Coupled solvers and more
Lecture within CFD with open source 2013 (TME050)

Klas Jareteg
Chalmers University of Technology

2013-09-17
DISCLAIMER: This offering is not approved or endorsed by OpenCFD Limited, the producer of the OpenFOAM software and owner of the OPEN-FOAM® and OpenCFD® trade marks. Following the trademark policy.

DISCLAIMER: The ideas and code in this presentation and all appended files are distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
Plan and outline

Pressure-velocity
  Theory
OpenFOAM code basics
  Mesh
  Matrices
Coupled solvers
  Basic idea
  Coupled format
  Example solver
Pressure-velocity coupling
  Coupled model
  Implementing pressure-velocity coupling
  Tutorial case
Miscellaneous
  Git
  Better software development
Python scripting
Learning objectives

At the end of this lesson you should (hopefully):

- better understand the basics of the pressure-velocity implementation in OpenFOAM
- be acquainted with the ideas of the block coupled format in OpenFOAM-1.6-ext
- have basic practical experience with git
- increased understanding of templating and object orientation in C++
Pressure-velocity

Pressure-velocity
Theory
OpenFOAM code basics
Mesh
Matrices
Coupled solvers
Basic idea
Coupled format
Example solver
Pressure-velocity coupling
Coupled model
Implementing pressure-velocity coupling
Tutorial case
Miscellaneous
Git
Better software development
Python scripting
Incompressible flow

Acknowledgement for description: Professor Hrvoje Jasak

- For low Mach numbers the density and pressure decouple.
- General Navier-Stokes equations simplify to:

\[ \nabla \cdot (\mathbf{U}) = 0 \]  \hspace{1cm} (1)

\[ \frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{U} \mathbf{U}) - \nabla (\nu \nabla \mathbf{U}) = -\frac{1}{\rho} \nabla p \]  \hspace{1cm} (2)

- Non-linearity in the equation \( \nabla \cdot (\mathbf{UU}) \) resolved by iteration
- Continuity equation requiring the flow to be divergence free
- No explicit pressure dependence for the divergence free criterium. Pressure equation must be derived.
Incompressible flow - equation coupling

- Pressure equation retrieved from the continuity equation.
- Start by a semi-discretized form of the momentum equation:

\[ a_P \mathbf{U}_P = \mathbf{H(U)} - \nabla P \]  

(3)

where:

\[ \mathbf{H(U)} = \sum_N a_N \mathbf{U}_N \]  

(4)

and rearranged to:

\[ \mathbf{U}_P = (a_P\mathbf{U})^{-1} \mathbf{H(U)} - (a_P\mathbf{U})^{-1} \nabla P \]  

(5)
Incompressible flow - equation coupling II

- Eq. (5) is then substituted into the continuity equation:

\[ \nabla \cdot ((a_P^U)^{-1} \nabla P) = \nabla \cdot ((a_P^U)^{-1} \mathbf{H}(\mathbf{U})) \]  

(6)

- Gives two equations: momentum and pressure equation

- Pressure equation will assure a divergence free flux, and consequently the face fluxes \((F = S_f \cdot \mathbf{U})\) must be reconstructed from the solution of the pressure equation:

\[ F = -(a_P^U)^{-1} S_f \cdot \nabla P + (a_P^U)^{-1} S_f \cdot \mathbf{H}(\mathbf{U}) \]  

(7)
SIMPLE algorithm is primarily used for steady state problems:

1. Guess the pressure field
2. Solve momentum equation using the guessed pressure field (eq. 5)
3. Compute the pressure based on the predicted velocity field (eq. 6)
4. Compute conservative face flux (eq. 7)
5. Iterate

In reality, underrelaxation must be used to converge the problem

Study the source code of simpleFoam:
- Try to recognize the above equations in the code
Rhie-Chow correction

- Rhie and Chow introduced a correction in order to be able to use collocated grids
- This is used also in OpenFOAM, but not in an explicit manner
- The Rhie-Chow like correction will occur as a difference to how the gradient and Laplacian terms in eq. (6) are discretized.

OpenFOAM tips

- Learn to find your way around the code:
  - grep keyword `find -iname "*.C`
  - Doxygen (http://www.openfoam.org/docs/cpp/)
  - CoCoons-project (http://www.cocoons-project.org/)

- Get acquainted with the general code structure:
  - Study the structure of the src-directory
  - Try to understand where some general

- When you are writing your own solvers study the available utilities:
  - find how to read variables from dicts scalars, booleans and lists
  - find out how to add an argument to the argument list
OpenFOAM code basics

Pressure-velocity coupling
  Theory
OpenFOAM code basics
  Mesh
  Matrices
Coupled solvers
  Basic idea
  Coupled format
  Example solver
Pressure-velocity coupling
  Coupled model
  Implementing pressure-velocity coupling
  Tutorial case
Miscellaneous
  Git
  Better software development
Python scripting
Meshes in OpenFOAM

Mesh:
- Based on an unstructured mesh format
- Collocated mesh (Rhie-Chow equivalent already mentioned)
- polyMesh and fvMesh: Face based computational cell:

fvMesh is the finite volume specialization
- \( V() \) Volumes of the cells. Numbered according to cell numbering.
- \( S_f() \) Surface normals with magnitude equal to the area. Numbered according to face numbers.
Matrix format in OpenFOAM I

