OpenFOAMソースコードの眺め方 : はじめの一歩　part 1

2013年6月8日

オープンCAE勉強会＠富山

中川慎二

この講習の目的

- OpenFOAMのソースコードを読むのに必要な，基礎的な知識を知る
- ソルバのソースコードから，その先で行われていることを探る方法を知る
- 基礎的なソルバの，大まかな流れを知る
- 有限体積法が実装されていそうなことを感じ取る？

OpenFOAMの定式化の詳しい資料

Hrvoje Jasak, PhD 1996, Error analysis and estimation in the Finite Volume method with applications to fluid flows.

- http://powerlab.fsb.hr/ped/kturbo/OpenFOAM/docs/HrvojeJasakPhD.pdf

Henrik Rusche, PhD 2002, Computational fluid dynamics of dispersed two-phase flows at high phase fractions.

- http://powerlab.fsb.hr/ped/kturbo/OpenFOAM/docs/HenrikRuschePhD2002.pdf

OpenFOAM　ユーザーガイド, プログラマーズガイド

CFDの基礎に関する書籍

- 数値流体力学[第2版]，H.K.Versteeg, W.Malalasekera，森北出版　2011
- J.H. Ferziger and M. Peric *Computational Methods for Fluid Dynamics* 3rd ed. Springer 2002.

OpenFOAMのソースコード

- C++ 言語
- オブジェクト指向プログラミング

オブジェクト指向プログラミング

一般的に以下の機能や特徴を活用したプログラミング技法のことをいう。

- カプセル化 (振る舞いの隠蔽とデータ隠蔽)
- インヘリタンス (継承) -- クラスベースの言語
- ポリモフィズム (多態性、多相性) -- 型付きの言語

C++　クラスとは

クラス

- 部品（オブジェクト）の設計図
- 様々な値（状態，属性），機能（function,メソッド，関数）を含む
- クラスは、値とメソッドの集まりである
- この設計図（クラス）に基づいて，プログラム実行時に，部品（インスタンス）が作られる

プログラム

- 様々な部品が，協調しながら，目的を果たす。
- 部品どうしは，適切な独立性を持っている。

オブジェクト指向プログラミング，C++ の機能

カプセル化

- 部品の内部構造を詳細に知らなくても，使えるようにする
- 部品の使い方だけを明確にしておく

- このおかげで，何となく知っている程度の状態で，OpenFOAMのソルバが改造できる

継承

- 既存クラスの機能、構造を共有する新たなクラスを派生することができ（サブクラス化）
- あるクラスのメンバを、他のクラスに引継ぐ（継承させる）

- // base クラスを継承する sub クラス
- class sub : public base
- {
- 　　// メンバは省略
- };

継承

- スーパークラスの構造と機能がサブクラスにそのまま引き継がれるため、サブクラスでスーパークラスのコードを再利用できる。

継承の例：basicKinematicCollidingCloud

icoUncoupledKinematicParcelFoamにおいて，
basicKinematicCollidingCloudは，次のように定義されている。

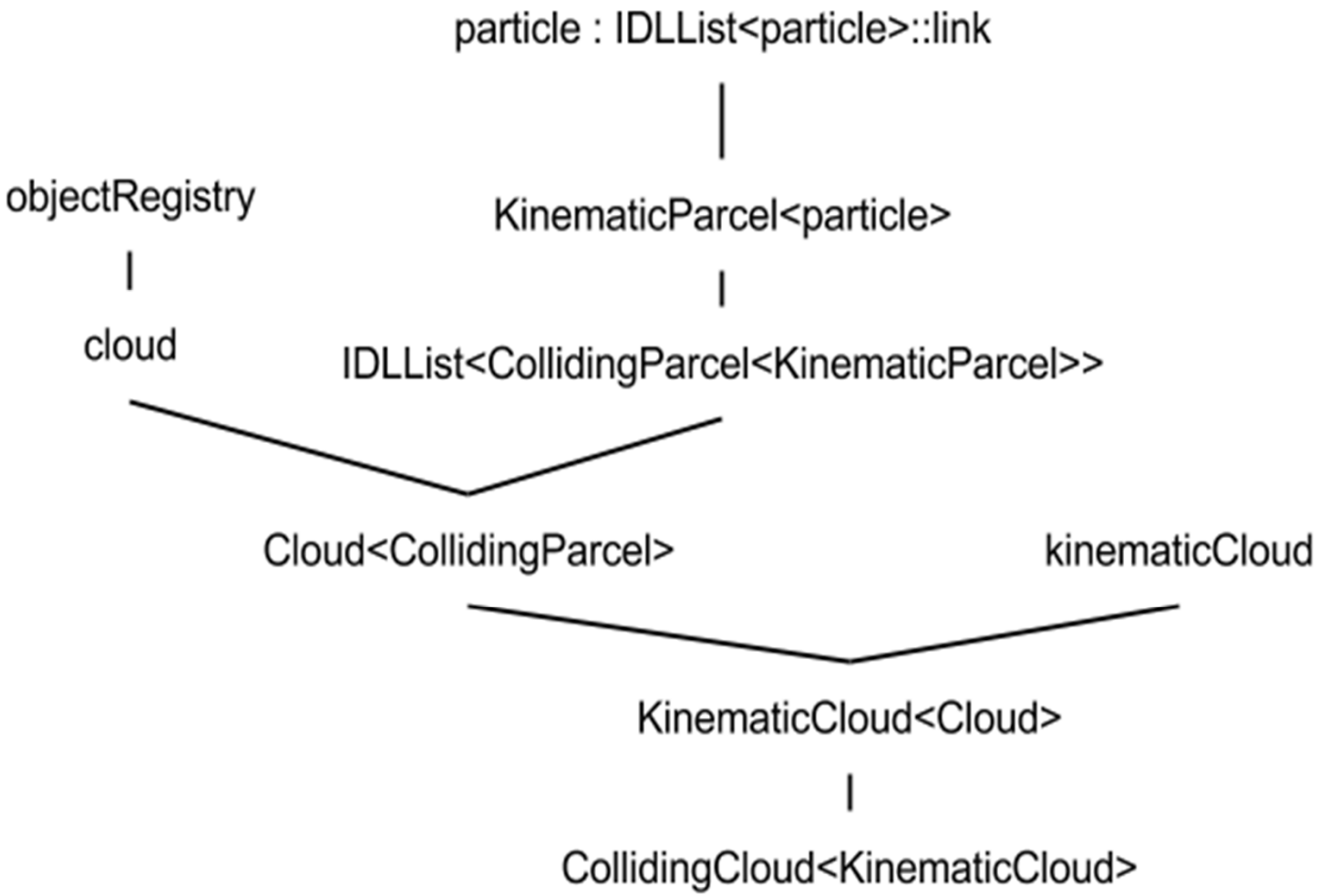
```
typedef CollidingCloud<
                KinematicCloud <
                        Cloud <
                                basicKinematicCollidingParcel
                                >
                        >
                >
                                basicKinematicCollidingCloud
```

Definition at line 53 of file basicKinematicCollidingCloud.H.

```
typedef CollidingParcel < KinematicParcel < particle >  >
                                basicKinematicCollidingParcel
```

Definition at line 48 of file basicKinematicCollidingParcel.H.

