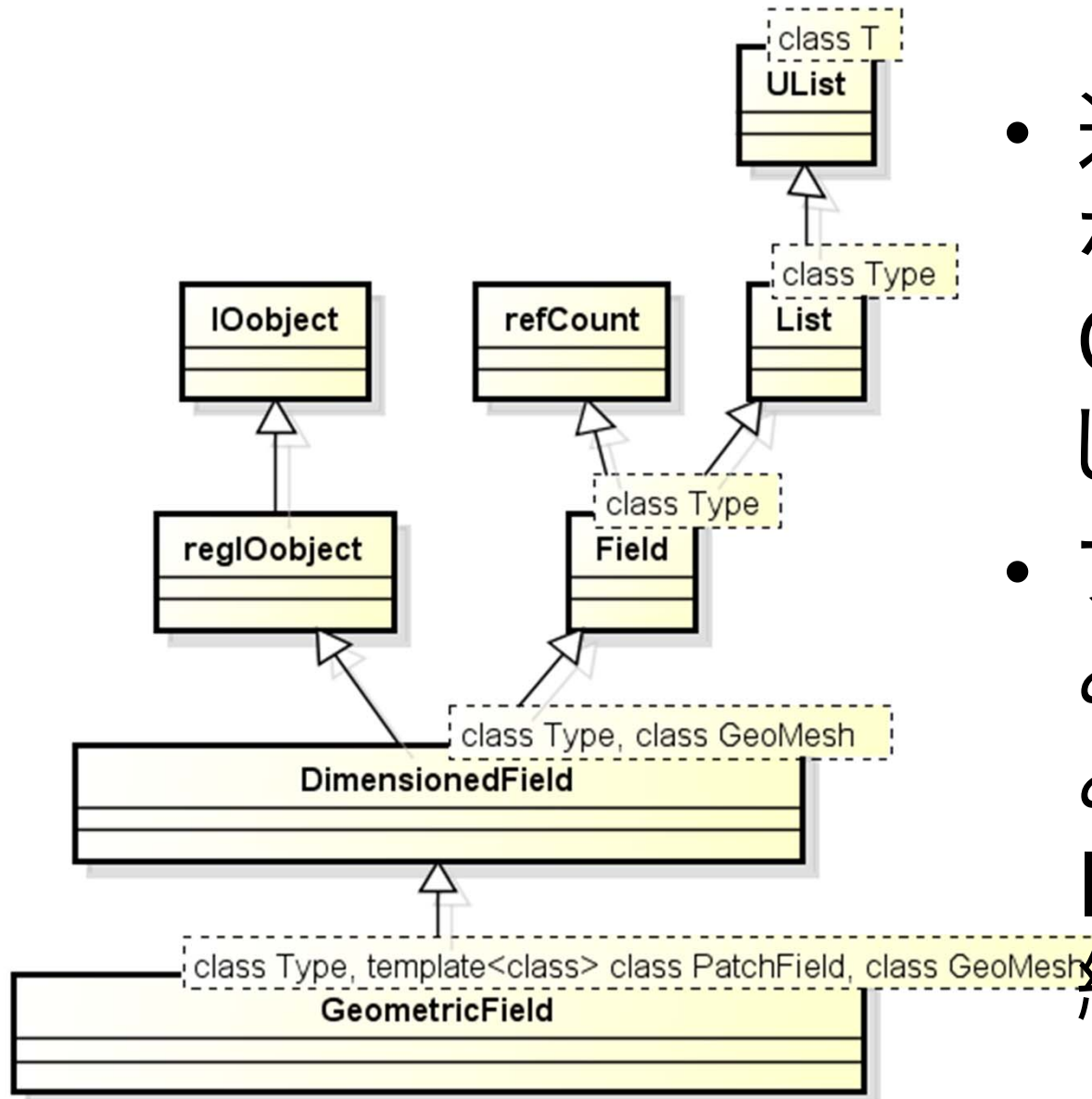
	非フィールド値 場所によらず一定	フィールド値 場所によって値が変わる	
		セル面での値	セル体積(中心)での値
スカラー	<code>dimensionedScalar</code> <code>dimensioned<scalar></code>	<code>surfaceScalarField</code> <code>GeometricField<scalar, fvsPatchField,</code> <code>surfaceMesh></code>	<code>volScalarField</code> <code>GeometricField<scalar, fvPatchField,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

