# ЛЕКЦИЯ 17.
Расширенные возможности OpenFOAM.

В данной лекции рассматриваются классы дискретизации границ, представления полей, представления матриц.

## ДИСКРЕТИЗАЦИЯ ГРАНИЦ В OpenFOAM
Доступ к границам: const fvBoundaryMesh& fvMesh::boundary() const
fvBoundaryMesh наследует от fvPatchList (List<fvPatch>) - список внешних границ (патчей) расчетной области, на которых задаются граничные условия для каждого из полей задачи
fvPatch содержит информацию о геометрии границы — центры граней Cf() и прилегающих к ним ячеек Cn(), их площади magSf(), Sf() и нормали nf(), вектора расстояний delta() от центров граней до центров прилегающих ячеек, ссылку на объект с топологией границы patch()
polyPatch — наследует от patchIdentifier, primitivePatch, содержит в себе топологию границы и методы для работы с ней (центры граней, связи между гранями, связи между гранями и прилегающими ячейками, адресацию между номерами граней на данной внешней границе (локальными номерами) и глобальными (во всей расчетной области)
patchIdentifier — идентификатор внешней границы (её номер в общем списке, имя и тип в виде строковой константы)
primitivePatch — устанавливает адресацию между узлами и натянутыми на них гранями данной внешней границы.

## ПРЕДСТАВЛЕНИЕ ПОЛЕЙ В OpenFOAM
Поле как список значений в расчетных точках (например, в центрах к.о.)
Field<T> наследует от refCount и List<T>
Поле физической величины определенной размерности DimensionedField <class T, class GeoMesh> наследует от Field<T> и regIOobject
Геометрическое поле GeometricField<class T, template<class> PatchField, class GeoMesh> наследует от DimensionedField<T, GeoMesh> определено на всей расчетной области и вычисляется в определенных точках (к.о., гранях, узлах)
Поле, определенное в центрах ячеек (volFields.H, volFieldsFwd.H):
typedef GeometricField<scalar, fvPatchField,volMesh> volScalarField;
typedef GeometricField<vector, fvPatchField,volMesh> volVectorField;
typedef GeometricField<tensor, fvPatchField,volMesh> volTensorField;
Поле, определенное в центрах граней (surfaceFields.H, surfaceFieldsFwd.H):
typedef GeometricField<scalar, fvPatchField,surfaceMesh> surfaceScalarField;
typedef GeometricField<vector, fvPatchField,surfaceMesh> surfaceVectorField
Также как и расчетная область (сетка), поля физических величин представлены значениями внутри расчетной области и значениями на границе.
Последние группируются по именованному множеству граней и выступают как численная реализация граничного условия определенного типа.
Поле, определенное на некоторой границе (fvPatch) определяется либо в шаблоне fvPatchField<T> (при дискретизации в центрах к.о., соответствует vol***Field) либо fvsPatchField<T> (при дискретизации в центрах граней, соответствует surface***Field).
Параметр шаблона T - это тип поля (вектор, скаляр, тензор и прочее)
Класс fv(s)PatchField обязательно содержит:
а) ссылку на сетку данной границы patch(),
б) регистр объектов данного уровня db(),
в) внутренее поле internalField(), г) производную по нормали snGrad(),

д) диагональные коэффициенты матрицы valueInternalCoeffs() и gradientInternalCoeffs(), е) коэффициенты правой части уравнений valueBoundaryCoeffs() и gradientBoundaryCoeffs(),
ж) операторы присваивания и простейшие арифметические операторы,
з) маркер, определяющий является ли это ГУ 1-го рода — fixesValue().

**ПРЕДСТАВЛЕНИЕ РАЗРЕЖЕННЫХ МАТРИЦ. ИСХОДНАЯ МАТРИЦА.**

$$A_{ii}\psi_i + \sum_j A_{ij}\psi_j = S_i$$

$A_{ii}$ - Диагональные элементы матрицы A

$\psi_j$ - Искомое поле, определенное в расчетных точках

$\sum_j A_{ij}\psi_j$ - Сумма недиагональных элементов в строке i

$S_i$ - Правая часть уравнения (источник)

Матрица содержится в классе lduMatrix:

Коэффициенты матрицы lowerPtr_, diagPtr_, upperPtr_
Адресация осуществляется через объект типа lduAddressing - ::lduAddr()
Доступ к матрице — lower(), diag(), upper()
Доступ к операторам H() и A()
Операторы сложения и вычитания -=, +=
Ссылку на расчетную область mesh()
Подклассы: а) solver; б) solverPerfomance; в) smoother; г) preconditioner
Имеем матрицу NxN, где N — число расчетных точек (контрольных объёмов)

**ПРЕДСТАВЛЕНИЕ РАЗРЕЖЕННЫХ МАТРИЦ В ПАМЯТИ. АДРЕСАЦИЯ**
COO (Coordinate storage) — все непустые значения хранятся в одном массиве (values). Помимо него используются ещё два вектора — номера непустых колонок (cols) и номера непустых рядов (rows)
OpenFOAM COO — отдельно хранятся диагональные элементы (D), верхние (U) — номера ячеек с индексом U > D, и нижние (L) с индексом L < D
В представлении матрицы OpenFOAM каждой ячейке соответствуют: а) диагональный элемент, б) список элементов верхнего треугольника (по строкам), в) список элементов нижнего треугольника (по рядам)
Структура представления расчетной сетки OpenFOAM соответствует представлению данных матрицы. Расчетная сетка представляет собой:
1) Список координат вершин, образующих контрольные объёмы;
2) Список граней, образующих контрольные объёмы, каждый элемент такого списка содержит массив номеров узлов из списка вершин пункта 1, сортированный таким образом, что нормаль грани «смотрит» во-вне контрольного объёма;
3) Список граней, принадлежащих контрольным объёмам (соединяющим один объём с другим, чей номер выше данного); нормаль такой грани направлена наружу упомянутого объёма (Owners). Число элементов — общее число граней.
4) Список граней, принадлежащих соседним контрольным объёмам (с номером, меньшим чем данный) — Neighbours. Число элементов количество внутренних граней < число Owners

Матрица содержится в классе lduMatrix:

Коэффициенты матрицы lowerPtr_, diagPtr_, upperPtr_
Адресация осуществляется через объект типа lduAddressing - ::lduAddr()
Доступ к матрице — lower(), diag(), upper()
Доступ к операторам H() и A()
Операторы сложения и вычитания -=, +=
Ссылку на расчетную область mesh()
Подклассы:
а) solver;
б) solverPerfomance;
в) smoother;
г) preconditioner.

**Решение СЛАУ**
Класс lduMatrix::solver:
Имя искомой величины
Ссылка на параметры поиска
Критерий выхода из итераций
Процедура решения — BICCG, ICCG, PCG, PbiCG
Для решения используется процедура Matrix::solve()
lduMatrix::solverPerfomance — информация о процессе решения системы линейных алгебраических уравнений (private):
Имя решателя solverName_
Имя поля fieldName_
Начальная невязка initialResidual_
Конечная невязка finalResidual_
Число итераций noIterations_
Сходимость решения converged_
Сингулярность решения singular_
Сглаживание — процедура подавления высокочастотных составляющих решения (релаксация) при использовании многосеточных методов. Целью сглаживателя (smoother) ставится сглаживание решения для улучшения приближения на грубой сетке
Базовый абстрактный класс сглаживателя определен в классе lduMatrix::smoother, его реализации в классах:
DICSmoother — сглаживание методом неполного разложения Холецкого
DICGaussSeidelSmoother — сглаживание методом неполного разложения Холецкого и Гаусса-Зейделя
DILUSmoother — сглаживание методом LU-разложения
DILUGaussSeidelSmoother — сглаживание методом LU-разложения и Гаусса-Зейделя
GaussSeidelSmoother — сглаживание методов Гаусса-Зейделя

**Предобуславливатели**
Базовый класс для всех предобуславливателей — lduMatrix::preconditioner, их реализации содержатся в классах:
noPreconditioner — без предобуславливания;
diagonalPreconditioner — диагональный предобуславливатель;
GAMGPreconditioner — многосеточный предобуславливатель;
DILUPreconditioner — предобусловливание методом LU-разложения;
DICPreconditioner — предобусловливание методом неполного разложения Холецкого;
FDICPreconditioner — предобусловливание методом быстрого неполного разложения Холецкого.