継承の例：basicKinematicCollidingCloud



ポリモフィズム (多態性、多相性)

- 同名のメソッドなどを，オブジェクトの種類に応じて使い分けることができること
- 同じ仕組みには，対象が違っても，同じメソッド名を付けられるので，覚えやすい・使いやすい
- OpenFOAMでは，「＋」という記号で，整数も，少数も，単位付スカラー量も，ベクトルも，マトリクスも，同じような計算ができる
- オーバーロードとオーバーライド

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

テンプレート クラス

多くの似たようなクラスを作成するとき，テンプレートを利用できる。
クラスの内部で持つメンバやメソッドで使うクラスを，テンプレート型として定義する。

```
template<class CloudType>
class KinematicCloud
 {
public:
      CloudType kinematicClouds;
}
```

template<class CloudType>は，テンプレートヘッダと呼ばれる。コンパイラに，これからテンプレートクラスを定義することを示す。このクラスの中でCloudType という名前を使うと，コンパイラはこのCloudTypeが何らかの型を表すと判断する。

テンプレート クラス　例

CollidingCloud.Hの60行付近に，クラスの定義が記述されている。簡潔に書くと次のようなものである。

00060 template<class CloudType>

00061 class CollidingCloud ：public CloudType

　これはCollidingCloudクラスが，CloudTypeクラスを継承することを表す。

　CloudTypeはテンプレートクラスなので，CollidingCloudを呼び出す時に使われるクラスとなる。

テンプレートクラスと継承　例

KinematicCloud.Hの92～96行に，クラスの定義が記述されている。簡潔に書くと次のようなものである。

00092 template<class CloudType>

00093 class KinematicCloud : public CloudType, public kinematicCloud

これは，KinematicCloudクラスが，CloudTypeとkinematicCloudの2つのクラスを継承（多重継承）することを表す。

CloudTypeはテンプレートクラスなので，インスタンス化するときに使用するクラスとなる。

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

コンストラクタ

- クラスと同じ名前の，特別なメソッド。
- クラスから新たなインスタンスを作成するときに，必ず実行される
- 初期化作業を行う
- クラスを生成するときの引数に応じて，複数のコンストラクタを用意することができる　→ OpenFOAMでも，実際に活用されまくっている

コンストラクタの例

laplacianScheme< Type, GType > Class Template Reference

http://foam.sourceforge.net/docs/cpp/a01091.html#details

Public Member Functions

virtual const word &	type () const =0 Runtime type information.
	declareRunTimeSelectionTable (tmp, laplacianScheme, Istream,(const fvMesh &mesh, Istream &schemeData),(mesh, schemeData))
	laplacianScheme (const fvMesh &mesh) Construct from mesh.
	laplacianScheme (const fvMesh &mesh, Istream &is) Construct from mesh and Istream.
	laplacianScheme (const fvMesh &mesh, const tmp< surfaceInterpolationScheme< GType > > &igs, const tmp< snGradScheme< Type > > &sngs) Construct from mesh, interpolation and snGradScheme schemes.
virtual	~laplacianScheme () Destructor.
const fvMesh &	mesh () const Return mesh reference.

具体的に：icoFoamを例に

ソルバ icoFoam

ソースコードの場所

- icoFoam 本体
  - /opt/openfoam220/applications/solvers/incompressible/icoFoam
    - icoFoam.C
    - createFields.H
- その他に必要なライブラリ
  - /opt/openfoam220/src

icoFoam.C

```
#include "fvCFD.H"

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

int main(int argc, char *argv[])
{
    #include "setRootCase.H"

    #include "createTime.H"
    #include "createMesh.H"
    #include "createFields.H"
    #include "initContinuityErrs.H"

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

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

    while (runTime.loop())
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;

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

        fvVectorMatrix UEqn
        (
            fvm::ddt(U)
          + fvm::div(phi, U)
          - fvm::laplacian(nu, U)
        );

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

        // --- PISO loop
        for (int corr=0; corr<nCorr; corr++)
        {
            volScalarField rAU(1.0/UEqn.A());

            volVectorField HbyA("HbyA", U);
            HbyA = rAU*UEqn.H();
            surfaceScalarField phiHbyA
            (
                "phiHbyA",
                (fvc::interpolate(HbyA) & mesh.Sf())
              + fvc::ddtPhiCorr(rAU, U, phi)
            );

            adjustPhi(phiHbyA, U, p);

            for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
            {
                fvScalarMatrix pEqn
                (
                    fvm::laplacian(rAU, p) == fvc::div(phiHbyA)
                );

                pEqn.setReference(pRefCell, pRefValue);
                pEqn.solve();

                if (nonOrth == nNonOrthCorr)
                {
                    phi = phiHbyA - pEqn.flux();
                }
            }

            #include "continuityErrs.H"

            U = HbyA - rAU*fvc::grad(p);
            U.correctBoundaryConditions();
        }

        runTime.write();

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }

    Info<< "End¥n" << endl;

    return 0;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

createFields.H

```
Info<< "Reading transportProperties¥n" << endl;

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

dimensionedScalar nu
(
    transportProperties.lookup("nu")
);

Info<< "Reading field p¥n" << endl;
volScalarField p
(
    IOobject
    (
        "p",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);


Info<< "Reading field U¥n" << endl;
volVectorField U
(
    IOobject
    (
        "U",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