上の行は, typedefで定義された別名
下の行が本来の定義

typedef

- 既存の型に 新しい名前(別名)を付ける
- コードが見やすくなる
- テンプレートなどを含んだ長い名前の型・クラスも,短く理解しやすい名前を付けることができる

GeometricFieldクラスの継承関係



- 速度U, 圧力p, 温度Tなどのデータは, GeometricField型として保存されている。
- ファイルへの書き込み/ファイルからの読み込みなどは, IObjectクラスから継承している。

TとDTの定義: createFields.H

volScalarField T

volScalarField T(IOobject, mesh);
Tという名前で、volScalarFieldクラスのオブジェクトを作成する。引数を2つ渡してコンストラクタ指定

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

dimensionedScalar DT();
DTという名前の dimensionedScalarクラスのオブジェクトを作成する。引数はIstream&。読み取った値で初期化する。

dimensioned (Istream & is)
Construct from Istream.
Definition at line 91 of file dimensionedType.C.

dimensionedScalar DT

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

transportPropertiesオブジェクトのlookup関数を使って、transportPropertiesディクショナリのDTという項目の値を読み込む。transportPropertiesは、IOdictionaryクラスのオブジェクト。

IOdictionary transportProperties
lookup関数の戻り値は、Istream&

transportPropertiesの定義： createFields.H

IOdictionary transportProperties

```
(  
  IOobject  
  (  
    "transportProperties",  
    runTime.constant(),  
    mesh,  
    IOobject::MUST_READ_IF_MODIFIED,  
    IOobject::NO_WRITE  
  )  
);
```

設定ファイル(ディクショナリ)クラス
のオブジェクトとして、
transportPropertiesを宣言している。

ディクショナリ名

格納場所

IOdictionary Class Reference

Public Member Functions inherited from dictionary

ITstream & lookup (const word &, bool recursive=false, bool patternMatch=true) const

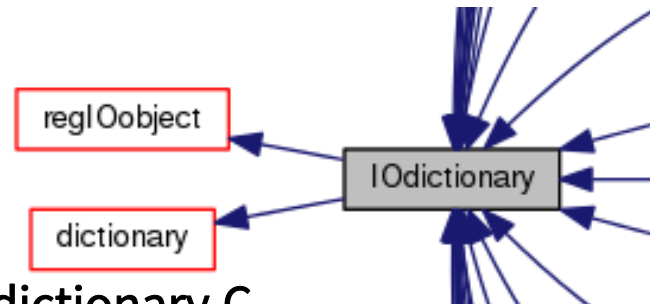
IOdictionary.H

54 class IOdictionary

55 :

56 public regIOobject,

57 public dictionary



dictionary.C

```
446 Foam::ITstream& Foam::dictionary::lookup
```

```
447 (
```

```
448     const word& keyword,
```

```
449     bool recursive,
```

```
450     bool patternMatch
```

```
451 ) const
```

```
452 {
```

```
453     return lookupEntry(keyword, recursive,
```

```
454     patternMatch).stream();
```

```
455 }
```

dictionary.H

```
316 ITstream& lookup
```

```
317 (
```

```
318     const word& keyword,
```

```
319     bool recursive=false,
```

```
320     bool patternMatch=true
```

```
321 ) const;
```

Class ITstream

Description: Input token stream.

```
class ITstream: public Istream,public tokenList
```

laplacianFoam.C メインループ

行列を作成(各項ごとの行列を作成した後, 加減算)

どちらも戻り値tmp<fvMatrix<Type>>

関数名

引数名

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

非定常項
引数: volScalarField
fvmDdt.C

拡散項
引数: dimensionedScalar と volScalarField
fvmLaplacian.C

fvmDdt.C

テンプレートクラス
を使う宣言

```
template<class Type>
```

戻り値
の型

```
tmp<fvMatrix<Type>>
```

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。

関数
名

```
ddt
```

引数

```
(  
)
```

```
const GeometricField<Type, fvPatchField, volMesh>& vf
```

関数の
中身

```
{
```

```
return fv::ddtScheme<Type>::New
```

volScalarField T

戻り値

```
(  
    vf.mesh(),  
    vf.mesh().ddtScheme("ddt(" + vf.name() + ')')  
)().fvmDdt(vf);
```

