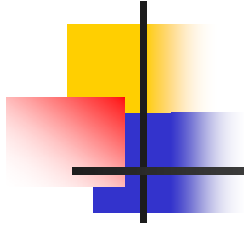


# OpenFOAMの構造とCFDの基礎

東京大学大学院新領域創成科学研究科  
丁 世珉 (ジョン セミン)



# Contents

1. OpenFOAMとCFD
2. OpenFOAMの構造
3. CFDの基礎 (OpenFOAM内での)
4. まとめ



# 1. OpenFOAMとCFD

## □ OpenFOAM (<http://www.ofwikija.org/>から)

OpenFOAM(Field Operation and Manipulation) **CFDツールボックス**は、化学反応や乱流、熱伝達を含む複雑な流体の流れから、固体力学や電磁力学、そして経済の**支配方程式までさまざまな現象をシミュレート**することができます。OpenFOAMはOpenCFD社が開発し、GNUのGeneral Public Licenceのライセンスの元で、**フリーかつオープンソース**として配布しています。

**効率的なC++モジュールの柔軟な組み合わせが、OpenFOAMのテクノロジーの要**です。これらのモジュールは以下のような様々なプログラムを作成するのに用いられます。

工業力学における特定の**問題をシミュレートするソルバー**可視化のための単純データ操作からメッシュ生成に到るまで**様々なプリ・ポスト処理**の仕事を行なう**ユーティリティ** 物理モデルのライブラリ群といった、ソルバやユーティリティから利用できる**ツールボックス**を形成している**ライブラリ群**



# 1. OpenFOAMとCFD

## □ OpenFOAM (<http://www.ofwikija.org/>から)

OpenFOAMは、多数の既成のソルバーやユーティリティ、ライブラリーと共に提供されているため、通常のシミュレーション・パッケージ・ソフトのようにも使うことができます。しかしながら、OpenFOAMは、ソースだけでなく、その構造や階層的なデザインすらもオープンであることから、OpenFOAMのソルバーやユーティリティ、ライブラリーは**全面的にカスタマイズが可能**です。

OpenFOAMは、多面体格子で構成される**3次元の非構造メッシュ**上で記述された**偏微分方程式**系を解くのに**有限体積法**を用います。流体のソルバーは、**圧力と速度の関係**について陰的かつ反復して解く頑丈なフレームワークを用いて開発しており、また、他の保存則のソルバーに対してはそれに応じた別のテクニックを採用しています。領域分割による**並列化**はOpenFOAMのデザインの基盤であり、低レベルのコードに組み込まれているため、一般にソルバーは、何の並列仕様のコーディングをしなくても開発することができます。

# 1. OpenFOAMとCFD

## □ OpenFOAM

Open Source Field Operation And Manipulation

Field

(物理)場

Operation And Manipulation

シミュレーション



実験・訓練を目的とし、  
複雑な事象・システムを定式化して行う  
模擬実験をいう

..... ツールボックスを形成しているC++ライブラリ群

# 1. OpenFOAMとCFD

## □ 物理現象の解析と予想

支配方程式

(xxx) 場で起こる物理現象を支配する法則

連続体力学



(熱)流(体)



偏微分方程式

## 物理現象の解析方法

実験

実際を良く描写、無次元係数、お金、時間、限界

(数学)解析

物理現象Base、解けるケースの限り

数値解法

物理現象Base、COSTの節約、条件設定が自由

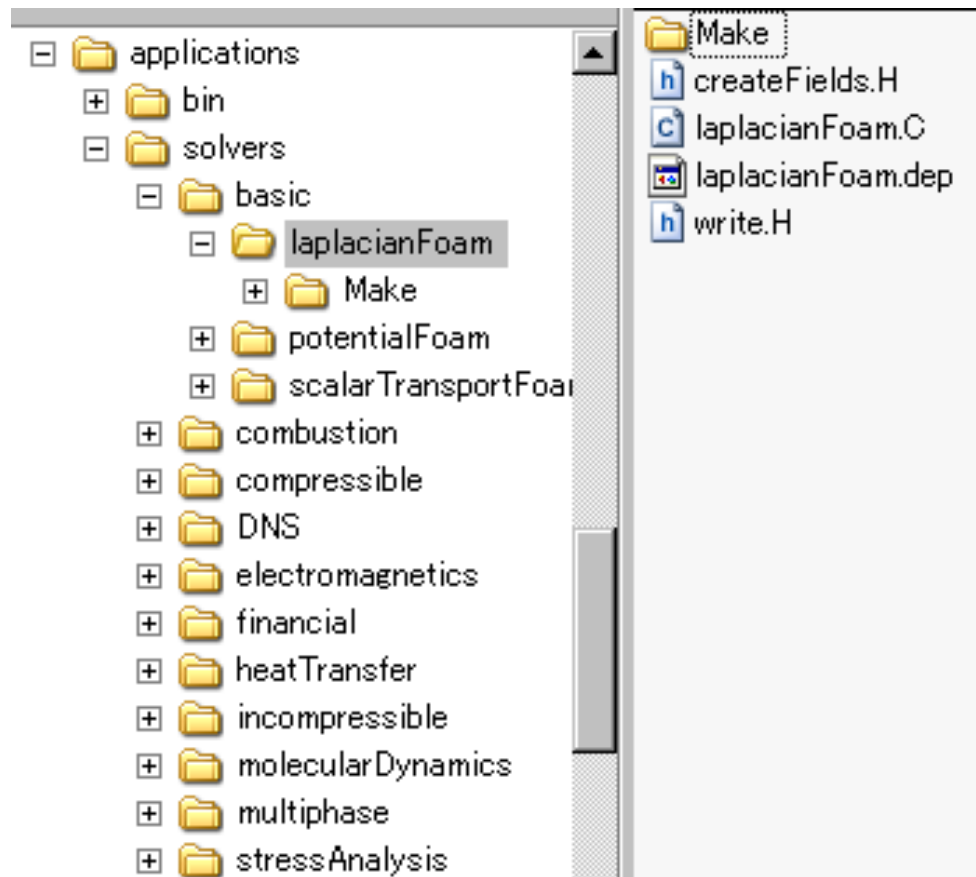
数値流体力学(CFD)

# 1. OpenFOAMとCFD

## □ OpenFOAMのSolver

`\OpenFOAM-1.5\applications\solvers\`

1. 基礎的なCFDコード (3種類)
2. 非圧縮性流れ (9種類)
3. 圧縮性流れ (12種類)
4. 多相流 (13種類)
5. DNS (1種類)
6. 燃焼 (8種類)
7. 熱輸送 (5種類)
8. 電磁流体 (2種類)
9. 固体応力解析 (2種類)
10. 金融工学 (1種類)
11. 分子力学 (2種類)



# 1. OpenFOAMとCFD

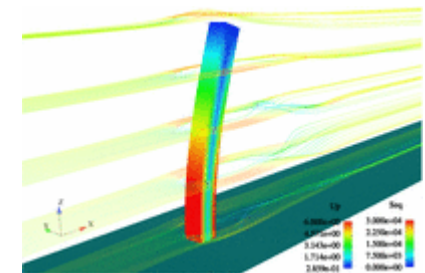
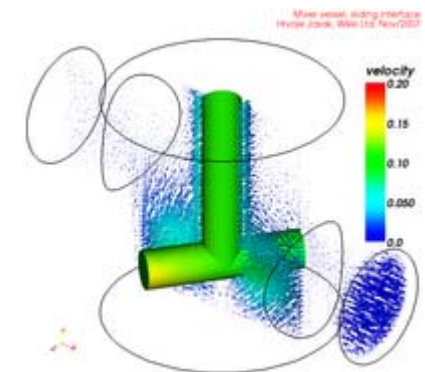
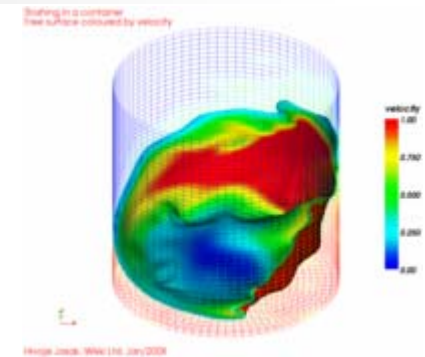
## □ 数値流体力学(CFD)

解析的には解けない物理的現象を**支配方程式**を  
時間的・空間的に**離散化**して**近似解**を求める手法。

1960年代から活発に研究が進行されて現在まで混状  
流や反応流、砕波などの特殊な流れ以外は技術的に  
ほとんど確立されている。

最近では**商用S/W**も大量提案されて実用化段階に到  
達している。(Fluent、StarCD、Nastran、Dyna。。。)

お金、精度、BlackBox、一般的な問題







# 1. OpenFOAMとCFD

---

## □ CFDの用途

Navier-Stokes 方程式の対流項

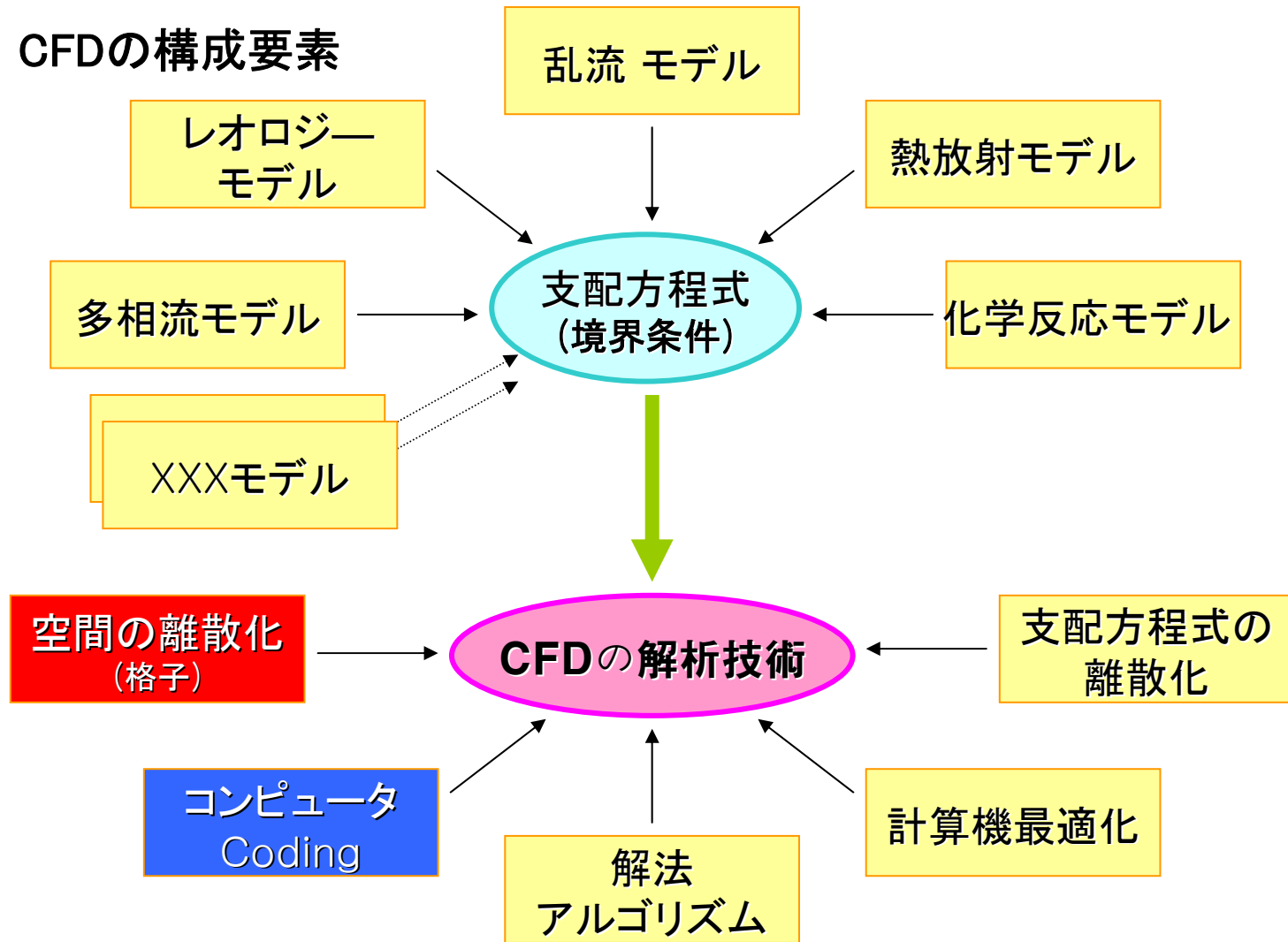
乱流や燃焼、あるいは**非線形性が強い**流体現象の解明のための手段  
実験の代替手段としての**設計道具**

## □ CFDの長所

長所実験が困難な条件でも解を求められる  
実験では計測が困難な物理量を容易に知られる  
実験に比べて、費用が安い  
施設、設備に束縛を受けない

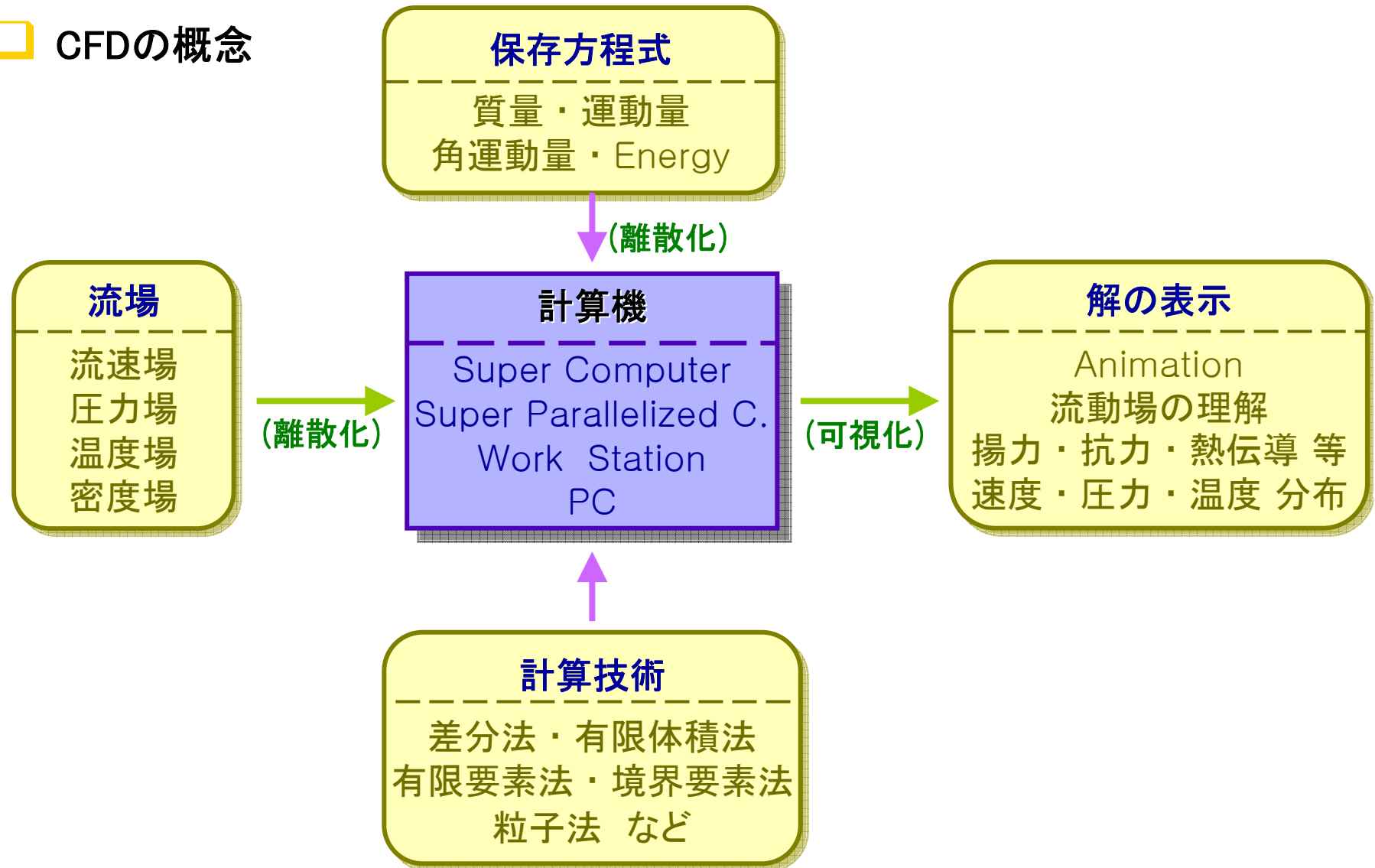
# 1. OpenFOAMとCFD

## □ CFDの構成要素



# 1. OpenFOAMとCFD

## □ CFDの概念





# 1. OpenFOAMとCFD

## □ OpenFOAMの特徴

これまで一般的に CFD コードは、**手続き型言語**、とりわけ FORTRAN によって書かれてきた。この場合、新しい物理モデルを組み込むためには、**方程式の離散化など下位のレベルからプログラミングをやり直さなければならなかった。**

それに対して、OpenFOAM には、**連続体力学で使用される標準的なベクトルおよびテンソルに関する演算子がオブジェクトとして定義されている。**

この演算子を使用することで、**数学的な記述としてプログラミングを行うことができる。**

このようなプログラミングスタイルのためには、**オブジェクト指向プログラミング (Object-Oriented Programming, 以後, OOP と略記) が応用できる。**



# 1. OpenFOAMとCFD

## □ OOPの簡単概念

### 1) 演算子の多重定義

OpenFOAM で採用している C++ では、プログラマにより、**演算子（のみならず関数）の多重定義が可能**となっている。これにより、C++ で表現された演算式は、`'object' + 'object'` と記述された場合、`'object'` と `'object'` を算術的に加えるという操作ではなく、`'object'` と `'object'` に対して、`'+'` で定義された処理を行うという操作になる。結果として、スカラー、ベクトル、行列の 4 則演算子の定義が、通常の数学的な記述と同じように行えるようになる。これは、プログラムの表記も数学的記述と同じような形式で記述可能になることを表している。

### 2) 基本クラスと派生クラス

これまで CFD で多く用いられている手続き型言語の場合、新しい物理モデルなどを組み込むためには方程式の離散化など下位のレベルからプログラミングをやり直さなければならなかった。