Matrix:

- Sparse matrix system:
  - No zeros stored
  - Only neighbouring cells will give a contribution

- Basic format of the lduMatrix:
  - diagonal coefficients
  - upper coefficients
  - lower coefficients (not necessary for symmetric matrices)

Study the code for lduMatrix:

- find the diagonal, upper and lower fields
Matrix format in OpenFOAM II

**lduMatrix**

Basic square sparse matrix. Stored in three arrays: the diagonal, the upper and the lower part:

```c++
// Coefficients (not including interfaces)
scalarField *lowerPtr_, *diagPtr_, *upperPtr_;
```

**Listing 1: lduMatrix.H**

- Diagonal elements: numbered as cell numbers
- Off-diagonal elements: are numbered according to faces.

```c++
const surfaceVectorField& Sf = p.mesh().Sf();
const unallocLabelList& owner = mesh.owner();
const unallocLabelList& neighbour = mesh.neighbour();
```

**Listing 2: Surface normal, owner and neighbour for each face**
Matrix format in OpenFOAM III

Sparsity of matrix:

\[ \mathbf{A} = A_{i,j} \] (8)

- \( i, j \): contribution from cell \( j \) on cell \( i \)
- \( j, i \): contribution from cell \( i \) on cell \( j \)
- \( i > j \): upper elements
- \( i < j \): lower elements
- \( i = j \): diagonal elements
Matrix format in OpenFOAM IV

fvMatrix

- Specialization for finite volume
- Adds source and reference to field
- Helper functions:

```cpp
volScalarField AU = UEqn().A();
```

Listing 3: Part of pEqn.H in simpleFoam
Lazy evaluation

- Lazy evaluation is used to avoid calculation and transfer of unnecessary data
- Example 1duMatrix:
  - Used for returning the upper part of the matrix (upper())
  - If upper part does not exist it will be created
  - If it already exists it is simply returned
- To achieve lazy evaluation you will see pointers used in OpenFOAM
Coupled solvers

Pressure-velocity
  Theory
OpenFOAM code basics
  Mesh
  Matrices
Coupled solvers
  Basic idea
  Coupled format
  Example solver
Pressure-velocity coupling
  Coupled model
  Implementing pressure-velocity coupling
  Tutorial case
Miscellaneous
  Git
  Better software development
Python scripting
What is a coupled solver?

**Coupling on many levels:**
- Model level (example: couple a turbulence model to your steady state solver)
- Equation level (example: couple the pressure equation to the velocity equation)
- Matrix level (example: GGI and regionCoupling)

**Differ between:**
- **explicit coupling:** solve one matrix for each equation, use fix values from all the other equations
- **implicit coupling:** directly couple linear dependencies in equations by including multiple equations in the same matrix system
Explicit coupling

Examples:
- Velocity components in simpleFoam and pisoFoam
- Turbulence and momentum equations in simpleFoam and pisoFoam
- Regions in chtMultiRegionFoam

Advantages:
- Requires less memory than implicit coupling
- Sometimes easier to implement (each equation solved separately)

Study simpleFoam to see how the explicit coupling is done:
- In which terms and expressions are the p and U equations coupled?
- How is the turbulence model connected to the velocity?
Implicit coupling

Examples:
- Regions in `regionCoupling` (OpenFOAM-1.6-ext)

Advantages:
- Can increase convergence rates as fewer iterations are anticipated
- Sometimes necessary in order for the system to converge
- Minimizing underrelaxation

Disadvantages:
- Increased memory cost, each matrix coefficient a tensor (rank two) instead of a scalar
- Convergence properties changed

Optional: Study the `regionCouple` boundary condition to see how the implicit coupling between different regions is achieved.
Implementing a block matrix format

**Possible choices of format:**

- Extend the matrix:
  - Sparsity pattern of matrix changed
  - General coupled matrix system:

  \[
  \begin{align*}
  A(y)x &= a \\
  B(x)y &= b
  \end{align*}
  \]  

- Solved together (still segregated):

  \[
  \begin{bmatrix}
  A(y) & 0 \\
  0 & B(x)
  \end{bmatrix}
  \begin{bmatrix}
  x \\
  y
  \end{bmatrix} = \begin{bmatrix}
  a \\
  b
  \end{bmatrix}
  \]  

- Coupled solution:

  \[
  \begin{bmatrix}
  A' & A_y \\
  B_x & B'
  \end{bmatrix}
  \begin{bmatrix}
  x \\
  y
  \end{bmatrix} = \begin{bmatrix}
  a \\
  b
  \end{bmatrix}
  \]  

- Important: non-linearities still left, must be treated explicitly
Implementing a block matrix format II

- Extend the size of each coefficient in the matrix:
  - Sparsity pattern preserved
  - Alternative formulation of eq. (12):

\[
Cz = c
\] (13)

\[
C = C_{i,j} = \begin{bmatrix} c_{a,a} & c_{a,b} \\ c_{b,a} & c_{b,b} \end{bmatrix}_{i,j}
\] (14)

\[
c = c_i = \begin{bmatrix} s_a & s_b \end{bmatrix}^T_i
\] (15)

\[
z = z_i = \begin{bmatrix} x & y \end{bmatrix}^T_i
\] (16)