T

Newの戻り値 tmp< ddtScheme< Type >>
tmpクラスで演算子()のオーバーロード
該当スキームのポインタが戻る

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

fvmDdt.C

テンプレート	template<class Type >
戻り値型	tmp<fvMatrix<Type>>
関数名	ddt
引数	(const GeometricField< Type , fvPatchField, volMesh> &vf)
関数の中身	{ return fv::ddtScheme<Type>::New (vf.mesh(), vf.mesh().ddtScheme("ddt(" + vf.name() + ")"))) return EulerDdtScheme< Type >::fvmDdt(vf); } } Newの戻り値 tmp< ddtScheme< Type > > tmpクラスで演算子()のオーバーロード 該当スキームのポインタが戻る

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。

scalar

volScalarField T

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

Run-Time Selection

- OpenFOAM guide/runTimeSelection mechanism – OpenFOAMWiki
http://openfoamwiki.net/index.php/OpenFOAM_guide/runTimeSelection_mechanism
- Run-time type selection in OpenFOAM - the type name system – sourceflux
<http://www.sourceflux.de/blog/runtime-type-selection-openfoam/>

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。
 - クラス名 EulerDdtScheme

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

line 350

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

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

```
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
        )
    );

    fvMatrix<Type>& fvm = tfvm.ref();
```

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

        fvm.diag() = rDeltaT*mesh().Vsc();

        if (mesh().moving())
        {
            fvm.source() = rDeltaT*vf.oldTime().
primitiveField()*mesh().Vsc0();
        }
        else
        {
            fvm.source() = rDeltaT*vf.oldTime().
primitiveField()*mesh().Vsc();
        }

        return tfvm;
    }
```

fvm.source() = rDeltaT*vf.oldTime().primitiveField()*mesh().Vsc();

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

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

行列

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

ソースコード

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

$$\begin{bmatrix} S_{P1} & 0 & 0 & 0 \\ 0 & S_{P2} & 0 & 0 \\ 0 & 0 & S_{P3} & 0 \\ 0 & 0 & 0 & S_{P4} \end{bmatrix} - \begin{bmatrix} -a_{E1} & a_{E1} & 0 & 0 \\ a_{W2} & -a_{W2} - a_{E2} & a_{E2} & 0 \\ 0 & a_{W3} & -a_{W3} - a_{E3} & a_{E3} \\ 0 & 0 & a_{W4} & -a_{W4} \end{bmatrix}$$

ここまでで設定済み
非定常項

これから設定
拡散項

$$\begin{bmatrix} S_{BC0} + S_{u1} \\ S_{u2} \\ S_{u3} \\ S_{BC5} + S_{u4} \end{bmatrix} = \begin{bmatrix} S_{u1_ddt} \\ S_{u2_ddt} \\ S_{u3_ddt} \\ S_{u4_ddt} \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} S_{BC0} \\ 0 \\ 0 \\ S_{BC5} \end{bmatrix}$$

今回は考えない: 境界
条件に応じた値が入る

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 拡散項

- 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)
 - 隣接するセル中心での勾配の値から, 面での値を補間する
- 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

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

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

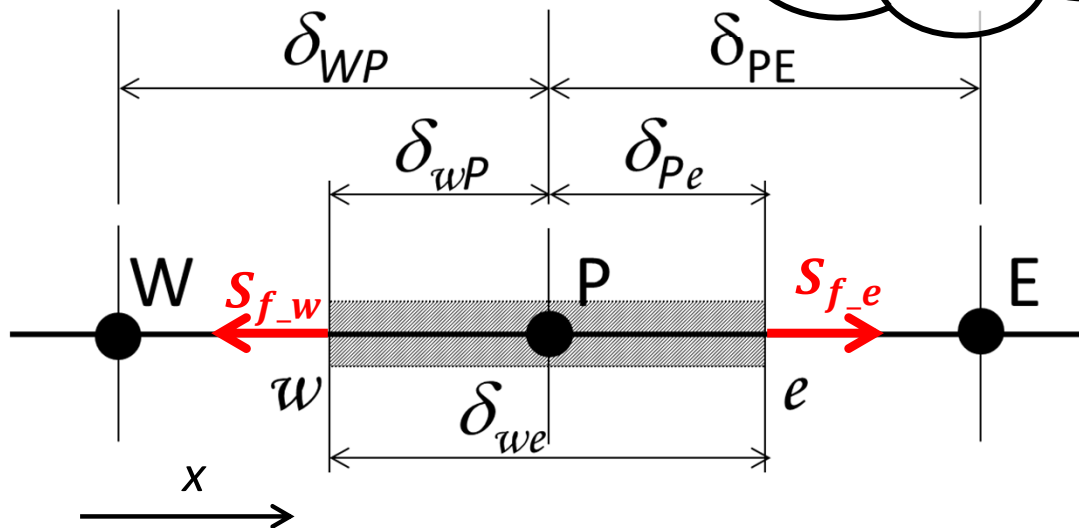
スカラー値の積

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



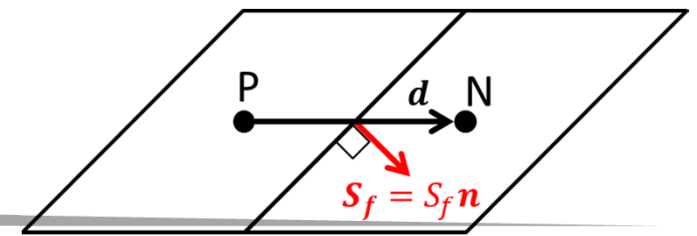
\mathbf{n} control volume界面での単位面積ベクトル
大きさ:1, 方向:面に垂直

\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



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

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

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

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

面fの面積

面fにおける $(\text{grad}T)$ の値(vector)と単位面ベクトルとの内積=面f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() 多数のオーバーロード

引数の数と種類によって多数あり

ソースコード `fvm::laplacian(DT, T)`