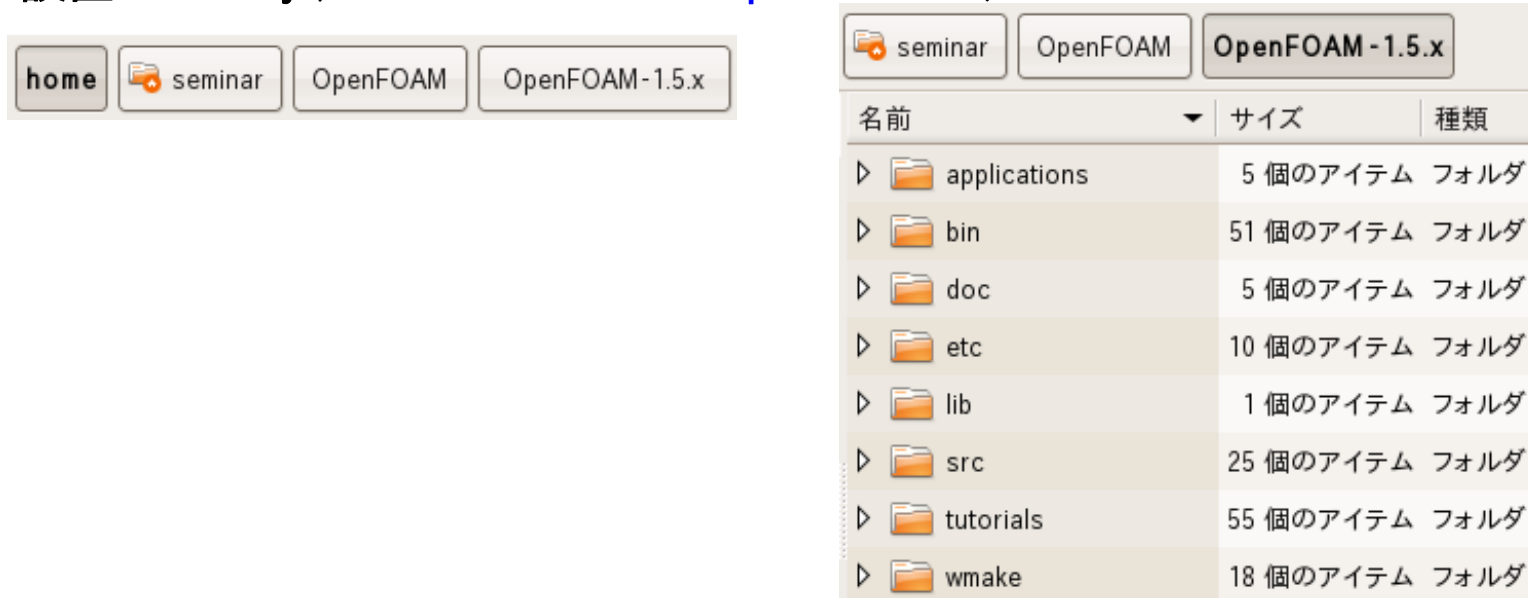


# 1. OpenFOAMとCFD

**継承**と呼ばれる OOP 言語の概念の一つを中心として、OOP の技法を有効に用いると、例えば実際に処理を行う関数の引数や戻り値などが変更されることがあっても、それを利用しているクラスには、修正の必要がないということが可能となる。このことは、関数の定義をしている部分と関数を利用している部分の分離が可能であることを示している。このためには、変更以前の関数が定義されていたクラスから、新しい物理モデルの導入などにより機能が追加された新しいクラスを、継承を用いて作成することが基本となる。**継承とは、クラスを最初から作成するのではなく、あるクラスを元に新たなクラスを作成することを指す**。そして、新たに作成されたクラスは、元のクラスの全機能を含み、さらに機能を追加することができる。このときの元となるクラスを基本クラスと呼び、基本クラスに機能を追加しているクラスを派生クラスと呼ぶ。また、派生クラスは別のクラスの基本クラスとなることが可能であり、全体として、**階層的な構造**が構成される。

## 2. OpenFOAMの構造

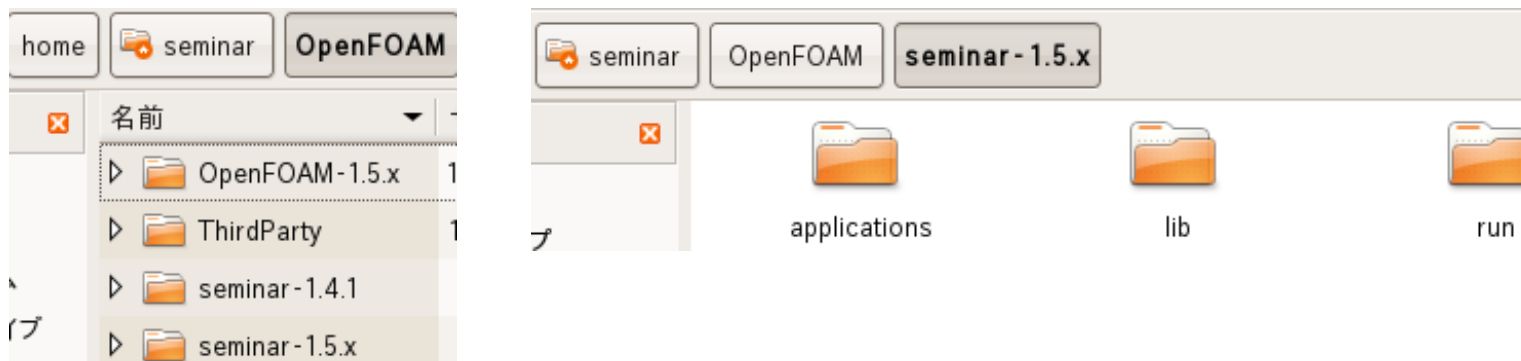
### □ 設置Directory ( /home/UserName/OpenFOAM/ )



The screenshot shows a file manager interface with a breadcrumb path: home > seminar > OpenFOAM > OpenFOAM-1.5.x. The main area displays a table of subdirectories:

名前	サイズ	種類
▷ applications	5 個のアイテム	フォルダ
▷ bin	51 個のアイテム	フォルダ
▷ doc	5 個のアイテム	フォルダ
▷ etc	10 個のアイテム	フォルダ
▷ lib	1 個のアイテム	フォルダ
▷ src	25 個のアイテム	フォルダ
▷ tutorials	55 個のアイテム	フォルダ
▷ wmake	18 個のアイテム	フォルダ

### □ UserDirectory ( /home/UserName/OpenFOAM/UserName-1.5?? )



The screenshot shows a file manager interface with a breadcrumb path: home > seminar > OpenFOAM > seminar-1.5.x. The main area displays three subdirectories:

- applications
- lib
- run



## 2. OpenFOAMの構造

---

### □ アプリケーションとライブラリ

OpenFOAMは実行のために、**基本的にC++のライブラリを用いている**。OpenFOAMはプリコンパイル済みの数多くのアプリケーションで構成されているがユーザーが独自に作成したり従来のものを修正しても構わない。

アプリケーションは大きく2つのカテゴリに分けられている。

**ソルバは数値連続体力学の特定の問題を解くためのもの**

**ユーティリティは主にデータ操作や代数計算を主に行うプリ・ポストプロセスを実行するもの**

OpenFOAMは一連のプリコンパイル済ライブラリに分けられ、それらはソルバとユーティリティの集合体を**ダイナミックにリンクする**。ユーザーが適宜独自のモデルをライブラリに追加できるように物理的モデルのこのようなライブラリはソースコードとして与えられる。





## 2. OpenFOAMの構造

---

### □ OpenFOAMのライブラリ

OpenFOAM配布のライブラリは\$FOAM\_LIB/\$WM\_OPTIONSディレクトリ内にあり、コマンド欄にlibと入力すればすぐに見つかります。一方、名前はlibを前につけて、例えばincompressibleTransportModelsが非圧縮性の輸送モデルのライブラリを含むというように合理的でかつ説明的です。表現を簡単にするためにライブラリは2つのタイプに分けられます。

#### 一般的ライブラリ

: これらは一般的なクラスや関連機能を備えている。

#### モデルライブラリ

: 計算連続体力学で使われるモデルを定める。

## 2. OpenFOAMの構造

### □ アプリケーションとライブラリ

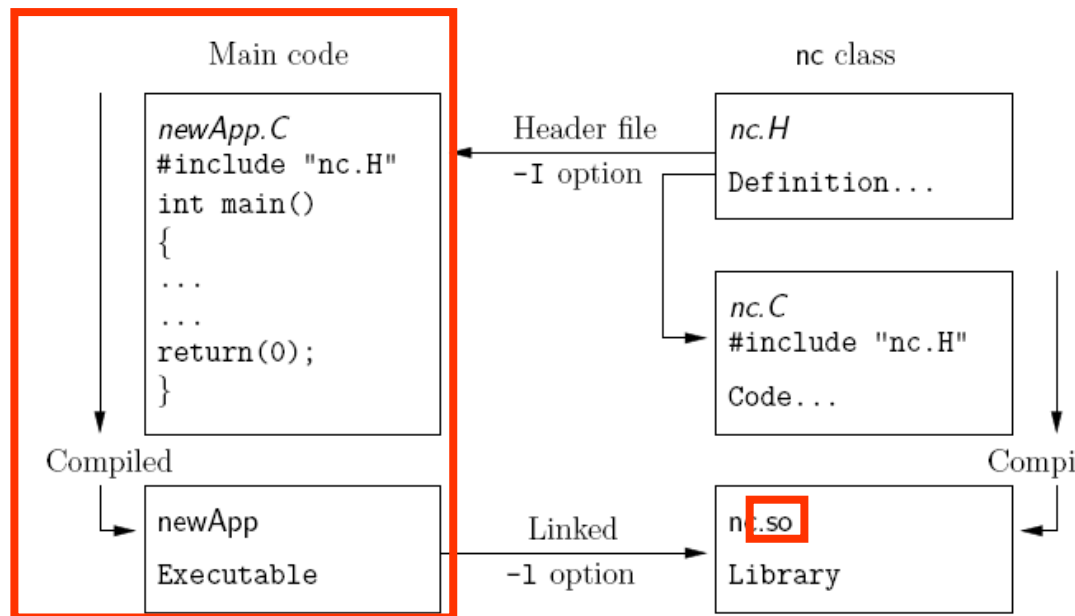


Figure 3.1: Header files, source files, compilation and linking.

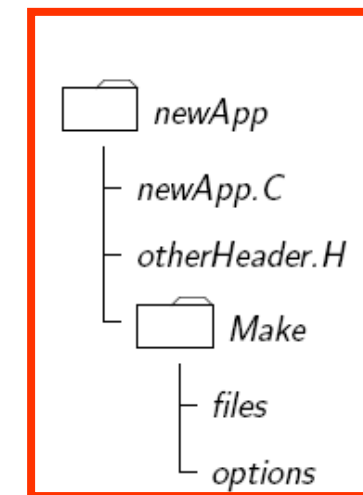


Figure 3.2: Directory structure for an application

```
wmake <optionalArguments> <optionalDirectory>
```

Argument	編集の種類
lib	静的にリンクされたライブラリの作成
libso	動的にリンクされたライブラリの作成
libo	静的にリンクされたオブジェクトファイルライブラリの作成
jar	JAVAアーカイブの作成
exe	特定のプロジェクトライブラリから独立したアプリケーションの作成

表3.1: wmakeのオプション編集Argument.

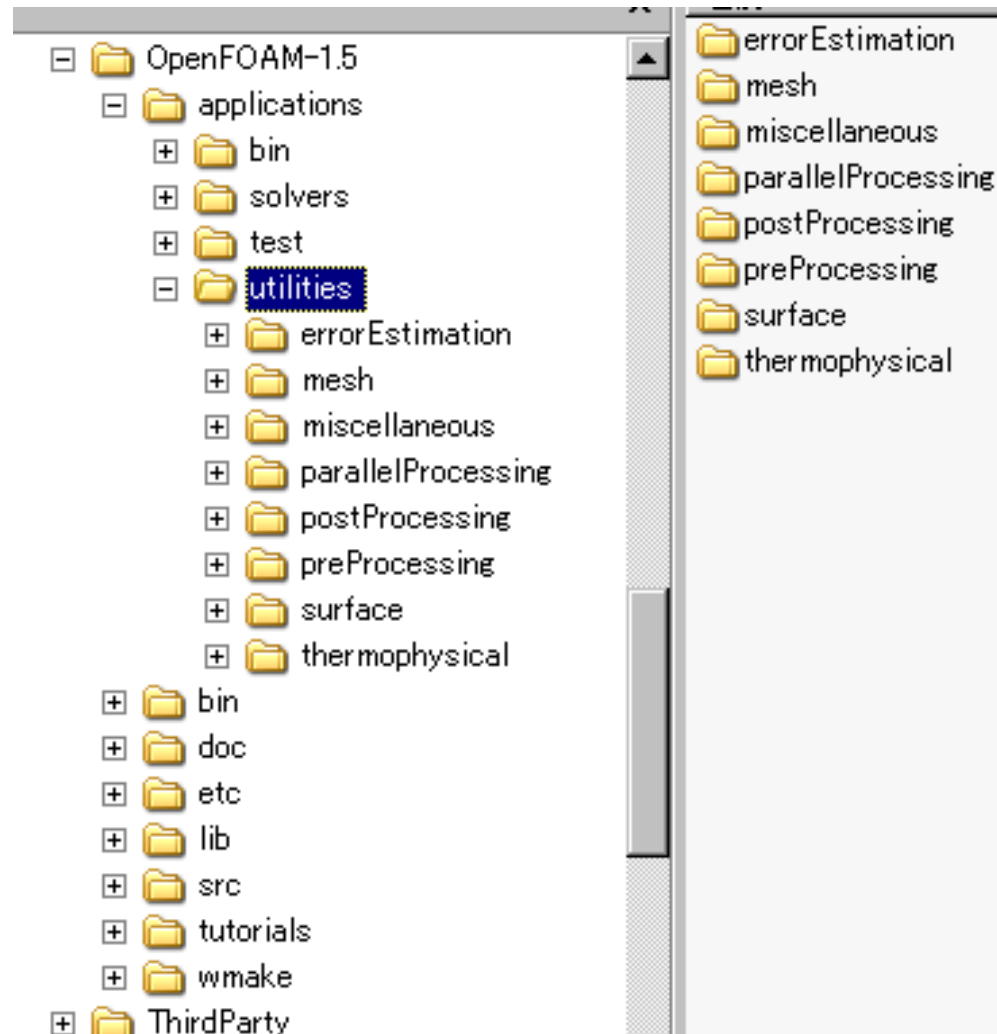
## 2. OpenFOAMの構造

- アプリケーション 端末上で「app」入力で「.../application/」に移動

The image shows a file explorer window displaying the OpenFOAM directory structure. The left pane shows a tree view with folders like 'jeong-1.5', 'lib', 'run', 'OpenFOAM-1.5', and 'applications'. The 'applications' folder under 'OpenFOAM-1.5' is highlighted with a red box. A blue callout box labeled '実行ファイル' (Executable File) points to the 'bin' folder. A blue callout box labeled 'Source Code' points to the 'solvers' and 'utilities' folders. The right pane shows a list of application files, with 'boundaryFoam' highlighted by a red box. Below the file list, a breadcrumb path is shown: 'OpenFOAM-1.5.x > applications > solvers > incompressible > boundaryFoam'. The bottom pane shows files like 'Make', 'boundaryFoam.C', and 'createFields.H'.

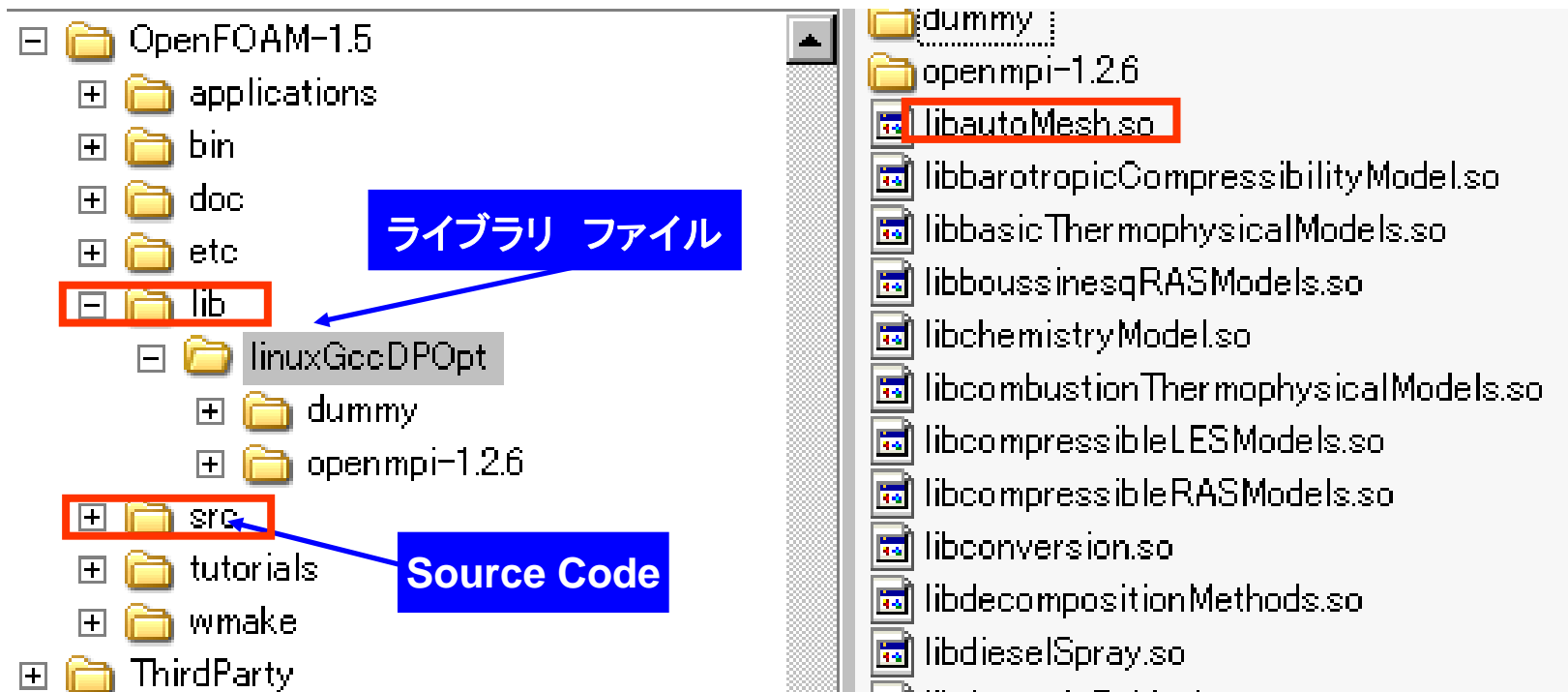
## 2. OpenFOAMの構造

□ Utilities 端末上で「util」入力で「.../utilities/」に移動



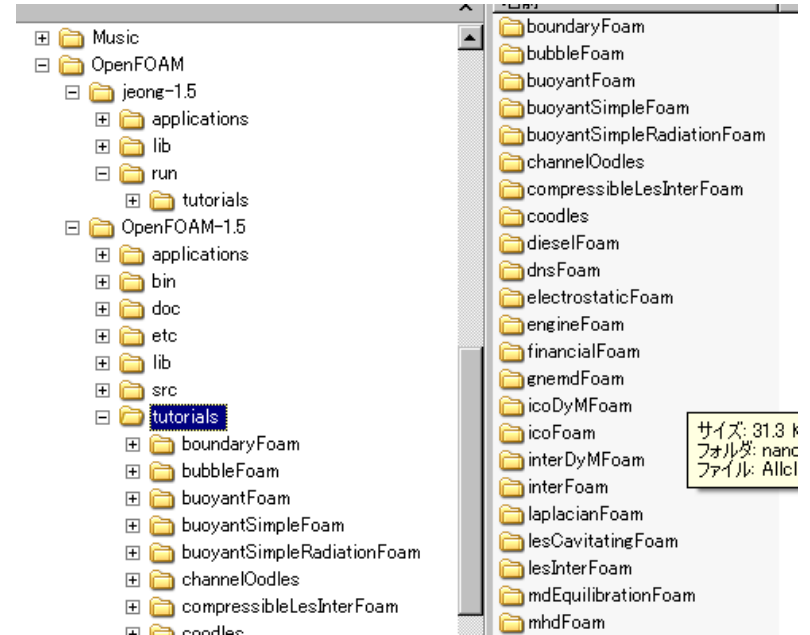
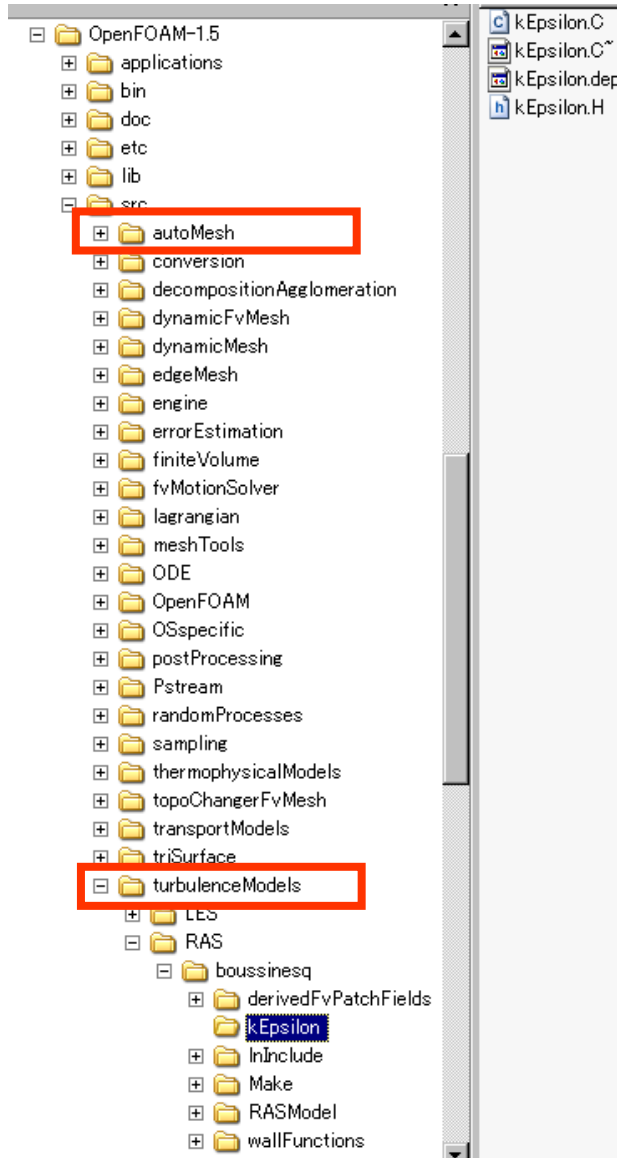
## 2. OpenFOAMの構造