- Element in vectors and matrices: vectors and tensors
Implementing a block matrix format III

Implementation in OpenFOAM-1.6-ext:

- Sparsity pattern preserved, each coefficient a tensor

Study the code for BlockLduMatrix:

- find the matrix format and how it relates to the mesh
C++ templates

- Templated functions and classes can operate with generic types.
- Templates are generated at compile time (compare to virtual functions)
- Allows reusing algorithms and classes which are common to many specific types

- Find a class which is templated!
- Find a function which is templated!
C++ templates II

Example: List

- A list could be used different type of contents → generic class needed
- ListI.H: included already in the header file
- Compilation done for each specific type (remember: generated during compile-time)

Example: BlockLduMatrix

- Allow matrix coefficients to be of generic size
- Each <Type> must have operators needed defined
- Compilation done for each specific type (remember generated during compile-time)
C++ templates III

Tips on templates

- Read the basics (and more):
  - Effective C++: 50 Specific Ways to Improve Your Programs and Designs
  - C++ Templates: The Complete Guide

- Look at existing code to see how the templating is implemented, used and compiled (’’code explains code’’


Example solver in OpenFOAM-1.6-ext:
blockCoupledScalarTransportFoam

Theory behind solver: coupled two phase heat transfer\(^1\):

\[ \nabla \cdot (U T) - \nabla (D \nabla \cdot T) = \alpha (T_s - T) \]
\[ - \nabla (D T_s \nabla \cdot D_s) = \alpha (T - T_s) \]

Velocity field \( U \) prescribed, \( T \) and \( T_s \) are fluid and solid temperatures
Studying blockCoupledScalarTransportFoam II

Study blockCoupledScalarTransportFoam to find:

- vector and matrix formats used,
- how the scalar equations are coupled in the block-coupled matrix,
- how the boundary conditions are transferred and
- how the system is solved

Run the blockCoupledSwirlTest

---

Pressure-velocity coupling
Implicit model

- Navier-Stokes, incompressible, steady-state:
  \[ \nabla \cdot (\mathbf{U}) = 0 \]  
  \[ \nabla \cdot (\mathbf{U} \mathbf{U}) - \nabla (\nu \nabla \mathbf{U}) = -\frac{1}{\rho} \nabla p \]  

- Semi-discretized form:
  \[ \sum_{\text{faces}} \mathbf{U}_f \cdot \mathbf{S}_f = 0 \]  
  \[ \sum_{\text{faces}} [\mathbf{U} \mathbf{U} - \nu \nabla \mathbf{U}]_f \cdot \mathbf{S}_f = -\sum_{\text{faces}} P_f \mathbf{S}_f \]  

- Modified pressure:
  \[ \frac{p}{\rho} = P \]  

- Rhie-Chow in continuity equation:
  \[ \sum_{\text{faces}} [\mathbf{U}_f - \mathbf{D}_f (\nabla P_f - \nabla P_f)] \cdot \mathbf{S}_f = 0 \]
fvm vs. fvc

Meanings:
- **fvm**: finite volume method, results in an implicit discretization (a system of equations)
- **fvc**: finite volume calculus, results in an explicit discretization (source terms)

Types:
- **fvm**: returns `fvMatrix<Type>`
- **fvc**: returns `geometricField<Type>`

\[
[A][x] = [b]
\] (25)

Study the Programmers guide to find the available:
- **fvm discretizations**
- **fvc discretizations**
Pressure-velocity discretization

Eqs. (24) and (22) in OpenFOAM format:

```
fvm::div(U)
- fvm::laplacian(D,p)
==
- fvc::div(D*fvc::grad(p))
```

```
fvm::div(phi,U)
+ turbulence->divDevReff(U)
==
- fvm::grad(p)
```

Problem:
- Implicit div and grad not generally desired → implementations not existing
Implementing the pressure-velocity coupling I

Solution vector:

\[ x^P = x^l = \begin{bmatrix} u^P \\ v^P \\ w^P \\ P^P \end{bmatrix} \]  (26)

```
// Block vector field for the pressure and velocity field to be solved for
volVector4Field pU
(
    IOobject
    (    
        "pU",  
        runTime.timeName(),  
        mesh,  
        I0object::NO_READ,  
        I0object::NO_WRITE
    ),
    mesh,  
    dimensionedVector4(word(), dimless, vector4::zero)
);  
```
Implementing the pressure-velocity coupling II

// Insert the pressure and velocity internal fields in to the volVector2Field
{
  vector4Field blockX = pU.internalField();

  // Separately add the three velocity components
  for (int i=0; i<3; i++)
  {
    tmp<scalarField> tf = U.internalField().component(i);
    scalarField& f = tf();
    blockMatrixTools::blockInsert(i,f,blockX);
  }

  // Pressure is the 2nd component
  scalarField& f = p.internalField();
  blockMatrixTools::blockInsert(3,f,blockX);
Implementing the pressure-velocity coupling III