#   include "createPhi.H"

label pRefCell = 0;
scalar pRefValue = 0.0;
setRefCell(p, mesh.solutionDict().subDict("PISO"), pRefCell, pRefValue);
```

ファイルがどこにあるかを探す

http://foam.sourceforge.net/docs/cpp/files.html

- fvCFD.H
  - src/finiteVolume/cfdTools/general/include/fvCFD.H
- setRootCase.H
  - src/OpenFOAM/include/setRootCase.H
- createTime.H
  - src/OpenFOAM/include/createTime.H
- createMesh.H
  - src/OpenFOAM/include/createMesh.H
- createFields.H
  - ソルバのディレクトリにある
- initContinuityErrs.H
  - src/finiteVolume/cfdTools/general/include/initContinuityErrs.H

fvCFD.H

```
00001 #ifndef fvCFD_H
00002 #define fvCFD_H
00003
00004 #include "parRun.H"
00005
00006 #include "Time.H"
00007 #include "fvMesh.H"
00008 #include "fvc.H"
00009 #include "fvMatrices.H"
00010 #include "fvm.H"
00011 #include "linear.H"
00012 #include "uniformDimensionedFields.H"
00013 #include "calculatedFvPatchFields.H"
00014 #include "fixedValueFvPatchFields.H"
00015 #include "adjustPhi.H"
00016 #include "findRefCell.H"
00017 #include "constants.H"
00018
00019 #include "OSspecific.H"
00020 #include "argList.H"
00021 #include "timeSelector.H"
00022
00023 #ifndef namespaceFoam
00024 #define namespaceFoam
00025     using namespace Foam;
00026 #endif
00027
00028 #endif
```

setRootCase.H

```
00001 //
00002 // setRootCase.H
00003 // ~~~~~~~~~~~~~~
00004
00005     Foam::argList args(argc, argv);
00006     if (!args.checkRootCase())
00007     {
00008         Foam::FatalError.exit();
00009     }
```

createTime.H

```
00001 //
00002 // createTime.H
00003 // ~~~~~~~~~~~~
00004
00005     Foam::Info<< "Create time¥n" <<
Foam::endl;
00006
00007     Foam::Time
runTime(Foam::Time::controlDictName, args);
```

createMesh.H

```
00001 //
00002 // createMesh.H
00003 // ~~~~~~~~~~~~
00004
00005     Foam::Info
00006         << "Create mesh for time = "
00007         << runTime.timeName() << Foam::nl << Foam::endl;
00008
00009     Foam::fvMesh mesh
00010     (
00011         Foam::IOobject
00012         (
00013             Foam::fvMesh::defaultRegion,
00014             runTime.timeName(),
00015             runTime,
00016             Foam::IOobject::MUST_READ
00017         )
00018     );
```

initContinuityErrs.H

```
00001 /*---------------------------------------------------------------------------*¥
00002  =========                 |
00003  ¥¥     /  F ield        | OpenFOAM: The Open Source CFD Toolbox
00004   ¥¥   /   O peration     |
00005    ¥¥ /    A nd          | Copyright (C) 2011 OpenFOAM Foundation
00006     ¥¥/    M anipulation  |
00007 -------------------------------------------------------------------------
00008 License
00009     This file is part of OpenFOAM.
00010
00011     OpenFOAM is free software: you can redistribute it and/or modify it
00012     under the terms of the GNU General Public License as published by
00013     the Free Software Foundation, either version 3 of the License, or
00014     (at your option) any later version.
00015
00016     OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
00017     ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
00018     FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
00019     for more details.
00020
00021     You should have received a copy of the GNU General Public License
00022     along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.
00023
00024 Global
00025     cumulativeContErr
00026
00027 Description
00028     Declare and initialise the cumulative continuity error.
00029
00030 ¥*---------------------------------------------------------------------------*/
00031
00032 #ifndef initContinuityErrs_H
00033 #define initContinuityErrs_H
00034
00035 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
00036
00037 scalar cumulativeContErr = 0;
00038
00039 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
00040
00041 #endif
00042
00043 // ************************************************************************* //
```

- src/finiteVolume/cfdTools/incompressible/createPhi.H [code]

createPhi.H

```
00001 /*---------------------------------------------------------------------------*¥
00002  =========                 |
00003  ¥¥     /  F ield         | OpenFOAM: The Open Source CFD Toolbox
00004   ¥¥   /   O peration     |
00005    ¥¥ /    A nd           | Copyright (C) 2011 OpenFOAM Foundation
00006     ¥¥/    M anipulation  |
00007 -------------------------------------------------------------------------------
00008 License
00009     This file is part of OpenFOAM.
00010
00011     OpenFOAM is free software: you can redistribute it and/or modify it
00012     under the terms of the GNU General Public License as published by
00013     the Free Software Foundation, either version 3 of the License, or
00014     (at your option) any later version.
00015
00016     OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
00017     ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or
00018     FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public
License
00019     for more details.
00020
00021     You should have received a copy of the GNU General Public License
00022     along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.
00023
00024 Global
00025     createPhi
00026
00027 Description
00028     Creates and initialises the relative face-flux field phi.
00029
00030 ¥*---------------------------------------------------------------------------*/
00031
00032 #ifndef createPhi_H
00033 #define createPhi_H
00034
00035 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
00036
00037 Info<< "Reading/calculating face flux field phi¥n" << endl;
00038
00039 surfaceScalarField phi
00040 (
00041     IOobject
00042     (
00043         "phi",
00044         runTime.timeName(),
00045         mesh,
00046         IOobject::READ_IF_PRESENT,
00047         IOobject::AUTO_WRITE
00048     ),
00049     linearInterpolate(U) & mesh.Sf()
00050 );
00051
00052 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
00053
00054 #endif
00055
00056 //
************************************************************************
******* //
```

- readPISOControls.H
  - src/finiteVolume/cfdTools/general/include/readPISOControls.H
- CourantNo.H
  - src/finiteVolume/cfdTools/incompressible/CourantNo.H
- continuityErrs.H
  - src/finiteVolume/cfdTools/incompressible/continuityErrs.H

readPISOControls.H

```
00001   const dictionary& pisoDict = mesh.solutionDict().subDict("PISO");
00002
00003   const int nOuterCorr =
00004       pisoDict.lookupOrDefault<int>("nOuterCorrectors", 1);
00005
00006   const int nCorr =
00007       pisoDict.lookupOrDefault<int>("nCorrectors", 1);
00008
00009   const int nNonOrthCorr =
00010       pisoDict.lookupOrDefault<int>("nNonOrthogonalCorrectors", 0);
00011
00012   const bool momentumPredictor =
00013       pisoDict.lookupOrDefault("momentumPredictor", true);
00014
00015   const bool transonic =
00016       pisoDict.lookupOrDefault("transonic", false);
00017
```

CourantNo.H