- Lib 端末上で「lib」入力で「.../lib/」に移動



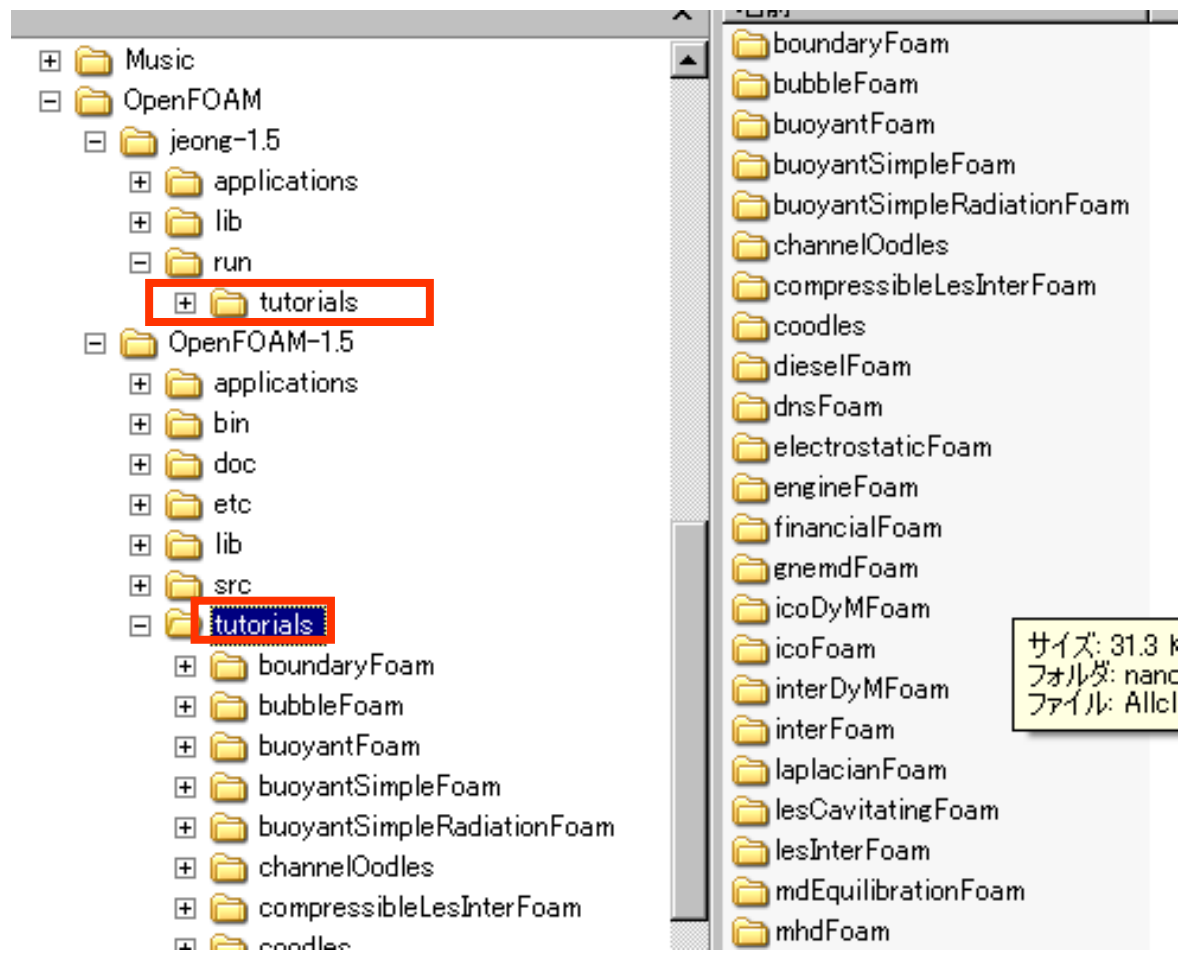
## 2. OpenFOAMの構造

SRC



## 2. OpenFOAMの構造

- Tutorial 端末上で「**tut**」 & Run 端末上で「**run**」



## 2. OpenFOAMの構造

### OpenFOAMの実行Directory

Solver名

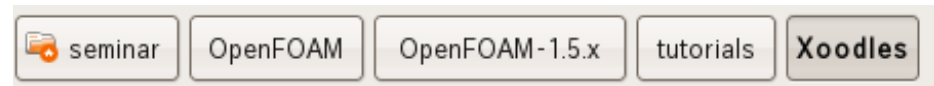
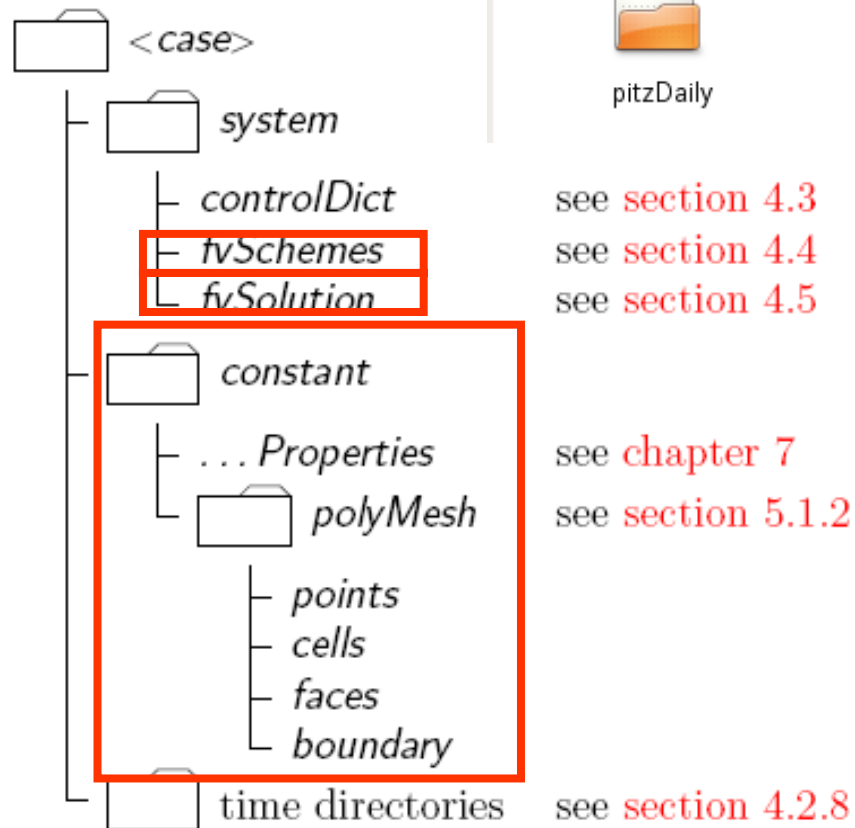


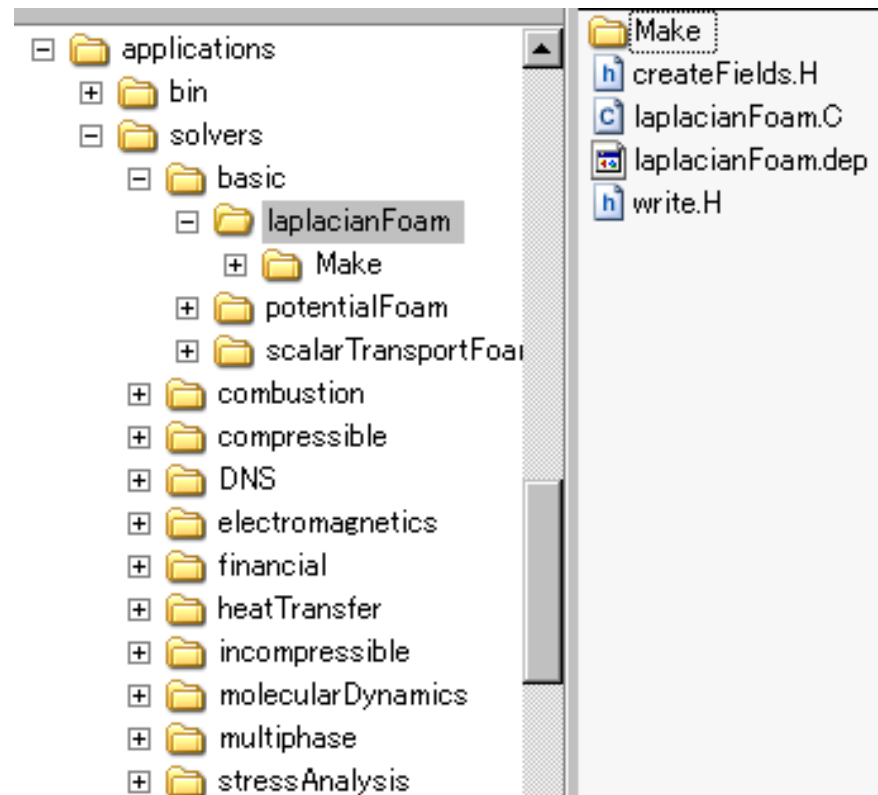
Figure 4.1: Case directory structure



## 2. OpenFOAMの構造

□ OpenFOAMのSolver `\OpenFOAM-1.5\applications\solvers\`

1. 基礎的なCFDコード (3種類)
2. 非圧縮性流れ (9種類)
3. 圧縮性流れ (12種類)
4. 多層流 (13種類)
5. DNS (1種類)
6. 燃焼 (8種類)
7. 熱輸送 (5種類)
8. 電磁流体 (2種類)
9. 固体応力解析 (2種類)
10. 金融工学 (1種類)
11. 分子力学 (2種類)





## 3. CFDの基礎

---

### □ CFDの具体化

物体外部・内部の流体領域を多数の格子あるいは検事体積(mesh or grid)で分けて計算する。(格子がない方法も有:MPS、SPH、MLSなど)

流体の流動を記述する偏微分方程式(支配方程式)をそれぞれの格子で周囲格子らの物理変数(圧力,速度,温度など)と関連した代数(algebraic)方程式として再表現される。

格子サイズを変化させて最適な解が(求める)救われる時まで詳述した方程式を数値的に解いていく。

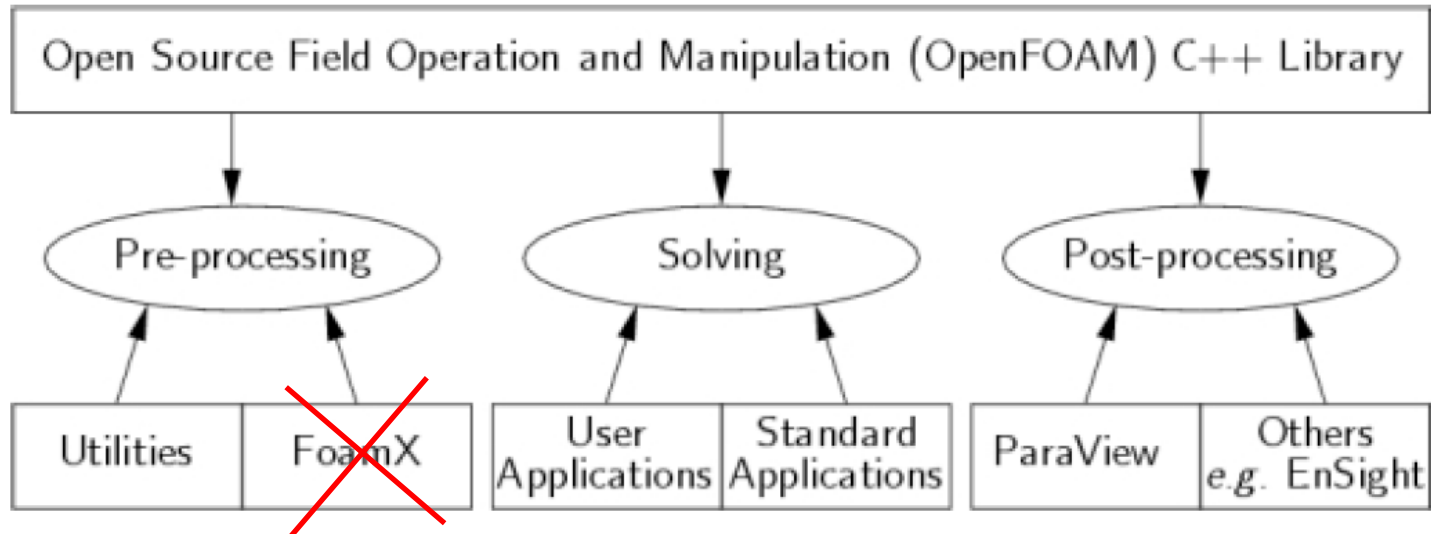
### 3. CFDの基礎

#### □ CFDの順序

モデルの形状に合う格子を生成(pre)

支配方程式を離散化された各格子点で解く(solve)

得られた結果を数値的・図式的に解析する(post)



### 3. CFDの基礎

- ◆ 離散化計算にはLagrangian記述とEulerian記述がある。
- ◆ **Lagrangian記述** : 時点を流れることと共に流速に移動させる。純粋な移流方程式では分布が保存されるから格子上の分布も変化しないで精度が高い計算が可能。

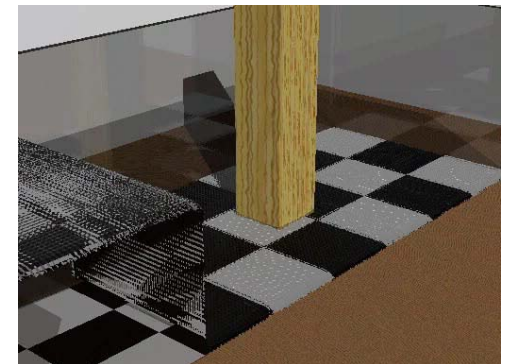
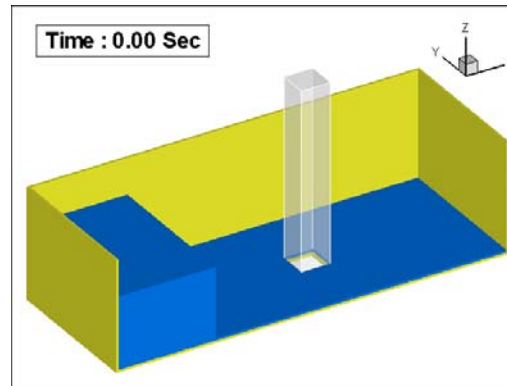
$$\frac{Du_i}{Dt} = -\frac{1}{\rho} \frac{\partial p}{\partial x_i} + \frac{\mu}{\rho} \frac{\partial^2 u_i}{\partial x_j^2} + f_i$$

- ◆ **Eulerian記述** : 格子を固定したまま移流を計算、移流方程式中の空間微分と時間微分を差分化する必要がある。

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{1}{\rho} \frac{\partial p}{\partial x_i} + \frac{\mu}{\rho} \frac{\partial^2 u_i}{\partial x_j^2} + f_i$$

対流

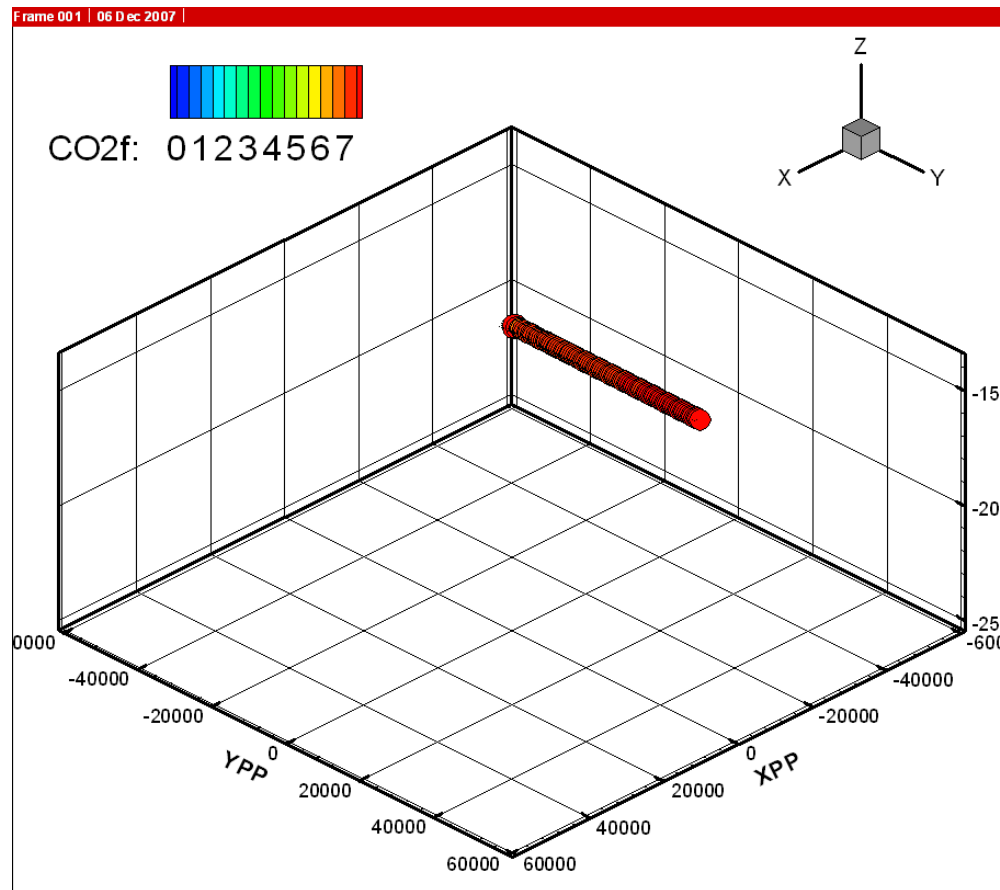
- ◆ ALE(arbitrary Lagrangian Eulerain)