Equation system to be formed:

$$\mathbf{A}^P \mathbf{x}^P + \sum_{\mathbf{F} \in \{\mathbf{N}\}} \mathbf{A}^F \mathbf{x}^F = \mathbf{b}^P$$  \hspace{1cm} (27)

$$A^X = \left[ a^X_{k,l} \right]_i \quad k, l \in \{u, v, w, p\}, \quad X \in \{P, F\}$$  \hspace{1cm} (28)

Construct block matrix:

```cpp
// Matrix block
BlockLduMatrix< vector4 > B(mesh);
```

Retrieve fields:

```cpp
// Diagonal is set separately
Field< tensor4 >& d = B.diag().asSquare();

// Off-diagonal also as square
Field< tensor4 >& u = B.upper().asSquare();
Field< tensor4 >& l = B.lower().asSquare();
```

Source:

```cpp
// Source term for the block matrix
Field< vector4 > s(mesh.nCells(), vector4::zero);
```
Discretizing the momentum equation

LHS: Turbulence is introduced by calling the \texttt{divDivReff(U)}

\begin{verbatim}
182    tmp<fvVectorMatrix> UEqnLHS
183    (   
184       fvm::div(phi,U)
185       + turbulence\rightarrow divDevReff(U)
186    );
\end{verbatim}

Retrieve matrix coefficients:

\begin{verbatim}
202    tmp<scalarField> tdiag = UEqnLHS().D();
203    scalarField& diag = tdiag();
204    scalarField& upper = UEqnLHS().upper();
205    scalarField& lower = UEqnLHS().lower();
\end{verbatim}

Add boundary contribution:

\begin{verbatim}
211    // Add source boundary contribution
212    vectorField& source = UEqnLHS().source();
213    UEqnLHS().addBoundarySource(source, false);
\end{verbatim}
Discretizing the momentum equation II

Considering RHS as separate problem:

\[ \sum_{\text{faces}} P_f S_f = 0 \] \hspace{1cm} (29)

Interpolation weights:

```cpp
tmp<surfaceInterpolationScheme<scalar> >
tinterpScheme_ ( surfaceInterpolationScheme<scalar>::New ( p.mesh(), p.mesh().interpolationScheme("grad(p)") ) );
```

```cpp
tmp<surfaceScalarField> tweights = tinterpScheme_.weights(p);
const surfaceScalarField& weights = tweights();
```

\[ w_N = 1 - w_P \] \hspace{1cm} (30)
Discretizing the momentum equation III

Equivalent to matrix fields:

```cpp
// Pressure gradient contributions — corresponds to an implicit
// gradient operator

tmp<vectorField> tpUv = tmp<vectorField>
(    
    new vectorField(upper.size(),pTraits<vector>::zero)
)
;
vectorField& pUv = tpUv();
tmp<vectorField> tpLv = tmp<vectorField>
(    
    new vectorField(lower.size(),pTraits<vector>::zero)
)
;
vectorField& pLv = tpLv();
tmp<vectorField> tpSv = tmp<vectorField>
(    
    new vectorField(source.size(),pTraits<vector>::zero)
)
;
vectorField& pSv = tpSv();
tmp<vectorField> tpDv = tmp<vectorField>
(    
    new vectorField(diag.size(),pTraits<vector>::zero)
)
;
vectorField& pDv = tpDv();
```
Discretizing the momentum equation IV

**Calcualte elements:**

```cpp
for(int i=0;i<owner.size();i++)
{
    int o = owner[i];
    int n = neighbour[i];
    scalar w = weights.internalField()[i];
    vector s = Sf[i];

    pDv[o]+=s*w;
    pDv[n]-=s*(1-w);
    pLv[i]=-s*w;
    pUv[i]=s*(1-w);
}
```

**Boundary contribution:**

```cpp
p.boundaryField().updateCoeffs();
forAll(p.boundaryField(),patchI)
{
    // Present fvPatchField
    fvPatchField<scalar> & fv = p.boundaryField()[patchI];

    // Retrieve the weights for the boundary
    const fvsPatchScalarField& pw = weights.boundaryField()[patchI];

    // Contributions from the boundary coefficients
    tmp<Field<scalar>> tic = fv.valueInternalCoeffs(pw);
    Field<scalar>& ic = tic();
    tmp<Field<scalar>> tbc = fv.valueBoundaryCoeffs(pw);
```
Discretizing the momentum equation $V$

Field<scalar>& bc = tbc();

// Get the fvPatch only
const fvPatch& patch = fv.patch();

// Surface normals for this patch
tmp<Field<vector>> tsn = patch.Sf();
Field<vector> sn = tsn();