```
00001 /*---------------------------------------------------------------------------*¥

00024 Global
00025     CourantNo
00026
00027 Description
00028     Calculates and outputs the mean and maximum Courant Numbers.
00029
00030 ¥*---------------------------------------------------------------------------*/
00031
00032 scalar CoNum = 0.0;
00033 scalar meanCoNum = 0.0;
00034
00035 if (mesh.nInternalFaces())
00036 {
00037     scalarField sumPhi
00038     (
00039         fvc::surfaceSum(mag(phi))().internalField()
00040     );
00041
00042     CoNum = 0.5*gMax(sumPhi/mesh.V().field())*runTime.deltaTValue();
00043
00044     meanCoNum =
00045         0.5*(gSum(sumPhi)/gSum(mesh.V().field()))*runTime.deltaTValue();
00046 }
00047
00048 Info<< "Courant Number mean: " << meanCoNum
00049     << " max: " << CoNum << endl;
00050
00051 // ************************************************************************* //
```

continuityErrs.H

```
00001 /*---------------------------------------------------------------------------*¥
00024 Global
00025     continuityErrs
00026
00027 Description
00028     Calculates and prints the continuity errors.
00029
00030 ¥*---------------------------------------------------------------------------*/
00031
00032 {
00033     volScalarField contErr(fvc::div(phi));
00034
00035     scalar sumLocalContErr = runTime.deltaTValue()*
00036         mag(contErr)().weightedAverage(mesh.V()).value();
00037
00038     scalar globalContErr = runTime.deltaTValue()*
00039         contErr.weightedAverage(mesh.V()).value();
00040     cumulativeContErr += globalContErr;
00041
00042     Info<< "time step continuity errors : sum local = " << sumLocalContErr
00043         << ", global = " << globalContErr
00044         << ", cumulative = " << cumulativeContErr
00045         << endl;
00046 }
00047
00048 // ************************************************************************* //
```

- 基礎式

$$\frac{\partial(\rho\phi)}{\partial t} + \mathrm{div}(\rho\phi\mathbf{u}) = \mathrm{div}(\Gamma\,\mathrm{grad}\,\phi) + S_\phi$$

流体要素の ϕ の増加割合	+	流体要素から流出する ϕ の正味の割合	=	拡散による ϕ の増加割合	+	生成による ϕ の増加割合

$$\frac{\partial \boldsymbol{U}}{\partial t} + \mathrm{div}(\boldsymbol{U}\boldsymbol{U}) = -\nabla\left(\frac{p}{\rho}\right) + \nabla^2(\nu\boldsymbol{U})$$

- ソースコード

```
fvm::ddt(U) + fvm::div(phi, U) - fvm::laplacian(nu, U) ==
-fvc::grad(p)
```

Uとは何か？

createFields.H で作られる。

volVectorField クラスから，U インスタンスを，IOobject, mesh を引数として，作成する。（コンストラクタは，GeometricField (const IOobject & io, const Mesh & mesh) となる）

```
volVectorField U
    (
        IOobject  ( “U”, runTime.timeName(), mesh,
                        IOobject::MUST_READ,
                         IOobject::AUTO_WRITE
        ),
        mesh
    );
```

volVectorField とは？

GeometricField<vector, fvPatchField, volMesh> の別名

volFieldsFwd.H

```
00048 template<class Type>
00049 class fvPatchField;
00050
00051 template<class Type, template<class> class PatchField, class GeoMesh>
00052 class GeometricField;
00053
00054 typedef GeometricField<scalar, fvPatchField, volMesh> volScalarField;
00055 typedef GeometricField<vector, fvPatchField, volMesh> volVectorField;
00056 typedef GeometricField<sphericalTensor, fvPatchField, volMesh>
00057     volSphericalTensorField;
00058 typedef GeometricField<symmTensor, fvPatchField, volMesh>
volSymmTensorField;
00059 typedef GeometricField<tensor, fvPatchField, volMesh> volTensorField;
```

Uとは何か？

createFields.H で作られる。

volVectorField クラスから，U インスタンスを，IOobject, mesh を引数として，作成する。（コンストラクタは，GeometricField (const IOobject & io, const Mesh & mesh) となる）

```
volVectorField U
    (
        IOobject  ( “U”, runTime.timeName(), mesh,
                          IOobject::MUST_READ,
                           IOobject::AUTO_WRITE
        ),
        mesh
    );
```

コンストラクタ GeometricField (const IOobject & io, const Mesh & mesh)

GeometricField (const IOobject & io, const Mesh & mesh)
Construct and read given IOobject.
Definition at line 332 of file GeometricField.C.

```
00330 template<class Type, template<class> class PatchField, class GeoMesh>
00331 Foam::GeometricField<Type, PatchField, GeoMesh>::GeometricField
00332 (
00333     const IOobject& io,
00334     const Mesh& mesh
00335 )
00336 :
00337     DimensionedField<Type, GeoMesh>(io, mesh, dimless, false),
00338     timeIndex_(this->time().timeIndex()),
00339     field0Ptr_(NULL),
00340     fieldPrevIterPtr_(NULL),
00341     boundaryField_(mesh.boundary())
00342 {
00343     readFields();
00344
00345     // Check compatibility between field and mesh
00346
00347     if (this->size() != GeoMesh::size(this->mesh()))
00348     {
00349         FatalIOErrorIn
00350         (
00351             "GeometricField<Type, PatchField, GeoMesh>::GeometricField"
00352             "(const IOobject&, const Mesh&)",
00353             this->readStream(typeName)
00354         )   << "   number of field elements = " << this->size()
00355             << " number of mesh elements = " << GeoMesh::size(this->mesh())
00356             << exit(FatalIOError);
00357     }
00358
00359     readOldTimeIfPresent();
00360
00361     if (debug)
00362     {
00363         Info<< "Finishing read-construct of "
00364                 "GeometricField<Type, PatchField, GeoMesh>"
00365             << endl << this->info() << endl;
00366     }
00367 }
```

fvVectorMatrix UEqn

```
fvVectorMatrix UEqn
    (
      fvm::ddt(U)
    + fvm::div(phi, U)
    - fvm::laplacian(nu, U)
    );
```

fvVectorMatrixクラスから UEqn インスタンスを作成。

tmp<fvMatrix<Type> > を引数とするコンストラクタ

fvVectorMatrix

fvMatricesFwd.H

Description　Forward declarations of fvMatrix specializations.