### 3. CFDの基礎

The color and size of Particle = value of Concentration

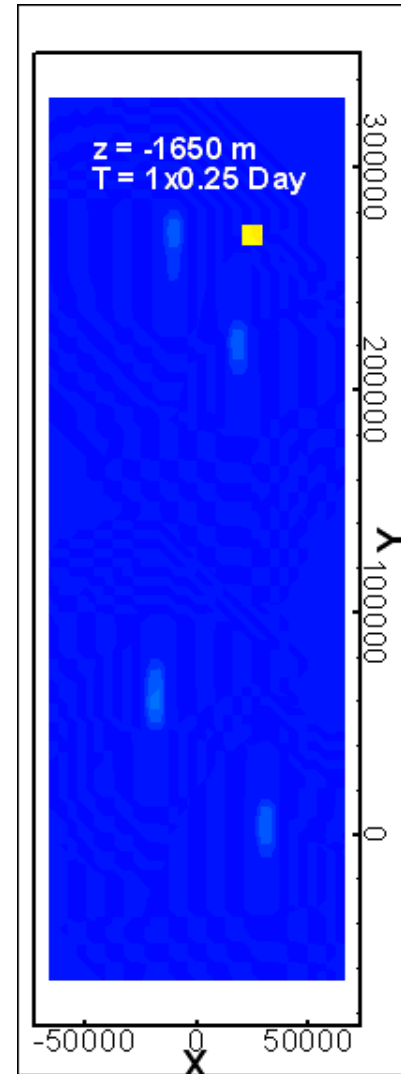
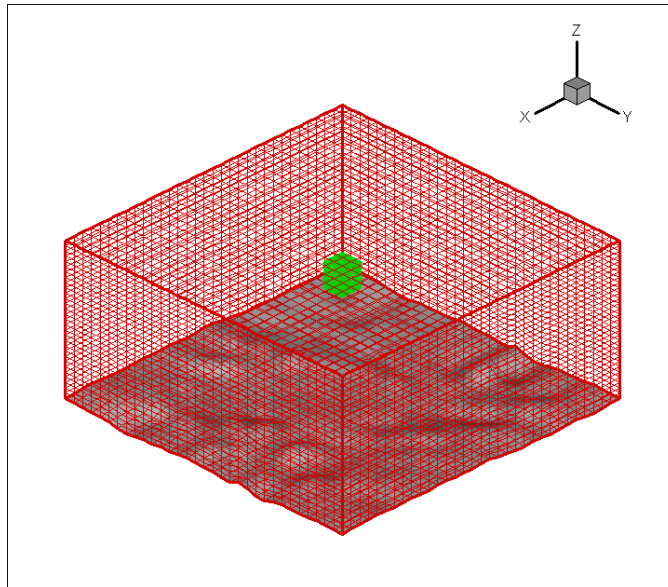
◆ ALE(arbitrary Lagrangian Eulerian)+粒子(濃度だけ)



**Time change of concentration by PLM in mesoscale domain near the injection plane ( $z=-2000$ ), Not initially placed case**

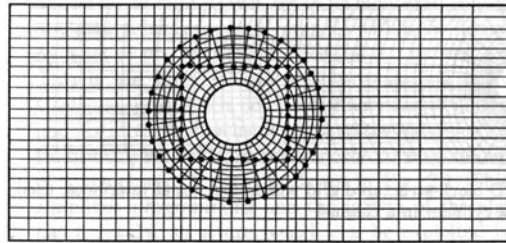
### 3. CFDの基礎

◆ ALE(arbitrary Lagrangian Eulerian)



### 3. CFDの基礎

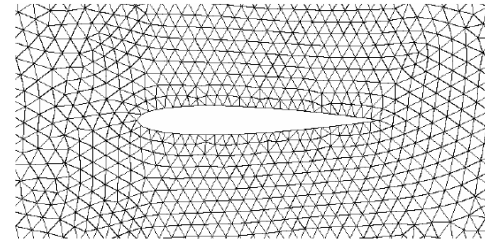
#### □ 格子 (Mesh、Grid) と変数の配置



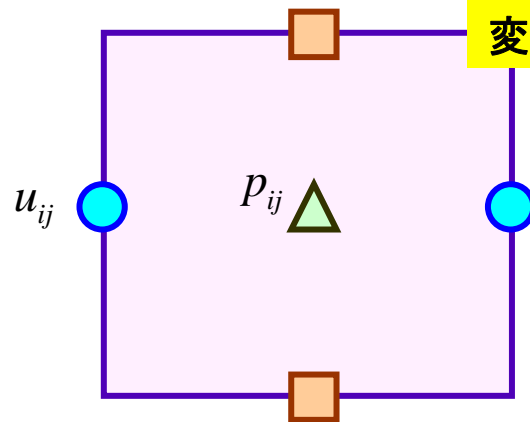
構造 (Structured) 格子  
直角 (Rectangular) 格子

物体適合 (body-fitted or general curvilinear) 格子

OpenFOAM

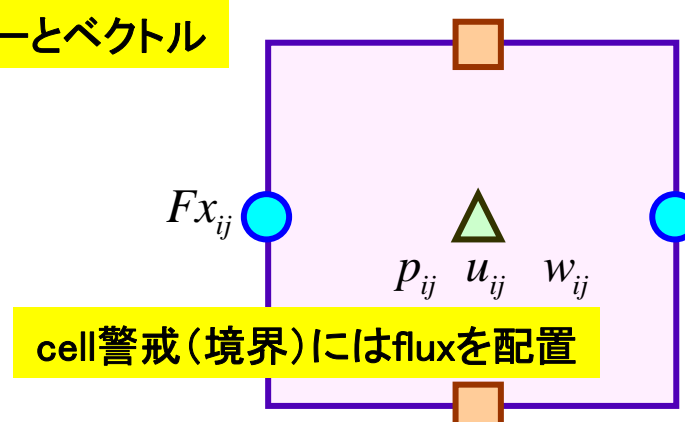


非構造格子  
(Unstructured Grid)



変数: スカラーとベクトル

Staggered 格子



cell警戒 (境界) には flux を配置

Co-located 格子



# 1. OpenFOAMとCFD

---

## □ OpenFOAMの特徴

これまで一般的に CFD コードは、手続き型言語、とりわけ FORTRAN によって書かれてきた。この場合、新しい物理モデルを組み込むためには、方程式の離散化など下位のレベルからプログラミングをやり直さなければならなかった。

それに対して、OpenFOAM には、連続体力学で使用される標準的なベクトルおよびテンソルに関する演算子がオブジェクトとして定義されている。

この演算子を使用することで、**数学的な記述としてプログラミングを行うことができる。**

このようなプログラミングスタイルのためには、オブジェクト指向プログラミング ( Object-Oriented Programming , 以後, OOP と略記) が応用できる。



### 3. CFDの基礎

#### □ 変数、Tensor、Fields

Tensor Class : Template Classは Field <type> (ex. Field<vector>)

読みやすい : typedef scalarField, tensorField... (in Instance)

離散化点との連携 : geometricField<type>

(Internal Field, Boundary Field, Mesh, Dimensions, Old Values..)

volField<Type> A field defined at cell centres;

surfaceField<Type> A field defined on cell faces;

pointField<Type> A field defined on cell vertices.

Rank	Common name	Basic class	Access functions
0	Scalar	scalar	
1	Vector	vector	x(), y(), z()
2	Tensor	tensor	xx(), xy(), xz()...

Table 1.1: Basic tensor classes in OpenFOAM

### 3. CFDの基礎

□ Meshの定義 :by Points, Faces, Cells, Boundary

-Basic Class : polyMesh

-Extended Class :

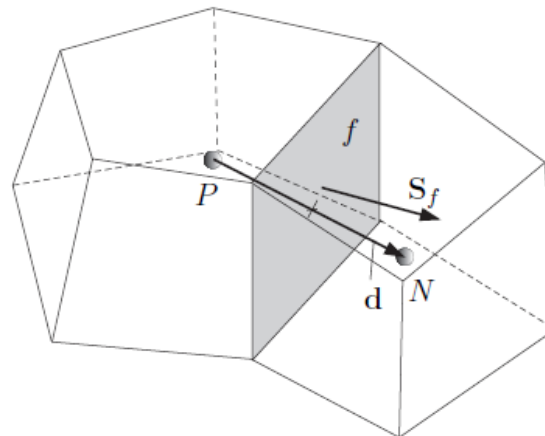


Figure 2.2: Parameters in finite volume discretisation

Class	Description	Symbol	Access function
volScalarField	Cell volumes	$V$	$V()$
surfaceVectorField	Face area vectors	$S_f$	$Sf()$
surfaceScalarField	Face area magnitudes	$ S_f $	$magSf()$
volVectorField	Cell centres	$C$	$C()$
surfaceVectorField	Face centres	$C_f$	$Cf()$
surfaceScalarField	Face motion fluxes **	$\phi_q$	$phi()$

Table 2.1: fvMesh stored data.



### 3. CFDの基礎

#### □ 各種演算子のVectorとTensorの表現

$$\nabla s = \partial_i s = \left( \frac{\partial s}{\partial x_1}, \frac{\partial s}{\partial x_2}, \frac{\partial s}{\partial x_3} \right)$$

$$\nabla \mathbf{a} = \partial_i a_j = \begin{pmatrix} \partial a_1 / \partial x_1 & \partial a_2 / \partial x_1 & \partial a_3 / \partial x_1 \\ \partial a_1 / \partial x_2 & \partial a_2 / \partial x_2 & \partial a_3 / \partial x_2 \\ \partial a_1 / \partial x_3 & \partial a_2 / \partial x_3 & \partial a_3 / \partial x_3 \end{pmatrix}$$

$$\nabla \cdot \mathbf{a} = \partial_i a_i = \frac{\partial a_1}{\partial x_1} + \frac{\partial a_2}{\partial x_2} + \frac{\partial a_3}{\partial x_3}$$

$$\nabla \cdot \mathbf{T} = \partial_i T_{ij} = \begin{pmatrix} \partial T_{11} / \partial x_1 + \partial T_{12} / \partial x_1 + \partial T_{13} / \partial x_1 \\ \partial T_{21} / \partial x_2 + \partial T_{22} / \partial x_2 + \partial T_{23} / \partial x_2 \\ \partial T_{31} / \partial x_3 + \partial T_{32} / \partial x_3 + \partial T_{33} / \partial x_3 \end{pmatrix}$$

$$\nabla^2 \equiv \partial^2 \equiv \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2} + \frac{\partial^2}{\partial x_3^2}$$

$$\nabla^2 s = \partial^2 s = \frac{\partial^2 s}{\partial x_1^2} + \frac{\partial^2 s}{\partial x_2^2} + \frac{\partial^2 s}{\partial x_3^2}$$

### 3. CFDの基礎

#### □ OpenFOAMの微分関数

陰解 (implicit)  
陽解 (explicit,?)

Term description	Implicit / Explicit	Text expression	fvm::/fvc:: functions
Laplacian	Imp/Exp	$\nabla^2 \phi$ $\nabla \cdot \Gamma \nabla \phi$	laplacian(phi) laplacian(Gamma, phi)
Time derivative	Imp/Exp	$\frac{\partial \phi}{\partial t}$ $\frac{\partial \rho \phi}{\partial t}$	ddt(phi) ddt(rho, phi)
Second time derivative	Imp/Exp	$\frac{\partial}{\partial t} \left( \rho \frac{\partial \phi}{\partial t} \right)$	d2dt2(rho, phi)
Convection	Imp/Exp	$\nabla \cdot (\psi)$ $\nabla \cdot (\psi \phi)$	div(psi, scheme)* div(psi, phi, word)* div(psi, phi)
Divergence	Exp	$\nabla \cdot \chi$	div(chi)
Gradient	Exp	$\nabla \chi$ $\nabla \phi$	grad(chi) gGrad(phi) lsGrad(phi) snGrad(phi) snGradCorrection(phi)
Grad-grad squared	Exp	$ \nabla \nabla \phi ^2$	sqrGradGrad(phi)
Curl	Exp	$\nabla \times \phi$	curl(phi)
Source	Imp Imp/Exp†	$\rho \phi$	Sp(rho, phi) SuSp(rho, phi)

†fvm::SuSp source is discretised implicit or explicit depending on the sign of rho.

†An explicit source can be introduced simply as a vol<Type>Field, e.g.rho\*phi.



### 3. CFDの基礎-偏微分方程式

#### □ 2次偏微分方程式の一般型

$$A \frac{\partial^2 \phi}{\partial x^2} + B \frac{\partial^2 \phi}{\partial x \partial y} + C \frac{\partial^2 \phi}{\partial y^2} + D \frac{\partial \phi}{\partial x} + E \frac{\partial \phi}{\partial y} + F \phi = 0$$

#### □ 偏微分方程式の判別式

$B^2 - 4AC > 0$  , 双曲線型 (hyperbolic)

$B^2 - 4AC = 0$  , 放物線型 (parabolic)

$B^2 - 4AC < 0$  , 楕円型 (elliptic)



### 3. CFDの基礎-偏微分方程式

□ 基本的な流動方程式の形態(圧縮性流体)

	定常流動	非定常流動
粘性流動	楕円型	放物線型
非粘性流動	双曲線型(超音速) 楕円型(亜音速)	双曲線型



### 3. CFDの基礎-偏微分方程式

#### □ 双曲線型

物理的に波動方程式(equation of wave)が対応.

$$\frac{\partial^2 \phi}{\partial t^2} - c^2 \frac{\partial^2 \phi}{\partial x^2} = 0$$

上記の式を次のような1次偏微分方程式で分解

$$\frac{\partial \phi}{\partial t} = -c \frac{\partial \phi}{\partial x}$$

$$\frac{\partial \phi}{\partial t} = +c \frac{\partial \phi}{\partial x}$$

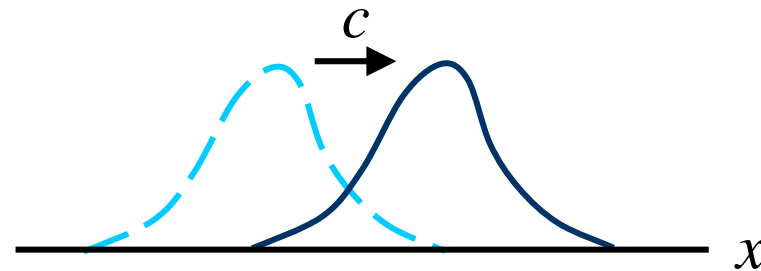
物理的意味:速度 $-c$ と $+c$ で伝播する波。

移流方程式(equation of advection)と呼ぶ。

初期条件と境界条件を与えて解が求められる。

### 3. CFDの基礎-偏微分方程式

- ◆ 時間 $t_0$ にある位置 $x_0$ の影響は、時間 $t > t_0$ にも有限領域 $x_0 - c(t - t_0) < x < x_0 + c(t - t_0)$ にだけ現れる。



- ◆ 移流方程式の離散化計算にはLagrangian記述とEulerian記述がある。
- ◆ **Lagrangian記述** : 時点を流れることと共に速度 $c$ に移動させる。純粋な移流方程式では分布が保存されるから格子上の分布も変化しないで精度が高い計算が可能。
- ◆ **Eulerian記述** : 格子を固定したまま移流を計算、移流方程式中の空間微分と時間微分を差分化する必要がある。





### 3. CFDの基礎-偏微分方程式

#### □ 放物線型

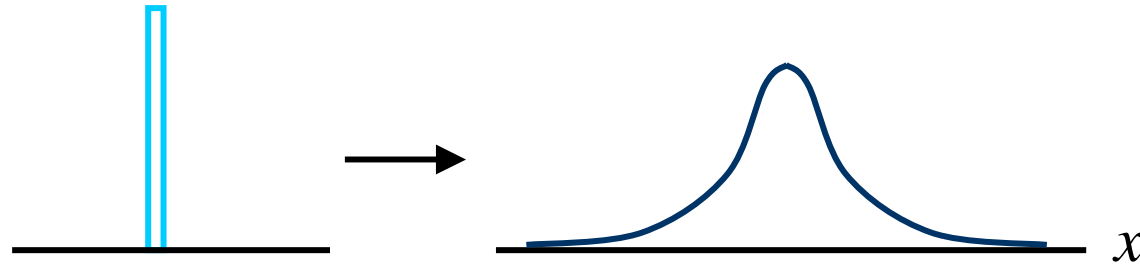
物理的に**非定常拡散方程式**が対応

$$\frac{\partial \phi}{\partial t} = \alpha \frac{\partial^2 \phi}{\partial x^2}$$

初期条件と境界条件を与えて解を得られう。ただ、双曲線型では影響領域が速度cに制限されるが**放物線型ではすべての計算領域が影響領域になる。**

初期条件で**delta関数**( $x, t_0$ )= $(x-x_0)$ を考慮すれば、非定常拡散方程式の解析しては**Gauss分布**になって、 $t > t_0$ のすべての位置xで増えたゼロでない値段を持つ。ただ、未来に起きることが過去に影響を与えることはなくて、時間方向に対しては一方的に影響を与える。

### 3. CFDの基礎-偏微分方程式



- ◆ ある位置での影響が一瞬に全領域に伝えられることは物理的にできない。
- ◆ 特殊相対性理論によって、いかなる影響でも光の速度を越えて伝えられることはないことが知られている。
- ◆ 拡散現象自体が統計的に導き出された2次元的な現象であり、近似的。
- ◆ したがって現象を支配する方程式が放物線型というのは無限の伝播速度を持つのを意味することだが、その方程式は実際の物理現象を近似するものでその物理現象にも必ず有限の伝播速度がある。



### 3. CFDの基礎-偏微分方程式

---

#### □ 楕円型

Laplace 方程式

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0$$

Poisson 方程式 : 速度・圧力連性によって誘導される式など

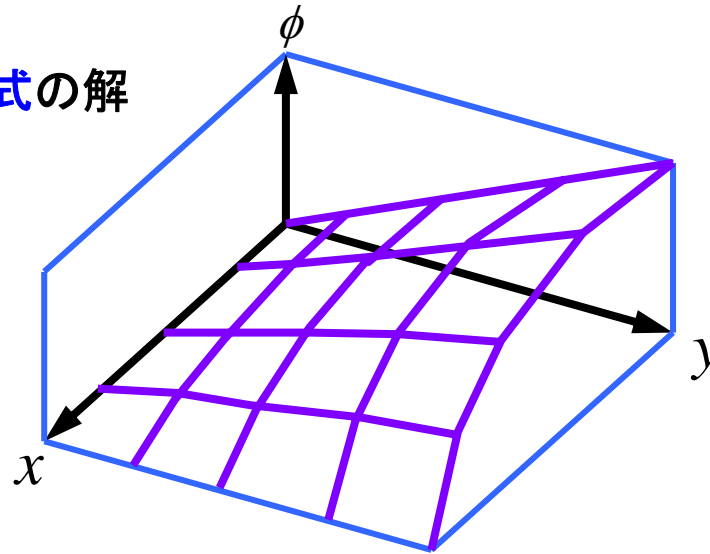
$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = S(x, y)$$

Helmholtz 方程式

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = \alpha \phi$$

### 3. CFDの基礎-偏微分方程式

#### ◆ 定常拡散方程式の解



- ◆ DirichletやNeumannの境界条件を与えて、解を得ることができる。(初期条件は?)
- ◆ 計算領域全体が互いに関係している。
- ◆ Poisson方程式右辺のsource項において任意の位置(x1,y1)で値S(x1,y1)を変化させれば、その影響は計算領域全体に広がる。放物線型では時間に対してだけは方向性があるが、楕円形ではそのような方向性を持つ変数はない。

## 1. 前回の要約

- ◆ 離散化計算にはLagrangian記述とEulerian記述がある。
- ◆ Lagrangian記述 : 時点を流れることと共に流速に移動させる。純粋な移流方程式では分布が保存されるから格子上の分布も変化しないで精度が高い計算が可能。

$$\frac{Du_i}{Dt} = -\frac{1}{\rho} \frac{\partial p}{\partial x_i} + \frac{\mu}{\rho} \frac{\partial^2 u_i}{\partial x_j^2} + f_i$$

- ◆ **Eulerian記述** : 格子を固定したまま移流を計算、移流方程式中の空間微分と時間微分を差分化する必要がある。

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{1}{\rho} \frac{\partial p}{\partial x_i} + \frac{\mu}{\rho} \frac{\partial^2 u_i}{\partial x_j^2} + f_i$$