**Решение СЛАУ**
fvMatrix — реализация lduMatrix, предназначенная для решения СЛАУ, полученных МКО, наследует от refCount и lduMatrix, осуществляет выбор решателя (fvMatrix::fvSolver). Уравнения могут решаться только для одной переменной (скаляр), решение уравнений для тензоров и векторов производится последовательно:
solve() - решение системы СЛАУ, полученных после дискретизации уравнений на сетке
psi() - искомое поле (скаляр)
source() - источник
другие функции связанные lduMatrix
Класс fvMatrix обладает свойствами самотождественности, аддитивности и коммутативности через переопределенные операторы:

$$LA = L_1 A + L_2 A = L_2 A + L_1 A = (L_1 + L_2) A$$

**Общий порядок интегрирования уравнений**
Решение задачи проводится на три этапа. Результат выполнения каждого из трех — fvMatrix. В случае неявной дискретизации (fvm::) - матрица коэффициентов в левой части, в случае явной (fvc::) — вектор правой части (поле, определенное в расчетных точках)
1) Неявная дискретизация слагаемых уравнений (fvm::)
2) Явная дискретизация слагаемых уравнений (fvc::)
3) Дискретизация по времени (fvm::ddt,fvm::d2dt2 и fvc::dtdt, fvc::d2tdt2)
4) Обновление граничных условий (возможно на шагах 1 и 2)
5) Решение системы уравнений

Дискретизация дивергенции

$$\int_V \nabla\cdot(\rho \boldsymbol{U}\,\psi)dV = \int_S d\boldsymbol{S}\cdot(\rho \boldsymbol{U}\,\psi) = \sum_f S_f\cdot(\rho \boldsymbol{U})_f\,\psi_f$$

$$\int_V \nabla\cdot\psi\,dV = \int_S d\boldsymbol{S}\cdot(\psi) = \sum_f S_f\cdot(\psi)_f$$

Дискретизация конвективного слагаемого осуществляется с помощью группы функций div(...) пространства имён fvc:: и fvm::,
файлы fvmDiv.H, fvcDiv.H
template<class Type> tmp<fvMatrix<Type> >
fvm::div (const surfaceScalarField& flux, GeometricField<Type, fvPatchField, volMesh>& vf, const word& name);
template<class Type> tmp <GeometricField<
typename innerProduct<vector, Type>::type, fvPatchField, volMesh> >
div (const GeometricField<Type, fvPatchField, volMesh>& vf, const word& name)
Классы-реализации операторов:
fv::convectionScheme (fvm::, fvc::), fv::divScheme (fvc)

**Дискретизация градиента**

$$\int_V \nabla(\psi)dV = \int_S (d\boldsymbol{S})\,\psi = \sum_f \boldsymbol{S}_f\,\psi_f$$

Дискретизация градиента (метод наименьших квадратов, теорема Гаусса и т. д.) осуществляется с помощью группы функций grad(...) пространства имён fvc::, для дискретизации градиента определенным методом используются fvc::gGrad(...), fvc::lsGrad(...) и т. д.
Файл fvcGrad.H
template<class Type> tmp <GeometricField<
typename outerProduct<vector,Type>::type, fvPatchField, volMesh> >
fvc::grad (
const GeometricField<Type, fvPatchField, volMesh>& vf,
const word& name)
Операторы реализуются в классе gradScheme

**Дискретизация диффузионного слагаемого**
Дискретизация диффузионного слагаемого (лапласиан в OpenFOAM) осуществляется с помощью группы функций laplacian(...) пространства имён fvc:: и fvm::,
файлы fvmLaplacian.H и fvcLaplacian.H
template<class Type, class GType> tmp<fvMatrix<Type> >
fvm::laplacian (const GeometricField<GType, fvsPatchField, surfaceMesh>& gamma, GeometricField<Type, fvPatchField, volMesh>& vf, const word& name)
template<class Type, class Gtype> tmp<GeometricField<Type, fvPatchField, volMesh> >
fvc::laplacian ( const GeometricField<GType, fvsPatchField, surfaceMesh>& gamma, const GeometricField<Type, fvPatchField, volMesh>& vf, const word& name);
Операторы реализуются в классе laplacianScheme