// Manually add the contributions from the boundary
// This what happens with addBoundaryDiag, addBoundarySource
forAll(fv, facei)
{
    label c = patch.faceCells()[facei];

    pDv[c]+=ic[facei]*sn[facei];
    pSv[c]=bc[facei]*sn[facei];
}


Discretizing the momentum equation VI

\[ a_{u,u}, a_{v,v}, a_{w,w}, a_{p,u}, a_{p,v}, a_{p,w} : \]

```csharp
forAll(d, i)
{
    d[i](0,0) = diag[i];
    d[i](1,1) = diag[i];
    d[i](2,2) = diag[i];
    d[i](0,3) = pDv[i].x();
    d[i](1,3) = pDv[i].y();
    d[i](2,3) = pDv[i].z();
}
forAll(l, i)
{
    l[i](0,0) = lower[i];
    l[i](1,1) = lower[i];
    l[i](2,2) = lower[i];
    l[i](0,3) = pLv[i].x();
    l[i](1,3) = pLv[i].y();
    l[i](2,3) = pLv[i].z();
}
forAll(u, i)
{
    u[i](0,0) = upper[i];
    u[i](1,1) = upper[i];
    u[i](2,2) = upper[i];
    u[i](0,3) = pUv[i].x();
    u[i](1,3) = pUv[i].y();
    u[i](2,3) = pUv[i].z();
}
forAll(s, i)
```
Discretizing the momentum equation VII

```
348  {
349    s[i](0) = source[i].x() + pSv[i].x();
350    s[i](1) = source[i].y() + pSv[i].y();
351    s[i](2) = source[i].z() + pSv[i].z();
352  }
```
Use of `tmp`

- `tmp` is used to minimize the computational effort in the code.
- In general, C++ will create objects in local scope, return a copy and destroy the remaining object.
- This is undesired for large objects which gives lots of data transfer.
- To avoid the local object to be out of scope the `tmp` container is used.

Source and more info:  
http://openfoamwiki.net/index.php/OpenFOAM_guide/tmp
Discretizing the continuity equation

One implicit and one explicit contribution:

```cpp
439  tmp<volScalarField> tA = UEqnLHS().A();
    volScalarField& A = tA();

442  tmp<volVectorField> texp = fvc::grad(p);
    volVectorField& exp = texp();
    tmp<volVectorField> texp2 = exp/A;
    volVectorField exp2 = texp2();

447  tmp<fvScalarMatrix> MEqnLHSp
(    -fvm::laplacian(1/A,p)
        ==
    -fvc::div(exp2)
);

454  // Add the boundary contributions
    scalarField& pMdiag = MEqnlHSp().diag();
    scalarField& pMupper = MEqnlHSp().upper();
    scalarField& pMlower = MEqnlHSp().lower();

459  // Add diagonal boundary contribution
    MEqnlHSp().addBoundaryDiag(pMdiag,0);

462  // Add source boundary contribution
    scalarField& pMsourse = MEqnlHSp().source();
    MEqnlHSp().addBoundarySource(pMsourse, false);
```
Discretizing the continuity equation II

Need implicit divergence scheme:

```cpp
// Again an implicit version not existing, now the div operator
tmp<surfaceInterpolationScheme<scalar>> UtinterpScheme_(
    surfaceInterpolationScheme<scalar>::New(
        U.mesh(),
        U.mesh().interpolationScheme("div(U)(implicit)")
    )
);

// 1) Setup diagonal, source, upper and lower
tmp<vectorField> tMUpper = tmp<vectorField>(
    new vectorField(upper.size(),pTraits<vector>::zero));
vectorField& MUpper = tMUpper();

tmp<vectorField> tMLower = tmp<vectorField>(
    new vectorField(lower.size(),pTraits<vector>::zero));
vectorField& MLower = tMLower();

tmp<vectorField> tMDiag = tmp<vectorField>(
    new vectorField(diag.size(),pTraits<vector>::zero));
vectorField& MDiag = tMDiag();

tmp<vectorField> tMSource = tmp<vectorField>(
    new vectorField
    (source.component(0)().size(),pTraits<vector>::zero)
    );
```
Discretizing the continuity equation III

```cpp
vectorField& MSource = tMSource();

// 2) Use interpolation weights to assemble the contributions

for (int i=0; i<owner.size(); i++)
{
    int o = owner[i];
    int n = neighbour[i];
    scalar w = Mweights.internalField()[i];
    vector s = Sf[i];

    MDia[0] += s*w;
    MDia[1] -= s*(1-w);
    MLow[i] -= s*w;
    MUp[i] = s*(1-w);
}

// Get boundary condition contributions for the pressure grad(P)
U.boundaryField().updateCoeffs();
forAll(U.boundaryField(), patchI)
{
    // Present fvPatchField
    fvPatchField<vector>& fv = U.boundaryField()[patchI];

    // Retrieve the weights for the boundary
    const fvsPatchScalarField& Mw =
        Mweights.boundaryField()[patchI];

    // Contributions from the boundary coefficients
```
Discretizing the continuity equation IV

Discretizing the continuity equation IV

412 \[
\text{tmp<const Field<vector>&} \\ \text{ tic } = \text{ fv.valueInternalCoeffs(Mw)};
\]
413 \[
\text{Field<vector>& ic } = \text{ tic();}
\]
414 \[
\text{tmp<const Field<vector>&} \\ \text{ tbc } = \text{ fv.valueBoundaryCoeffs(Mw)};
\]
415 \[
\text{Field<vector>& bc } = \text{ tbc();}
\]
416
417 \[
\text{// Get the fvPatch only}
\]
418 \[
\text{const fvPatch& patch } = \text{ fv.patch();}
\]
419
420 \[
\text{// Surface normals for this patch}
\]
421 \[
\text{tmp<const Field<vector>&} \\ \text{ tsn } = \text{ patch.Sf();}
\]
422 \[
\text{Field<vector> sn } = \text{ tsn();}
\]
423
424 \[
\text{// Manually add the contributions from the boundary}
\]
425 \[
\text{// This what happens with addBoundaryDiag, addBoundarySource}
\]
426 \[
\text{forAll(fv,facei)}
\]
427 \[
\{\]
428 \[
\text{label c } = \text{ patch.faceCells()[facei];}
\]
429 \[
\text{MDiag[c]+=cmptMultiply(ic[facei],sn[facei]);}
\]
430 \[
\text{MSource[c]-=}\text{cmptMultiply(bc[facei],sn[facei]);}
\]
431 \[
\}
\]
Discretizing the continuity equation V