```
00041 template<class Type>
00042 class fvMatrix;
00043
00044 typedef fvMatrix<scalar> fvScalarMatrix;
00045 typedef fvMatrix<vector> fvVectorMatrix;
00046 typedef fvMatrix<sphericalTensor>
fvSphericalTensorMatrix;
00047 typedef fvMatrix<symmTensor> fvSymmTensorMatrix;
00048 typedef fvMatrix<tensor> fvTensorMatrix;
```

名前空間　namespace

- プログラムの一部を，特定の名前を付けてグループ分けする。
- 名前空間でグループ化されたメンバーは，名前空間名::メンバ名で識別される。

```
namespace Foam
{

namespace fvm
{

template<class Type>
tmp<fvMatrix<Type> >
d2dt2 ( const GeometricField<Type, fvPatchField, volMesh>& vf )
{
   return fv::d2dt2Scheme<Type>::New  ( vf.mesh(), vf.mesh().d2dt2Scheme("d2dt2(" + vf.name() + ')')   )().fvmD2dt2(vf);
}

} // End namespace fvm

} // End namespace Foam
```

fvm::

Foam::fvm Namespace Reference

- Namespace of functions to calculate implicit derivatives returning a matrix.
- Temporal derivatives are calculated using Euler-implicit, backward differencing or Crank-Nicolson. Spatial derivatives are calculated using Gauss' Theorem.

http://foam.sourceforge.net/docs/cpp/a09373.html

Foam::fv

Namespace for finite-volume

Foam::fvc

Namespace of functions to calculate explicit derivatives

Foam::fvm

Namespace of functions to calculate implicit derivatives returning a matrix

$$\frac{\partial(\rho\phi)}{\partial t} + \mathrm{div}(\rho\phi\mathbf{u}) = \mathrm{div}(\Gamma\,\mathrm{grad}\,\phi) + S_\phi$$

流体要素の ϕ の増加割合	+	流体要素から流出する ϕ の正味の割合	=	拡散による ϕ の増加割合	+	生成による ϕ の増加割合

- 有限体積法

$$\int_{\mathrm{CV}} \frac{\partial(\rho\phi)}{\partial t}\,\mathrm{d}V + \int_{\mathrm{CV}} \mathrm{div}(\rho\phi\mathbf{u})\,\mathrm{d}V = \int_{\mathrm{CV}} \mathrm{div}(\Gamma\,\mathrm{grad}\,\phi)\,\mathrm{d}V + \int_{\mathrm{CV}} S_\phi\,\mathrm{d}V$$

$$\int_{\mathrm{CV}} \mathrm{div}(\mathbf{a})\,\mathrm{d}V = \int_{\mathrm{A}} \mathbf{n}\cdot\mathbf{a}\,\mathrm{d}A$$

$$\frac{\partial}{\partial t}\left(\int_{\mathrm{CV}} \rho\phi\,\mathrm{d}V\right) + \int_{\mathrm{A}} \mathbf{n}\cdot(\rho\phi\mathbf{u})\,\mathrm{d}A = \int_{\mathrm{A}} \mathbf{n}\cdot(\Gamma\,\mathrm{grad}\,\phi)\,\mathrm{d}A + \int_{\mathrm{CV}} S_\phi\,\mathrm{d}V$$

コントロールボリューム内の ϕ の増加割合	+	コントロールボリューム境界を通過する対流による ϕ の正味の減少割合	=	コントロールボリューム境界を通過する拡散による ϕ の正味の増加割合	+	コントロールボリューム内の ϕ の正味の生成割合

第８章

非定常流れに対する有限体積法

$$\frac{\partial}{\partial t}(\rho\phi) + \mathrm{div}(\rho\mathbf{u}\phi) = \mathrm{div}(\Gamma\,\mathrm{grad}\,\phi) + S_\phi \qquad (8.1)$$

この式の第１項は非定常項を表し，定常流れではゼロである．非定常問題を解析するためには，離散化の過程でこの項を取り扱わなければならない．そのため，式 (8.1) をコントロールボリューム（control volume，CV）において有限の体積で積分することに加え，さらに，有限の時間刻み Δt において積分しなければならない．これまでのように，対流項と拡散項の体積の積分を界面の面積の積分に置き換え（2.5 節参照），非定常項の積分の順序を入れ換えることで次式を得る．

$$\int_{\mathrm{CV}}\left[\int_t^{t+\Delta t}\frac{\partial}{\partial t}(\rho\phi)\,\mathrm{d}t\right]\mathrm{d}V + \int_t^{t+\Delta t}\left[\int_A \mathbf{n}\cdot(\rho\mathbf{u}\phi)\,\mathrm{d}A\right]\mathrm{d}t$$
$$= \int_t^{t+\Delta t}\left[\int_A \mathbf{n}\cdot(\Gamma\,\mathrm{grad}\,\phi)\,\mathrm{d}A\right]\mathrm{d}t + \int_t^{t+\Delta t}\int_{\mathrm{CV}} S_\phi\,\mathrm{d}V\,\mathrm{d}t \qquad (8.2)$$