対流

## 2. 偏微分方程式

### □ 対流と拡散 (Passive Scalarの移流拡散問題の場合)

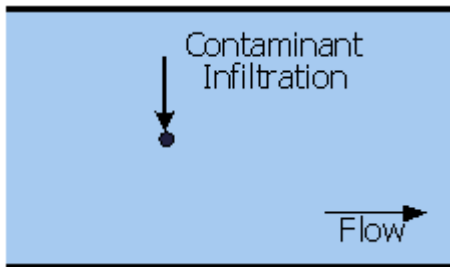


Fig. 1 河への汚染物質注入

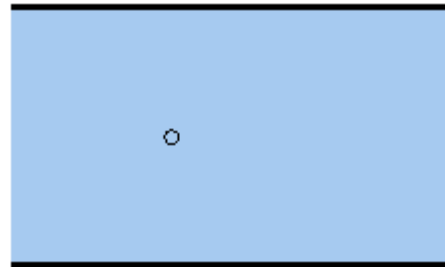


Fig. 2 移流拡散現象

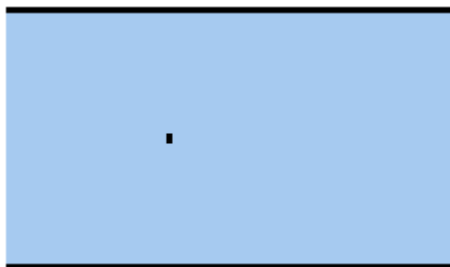


Fig. 3 移流現象

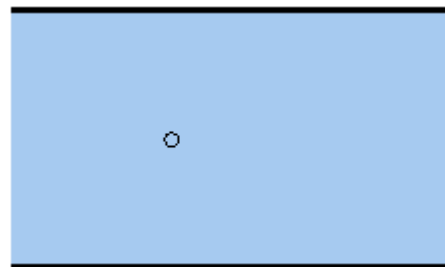


Fig. 4 拡散現象

1. 水の流れによる要因 (Fig. 2において右方向の運動)
  2. 濃度を均一化しようとする要因 (Fig. 2において上下左右の運動)
- この2つの要因を別々に表したものが, Fig. 3とFig. 4です。

### 3. CFDの基礎-支配方程式

□ 非圧縮性流体運動に対する支配方程式はN-S方程式と連続方程式。

一般(3次元)

$$\frac{\partial u}{\partial t} + \underbrace{u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} + w \frac{\partial u}{\partial z}}_{\text{対流}} = -\frac{1}{\rho} \frac{\partial p}{\partial x} + \underbrace{\frac{\mu}{\rho} \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right)}_{\text{拡散}} + f_x$$

$$\frac{\partial v}{\partial t} + u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} + w \frac{\partial v}{\partial z} = -\frac{1}{\rho} \frac{\partial p}{\partial y} + \frac{\mu}{\rho} \left( \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2} \right) + f_y$$

$$\frac{\partial w}{\partial t} + u \frac{\partial w}{\partial x} + v \frac{\partial w}{\partial y} + w \frac{\partial w}{\partial z} = -\frac{1}{\rho} \frac{\partial p}{\partial z} + \frac{\mu}{\rho} \left( \frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2} \right) + f_z$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0$$

$$\frac{\partial(\rho\alpha^n u)}{\partial t} + \frac{\partial(\rho\alpha^n u)}{\partial x} + \frac{\partial(\rho\alpha^n v)}{\partial y} + \frac{\partial(\rho\alpha^n w)}{\partial z} = \dot{m}$$

二層流の場合

Tensor表現

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{1}{\rho} \frac{\partial p}{\partial x_i} + \frac{\mu}{\rho} \frac{\partial^2 u_i}{\partial x_j^2} + f_i$$

$$\frac{\partial u_i}{\partial x_i} = 0$$

時間項無: 定常  
 密度が常数: 非圧縮性  
 拡散項(粘性)無: Eulerian Flow

Vector表現

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{1}{\rho} \nabla p + \frac{\mu}{\rho} \nabla^2 \mathbf{u} + \mathbf{f}$$

$$\nabla \cdot \mathbf{u} = 0$$

### 3. CFDの基礎-支配方程式

#### □ 2次元等方塑性構造物の支配方程式

#### Governing Equation

$$\rho \frac{Du}{Dt} = \frac{\partial}{\partial x} (\lambda \varepsilon_{kk} \delta_{ij} + 2\mu \varepsilon_{ij}) + K$$

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}, \quad \mu = \frac{E}{2(1+\nu)}$$

#### Derivation of a G.E.

term

rm

constants

- Lamé's constants are material constants that are derived Young Modulus and Poisson's Ratio



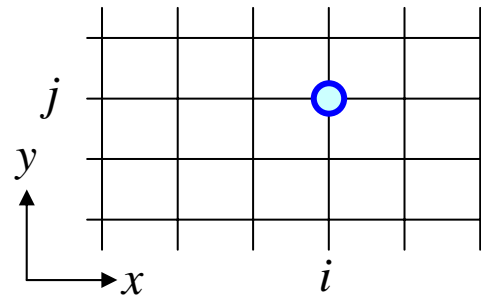
### 3. CFDの基礎-離散化

#### □ 離散化 (Discretization)

数値解析では連続の時間および空間を離散化する。

離散化された変数の表記法:

$\phi_{ij}^n$  ← 時間ステップ  
← 空間位置 Index



位置 $ij$ の右側は $i+1j$ ,左側は $i-1j$ 、上は $ij-1$ ,下は $ij+1$ 。

位置 $i-1/2j$ は格子点 $ij$ と $i+1j$ の間。

### 3. CFDの基礎-離散化

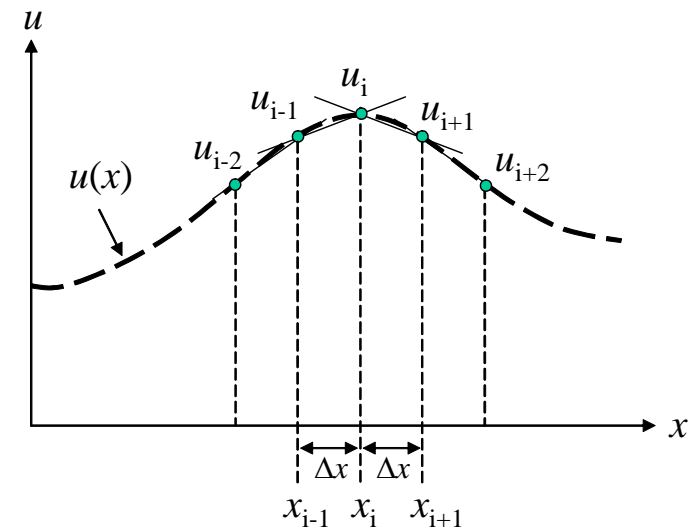
#### 有限差分法 (finite difference method, FDM)

差分(difference)は離散化された変数の代数式によって、微分(differentiation)を近似すること。

支配方程式の離散化のために  $\partial\phi/\partial x$  ,  $\partial^2\phi/\partial x^2$  などの微分演算は離散化値を使って整理する必要がある。

関数 $u(x)$ を点 $x=x_i$ 周囲のTaylor級数で展開。

$$\frac{\partial \mathbf{u}}{\partial t} \Big|_{x=x_i} + \nabla \cdot (\mathbf{u}\mathbf{u}) \Big|_{x=x_i} = \left( \frac{1}{\text{Re}} \nabla \mathbf{u} - \nabla p + \mathbf{f} \right) \Big|_{x=x_i}$$
$$\nabla \cdot \mathbf{u} \Big|_{x=x_i} = 0$$



### 3. CFDの基礎-離散化「精度と誤差」

$$u_{i+1} = u(x_{i+1}) = u(x_i + \Delta x) = u_i + \Delta x \left. \frac{\partial u}{\partial x} \right|_{x=x_i} + \frac{\Delta x^2}{2!} \left. \frac{\partial^2 u}{\partial x^2} \right|_{x=x_i} + \frac{\Delta x^3}{3!} \left. \frac{\partial^3 u}{\partial x^3} \right|_{x=x_i} + O(\Delta x^4) \quad (1)$$

$$u_{i-1} = u(x_{i-1}) = u(x_i - \Delta x) = u_i - \Delta x \left. \frac{\partial u}{\partial x} \right|_{x=x_i} + \frac{\Delta x^2}{2!} \left. \frac{\partial^2 u}{\partial x^2} \right|_{x=x_i} - \frac{\Delta x^3}{3!} \left. \frac{\partial^3 u}{\partial x^3} \right|_{x=x_i} + O(\Delta x^4) \quad (2)$$

$O(\Delta x^n)$ は  $\Delta x^n$ に比例する微小量。

$$\text{式(1)} - \text{式(2)} \quad u_{i+1} - u_{i-1} = 2\Delta x \left. \frac{\partial u}{\partial x} \right|_{x=x_i} + O(\Delta x^3)$$

$$\text{式(1)} + \text{式(2)} \quad u_{i+1} + u_{i-1} = 2u_i + \Delta x^2 \left. \frac{\partial^2 u}{\partial x^2} \right|_{x=x_i} + O(\Delta x^4)$$

微分の定義が無限要素の間隔をとった差分であるから微分と差分の対応関係がわかりやすい。

$$\frac{\delta u}{\delta x} \approx \frac{u_{i+1} - u_{i-1}}{2\Delta} = \frac{\partial u}{\partial x} + \frac{\Delta^2}{6} \frac{\partial^3 u}{\partial x^3} + \dots = \frac{\partial u}{\partial x} + O(\Delta^2) \quad \text{2nd-order accuracy}$$

$$\frac{\delta^2 u}{\delta x^2} \approx \frac{\frac{u_{i+1} - u_i}{\Delta} - \frac{u_i - u_{i-1}}{\Delta}}{\Delta} = \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\Delta^2}{12} \frac{\partial^4 u}{\partial x^4} + \dots = \frac{\partial^2 u}{\partial x^2} + O(\Delta^2)$$

2nd-order accuracy

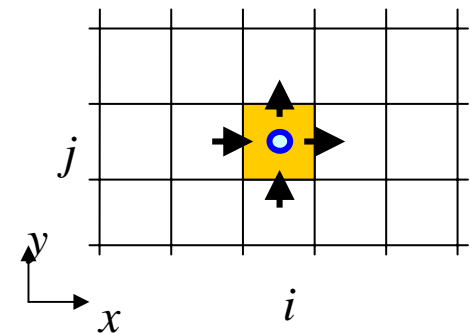
### 3. CFDの基礎-離散化

#### □ 有限体積法 (finite volume method, FVM)

格子点に変数を配置するのではなく格子によって囲まれた有限な領域(cell or control volume)に変数を配置する

FVMではcellとcellの間の境界にも変数を配置する。

例えば、移流方程式を解く時にcell  $ij$ と $i+1j$ 間にfluxの $F_{i+1/2 j}$ を配置する。このようにすることによって片方cellから流出する物理量と隣接するcellに流入する物理量が完全に一致する。これは数値流体力学で重要である**保存性**(conservation)を満足させることになる。このようにすれば支配方程式の発散(divergence)、勾配(gradient)、回転(rotation)をGaussの整理などによって、各cellの警戒(境界)積分に変換させることができる。



### 3. CFDの基礎-離散化

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{u}\mathbf{u}) = \frac{1}{\text{Re}} \nabla^2 \mathbf{u} - \nabla p + \mathbf{f}$$

積分

$$\int_V \frac{\partial \mathbf{u}}{\partial t} dV + \int_V \nabla \cdot (\mathbf{u}\mathbf{u}) dV = \int_V \frac{1}{\text{Re}} \nabla \cdot \nabla \mathbf{u} dV - \int_V \nabla \cdot p \mathbf{I} dV + \int_V \mathbf{f} dV$$

$$\frac{\partial V \mathbf{u}}{\partial t} + \int_V \nabla \cdot (\mathbf{u}\mathbf{u}) dV = \int_V \frac{1}{\text{Re}} \nabla \cdot \nabla \mathbf{u} dV - \int_V \nabla \cdot p \mathbf{I} dV + V \mathbf{f}$$

Gaussの整理

$$\frac{\partial V \mathbf{u}}{\partial t} + \int_S (\mathbf{u}\mathbf{u}) \cdot \mathbf{n} dS = \int_S \frac{1}{\text{Re}} (\nabla \mathbf{u}) \cdot \mathbf{n} dS - \int_S (p \mathbf{I}) \cdot \mathbf{n} dS + V \mathbf{f}$$

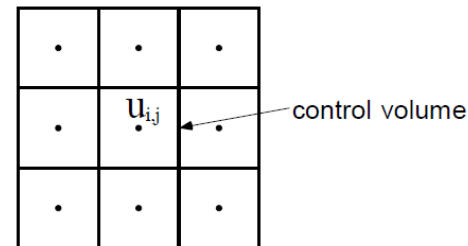
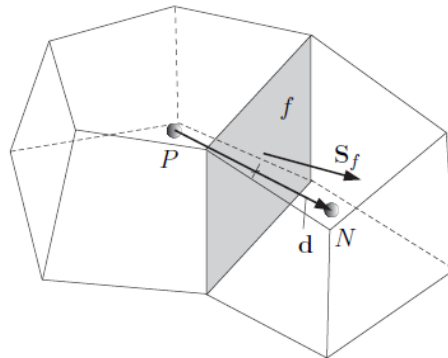


Figure 2.2: Parameters in finite volume discretisation

### 3. CFDの基礎-離散化

□ 有限要素法 (finite element method, FEM, Galerkin Method)

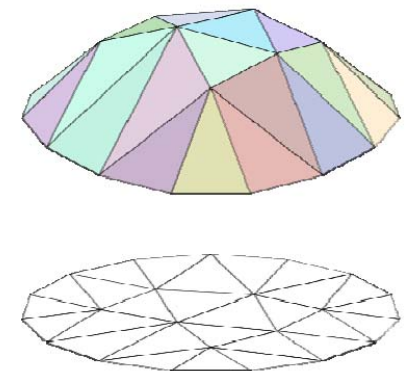
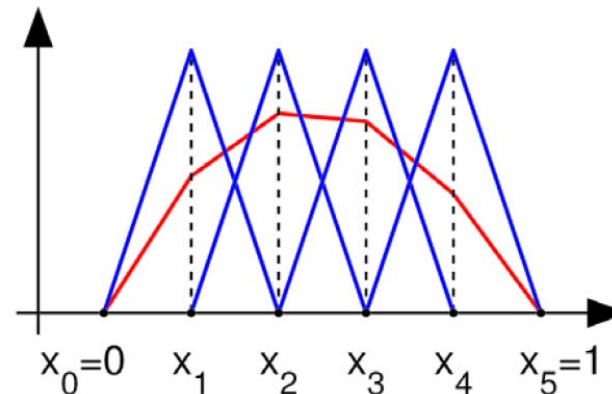
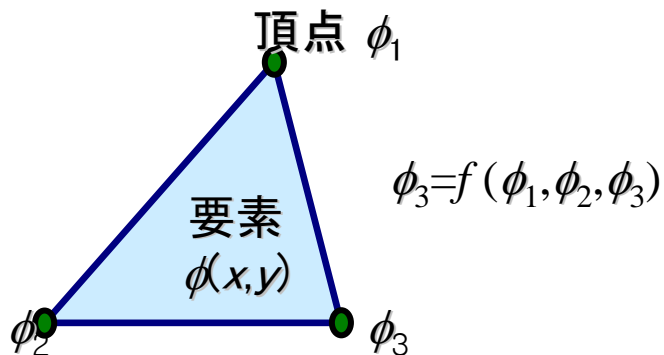
構造解析分野で幅広く使われる方法。

(熱)流動にも利用される。

変数を頂点上に配置。

要素は頂点で囲まれた面(2D)あるいは体積(3D)。

要素内の変数分布を頂点上に配置された変数の値から内挿。変数分布を支配方程式に代入した後、weight functionをかけて積分すれば、結局頂点上に配置された変数の関係式が得られる。変数の自由度の程度の独立したweight functionを準備しておけば必要な数の関係式を得ることができる。





### 3. CFDの基礎-離散化

#### □ 時間差分Scheme

$$\frac{\partial \phi}{\partial t} = f(\phi)$$

- オイラー陽解法、前進オイラー法(1次精度)

$$\phi^{n+1} = \phi^n + \Delta t f(\phi^n)$$

- オイラー陰解法、後退オイラー法(1次精度、行列解く必要)

$$\phi^{n+1} = \phi^n + \Delta t f(\phi^{n+1})$$

- Crank-Nicholson法(2次精度、行列解く必要)

$$\phi^{n+1} = \phi^n + \frac{1}{2} \Delta t f(\phi^{n+1} + \phi^n)$$

- Multi-Stage(修正オイラー、改良オイラー、Runge-Kuttaなど)

$$\phi^* = \phi^n + \frac{\Delta t}{2} f(\phi), \quad \phi^{n+1} = \phi^n + \Delta t f(\phi^*) \quad \phi^* = \phi^n + \Delta t f(\phi^n), \quad \phi^{n+1} = \phi^n + \frac{\Delta t}{2} [f(\phi^n + \phi^*)]$$

### 3. CFDの基礎-離散化

- CFD : 連続変微分を複数の代数式に変更して解く

$$[A] [x] = [b]$$

A: Geometryに定義された変数 (geometricField <type> FVMの場合、**volFiled<type>** )

B: 係数 (fvMatrix <type>, geometric<Type>Fieldの離散化で生成)

C: 生成項

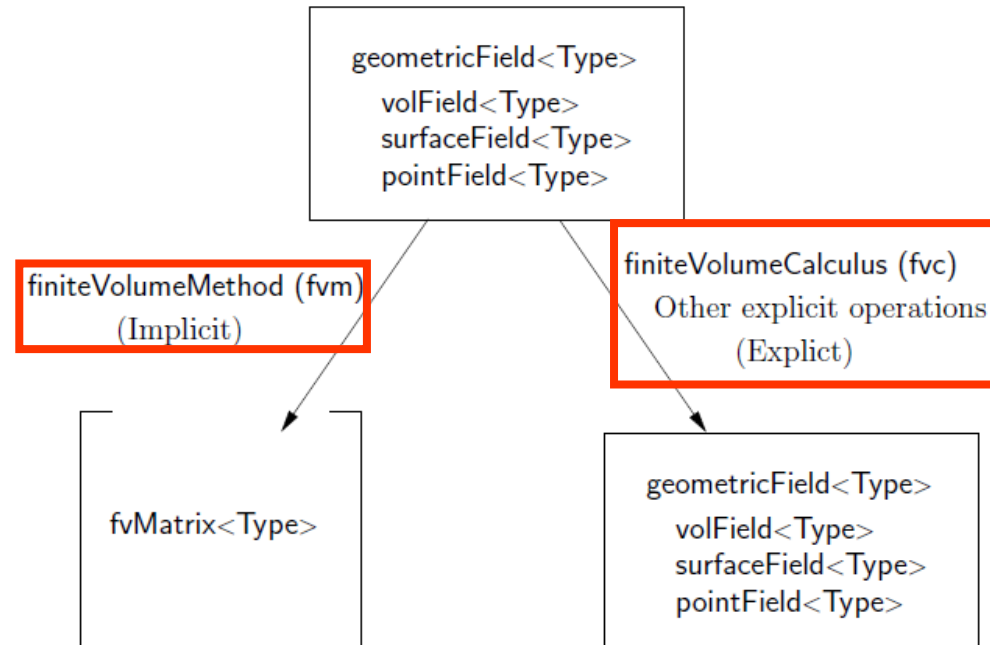
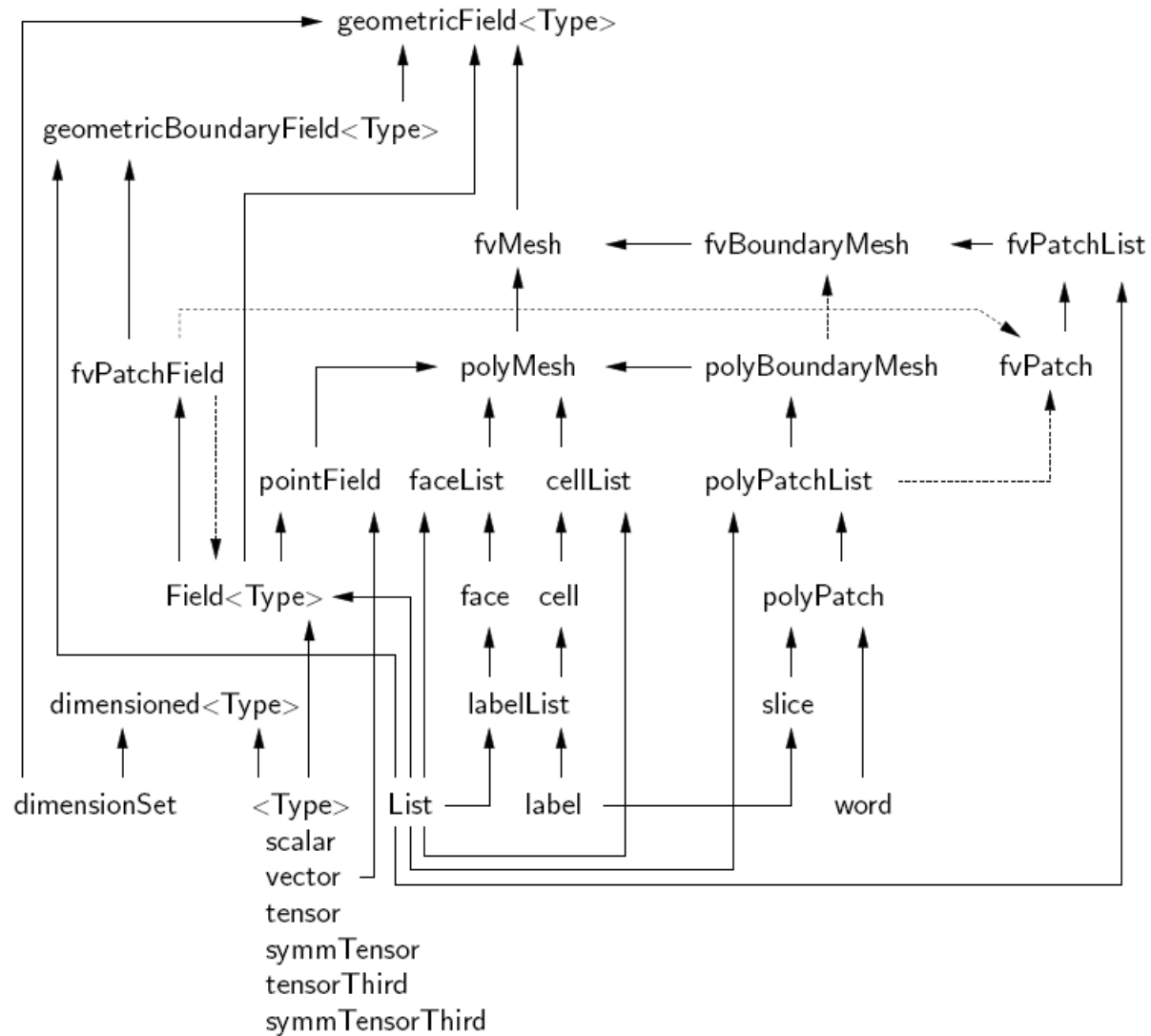