\[ a_{u,p}, a_{v,p}, a_{w,p}, a_{p,p}: \]

```
forall(d,i)
{
  d[i](3,0) = MDiag[i].x();
  d[i](3,1) = MDiag[i].y();
  d[i](3,2) = MDiag[i].z();
  d[i](3,3) = pMdiag[i];
}
forall(l,i)
{
  l[i](3,0) = MLower[i].x();
  l[i](3,1) = MLower[i].y();
  l[i](3,2) = MLower[i].z();
  l[i](3,3) = pMlower[i];
}
forall(u,i)
{
  u[i](3,0) = MUpper[i].x();
  u[i](3,1) = MUpper[i].y();
  u[i](3,2) = MUpper[i].z();
  u[i](3,3) = pMupper[i];
}
forall(s,i)
{
  s[i](3) = MSource[i].x();
  +MSource[i].y();
  +MSource[i].z();
  +pMsoure[i];
}
```
OpenFOAM programming tips

- To get more information from a floating point exception:
  - export FOAM_ABORT=1
- If you are compiling different versions of OpenFOAM back and forth the compiling is accelerated by using ccache (http://ccache.samba.org/)
Miscallaneous

Pressure-velocity
  Theory
OpenFOAM code basics
  Mesh
  Matrices
Coupled solvers
  Basic idea
  Coupled format
  Example solver
Pressure-velocity coupling
  Coupled model
  Implementing pressure-velocity coupling
  Tutorial case
Miscallaneous
  Git
    Better software development
Python scripting
Git

- Version control system\(^2\) - meant to manage changes and different versions of codes
- Distributed - each directory is a fully functioning repository without connection to any servers
- Multiple protocols - code can be pushed and pulled over HTTP, FTP, ssh ...

\(^2\)Many more version control systems exists, e.g. Subversion and Mercurial
Git - Hands on I

**Basics:**

- Initialize a repository in the current folder:
  ```
  git init
  ```

- Check the current status of the repository:
  ```
  git status
  ```

- Add a file to the revision control:
  ```
  git add filename
  ```

- Now again check the status:
  ```
  git status
  ```

- In order to commit the changes:
  ```
  git commit -m "Message that will be stored along with the commit"
  ```

- List the current commits using log:
  ```
  git log
  ```
Branches:

- When developing multiple things or when multiple persons are working on the same code it can be convenient to use branches.
- To create a branch:
  ```
  git branch name_new_branch
  ```
- List the available branches:
  ```
  git branch
  ```
- Switch between branches by:
  ```
  git checkout name_new_branch
  ```
- Branches can be merged so that developments of different branches are brought together.
Ignore file:

- Avoid including compiled files and binary files in the revision tree.
- Add a `.gitignore` file. The files and endings listed in the file will be ignored. Example:

```
# Skip all the files ending with .o (object files)
*.o

# Skip all dependency files
*.dep
```

- When looking at the status of the repository the above files will be ignored.
Git - Information and software

Some documentation:
- Code School - Try Git: http://try.github.io/levels/1/challenges/1
- ... google!

Examples of software:
- Meld - merging tool, can be used to merge different branches and commits (http://meldmerge.org/)
- Giggle - example of a GUI for git (https://wiki.gnome.org/Apps/giggle)
Better software development

- Write small, separable segments of code
- Test each unit in the code, test the code often
- Setup a test case structure to continuously test the code
- Comment your code
- Use a version control system
- Use tools that you are used to, alternatively get used to them!
Python scripting

Pressure-velocity
  Theory
OpenFOAM code basics
  Mesh
  Matrices
Coupled solvers
  Basic idea
  Coupled format
  Example solver
Pressure-velocity coupling
  Coupled model
  Implementing pressure-velocity coupling
  Tutorial case
Miscellaneous
  Git
  Better software development

Python scripting
Why? What? How?

**What is a script language?**
- Interpreted language, not usually needed to compile
- Aimed for rapid execution and development
- Examples: Python, Perl, Tcl ...