```
template<class Type >
```

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

```
laplacian (const GeometricField< Type, fvPatchField, volMesh > &vf)
```

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

```
laplacian (const zero &, const GeometricField< Type, fvPatchField, volMesh > &vf)
```

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

```
laplacian (const one &, const GeometricField< Type, fvPatchField, volMesh > &vf)
```

```
laplacian (const dimensioned< GType > &gamma, const GeometricField< Type, fvPatchField, volMesh > &vf, const word &n
```

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

```
laplacian (const GeometricField< GType, fvPatchField, volMesh > &gamma, const GeometricField< Type, fvPatchField, volM
```

```
laplacian (const tmp< GeometricField< GType, fvPatchField, volMesh > > &tgamma, const GeometricField< Type, fvPatchFi
```

他多数

引数の数と種類が合わない？

- 関数を検索するとき、「引数の数と種類が一致するものがない!」と困惑することがある。
- 引数のデフォルト値を,関数宣言時(*.H)に与える場合,使用時に省略可能であることに注意。
- デフォルト値を記入するのは宣言時だけであり,定義時(*.C)には不要であることに,さらに注意。

GeometricField.C

```
188 Foam::GeometricField<Type, PatchField, GeoMesh>::GeometricField
189 (
190   const IObject& io,
191   const Mesh& mesh,
192   const dimensionSet& ds,
193   const word& patchFieldType
194 )
```

GeometricField.H

```
287 GeometricField
288 (
289   const IObject&,
290   const Mesh&,
291   const dimensionSet&,
292   const word& patchFieldType = PatchField<Type>::calculatedType()
293 );
```

laplacian(DT, T)から laplacianSchemeまで

line 179 スカラ値 (dimensionalScalar) DT と スカラ場 (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. → 次の laplacian を呼び出す

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

line 303 DT を スカラ場 (surfaceScalarField) にした gamma と スカラ場 (volScalarField) T とが引数; laplacian の名前を付ける → 次の laplacian を呼び出す

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

line 271 スカラ場 (surfaceScalarField) gamma と スカラ場 (volScalarField) T と名前 laplacian(DT, T) が引数; laplacianScheme を実行する → ディクショナリで設定したスキーム

```
tmp<fvMatrix<Type>>  
laplacian  
(  
  const GeometricField<GType, fvPatchField, volMesh>& 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 (**dimensionedScalar**,
GeometricField) とする laplacian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 238

引数を (**GeometricField** , GeometricField)
とするlaplacian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>

line 154

tmp<fvMatrix<Type>>

gaussLaplacianScheme<Type, GType>::fvmLaplacian

```
(
  const GeometricField<GType, fvsPatchField, surfaceMesh>& 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
);
```

SfGamma の面に垂直な成分(大きさ)

```
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;
```

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> line 44
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.ref();
    
```

```

fvm.upper() =
deltaCoeffs.primitiveField()*gammaMagSf.primitiveField();
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

src¥OpenFOAM¥matrices¥LduMatrix¥LduMatrix¥
LduMatrixOperations.C

line 55

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

$a_P = -(-a_E) + -(-a_W)$
対角成分と下三角, 上三角とでは, 符号が逆になることに注意。

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]]);
    }

    surfaceScalarField::Boundary& deltaCoeffsBf =
    deltaCoeffs.boundaryFieldRef(),

    forAll(deltaCoeffsBf, patchi)
    {
        deltaCoeffsBf[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