Figure 2.6: A geometricField<Type> and its operators



### 3. CFDの基礎-離散化

Classes for geometricField <type> class



### 3. CFDの基礎-離散化

#### □ OpenFOAMでの時間差分

<case>/system/fvSchemes

```

ddtSchemes
{
  default      Euler;
}

gradSchemes
{
  default      Gauss linear;
  grad(p)      Gauss linear;
}

divSchemes
{
  default      none;
  div(phi,U)   Gauss linear;
}

laplacianSchemes
{
  default      none;
  laplacian(nu,U) Gauss linear corrected;
  laplacian((1/A(U)),p) Gauss linear corrected;
}

interpolationSchemes
{
  default      linear;
  interpolate(HbyA) linear;
}
    
```

キーワード	数学的タームのカテゴリ
interpolationSchemes	2点間の値の補間
snGradSchemes	格子界面の法線方向勾配の各要素
gradSchemes	勾配 $\nabla$
divSchemes	発散 $\nabla \cdot$
laplacianSchemes	ラプラシアン $\nabla^2$
timeScheme	1次と2次の時間導関数 $\partial/\partial t, \partial^2/\partial t^2$
fluxRequired	フラックスの生成が必要な物理量

表4.5: fvSchemesで使用する主なキーワード

スキーム	説明
Euler	一次、制限、陰的
CrankNicholson $\psi$	二次、制限、陰的
backward	二次、陰的
steadyState	時間導関数について解かない

表4.11: ddtSchemesにおいて使用可能な離散化スキーム



### 3. CFDの基礎-数値安定性

#### □ 数値安定性

数値的に不安定な状態になれば変数分布が振動を始めてますます増加して行く。  
数値的な振動は差分スキームが原因で物理的な現象とは全く関係ない。したがって、差分スキームは数値的に安定した範囲以外には使うことはできない。

一般的に陽解法は数値安全性が悪くて陰解法は数値安全性が良い。

Euler陽解法と完全陰解法を比較した場合、どちらも1次精度だが数値安全性に関しては完陰解法が優れる。その代わり陰解法では行列方程式を解かなければならないし計算時間が長くなるという短所がある。

時間間隔幅が短いほど数値安全性が良くて長いほど悪い。したがって数値安全性を確保するためには時間間隔幅をある制限値段以下で設定しなければならない。

数値的に安定した範囲を分かる方法としてはNeumannの安全性解析法や行列の固有値解析法が知られている。だが、これら解析法は理想的な条件だけで解答を得ることができる。



### 3. CFDの基礎-数値安定性

#### □ 数値安定性

実際の計算コードではCourant数、拡散数(diffusion number)という簡単なパラメータを評価して概略的な安全性の限界が分かって、これらに安全計数をかけるなどDtを自動的に決めながら計算を進行させていく。

またDtの上限は非定常現状の解像度を確保するという点で制約を受けることになる。

数値安全性は時間差分スキームと空間差分スキームの両方に依存し、Courant数や拡散数は空間差分にも関与するパラメータ。

今まで説明した時間差分スキームに対し概略的に比較すると

完全陰解法 > C-N法 > R-K法 > A-B法 > Euler了解法

時間差分スキームに関しては陰的であるほど数値安全性が優れる。

### 3. CFDの基礎-数値安定性

#### □ Courant 数

$$C = \frac{\Delta t}{\Delta x} u$$

差分計算上に現れる無次元数。

物理的な意味はDtの間に格子を何ヶ分移流するのがCourant数の意味。

上流差分では時間差分スキームをEuler陽解法で適用した場合数値安全性の条件が $C < 1$ になる。この条件は結局tの間に許される移流距離は格子1ヶ分未満というものを意味。この条件をCFL条件(Courant-Friedrichs-Levy condition)と呼ぶ。

計算時間を節約するためにはできるだけ大きい時間間隔を利用することが望ましい。そのために計算コードでは通常Courant 数をコード中で計算してtを自動的に設定しながら計算を進行させていく。数値不安定性は局所的に起きるから各計算点のuとxからCourant 数を計算してtの最小値を取らなくてはならない。3次元の場合には各方向のCourant 数を加えてその値が1を越えないことが制約条件。

$$C_x + C_y + C_z < 1 \quad C_x = \frac{\Delta t}{\Delta x} u, \quad C_y = \frac{\Delta t}{\Delta y} v, \quad C_z = \frac{\Delta t}{\Delta z} w$$

### 3. CFDの基礎-離散化

#### □ 空間差分Scheme

空間差分で問題になるのは双曲線型方程式(移流方程式)中の1次微分であり、放物線型方程式(非定常拡散方程式)や楕円形方程式(定常拡散方程式)中の2次微分はほとんど問題にならない。

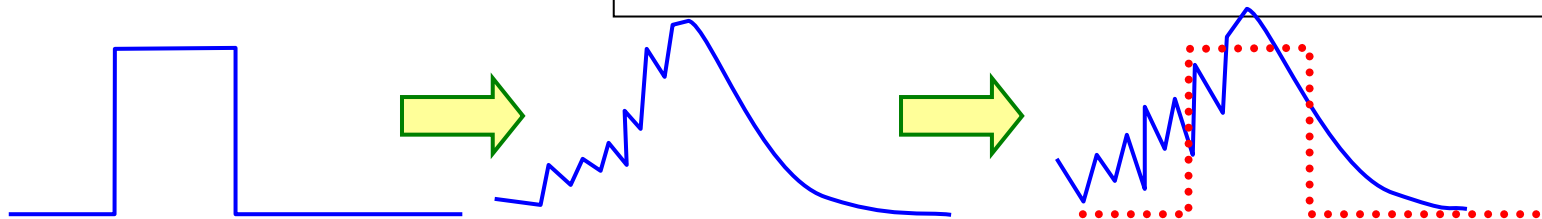
$$\frac{\partial \phi}{\partial t} = -u \frac{\partial \phi}{\partial x} \quad \phi \text{は任意のスカラー量、時間項はEuler陽解法}$$

#### ➤ 中心差分 (Central difference scheme、2次)

$$\phi^{n+1} = \phi_i^n - \Delta t u \frac{\phi_{i+1}^n - \phi_{i-1}^n}{\Delta x}$$

オーバーシュートが生じる  
( $\Delta t$ を小さくとっても消えない)

中心差分法に起因する数値不安定性の影響



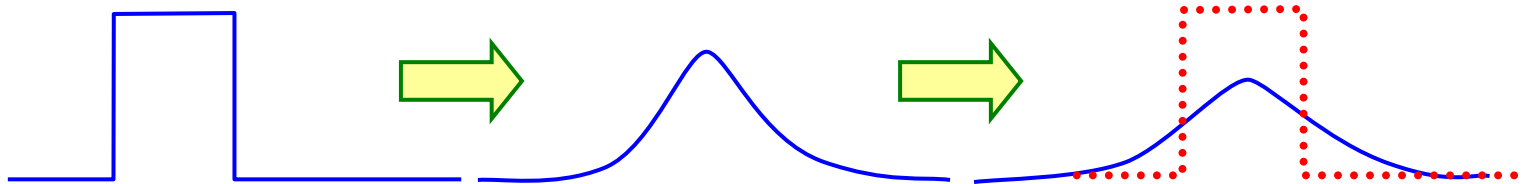
### 3. CFDの基礎-離散化

#### ➤ 上流差分 (Upwind difference scheme、1次)

$$\phi^{n+1} = \begin{cases} \phi_i^n - \Delta t u \frac{\phi_i^n - \phi_{i-1}^n}{\Delta x} & u \geq 0 \\ \phi_i^n - \Delta t u \frac{\phi_{i+1}^n - \phi_i^n}{\Delta x} & u < 0 \end{cases}$$

厳密解は初期分布そのまま一定速度  $u$  で右側に進行する

数値拡散が含まれるため、初期の形状がやわらかい曲線に



$$\phi_i^{n+1} = \phi_i^n - \Delta t u \frac{\phi_{i+1}^n - \phi_{i-1}^n}{2\Delta x} + \Delta t \left| \frac{u\Delta x}{2} \right| \frac{\phi_{i+1}^n - 2\phi_i^n + \phi_{i-1}^n}{\Delta x^2}$$

2次中心差分 + 数値拡散

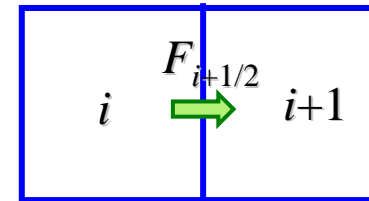
流動の主要な無次元数の Reynolds 数は対流項と拡散項の大きさに対する比。  
もし数値拡散があると、実際の物理的な Reynolds 数よりも小さい Reynolds 数の流動を計算したことになるのでこれは重大な問題になる。一方中心差分の場合は数値拡散はなくなるが計算結果に数値的な振動が起きる。これも実際の物理的な振動ではないので問題になる。

### 3. CFDの基礎-離散化

#### ➤ QUICK(3次)

QUICK (quadratic upstream interpolation for convective kinematics)スキームはcell境界の物理量 $i+1/2$ の内挿にその境界の隣接した2点だけでなくその外側の他の一点も利用した2次多項式による方法。

$$\phi_i^{n+1} = \begin{cases} \phi_i^n - \Delta t u \frac{(3\phi_{i+1} + 3\phi_i - 7\phi_{i-1} + \phi_{i-2})}{8\Delta x} & u \geq 0 \\ \phi_i^n - \Delta t u \frac{(3\phi_{i-1} + 3\phi_i - 7\phi_{i+1} + \phi_{i+2})}{8\Delta x} & u < 0 \end{cases}$$



$$\phi_i^{n+1} = \phi_i^n - \Delta t \frac{F_{i+1/2}^n - F_{i-1/2}^n}{V_i}$$
$$F_{i+1/2}^n = u_{i+1/2}^n \phi_{i+1/2}^n S_{i+1/2}$$

これは**実際に2次精度**しかない。その理由は $i-1/2$ と $i+1/2$ のflux計算が2次正確度であるため。だが、2次の誤差項を評価してみると、**中心差分**では計数が $1/6$ であるが、**QUICK**では $1/24$ で小さい。したがって同じ2次正確度といってもQUICK方が誤差が小さい。





### 3. CFDの基礎-離散化

➤ TVD (Total Variation Diminishing)

TVD条件を満足するスキーム

$$TV^n = \sum_j |\phi_{j+1}^n - \phi_j^n|$$

TVD条件は時間の進展にしたがってTVが減少しなくてはならない。具体的にcell境界での値を内挿する時、数値振動が生じやすいところには1次精度の上流差分を使ってその以外では高次精度のスキームに変えてくれる。

$$\phi_{i+1/2}^{n+1} = \begin{cases} \phi_i^n + \frac{1}{2} \Psi_{i-1/2}^+ (\phi_i^n - \phi_{i-1}^n) & u \geq 0 \\ \phi_{i+1}^n - \frac{1}{2} \Psi_{i+3/2}^- (\phi_{i+2}^n - \phi_{i+1}^n) & u < 0 \end{cases}$$

ここで $\Psi$ はlimiterで  
近辺の $\phi$ 分布により値を変化させる。

$$\begin{aligned} \Psi_{i-1/2}^+ &= \Psi_{i-1/2}^+(r_{i-1/2}^+) & r_{i-1/2}^+ &= \frac{\phi_{i+1}^n - \phi_i^n}{\phi_i^n - \phi_{i-1}^n} \\ \Psi_{i+3/2}^- &= \Psi_{i+3/2}^-(r_{i+3/2}^-) & r_{i+3/2}^- &= \frac{\phi_{i+1}^n - \phi_i^n}{\phi_{i+2}^n - \phi_{i+1}^n} \end{aligned}$$



### 3. CFDの基礎-離散化

#### □ 拡散数

放物線型方程式(非定常拡散方程式)にも同じ数値安全性の条件が存在する。

$$\frac{\partial \phi}{\partial t} = \alpha \frac{\partial^2 \phi}{\partial x^2} \quad \phi^{n+1} = \phi_i^n - \Delta t \alpha \frac{\phi_{i+1}^n - 2\phi_i^n + \phi_{i-1}^n}{\Delta x^2}$$

この時数値計算上の無次元数として拡散計数が現れる。

$$d = \alpha \frac{\Delta t}{\Delta x^2}$$

数値安全性の条件は $d < 0.5$ これやこれを時間間隔に対する制約条件として,

$$\Delta t = \frac{\Delta x^2}{2\alpha}$$

3次元では

$$d_x + d_y + d_z < \frac{1}{2}$$



### 3. CFDの基礎-離散化

#### □ 対流項と拡散項が共存するとき

両者の効果を付け加えたのが制約条件になる。重力項などの他の項らが含まれると、数値安全性はそれらからも制約される。

実際問題の数値安全性条件を正確に評価するのはだいぶ複雑で、通常的に普通の計算コードでは簡単にCourant数や拡散数の評価を行って近似的な数値安定条件を求めてそれに安全計数( $<1.0$ )をかけて、時間間隔を設定。

Courant 数と拡散数を比較してみると、Courant 数は格子間隔 $x$ の1乗に、拡散数は2乗に依存することから拡散数が格子間隔を狭めることに対して敏感に反応する。流動場解析では境界層の計算正確度を高めるために壁近辺に格子間隔を局所的に小さくする場合がある。このような場合に陽解法を利用するとCourant数と拡散数をだいぶ小さく取らなくてはならない。一方陰解法を適用すると、計算時間がかかることになる。また非定常問題を計算する場合には物理的現状の時間整数があってこの値より小さいを取らなくてはならない。したがって陰解法や了解法の選択は解こうとする流動場の特性に依存する。

### 3. CFDの基礎-離散化

#### □ OpenFOAMでの空間差分

##### ➤ 対流項 [<case>/system/fvSchemes](#)

$$\int_V \nabla \cdot (\Gamma \nabla \phi) dV = \int_S dS \cdot (\Gamma \nabla \phi) = \sum_f \Gamma_f S_f \cdot (\nabla \phi)_f$$

##### ➤ 拡散項

$$\int_V \nabla \cdot (\rho \mathbf{U} \phi) dV = \int_S dS \cdot (\rho \mathbf{U} \phi) = \sum_f S_f \cdot (\rho \mathbf{U})_f \phi_f = \sum_f F \phi_f$$

##### ➤ 発散

$$\int_V \nabla \cdot \phi dV = \int_S dS \cdot \phi = \sum_f S_f \cdot \phi_f$$

##### ➤ 勾配

$$\int_V \nabla \phi dV = \int_S dS \phi = \sum_f S_f \phi_f$$

```
ddtSchemes
{
    default          Euler;
}

gradSchemes
{
    default          Gauss linear;
    grad(p)          Gauss linear;
}

divSchemes
{
    default          none;
    div(phi,U)       Gauss linear;
}

laplacianSchemes
{
    default          none;
    laplacian(nu,U)  Gauss linear corrected;
    laplacian((1/A(U)),p) Gauss linear corrected;
}

interpolationSchemes
{
    default          linear;
    interpolate(HbyA) linear;
}
```

### 3. CFDの基礎-離散化

#### □ OpenFOAMの微分関数

陰解 (implicit)  
陽解 (explicit,?)

Term description	Implicit / Explicit	Text expression	fvm::/fvc:: functions
Laplacian	Imp/Exp	$\nabla^2\phi$ $\nabla \cdot \Gamma \nabla \phi$	laplacian(phi) laplacian(Gamma, phi)
Time derivative	Imp/Exp	$\frac{\partial \phi}{\partial t}$ $\frac{\partial \rho \phi}{\partial t}$	ddt(phi) ddt(rho, phi)
Second time derivative	Imp/Exp	$\frac{\partial}{\partial t} \left( \rho \frac{\partial \phi}{\partial t} \right)$	d2dt2(rho, phi)
Convection	Imp/Exp	$\nabla \cdot (\psi)$ $\nabla \cdot (\psi \phi)$	div(psi, scheme)* div(psi, phi, word)* div(psi, phi)
Divergence	Exp	$\nabla \cdot \chi$	div(chi)
Gradient	Exp	$\nabla \chi$ $\nabla \phi$	grad(chi) gGrad(phi) lsGrad(phi) snGrad(phi) snGradCorrection(phi)
Grad-grad squared	Exp	$ \nabla \nabla \phi ^2$	sqrGradGrad(phi)
Curl	Exp	$\nabla \times \phi$	curl(phi)
Source	Imp Imp/Exp†	$\rho \phi$	Sp(rho, phi) SuSp(rho, phi)

†fvm::SuSp source is discretised implicit or explicit depending on the sign of rho.

†An explicit source can be introduced simply as a vol<Type>Field, e.g.rho\*phi.