$$\int_{\mathrm{CV}}\left(\int_t^{t+\Delta t}\rho c\frac{\partial T}{\partial t}\,\mathrm{d}t\right)\mathrm{d}V=\rho c(T_P-T_P^o)\Delta V$$

$$I_T=\int_t^{t+\Delta t}T_P\,\mathrm{d}t=[\theta T_P+(1-\theta)T_P^o]\Delta t \tag{8.8}$$

θ	0	1/2	1
I_T	$T_P^o\,\Delta t$	$\frac{1}{2}(T_P+T_P^o)\Delta t$	$T_P\,\Delta t$

積分 I_T の値は以下のようになる．すなわち，$\theta=0$ の場合，（過去の）時刻 t での温度を用い，$\theta=1$ の場合，新しい時刻 $t+\Delta t$ での温度を用い，そして $\theta=1/2$ の場合，時刻 t と $t+\Delta t$ での温度を同じ重みで用いる．

$$\boxed{\begin{aligned}a_PT_P=a_W[\theta T_W+(1-\theta)T_W^o]+a_E[\theta T_E+(1-\theta)T_E^o]\\+[a_P^o-(1-\theta)a_W-(1-\theta)a_E]T_P^o+b\end{aligned}} \tag{8.11}$$

最終的な離散化方程式の形は θ の値に依存する．θ がゼロの場合，新しい時刻の温度 T_P を計算するために，式 (8.11) の右辺の過去の時刻 t の温度 T_P^o，T_W^o，T_E^o のみを用い，このスキームは**陽解法（explict method）**とよばれる．$0<\theta\leq 1$ の場合，式 (8.11) の両辺の新しい時刻の温度を用い，このスキームは**陰解法（implicit method）**とよばれる．とくに，$\theta=1$ の場合は**完全陰解法（fully implicit method）**，$\theta=1/2$ の場合は**クランク－ニコルソン法（Crank-Nicolson method）**とよばれる（Crank と Nicolson, 1947）．

速度場 $\mathbf{u}$ での一般化した変数 ϕ に対する非定常 3 次元対流－拡散の基礎式は，次式で与えられる．

$$\frac{\partial(\rho\phi)}{\partial t}+\frac{\partial(\rho u\phi)}{\partial x}+\frac{\partial(\rho v\phi)}{\partial y}+\frac{\partial(\rho w\phi)}{\partial z}=\frac{\partial}{\partial x}\left(\Gamma\frac{\partial\phi}{\partial x}\right)+\frac{\partial}{\partial y}\left(\Gamma\frac{\partial\phi}{\partial y}\right)+\frac{\partial}{\partial z}\left(\Gamma\frac{\partial\phi}{\partial z}\right)+S \tag{8.30}$$

OpenFOAM の生成項

また，完全陰解法の離散化方程式は，次式のようになる．

$$a_P\phi_P=a_W\phi_W+a_E\phi_E+a_S\phi_S+a_N\phi_N+a_B\phi_B+a_T\phi_T+a_P^o\phi_P^o+S_u \tag{8.31}$$

ここで，それぞれの係数は，次に示すとおりである．

$$a_P=a_W+a_E+a_S+a_N+a_B+a_T+a_P^o+\Delta F-S_P$$

$$a_P^o=\frac{\rho_P^o\Delta V}{\Delta t}$$

$$\overline{S}\,\Delta V=S_u+S_P\phi_P$$

行列の対角成分となる

8.2.3 完全陰解法

θ の値を 1 とした場合，完全陰解法となる．ここで，生成項を $b = S_u + S_P T_P$ と線形化する．この離散化方程式は，次式のようになる．

$$\boxed{a_P T_P = a_W T_W + a_E T_E + a_P^o T_P^o + S_u} \tag{8.16}$$

ここで，係数は次のようになる．

$$\boxed{a_P = a_P^o + a_W + a_E - S_P}$$

$$\boxed{a_P^o = \rho c \frac{\Delta x}{\Delta t}}$$

a_W	a_E
$\frac{k_w}{\delta x_{WP}}$	$\frac{k_e}{\delta x_{PE}}$

例題 8.2

完全陰解法を用いて例題8.1の問題をもう一度解き，時間刻みを8sとした場合の陽解法と陰解法の数値解を比較せよ．

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

格子点	a_W	a_E	S_P	S_u
1	0	$\frac{k}{\Delta x}$	0	0
2, 3, 4	$\frac{k}{\Delta x}$	$\frac{k}{\Delta x}$	0	0
5	$\frac{k}{\Delta x}$	0	$-\frac{2k}{\Delta x}$	$\frac{2k}{\Delta x}T_B$

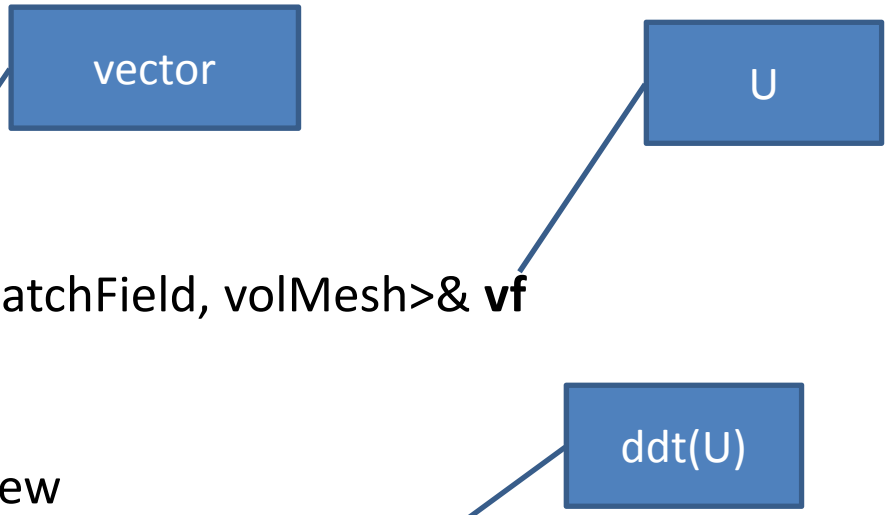
$T_B = 0$ であることに注意すると，それぞれの時間刻みで解くべき連立方程式は，次式のようになる．

$$\begin{bmatrix} 225 & -25 & 0 & 0 & 0 \\ -25 & 250 & -25 & 0 & 0 \\ 0 & -25 & 250 & -25 & 0 \\ 0 & 0 & -25 & 250 & -25 \\ 0 & 0 & 0 & -25 & 275 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{bmatrix} = \begin{bmatrix} 200T_1^o \\ 200T_2^o \\ 200T_3^o \\ 200T_4^o \\ 200T_5^o \end{bmatrix} \quad (8.26)$$

ddt(U)を解読したい

fvm::ddt(U)