**Why using a script language?**
- Automatization of sequences of commands
- Easy to perform data and file preprocessing
- Substitute for more expansive software
- Rapid development

**How to run a script language?**
- Interactive mode; line-by-line
- Script mode; run a set of commands written in a file
Python basics

- Interpreted language, no compilation by the user
- Run in interactive mode or using scripts
- Dynamically typed language: type of a variable set during runtime

```python
1 foo = "1"
2 bar = 5
```

- Strongly typed language: change of type requires explicit conversion

```python
1 >>> foo=1
2 >>> bar="a"
3 >>> foobar=foo+bar
4 Traceback (most recent call last):
5   File "<stdin>", line 1, in <module>
6   TypeError: unsupported operand type(s) for +: 'int' and 'str'
```
• Commented lines start with "#"

• Loops and conditional statements controlled by indentation

```python
if 1==1:
    print "Yes, 1=1"
print "Will always be written"
```

• Three important data types:
  • Lists:

```python
>>> foo = [1,"a"]
>>> bar = [1, 2, 3, 4]
>>> print foo[0]
1
>>> print bar[:]
[1, 2, 3, 4]
>>> print bar[1:2]
[2]
>>> print bar[-1]
4
>>> bar.append(4)
>>> print bar
[1, 2, 3, 4, 4]
```
Python syntax II

- Tuples:

```python
>>> foo = (1, 2, 3)
>>> print "Test %d use %d of tuple %d" % foo
Test 1 use 2 of tuple 3
```

- Dictionaries:

```python
>>> test = {}
>>> test['value'] = 4
>>> test['name'] = "test"
>>> print test
{'name': 'test', 'value': 4}
```
Auxiliary code can be included from modules. Examples:

- **os**: Operating system interface. Example:

```python
import os
# Run a command
os.system("run command")
```

- **shutil**: High-level file operations

```python
import shutil
# Copy some files
shutil.copytree('template','runfolder')
```
Case study: Running a set of simulations

- Multiple OpenFOAM runs with different parameters
- Example: edits in fvSolution:
  - Make a copy of your dictionary.
  - Insert keywords for the entries to be changed
  - Let the script change the keywords and run the application

```python
#!/usr/bin/python
import os
import shutil

presweeps = [2, 4]
cycles = ['W', 'V']

for p in presweeps:
    for c in cycles:
        os.system('rm -rf runfolder')
        shutil.copytree('template', 'runfolder')

        os.chdir('runfolder')
        os.system(f'"sed -i s/PRESWEEPS/%d/ system/fvSolution"%p')
        os.system(f'"sed -i s/CYCLETYPE/%s/ system/fvSolution"%c')
        os.system(f"mpirun --np 8 steadyNavalFoam --parallel > log.steadyNavalFoam")

        os.chdir('..')
```
Case study: Extract convergence results

- Run cases as in previous example and additionally extract some running time

```python
#!/usr/bin/python
import os
import shutil

presweeps = [2,4]
cycles = ['W','V']

for p in presweeps:
    for c in cycles:
        os.system('rm -rf runfolder')
        shutil.copytree('template','runfolder')

os.chdir('runfolder')
os.system("sed -i 's/PRESWEEPS/%d/' system/fvSolution"
)
os.system("sed -i 's/CYCLETYPE/%s/' system/fvSolution"
)
os.system("mpirun -np 8 steadyNavalFoam -parallel > log.steadyNavalFoam")

f = open('log.steadyNavalFoam','r')
for line in f:
    linsplit = line.rsplit()
    if len(linsplit)>7:
        if ls[0]=="ExecutionTime":
            execTime = float(ls[2])
            clocktime = float(ls[6])

f.close()
print "Cycle=%s, presweeps=%d, execution time=%.2f, clocktime=%.2f"%(c,p,execTime,clocktime)
os.chdir('..')
```
Case study: Setting up large cases

```python
#!/usr/bin/python
# Klas Jareteg
# 2013-08-30
#
# Desc:
# Setting up the a case with a box

import os, sys, shutil
opj = os.path.join
from optparse import OptionParser
import subprocess

MESH = '/home/klas/OpenFOAM/klas-16-ext-git/run/krjPbe/2D/meshes/box/coarse/moderator.blockMesh'
FIELDS = '/home/klas/OpenFOAM/klas-16-ext-git/run/krjPbe/2D/meshes/box/coarse/0'

parser = OptionParser()
p = parser.add_option("c", "--clean", dest="clean",
                       action="store_true", default=False)
p = parser.add_option("s", "--setup", dest="setup",
                       action="store_true", default=False)
(options, args) = parser.parse_args()

if options.clean:
    os.system('rm -fr 0')
    os.system('rm -fr [0-9]*')
```

---

Case study: Setting up large cases

```python
#!/usr/bin/python
# Klas Jareteg
# 2013-08-30
#
# Desc:
# Setting up the a case with a box

import os, sys, shutil
opj = os.path.join
from optparse import OptionParser
import subprocess

MESH = '/home/klas/OpenFOAM/klas-16-ext-git/run/krjPbe/2D/meshes/box/coarse/moderator.blockMesh'
FIELDS = '/home/klas/OpenFOAM/klas-16-ext-git/run/krjPbe/2D/meshes/box/coarse/0'

parser = OptionParser()
p = parser.add_option("c", "--clean", dest="clean",
                       action="store_true", default=False)
p = parser.add_option("s", "--setup", dest="setup",
                       action="store_true", default=False)
(options, args) = parser.parse_args()

if options.clean:
    os.system('rm -fr 0')
    os.system('rm -fr [0-9]*')
```
Case study: Setting up large cases II

```python
if options.setup:
    shutil.copy(MESH, 'constant/polyMesh/blockMeshDict')

    p = subprocess.Popen(['blockMesh'],
                         stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    out, error = p.communicate()

    if error:
        print(bcolors.FAIL + "ERROR: blockMesh failing" + bcolors.ENDC)
        print(bcolors.ENDC + "ERROR MESSAGE: %s" % error + bcolors.ENDC)

    try:
        shutil.rmtree('0')
    except OSError:
        pass

    shutil.copytree(FIELDS, '0')
```
Plotting with Python - matplotlib

```python
#!/usr/bin/python

import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0,1)
y = np.linspace(0,2)
y = y**2

plt.figure()
plt.plot(x,y)
plt.title('Test of matplotlib')
plt.xlabel('x')
plt.ylabel('y')
plt.savefig('Test.pdf', format='pdf')
```