### 3. CFDの基礎-離散化

- OpenFOAMの補間スキーム(ベクトル場に対するスキームのみ示す)

中心スキーム	
linear	線形補間 (中心差分)
cubicCorrection	体積スキーム
midPoint	均等重み付け線形補間

TVDスキーム	
limitedLinear	有限線形差分
vanLeer	van Leer リミッター
MUSCL	MUSCL リミッター
limitedCubic	体積リミッター
NVDスキーム	
SFCD	自動フィルター中心差分
Gamma $\psi$	ガンマ差分

風上対流スキーム	
upwind	風上差分
linearUpwind	線形風上差分
skewLinear	ひずみ補正付き線形
QUICK	二次風上差分

### 3. CFDの基礎-離散化

#### □ OpenFOAM空間差分

離散化スキーム	説明
Gauss	一次のガウス積分
leastSquares	二次の最小二乗法
fourth	四次の最小二乗法
limited	上記のスキームの制限バージョン

表4.8: gradSchemesにおいて使用できる離散化スキーム

スキーム	数値的性質
corrected	無制限、二次、保守的
uncorrected	制限、一次、非保守的
limited $\psi$	補正と非補正の混合
bounded	制限スカラーの一次
fourth	無制限、四次、保守的

表4.9: laplacianSchemesにおける表面法線方向スキームの性質

スキーム	数値的性質
linear	二次、無制限
skewLinear	二次、(より) 無制限、ひずみ補正
cubicCorrected	四次、無制限
upwind	四次、制限
linearUpwind	一次/二次、制限
QUICK	一次/二次、制限
TVD schemes	一次/二次、制限
SFCD	二次、制限
NVD schemes	一次/二次、制限

表4.10: divSchemesにおいて使用される補間スキームの性質

### 3. CFDの基礎-連性Algorithm

MAC法では、ナビエ・ストークス方程式と連続の式から圧力に関するポアソン方程式を得る。

ナビエ・ストークス方程式をそれぞれ微分して和を取る。

圧力に関するポアソン方程式

$$\begin{aligned} \frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} &= -\frac{1}{\rho} \frac{\partial p}{\partial x} + \nu \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \quad \xrightarrow{x\text{微分}} \\ \frac{\partial v}{\partial t} + u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} &= -\frac{1}{\rho} \frac{\partial p}{\partial y} + \nu \left( \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) \quad \xrightarrow{y\text{微分}} \end{aligned}$$

$$\Delta p = \left\{ \left( \frac{\partial u^n}{\partial x} \right)^2 + 2 \left( \frac{\partial u^n}{\partial x} \right) \left( \frac{\partial v^n}{\partial y} \right) + \left( \frac{\partial v^n}{\partial y} \right)^2 \right\} - \frac{D^{n+1} - D^n}{\Delta t} - \frac{\partial D^n}{\partial x} - \frac{\partial D^n}{\partial y} + \frac{1}{Re} \Delta D^n$$

$$\left[ \Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}, D = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right]$$

本来であればD=0となるはずであるが、数値計算ではそうはならない。

そこで、 $D^{n+1} = 0$  とおく。

$$\Delta p = \left\{ \left( \frac{\partial u^n}{\partial x} \right)^2 + 2 \left( \frac{\partial u^n}{\partial x} \right) \left( \frac{\partial v^n}{\partial y} \right) + \left( \frac{\partial v^n}{\partial y} \right)^2 \right\} + \frac{D^n}{\Delta t} - \frac{\partial D^n}{\partial x} - \frac{\partial D^n}{\partial y} + \frac{1}{Re} \Delta D^n$$

この圧力ポアソン方程式を解くことにより、 $D$ を小さな値に留める、すなわち連続の式が満足されるような速度場となるような圧力場を求めることができる。



## 3. CFDの基礎-連性Algorithm

### □ OpenFOAMの連性Algorithm

OpenFOAMのほとんどの流体力学ソルバアプリケーションはpressure-implicit splitoperator(PISO)もしくはsemi-implicit method for pressure-linked equations(SIMPLE)アルゴリズムを使用します。PIMPLE (PISO+SIMPLE)も有

**PISO** : R. I. Issa により開発された**圧力-流速分離型解法**の一種、少ない反復回数で非定常問題を高精度に解く、**三段階**に分かれており以下のようなになる。

- (1) 前の時間ステップの圧力場を初期推測値として、  
運動方程式が解かれ、**速度場の予測値**を得る。
- (2) 予測された**速度場の発散**をソース項とする**圧力ポアソン方程式**が解かれて、  
新しい修正圧力場が得られる。
- (3) **修正圧力場**を使用して**速度場**が修正される。

ステップ(2)と(3)は、収束解が得られるまで必要な回数(nCorr 回など)繰り返される。この反復回数は、時間ステップに依存しているが、それぞれの時間ステップごとに通常二回のループで十分である。

### 3. CFDの基礎-連性Algorithm

#### SIMPLE法

対流項

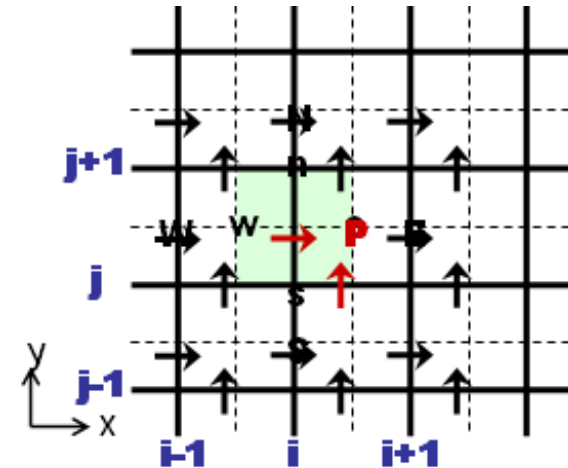
$$\begin{aligned} \iiint U \frac{\partial U}{\partial x} dx dy dz &= \Delta y \Delta z [U U_e - U U_w] \\ &= DY \cdot DZ \left[ \frac{U_{i,j,k} + U_{i+1,j,k}}{2} U_e - \frac{U_{i-1,j,k} + U_{i,j,k}}{2} U_w \right] \\ &= CE \cdot \frac{U_{i,j,k} + U_{i+1,j,k}}{2} - CW \cdot \frac{U_{i-1,j,k} + U_{i,j,k}}{2} \end{aligned}$$

$$\begin{aligned} \iiint V \frac{\partial U}{\partial y} dx dy dz &= \Delta x \Delta z [V U_n - V U_s] \\ &= DX \cdot DZ \left[ \frac{V_{i,j+1,k} + V_{i-1,j+1,k}}{2} U_n - \frac{V_{i,j,k} + V_{i-1,j,k}}{2} U_s \right] \\ &= AY [VN \cdot U_n - VS \cdot U_s] = CN \cdot \frac{U_{i,j+1,k} + U_{i,j,k}}{2} - CS \cdot \frac{U_{i,j,k} + U_{i,j-1,k}}{2} \end{aligned}$$

拡散項

$$\begin{aligned} \iiint v \frac{\partial^2 U}{\partial x^2} dx dy dz &= \Delta y \Delta z \left[ \left( v \frac{\partial U}{\partial x} \right)_e - \left( v \frac{\partial U}{\partial x} \right)_w \right] = AX \left[ VIS_e \frac{U_{i+1,j,k} - U_{i,j,k}}{DX_i} - VIS_w \frac{U_{i,j,k} + U_{i-1,j,k}}{DX_{i-1}} \right] \\ &= DE \cdot (U_{i+1,j,k} - U_{i,j,k}) - DW \cdot (U_{i,j,k} - U_{i-1,j,k}) \end{aligned}$$

圧力項  $-\iiint \frac{1}{\rho} \frac{\partial p}{\partial x} dx dy dz = -\Delta y \Delta z \frac{1}{\rho} [P_e - P_w] = -AX [P_{i,j,k} - P_{i-1,j,k}] / \left( \frac{R_{i,j,k} + R_{i-1,j,k}}{2} \right) = SU$



スタガード格子、  
有限体積法 (FVM)

### 3. CFDの基礎-連性Algorithm

#### SIMPLE法

NS式に代入

$$\begin{aligned}
 & CE \cdot \frac{U_{i,j,k} + U_{i+1,j,k}}{2} - CW \cdot \frac{U_{i-1,j,k} + U_{i,j,k}}{2} + CN \cdot \frac{U_{i,j+1,k} + U_{i,j,k}}{2} - CS \cdot \frac{U_{i,j-1,k} + U_{i,j,k}}{2} \\
 & + CT \cdot \frac{U_{i,j,k} + U_{i,j,k+1}}{2} - CB \cdot \frac{U_{i,j,k-1} + U_{i,j,k}}{2} \\
 & = DE \cdot (U_{i+1,j,k} - U_{i,j,k}) - DW \cdot (U_{i,j,k} - U_{i-1,j,k}) + DN \cdot (U_{i,j+1,k} - U_{i,j,k}) - DS \cdot (U_{i,j,k} - U_{i,j-1,k}) \\
 & + DT \cdot (U_{i,j,k+1} - U_{i,j,k}) - DB \cdot (U_{i,j,k} - U_{i,j,k-1}) + SU
 \end{aligned}$$



$$\begin{aligned}
 & [AE + AW + AN + AS + AT + AB + (CE - CW + CN - CS + CT - CB)] \cdot U_{i,j,k} \\
 & = AE \cdot U_{i+1,j,k} + AW \cdot U_{i-1,j,k} + AN \cdot U_{i,j+1,k} + AS \cdot U_{i,j-1,k} + AT \cdot U_{i,j,k+1} + AB \cdot U_{i,j,k-1} + SU
 \end{aligned}$$

$$\begin{aligned}
 AP \cdot U_{i,j,k} & = AE \cdot U_{i+1,j,k} + AW \cdot U_{i-1,j,k} + AN \cdot U_{i,j+1,k} + AS \cdot U_{i,j-1,k} + AT \cdot U_{i,j,k+1} + AB \cdot U_{i,j,k-1} \\
 & + SP \cdot U_{i,j,k} + SU
 \end{aligned}$$

Hybrid法

$$\begin{aligned}
 AE & = -0.5CE + DE = \max(-CE, 0) + DE \\
 AW & = 0.5CW + DW = \max(CW, 0) + DW \\
 AN & = -0.5CN + DN = \max(-CN, 0) + DN \\
 AS & = 0.5CS + DS = \max(CS, 0) + DS \\
 AT & = -0.5CT + DT = \max(-CT, 0) + DT \\
 AB & = 0.5CB + DB = \max(CB, 0) + DB
 \end{aligned}$$

$$AP = AE + AW + AN + AS + AT + AB$$

$$SP = -(CE - CW + CN - CS + CT - CB)$$

Divergence Freeにより0となるが  
安定性のため残すときもある

### 3. CFDの基礎-連性Algorithm

#### SIMPLE法

NS式の離散式を書き直すと、

$$a_e U_e = \sum a_{nb} U_{nb} + b + A_e (P_P - P_E)$$

ここで、推定圧力 $P^*$ を与えると、  
速度の近似値 $U^*, V^*, W^*$ を求めることができる。

$$a_e U_e^* = \sum a_{nb} U_{nb}^* + b + A_e (P_P^* - P_E^*)$$

$P^*$ に対する補正量 $P'$ 、  
それに対応する速度補正を $U', V', W'$ とする

$$\begin{cases} P = P^* + P' \\ U = U^* + U', V = V^* + V', W = W^* + W' \end{cases}$$

$$a_e U_e' = \sum a_{nb} U_{nb}' + A_e (P_P' - P_E')$$

$$\sum a_{nb} U_{nb}' \approx 0 \text{ とすると、 } U_e' = d_e (P_P' - P_E')$$

$$\text{ただし、 } d_e = A_e / a_e$$

$$U_e = U_e^* + d_e (P_P' - P_E')$$

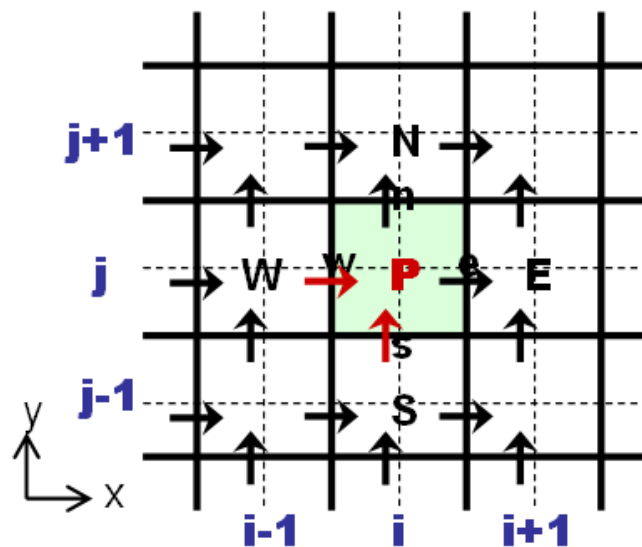
$$\text{ところで、連続の式は } [U_e - U_w] \Delta y \Delta z + [V_n - V_s] \Delta x \Delta z + [W_t - W_b] \Delta x \Delta y = 0$$

と離散化されるが、

ここに上式を代入すると

#### 圧力と速度の連成: SIMPLE

$$a_P P_P' = a_E P_E' + a_W P_W' + a_N P_N' + a_S P_S' + a_T P_T' + a_B P_B' + b$$



### 3. CFDの基礎-連性Algorithm

#### SIMPLE法

<三重対角行列解法 TDMA>

$$\text{三重対角行列 } A_i \cdot X_i = B_i \cdot X_{i+1} + C_i \cdot X_{i-1} + D_i$$

$$X_{i-1} = a_{i-1} \cdot X_i + b_{i-1} \text{ とおくと}$$

$$X_i = a_i \cdot X_{i+1} + b_i \text{ から}$$

$$a_i = \frac{B_i}{A_i - C_i \cdot a_{i-1}}$$

$$b_i = \frac{D_i + C_i \cdot b_{i-1}}{A_i - C_i \cdot a_{i-1}}$$

➡ 前進代入により、 $a_i, b_i$ を求める  
( $i=2,3,\dots,N$ )

$$\left[ a_1 = \frac{B_1}{A_1} \quad b_1 = \frac{D_1}{A_1} \right]$$

$$\begin{cases} X_{N-1} = a_{N-1} \cdot X_N + b_{N-1} \\ A_N \cdot X_N = C_N \cdot X_{N-1} + D_N \end{cases} \text{ より}$$

$$X_N = \frac{D_N + C_N \cdot b_{N-1}}{A_N - C_N \cdot a_{N-1}} = b_N$$

➡ 後退代入により、 $X_i$ を求める  
( $i=N-1, N-2, \dots, 2, 1$ )

結局、連続偏微分方程式を  
連立代数式に変え、解くことに  
この行列を効率よく解くのが重要

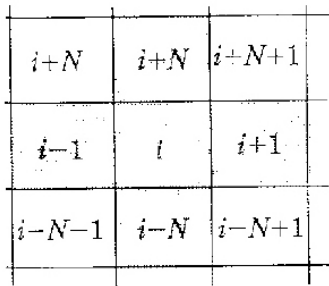
$$A_i = AP - SP$$

$$B_i = AE$$

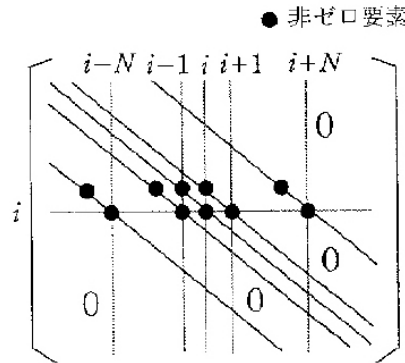
$$C_i = AW$$

$$D_i = SU + AN \cdot U_{i,j+1,k} + AS \cdot U_{i,j-1,k} + AT \cdot U_{i,j,k+1} + AB \cdot U_{i,j,k-1}$$

### 3. CFDの基礎-連性Algorithm



(a) 2次元矩形格子



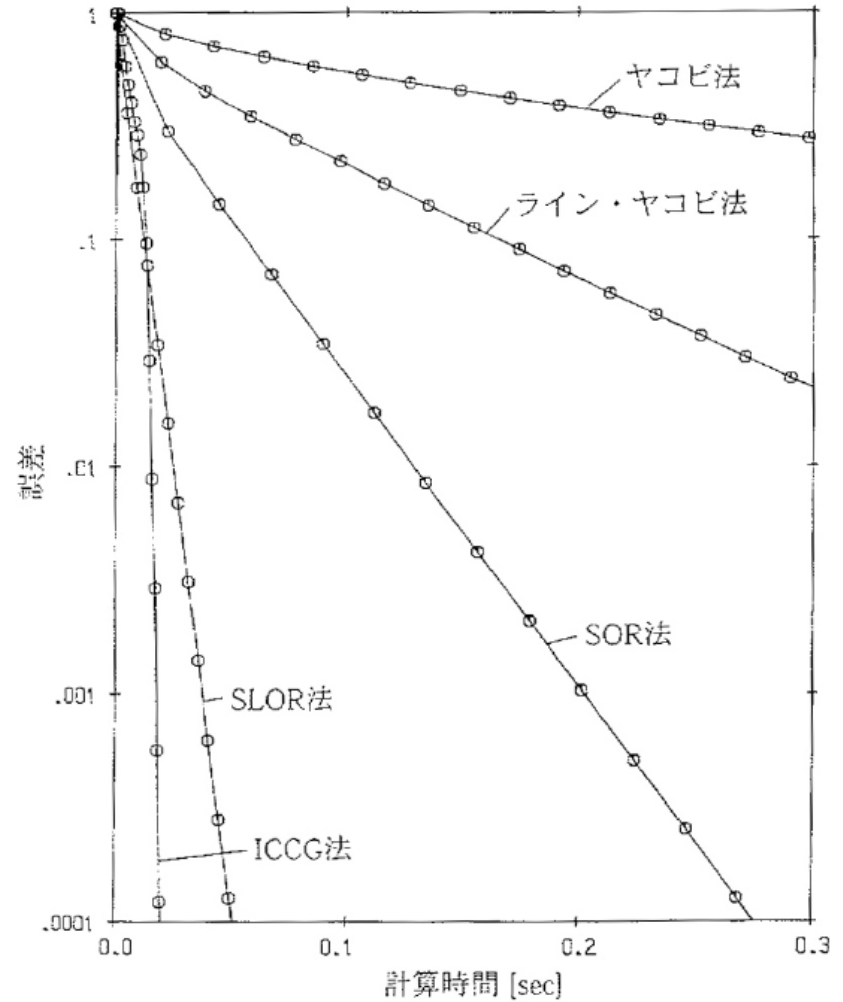
(b) 行列の構造

図 4.1 2次元矩形格子と行列方程式の関係

緩和法  $P = P^* + urfp \cdot P'$

$(\frac{1}{urfu})AP \cdot U =$

$\sum A_{nb}U_{nb} + SU + \frac{1-urfu}{urfu} (AP \cdot U)^{old}$

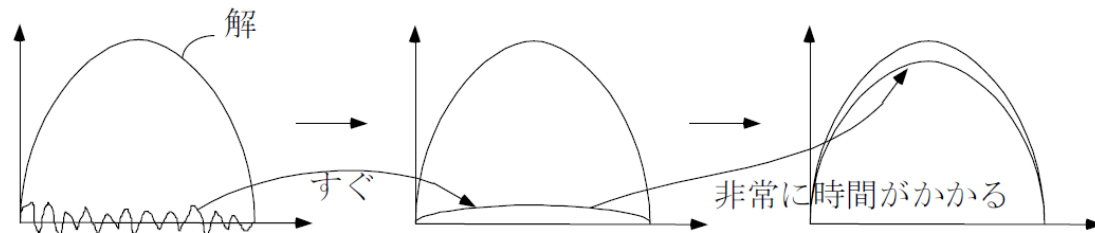


(b) 相対残差の減少

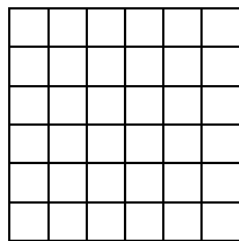
図 4.5 反復解法の比較

### 3. CFDの基礎-連性Algorithm

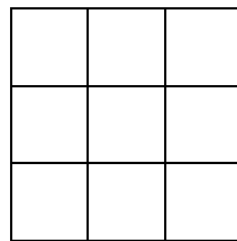
- Multigrid Method : to accelerate the convergence of a linear equation system  
 Iterative solvers can very effectively remove error components of which the wavelength is comparable with the mesh size.