```
00043 template<class Type>
00044 tmp<fvMatrix<Type> >
00045 ddt
00046 (
00047     const GeometricField<Type, fvPatchField, volMesh>& vf
00048 )
00049 {
00050     return fv::ddtScheme<Type>::New
00051     (
00052         vf.mesh(),
00053         vf.mesh().ddtScheme("ddt(" + vf.name() + ')')
00054     )().fvmDdt(vf);
00055 }
```



fv::ddtScheme<Type>().fvmDdt(vf)

template<class Type>

class Foam::fv::ddtScheme< Type >

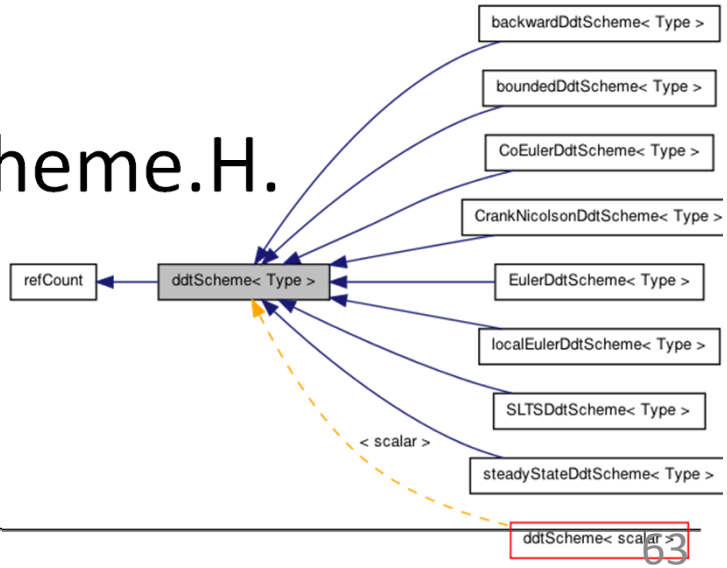
Abstract base class for ddt schemes.

Source files

ddtScheme.H

ddtScheme.C

Definition at line 65 of file ddtScheme.H.



virtual tmp<fvMatrix<Type> > fvmDdt (const GeometricField< Type, fvPatchField, volMesh > &) [pure virtual]

Implemented in backwardDdtScheme< Type >, boundedDdtScheme< Type >, CoEulerDdtScheme< Type >, CrankNicolsonDdtScheme< Type >, EulerDdtScheme< Type >, localEulerDdtScheme< Type >, SLTSDdtScheme< Type >, and steadyStateDdtScheme< Type >.

EulerDdtScheme< Type >

template<class Type>

class Foam::fv::EulerDdtScheme< Type >

Basic first-order Euler implicit/explicit ddt using only the current and previous time-step values.

Source files

EulerDdtScheme.H

EulerDdtScheme.C

Definition at line 56 of file EulerDdtScheme.H.

Definition at line 56 of file EulerDdtScheme.H

```
00055 template<class Type>
00056 class EulerDdtScheme
00057 :
00058     public ddtScheme<Type>
00059 {
00060     // Private Member Functions
00061
00062         //- Disallow default bitwise copy construct
00063         EulerDdtScheme(const EulerDdtScheme&);
00064
00065         //- Disallow default bitwise assignment
00066         void operator=(const EulerDdtScheme&);
00067
```

EulerDdtScheme<Type>::fvmDdt

```
00260 template<class Type>
00261 tmp<fvMatrix<Type> >
00262 EulerDdtScheme<Type>::fvmDdt
00263 (
00264     const GeometricField<Type, fvPatchField,
volMesh>& vf
00265 )
00266 {
00267     tmp<fvMatrix<Type> > tfvm
00268     (
00269         new fvMatrix<Type>
00270         (
00271             vf,
00272             vf.dimensions()*dimVol/dimTime
00273         )
00274     );
00275
00276     fvMatrix<Type>& fvm = tfvm();
00277
00278     scalar rDeltaT =
1.0/mesh().time().deltaTValue();
00279
00280     fvm.diag() = rDeltaT*mesh().V();
00281
00282     if (mesh().moving())
00283     {
00284         fvm.source() =
rDeltaT*vf.oldTime().internalField()*mesh().V0();
00285     }
00286     else
00287     {
00288         fvm.source() =
rDeltaT*vf.oldTime().internalField()*mesh().V();
00289     }
00290
00291     return tfvm;
00292 }
```

backwardDdtScheme<Type>::fvmDdt