**Производная по времени**

$$\frac{\partial}{\partial t}\int_V \rho\psi\, dV$$

Дискретизация первой производной осуществляется с помощью группы функций ddt(...) пространства имён fvc:: и fvm::, используются схемы первого порядка (Эйлера) и второго порядка (обратного дифференцирования), файлы fvmDdt.H, fvcDdt.H
template<class Type> tmp<fvMatrix<Type> >
fvm::ddt ( GeometricField<Type, fvPatchField, volMesh>& vf)
template<class Type> tmp<GeometricField<Type, fvPatchField, volMesh> >
fvc::ddt (const dimensioned<Type> dt, const fvMesh& mesh)
Операторы реализуются в классе ddtScheme

**Параллельное управление**
Класс Pstream (почти все члены статические)
bool parRun()
nProcs()
master()
masterNo()
myProcNo()
Классы IPstream (ввод данных, оператор >>), OPstream (вывод данных, оператор <<)
**Динамические библиотеки. Модели турбулентности.**

$$R^{Eff} = (\mu+\mu^t)\left[\nabla \boldsymbol{U} + (\nabla \boldsymbol{U})^T\right]$$

(mu + mut) * (fvc::grad(U) + fvc::grad(U).T())
1) Все RAS-модели опираются на одно и тоже предположение Буссинеска
2) Расчет эффективного тензора Рейнольдса зависит только от параметров конкретной модели
3) Эти особенности выносятся в отдельную библиотеку, например, libincompressibleRASModels
4) Множественность вариантов одной и той же функции осуществляется за счет полиморфизма — механизма виртуальных функций
5) Для включения в динамическую библиотеку информации о возможных вариантах виртуальной функции используются макросы:
defineTypeNameAndDebug(kEpsilon,0)
addToRunTimeSelectionTable(RASModel,kEpsilon,dictionary)
defineTemplateTypeNameAndDebugWithName(<name>, <debug level>)
<debug level> - уровень отладки. 0 — без отладки, 1 — минимальный вывод
2 — вывод всех сообщений

**Обзор библиотек в папке src**
ODE — методы интегрирования обыкновенных дифференциальных уравнений
OSspecific — системные вызовы и функции
OpenFOAM — ядро программы, её основные классы
Pstream — работа с потоками исполнения
autoMesh — алгоритмы автоматического создания сеток
conversion — алгоритмы конвертации форматов сеток
decompositionMethods — методы декомпозиции задачи по пространству
dummyThirdParty — интерфейсы к сторонним методам декомпозиции по пространству
dynamicFvMesh — конечно-объёмная сетка с поддержкой её деформации
dynamicMesh — методы деформации и изменения сетки
edgeMesh — сетка, состоящая из отрезков (рёбер)
engine — библиотека для решения задач, связанных с горением в двигателе
errorEstimation — оценка погрешностей, ошибок аппроксимации, невязок
finiteVolume — реализация метода конечных объёмов в OpenFOAM
fvAgglomerationMethods — методы укрупнения сеток при использовании многосеточных алгоритмов решения систем линейных алгебраических уравнений
fvMotionSolver — методы деформации сетки без изменения топологии по заданным перемещениям внешних границ расчетной области
genericPatchFields — описание нетипизированного (абстрактного) граничного условия
lagrangian — решение задач в переменных Лагранжа
meshTools — утилиты для работы с сеткой
postProcessing — анализ расчетных результатов
randomProcesses — анализ случайных процессов (например, БПФ)
sampling — анализ выборки из общих расчетных данных (полей)
surfMesh — поверхностная сетка
thermophysicalModels — термодинамические и теплофизические модели
topoChangerFvMesh — изменение топологии конечно-объёмной сетки
transportModels — модели транспорта (Ньютоновская и не-Ньютоновские)
triSurface — триангулированная поверхность
turbulenceModels — сжимаемые и несжимаемые RAS, LES и DES модели турбулентности

**Приложения OpenFOAM. Основные этапы.**
1) Инициализация структуры каталогов рабочей задачи

2) Инициализация потоков выполнения (MPI)
3) Инициализация физического времени
4) Инициализация расчетной сетки
5) Инициализация искомых полей
6)Инициализация цикла интегрирования (время)
7) Интегрирование уравнений и запись результатов
8) Завершение работы.