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

$$\begin{bmatrix} S_{P1} & 0 & 0 & 0 \\ 0 & S_{P2} & 0 & 0 \\ 0 & 0 & S_{P3} & 0 \\ 0 & 0 & 0 & S_{P4} \end{bmatrix} - \begin{bmatrix} -a_{E1} & a_{E1} & 0 & 0 \\ a_{W2} & -a_{W2} - a_{E2} & a_{E2} & 0 \\ 0 & a_{W3} & -a_{W3} - a_{E3} & a_{E3} \\ 0 & 0 & a_{W4} & -a_{W4} \end{bmatrix}$$

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

$$\begin{bmatrix} S_{BC0} + S_{u1} \\ S_{u2} \\ S_{u3} \\ S_{BC5} + S_{u4} \end{bmatrix} = \begin{bmatrix} S_{u1_ddt} \\ S_{u2_ddt} \\ S_{u3_ddt} \\ S_{u4_ddt} \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} S_{BC0} \\ 0 \\ 0 \\ S_{BC5} \end{bmatrix}$$

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。
- 今回は, 境界条件の寄与については考えていない。実際には, 境界条件に応じた値が行列に追加される。

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, 繰り返す, 繰り返す
- 基礎を学習
 - 理論とソースの両方を学ぶ

参考資料

- OpenFOAM Programmers Guide, User Guide, ソースコード
- Imperial College 博士論文など
 - Hrvoje Jasak, Henrik Rusche, Franjo Juretic などなど
 - <http://powerlab.fsb.hr/ped/kturbo/OpenFOAM/docs/>
- PENGUINITIS サイト(OpenFOAMたんけんたい)
 - <http://penguinitis.g1.xrea.com/study/OpenFOAM/tankentai/index.html>
- 野崎さんSlideShare
 - <http://www.slideshare.net/fumiyanozaki96/>
 - OpenFOAM 空間の離散化と係数行列の取り扱い
- <http://openfoamwiki.net/>
- <http://www.cfd-online.com>

追記:runTimeSelectionの理解に 向けて

C++ のマクロ

マクロは、

ロベールのC++教室 第28章

<http://www7b.biglobe.ne.jp/~robe/cpphtml/html01/cpp01028.html>

- テキストを差し込むもの、
- 引数もとれる。

書式

```
#define <マクロ名> <差し込むテキスト>
```

注意: 改行したい時は、行の終わりに バックスラッシュ(\ or ¥) を付ける

トークン連結演算子 (##)

<https://docs.microsoft.com/ja-jp/cpp/preprocessor/token-pasting-operator-hash-hash?view=vs-2017>

- 二重シャープ記号または"トークン連結演算子 (##)、両方のオブジェクトのような関数に似たマクロで使用されて、「マージ」演算子と呼ばれることがあります。この演算子を使用すると、別々のトークンを1つのトークンに結合できます。そのため、これをマクロ定義の最初または最後のトークンにすることはできません。
- マクロ定義の仮パラメーターの前または後ろにトークン連結演算子がある場合、仮パラメーターは展開されていない実引数に即座に置き換えられます。引数に対するマクロ展開が置換の前に行われることはありません。

トークン連結演算子 (##)

<https://docs.microsoft.com/ja-jp/cpp/preprocessor/token-pasting-operator-hash-hash?view=vs-2017>

```
#define SOME_MACRO(TYPE, arg) ¥  
typedef vector<TYPE> vector##arg;
```

- 例えば,上記のマクロを定義し,コード中に
SOME_MACRO(int, INT)
と書かれていると,そこは下記に置き換わる。

```
typedef vector<int> vectorINT;
```

runTimeSelectionTables.H

Detailed Description

Macros to ease declaration of run-time selection tables.

[declareRunTimeSelectionTable](#) is used to create a run-time selection table for a base-class which holds constructor pointers on the table.

`declareRunTimeNewSelectionTable` is used to create a run-time selection table for a derived-class which holds "New" pointers on the table.