```
00359 template<class Type>
00360 tmp<fvMatrix<Type> >
00361 backwardDdtScheme<Type>::fvmDdt
00362 (
00363     const GeometricField<Type, fvPatchField, volMesh>& vf
00364 )
00365 {
00366     tmp<fvMatrix<Type> > tfvm
00367     (
00368         new fvMatrix<Type>
00369         (
00370             vf,
00371             vf.dimensions()*dimVol/dimTime
00372         )
00373     );
00374
00375     fvMatrix<Type>& fvm = tfvm();
00376
00377     scalar rDeltaT = 1.0/deltaT_();
00378
00379     scalar deltaT = deltaT_();
00380     scalar deltaT0 = deltaT0_(vf);
00381
00382     scalar coefft   = 1 + deltaT/(deltaT + deltaT0);
00383     scalar coefft00 = deltaT*deltaT/(deltaT0*(deltaT + deltaT0));
00384     scalar coefft0  = coefft + coefft00;
00385
00386     fvm.diag() = (coefft*rDeltaT)*mesh().V();
00387
00388     if (mesh().moving())
00389     {
00390         fvm.source() = rDeltaT*
00391         (
00392             coefft0*vf.oldTime().internalField()*mesh().V0()
00393           - coefft00*vf.oldTime().oldTime().internalField()
00394            *mesh().V00()
00395         );
00396     }
00397     else
00398     {
00399         fvm.source() = rDeltaT*mesh().V()*
00400         (
00401             coefft0*vf.oldTime().internalField()
00402           - coefft00*vf.oldTime().oldTime().internalField()
00403         );
00404     }
00405
00406     return tfvm;
00407 }
00408
```

PROGRAMMER'S GUIDE
VERSION 2.2.0
22ND FEBRUARY 2013

P-29

Class	Description	Symbol	Access function
volScalarField	Cell volumes	V	`V()`
surfaceVectorField	Face area vectors	$\mathbf{S}_f$	`Sf()`
surfaceScalarField	Face area magnitudes	$\lvert\mathbf{S}_f\rvert$	`magSf()`
volVectorField	Cell centres	$\mathbf{C}$	`C()`
surfaceVectorField	Face centres	$\mathbf{C}_f$	`Cf()`
surfaceScalarField	Face motion fluxes **	ϕ_g	`phi()`

Table 2.1: fvMesh stored data.

P-29

2.4 Equation discretisation

Equation discretisation converts the PDEs into a set of algebraic equations that are commonly expressed in matrix form as:

$$[A]\,[x] = [b] \tag{2.12}$$

where $[A]$ is a square matrix, $[x]$ is the column vector of dependent variable and $[b]$ is the source vector. The description of $[x]$ and $[b]$ as 'vectors' comes from matrix terminology rather than being a precise description of what they truly are: a list of values defined at locations in the geometry, *i.e.* a geometricField<Type>, or more specifically a volField<Type> when using FV discretisation.

$[A]$ is a list of coefficients of a set of algebraic equations, and cannot be described as a geometricField<Type>. It is therefore given a class of its own: fvMatrix. fvMatrix<Type> is created through discretisation of a geometric<Type>Field and therefore inherits the <Type>. It supports many of the standard algebraic matrix operations of addition +, subtraction - and multiplication *.

P-29, P-30

volField<Type> A field defined at cell centres;

surfaceField<Type> A field defined on cell faces;

pointField<Type> A field defined on cell vertices.

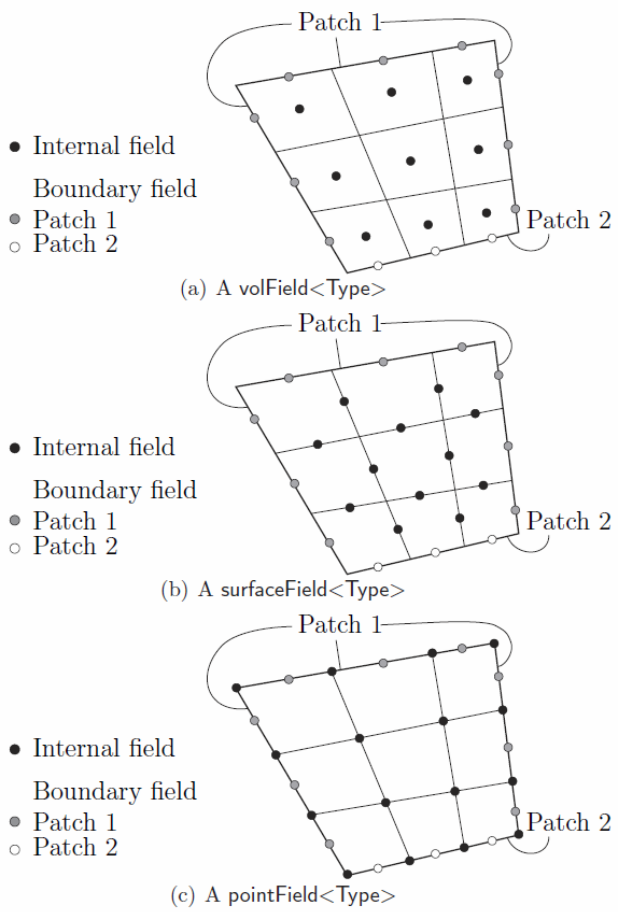


Figure 2.4: Types of geometricField<Type> defined on a mesh with 2 boundary patches (in 2 dimensions for simplicity)

fvm と fvc　P-32

- functions of fvm that calculate implicit derivatives of and return an fvMatrix<Type>
- some functions of fvc that calculate explicit derivatives and other explicit calculations, returning a geometricField<Type>.

Figure 2.6 shows a geometricField<Type> defined on a mesh with 2 boundary patches and illustrates the explicit operations merely transform one field to another and drawn in 2D for simplicity.

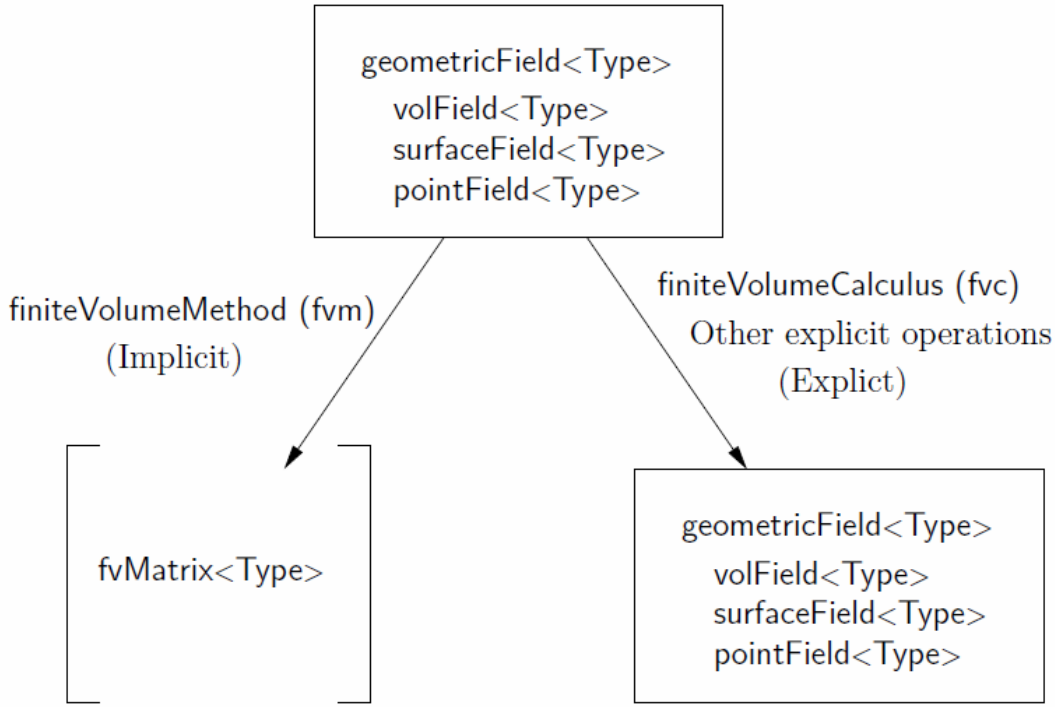


Figure 2.6: A geometricField<Type> and its operators

tmpについて

- tmpについて
- ガベージコレクションを実現するためのもの
- http://www.geocities.jp/penguinitis2002/study/OpenFOAM/tankentai/19-autoPtr_and_tmp.html
- tmp<fvVectorMatrix> UEqn となっていれば，UEqnはfvVectorMatrixだと考えればよい。それに，参照数などのガベージコレクション機能が付加されている。