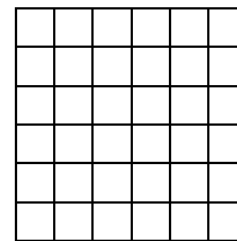
#### GAMG in OpenFOAM with pre-conditioner



$A_f \bar{x}_f = \bar{b}_f$   
 波長が短い成分を除く。  
 $A_f(\bar{x}_f + \delta \bar{x}_f) = \bar{b}_f$   
 $A_f \delta \bar{x}_f = \bar{b}_f - A_f \bar{x}_f = \bar{r}_f$   
 これを粗い格子で解く。



$A_c \delta \bar{x} = \bar{r}_c$  をとく  
 $\bar{r}_c = \underset{\downarrow}{R} \bar{r}_f$   
 細かい → 粗い格子  
 への補間演算子  
 細かい → 粗い格子への補間  
 を制限補間 (Restriction)  
 と呼ぶ



$\bar{x}_f = \bar{x}_f + \delta \bar{x}_f$  解を修正  
 $\delta \bar{x}_f = \underset{\downarrow}{L} \delta \bar{x}_c$   
 粗い → 細かい格子  
 への補間演算子  
 粗い → 細かい格子への補間  
 を延長補間 (Prolongation)  
 と呼ぶ。

### 3. CFDの基礎-連性Algorithm

#### □ OpenFOAMでの連性(Algorithm) <case>/system/fvSolution

- 線形ソルバ制御 :各離散化方程式に使用されるそれぞれの線形ソルバを指定
- 共役勾配ソルバの前提条件(Pre-Conditioner)
- 平滑化ソルバ(Smoother)
- 代数幾何マルチグリッドソルバ (GAMG)

```
SIMPLE
{
    nNonOrthogonalCorrectors 0;
}

relaxationFactors
{
    p 0.3;
    U 0.7;
    k 0.7;
    epsilon 0.7;
    R 0.7;
    nuTilda 0.7;
}
```

```
17 solvers
18 {
19     p PCG
20     {
21         preconditioner DIC;
22         tolerance 1e-06;
23         relTol 0;
24     };
25
26     U PBiCG
27     {
28         preconditioner DILU;
29         tolerance 1e-05;
30         relTol 0;
31     };
32 }
33
34 PISO
35 {
36     nCorrectors 2;
37     nNonOrthogonalCorrectors 0;
38     pRefCell 0;
39     pRefValue 0;
40 }
41
```



### 3. CFDの基礎-連性Algorithm

□ OpenFOAMでの連性(Algorithm) `<case>/system/fvSolution`

- 線形ソルバ制御 : 各離散化方程式に使用されるそれぞれの線形ソルバを指定
- 共役勾配ソルバの前提条件(Pre-Conditioner)
- 平滑化ソルバ(Smoother)
- 代数幾何マルチグリッドソルバ (GAMG)

ソルバ	キーワード
初期条件付き共役勾配	PCG/PBiCG †
スムーサーを使ったソルバ	smoothSolver
汎用幾何学的代数マルチグリッド	GAMG
†PCGは対称用、PBiCGは非対称用	

表4.12:線形ソルバ

スムーサー	キーワード
ガウスシーデル	GaussSeidel
対角不完全コレスキー分解 (対称)	DIC
ガウスシーデル 対角不完全コレスキー分解(対称)	DICGaussSeidel

表4.14:スムーサー・オプション

Preconditioner	キーワード
対角不完全コレスキー分解 (対称)	DIC
高速対角不完全コレスキー分解 (キャッシング付きDIC)	FDIC
対角不完全LU (非対称)	DILU
対角	diagonal
幾何学的代数マルチグリッド	GAMG
preconditioningなし	none

表4.13 Preconditionerオプション

### 3. CFDの基礎-連性Algorithm

#### □ OpenFOAMでの連性-(圧力式)

(1)  $\nabla \cdot (\rho \vec{U}) = 0$       **Matrixの対角項(計算)の係数**

(2)  $\frac{\partial U}{\partial t} + \nabla \cdot (\vec{v}\vec{v}) - \nabla \cdot (\nu \nabla \vec{v}) = -\nabla p$

$$a_p \vec{U}_p = H(\vec{U}) - \nabla p \iff \vec{U}_p = \frac{H(\vec{U})}{a_p} - \frac{\nabla p}{a_p}$$

$$H(\vec{U}) = \boxed{-\sum_n a_n \vec{U}_n} + \boxed{\frac{\vec{U}^o}{\Delta t}}$$

時間項+圧力項以外のSource項

周辺点のMatrixの係数 \* 速度

$$\vec{U}_f = \left( \frac{H(\vec{U})}{a_p} \right)_f - \frac{(\nabla p)_f}{(a_p)_f}$$

境界面での速度

$$\nabla \cdot \left( \frac{1}{a_p} \nabla p \right) = \nabla \cdot \left( \frac{H(\vec{U})}{a_p} \right) = \sum_f \vec{S} \left( \frac{H(\vec{U})}{a_p} \right)_f$$

by substituting into 連続式(1)

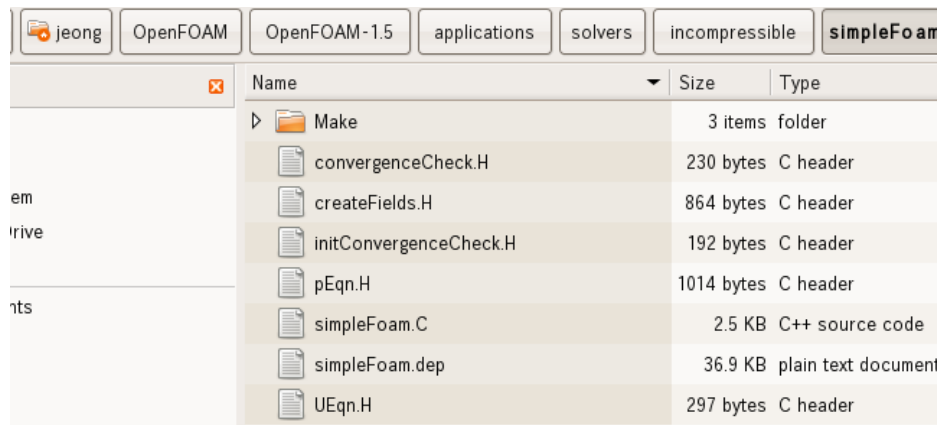
# 3. CFDの基礎-連性Algorithm

## simpleFOAM

### Matrixの対角項(計算)の係数

#### OpenFOAMでの連性-SIMPLE(定常計算)

1. 境界条件の設定
2. 離散化方程式を解いて、中間速度を求め
3. Cell境界でのFluxを計算.
4. 圧力式を解いてunder-relaxation
5. Cell境界での実量Fluxを修正
6. 新しい圧力場から速度を更新
7. 境界値を更新
8. 収束まで繰り返し反復



Name	Size	Type
Make	3 items	folder
convergenceCheck.H	230 bytes	C header
createFields.H	864 bytes	C header
initConvergenceCheck.H	192 bytes	C header
pEqn.H	1014 bytes	C header
simpleFoam.C	2.5 KB	C++ source code
simpleFoam.dep	36.9 KB	plain text document
UEqn.H	297 bytes	C header

```
-----*/
#include "fvCFD.H"
#include "incompressible/singlePhaseTransportModel/singlePhaseTransportModel.H"
#include "incompressible/RASModel/RASModel.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;

    for (runTime++; !runTime.end(); runTime++)
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;

        # include "readSIMPLEControls.H"
        # include "initConvergenceCheck.H"

        p.storePrevIter();
        // Pressure-velocity SIMPLE corrector
        {
            include "UEqn.H"
            include "pEqn.H"
        }

        turbulence->correct();

        runTime.write();

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

        # include "convergenceCheck.H"
    }

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

    return(0);
}
```

### 3. CFDの基礎-連性Algorithm

### simpleFOAM

#### □ OpenFOAMでの連性-SIMPLE(定常計算)

**.A() : Matrixの対角項(計算点)の係数**

1. 境界条件の設定
2. 離散化方程式を解いて、中間速度を求め
3. Cell境界でのFluxを計算.
4. 圧力式を解いてunder-relaxation
5. Cell境界でのFluxを修正
6. 新しい圧力場から速度を更新
7. 境界値を更新
8. 収束まで繰り返し反復

**mesh.Sf() : 面積 vector**

**& : 内積(a dot-product)**

**Cell中央点から界面へUを内挿、その後内積**

```
p.storePrevIter();
tmp<fvVectorMatrix> UEqn
(
    fvm::div(phi, U) - laplacian(nu, U)
);

UEqn.relax();

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

p.boundaryField().updateCoeffs();

volScalarField AU = UEqn().A();
U = UEqn().H()/AU;
UEqn.clear();

phi = fvc::interpolate(U) & mesh.Sf();
adjustPhi(phi, U, p);

fvScalarMatrix pEqn
(
    fvm::laplacian(1.0/AU, p) == fvc::div(phi)
);
pEqn.setReference(pRefCell, pRefValue);
pEqn.solve();

phi -= pEqn.flux();

# include "continuityErrs.H"

p.relax();
U -= fvc::grad(p)/AU;
U.correctBoundaryConditions();
```



## 3. CFDの基礎-連性Algorithm

---

### □ OpenFOAMでの連性-PISO(非定常計算)

1. 境界条件の設定
2. 離散化方程式を解いて、中間速度を求め
3. Cell境界でのFluxを計算.
4. 圧力式を解く(under-relaxation X)
5. Cell境界でのFluxを修正
6. 新しい圧力場から速度を更新
7. 境界値を更新
8. 3から指定された振り返し数まで反復
9. 時間を増加し、1から振り返し反復

### 3. CFDの基礎-連性Algorithm

#### □ OpenFOAMでの連性-PISO(非定常計算)

```
for(runTime++; !runTime.end(); runtime++)
{
    Info << "Time = " << runTime.curTime() nl << endl;
    fvMatrixVector 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++)
    {
        phi=Interpolate(Ueqn.H()/Ueqn.A()) & mesh.areas();
        fvMatrixScalar peqn
        (
            fvm::laplacian (1.0/Ueqn.A(), p) == fvc::div(phi)
        );
        peqn.setReference(0, 0.0);
        peqn.solve();
        phi -=peqn.flux();
        U = (Ueqn.H() - fvc::grad(p))/Ueqn.A();
        U.correctBoundaryConditions();
    }
}
```

時間ループ

速度場予測式の組立

速度場の予測値 Ueqn

nCorr 反復

圧力ポアソン式の組立

圧力ポアソン式反復

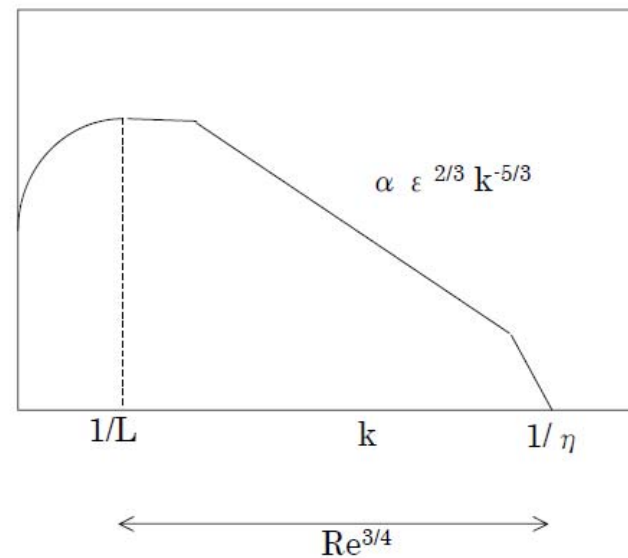
速度場  
修正 U

### 3. CFDの基礎-乱流

#### (1) Energy Cascade and Kolmogorov's Law

- Energy Cascade

Energy Contain Range S  
Inertial Subrange  
Viscous Dissipation Range



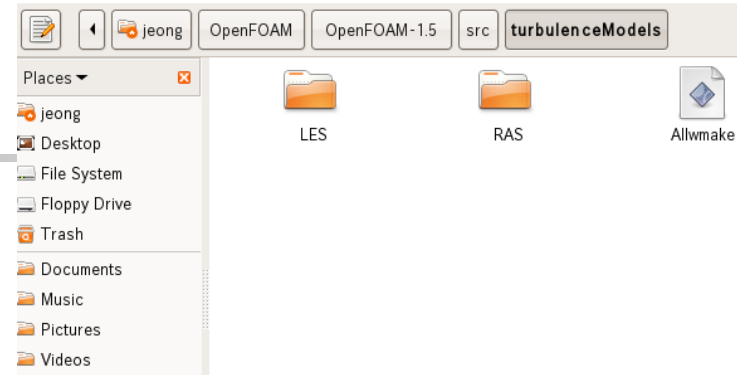
- Energy Contain Range

$$F\left(\frac{k}{k_0}\right) = \frac{\epsilon(k)}{Kl}$$

### 3. CFDの基礎-乱流

#### RANSとLES

#### 乱流の解析(RANSとLES)



RANSは時間平均と変動分、LESは格子で再現できる成分と小さい成分

- RANSはもともと定常流の解析に
- 普通LESは陽解法で計算時間はRANSより10倍

NS式に代入して整理

$$\frac{\mu}{\rho} \frac{\partial^2 u_i}{\partial x_j \partial x_j} \rightarrow \frac{1}{\rho} \frac{\partial}{\partial x_j} \left( \mu \frac{\partial \bar{u}_i}{\partial x_j} - \overline{\rho u'_i u'_j} \right) \frac{1}{\rho} \frac{\partial}{\partial x_j} \left( (\mu + \mu_t) \frac{\partial \bar{u}_i}{\partial x_j} \right)$$
$$-\overline{\rho u'_i u'_j} = \mu_t \left( \frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} \right)$$

結局、eddy viscosityを求めることになる

#### 乱流モデル

kとεの運送方程式

$$\mu_t = \frac{C_\mu \rho k^2}{\varepsilon}$$

- RANS: k-εモデルなど
- LES: Smagorinsky(SGS)、DSGSなど



## \* Codeの例

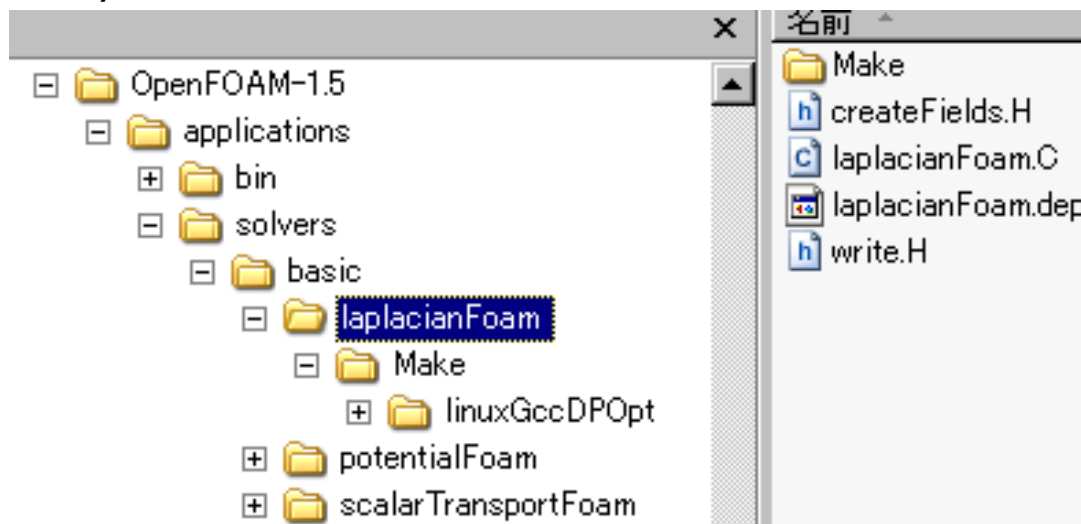
### □ OpenFOAMのSolver

#### 1. 基礎的なCFDコード (3種類)

##### - Laplacian Foam (例えば、熱の拡散方程式)

$$\rho c \frac{\partial T}{\partial t} = \lambda \left( \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} \right) \quad \rho c \frac{\partial T}{\partial t} = \lambda \frac{\partial^2 T}{\partial x_j^2} \quad \rho c \frac{\partial T}{\partial t} = \lambda \nabla^2 T$$

$\alpha = \frac{\lambda}{\rho c}$  : 熱拡散率     $\lambda$  : 熱伝度率     $C$  : 比熱



# \* Codeの例

```

#include "fvCFD.H"
// *****

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

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

    for (runTime++; !runTime.end(); runTime++)
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;

        # include "readSIMPLEControls.H"

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

```

Laplacian	Imp/Exp	$\nabla^2\phi$ $\nabla \cdot \Gamma \nabla \phi$	laplacian(phi) laplacian(Gamma, phi)
Time derivative	Imp/Exp	$\frac{\partial \phi}{\partial t}$ $\frac{\partial \rho \phi}{\partial t}$	ddt(phi) ddt(rho, phi)

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T$$

```

# include "write.H"

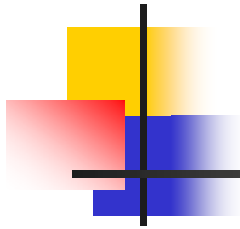
Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
    << " ClockTime = " << runTime.elapsedClockTime() << " s"
    << nl << endl;

}

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

return(0);

```



```

createFields.H
Info<< "Reading field T%n" << endl;

volScalarField T
(
    IObject
    (
        "T",
        runTime.timeName(),
        mesh,
        IObject::MUST_READ,
        IObject::AUTO_WRITE
    ),
    mesh
);

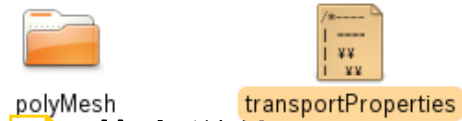
Info<< "Reading transportProperties%n" << endl;

IObject transportProperties
(
    IObject
    (
        "transportProperties",
        runTime.constant(),
        mesh,
        IObject::MUST_READ,
        IObject::NO_WRITE
    )
);

Info<< "Reading diffusivity DT%n" << endl;

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

```



## 基本単位

No.	プロパティ	SI単位系	USCS単位系
1	質量	キログラム(kg)	ポンド(lbm)
2	長さ	メートル(m)	フィート(ft)
3	時間	秒(s)	
4	温度	ケルビン(K)	ランキン温度(o)
5	物質量	キログラム-モル(kgmol)	ポンド-モル(lbmol)
6	電流	アンペア(A)	
7	光度	カンデラ(cd)	

表 4.2: SIとUSCSの基本単位

```

transportProperties
/*----- C++ -----*/
|=====|
|  ¥¥ /  Field      | OpenFOAM: The Ope
|  ¥¥ /  Operation | Version:  1.5
|  ¥¥ /  And       | Web:      http://
|  ¥¥ /  Manipulation |
|*-----|
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    object       transportProperties;
}
// *****

DT          DT [0 2 -1 0 0 0 0] 4e-05;

// *****

```



## 7. まとめ

---

- CFDに関する一般的な内容をOpenFOAMと連関して簡単に紹介

下記は「流れ 2007年8月号」から

- これまでのような、手続き型言語を使用したコーディングにおいては、実際の支配方程式群はあらかじめ机上において離散化されていなければならない、さらにコーディングはミスを誘発しやすい難しい作業であった。典型的な汎用 CFD コードは、100,000 行にもなるコードであるので、その実際のコーディングは大変な困難を伴う。
- 数学的記述に直接対応する OpenFOAM のプログラミングスタイルは、これまでよりもずっと直感的にコーディングを行うことができる。OpenFOAM は数学的記述を採用した高水準言語を数値シミュレーションの分野に導入することが可能であること。。。。

## 2. OpenFOAMの構造

The image shows the directory structure of OpenFOAM. The top part displays the 'linuxGccDPOpt' directory containing various library files (lib\*) and their sizes. The middle part shows the 'kEpsilon' directory with sub-directories for turbulence models and RAS models, and source files for kEpsilon.C, kEpsilon.dep, and kEpsilon.H. The bottom part shows the 'incompressible' directory with sub-directories for boussinesq and Allwmake, and a terminal window showing the command to build the libraries.

Name	Size
libcombustionThermophysicalModels.so	1.7 MB
libcompressibleLESModels.so	1.2 MB
libcompressibleRASModels.so	1.7 MB
libconversion.so	464.2 KB
libdecompositionMethods.so	129.6 KB
libdieselSpray.so	900.1 KB
libdynamicFvMesh.so	399.7 KB
libdynamicMesh.so	3.2 MB
libedgeMesh.so	56.4 KB
libengine.so	310.4 KB
liberrorEstimation.so	153.4 KB
libEulerianInterfacialModels.so	243.0 KB

```
#!/bin/sh
set -x

wmake libso incompressible
wmake libso compressible
wmake libso boussinesq
```