Definition in file `runTimeSelectionTables.H`.

ddtScheme.H

マクロ

src/OpenFOAM/db/runTimeSelection/construction/runTimeSelectionTables.H

declareRunTimeSelectionTable

```
(  
    tmp,  
    ddtScheme,  
    Istream,  
    (const fvMesh& mesh, Istream& schemeData),  
    (mesh, schemeData)  
);
```

src/OpenFOAM/db/runTimeSelection/construction/ runTimeSelectionTables.H

```
tmp, ddtScheme, Istream, (const fvMesh& mesh, Istream& schemeData), (mesh, schemeData)
```

```
//- Declare a run-time selection
```

```
#define declareRunTimeSelectionTable(autoPtr, baseType, argNames, argList, parList) ¥
```

```
/* Construct from argList function pointer type */ ¥
```

```
typedef autoPtr<baseType> (*argNames##ConstructorPtr)argList; ¥
```

```
/* Construct from argList function table type */ ¥
```

```
typedef HashTable<argNames##ConstructorPtr, word, string::hash> ¥
```

```
argNames##ConstructorTable; ¥
```

```
IstreamConstructorPtr  
IstreamConstructorTable
```

中略

```
/* Class to add constructor from argList to table */ ¥
```

```
template<class baseType##Type> ¥
```

```
class add##argNames##ConstructorToTable ¥
```

```
{ ¥
```

```
public: ¥
```


src/finiteVolume/finiteVolume/ddtSchemes/ddtScheme/ ddtSchemes.C

ここから, さらに
#include "runTimeSelectionTables.H
src/OpenFOAM/db/runTimeSelection/construction/

#include "ddtScheme.H"

// Define the constructor function hash tables

```
defineTemplateRunTimeSelectionTable(ddtScheme<scalar>, Istream);  
defineTemplateRunTimeSelectionTable(ddtScheme<vector>, Istream);  
defineTemplateRunTimeSelectionTable(ddtScheme<sphericalTensor>, Istream);  
defineTemplateRunTimeSelectionTable(ddtScheme<symmTensor>, Istream);  
defineTemplateRunTimeSelectionTable(ddtScheme<tensor>, Istream);
```

src/OpenFOAM/db/runTimeSelection/construction/
runTimeSelectionTables.H

```
defineTemplateRunTimeSelectionTable(ddtScheme<scalar>, Istream);
```

```
// - Define run-time selection table for template classes  
// use when baseType doesn't need a template argument (eg, is a typedef)
```

```
#define defineTemplateRunTimeSelectionTable(baseType, argNames)
```

¥
¥
¥
¥
¥
¥
¥
¥

```
ddtScheme<scalar>
```

```
template<>
```

```
defineRunTimeSelectionTablePtr(baseType, argNames);
```

```
Istream
```

```
template<>
```

```
defineRunTimeSelectionTableConstructor(baseType, argNames);
```

```
template<>
```

```
defineRunTimeSelectionTableDestructor(baseType, argNames)
```

```
// Create pointer to hash-table of functions
```

```
#define defineRunTimeSelectionTablePtr(baseType, argNames)
```

```
/* Define the constructor function table */
```

```
baseType::argNames##ConstructorTable*
```

```
baseType::argNames##ConstructorTablePtr_ = nullptr
```

```
ddtScheme<scalar>::IstreamConstructorTablePtr_
```

src/finiteVolume/Make/filesの一部

```
ddtSchemes = finiteVolume/ddtSchemes
$(ddtSchemes)/ddtScheme/ddtSchemes.C
$(ddtSchemes)/steadyStateDdtScheme/steadyStateDdtSchemes.C
$(ddtSchemes)/EulerDdtScheme/EulerDdtSchemes.C
$(ddtSchemes)/CoEulerDdtScheme/CoEulerDdtSchemes.C
```


Eulerスキームは？

EulerDdtScheme.s.C

```
makeFvDdtScheme(EulerDdtScheme)
```

makeFvDdtSchemeはddtScheme.Hで定義されたマクロ。

ddtスキームのrunTimeSelectionTableに、EulerDdtSchemeが追加される。

EulerDdtSchemeのtypeNameはEulerである。
(EulerDdtScheme.Hで定義している。)

他の runTimeSelectionTable

- ライブラリ・機能ごとに, runTimeSelectionTable が宣言されている。
 - declareRunTimeSelectionTable
 - abstract base classが担当することが多い?
- 各クラスをそのtableに追加するためのマクロが存在する
 - addToRunTimeSelectionTable
 - モデル名s.Cで呼び出すことがある
 - 複数のモデルを一括して登録する場合がある(乱流)
- 新クラス作成時には, その仕組みを使うとよい