**Figure:** Example plot from matplotlib
More on plotting

- **matplotlib** ([http://matplotlib.org/](http://matplotlib.org/)):
  - Plotting package with MATLAB equivalent syntax
  - Primarily 2D plots
  - Plots 3D
  - Works with VTK, possible complement to ParaView
Read more

Python introduction material:
  - Python tutorial: http://docs.python.org/2/tutorial/

Python and high performance computing:
  - http://www.c3se.chalmers.se/index.php/Python_and_High_Performance_Computing
PyFoam

From documentation:

“This library was developed to control OpenFOAM-simulations with a decent (sorry Perl) scripting language to do parameter-variations and results analysis. It is an ongoing effort. I add features on an As-Needed basis but am open to suggestions.”

Abilities:

- Parameter variation
- Manipulation directories
- Setting fields and boundary conditions
- Generate results and plots
- ....

http://openfoamwiki.net/index.php/Contrib_PyFoam
More modules

- **logging**: Flexible logging which could be used also for modules.
- **optparse**: Parser for command line options. Example from http://docs.python.org/2/library/optparse.html:

```python
from optparse import OptionParser

parser = OptionParser()
parser.add_option("-f", "--file", dest="filename",
                   help="write report to FILE", metavar="FILE")
parser.add_option("-q", "--quiet",
                   action="store_false", dest="verbose", default=True,
                   help="don’t print status messages to stdout")

(options, args) = parser.parse_args()
```

  - Array and matrix operations
  - Linear algebra
Case study: Meshing with Python

- Library of objects and functions to read a config file and produce a set of meshes and fields
Case study: Meshing with Python II

Needed for simulation:
- All meshes (16×4+1+1=66)
- All fields (≈400)
- All coupled patches

Reasons to automatize:
- Changes in mesh configurations (mesh independence tests etc.)
- Change in geometrical configurations
- Change in field initial and boundary conditions
- ....
Case study: Meshing with Python III

Meshes and fields produced from a configuration file read by Python application:

```
[general]
dimensions: 3
convert: 0.01
time: 0

[GeneralAssembly]
name: Generalized assembly mesh
symmetry: 4
nx: 7
lattice: guid pin0 guid pin0
        pin0 pin0 pin0 pin0
        guid pin0 guid pin0
        pin0 pin0 pin0 pin0
dphi: 8
pitch: 1.25
H: 1.0
dz: 1.0
gz: 1.0
ref: 0.0
ref_dz: 1.0
ref_gz: 1.0

moderatorfields: T p K k epsilon U G
modinnfields: T p K k epsilon U G
neutronicsmultiples: Phi Psi
fuefields: T rho K h p
clafields: T rho K h p
gapfields: T p_gap K k_gap epsilon_gap U_gap G
```
Case study: Meshing with Python IV

```python
[pin0]
type: FuelPin
fue_ro: 0.41
fue_ri: 0.12
fue_dr: 4
....
```
Case study: Meshing with Python V

blockMeshDict

```
....
convertToMeters 0.010000;

vertices
(
  (0.000000 0.000000 0.000000)
  (0.070711 0.070711 0.000000)
  (0.055557 0.083147 0.000000)
  ....
  (4.375000 4.167612 0.000000)
  (4.375000 4.167612 1.000000)
  (1000.000000 1000.000000 1000.000000)
);

blocks
(
  hex ( 0 1 2 5 6 7 7 ) ( 1 1 1 ) simpleGrading (1.000000 1.000000 1.000000 )
  hex ( 0 2 10 10 5 13 13 ) ( 1 1 1 ) simpleGrading (1.000000 1.000000 1.000000 )
  hex ( 0 10 16 16 5 19 19 ) ( 1 1 1 ) simpleGrading (1.000000 1.000000 1.000000 )
  hex ( 0 16 22 22 5 19 25 25 ) ( 1 1 1 ) simpleGrading (1.000000 1.000000 1.000000 )
  hex ( 0 166 172 172 5 169 175 175 ) ( 1 1 1 ) simpleGrading (1.000000 1.000000 1.000000 )
  hex ( 0 172 178 178 5 175 181 181 ) ( 1 1 1 ) simpleGrading (1.000000 1.000000 1.000000 )
  hex ( 0 178 184 184 5 181 187 187 ) ( 1 1 1 ) simpleGrading (1.000000 1.000000 1.000000 )
  ....
```
Case study: Meshing with Python

Summary:
- Using blockMesh for structured meshes with many regions
- Need for a script in order to be able to reproduce fast and easy
- Object oriented library written in Python