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 OPENFOAM® 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

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

Incompressible flow

Acknowledgement for description: Professor Hrvoje Jasak

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

$$\nabla \cdot (\mathbf{U}) = 0 \tag{1}$$

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{U}\mathbf{U}) - \nabla(\nu \nabla \mathbf{U}) = -\frac{1}{\rho}\nabla p \tag{2}$$

- Non-linearity in the equation $(\nabla \cdot (\mathbf{U}\mathbf{U}))$ 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 I

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

$$a_P \mathbf{U}_P = \mathbf{H}(\mathbf{U}) - \nabla P \tag{3}$$

where:

$$\mathbf{H}(\mathbf{U}) = \sum_N a_N^{\mathbf{U}} \mathbf{U}_N \tag{4}$$

and rearranged to:

$$\mathbf{U}_P = (a_P^{\mathbf{U}})^{-1} \mathbf{H}(\mathbf{U}) - (a_P^{\mathbf{U}})^{-1} \nabla P \tag{5}$$

Incompressible flow - equation coupling II

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

$$\nabla \cdot ((a_P^{\mathbf{U}})^{-1} \nabla P) = \nabla \cdot ((a_P^{\mathbf{U}})^{-1} \mathbf{H}(\mathbf{U})) \tag{6}$$

- Gives two equations: momentum and pressure equation
- Pressure equation will assure a divergence free flux, and consequently the face fluxes ($F = \mathbf{S}_\mathrm{f} \cdot \mathbf{U}$) must be reconstructed from the solution of the pressure equation:

$$F = -(a_P^{\mathbf{U}})^{-1} \mathbf{S}_\mathrm{f} \cdot \nabla P + (a_P^{\mathbf{U}})^{-1} \mathbf{S}_\mathrm{f} \cdot \mathbf{H}(\mathbf{U}) \tag{7}$$

SIMPLE

Acknowledgement for description: Professor Hrvoje Jasak

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.

Source and longer description: Peng-Kärrholm: `http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2007/rhiechow.pdf`

OpenFOAM tips

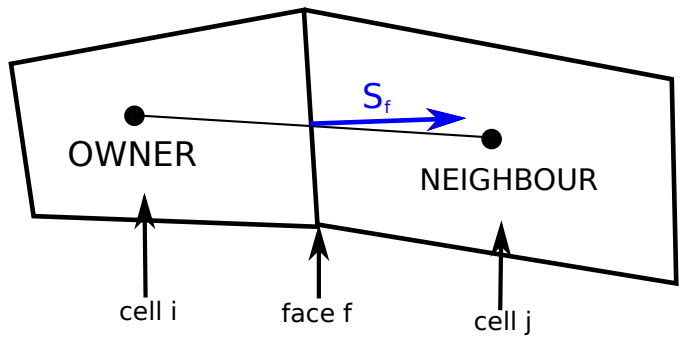
- Learn to find your way around the code:
  - `grep keyword `find -iname "*.C"``
  - Doxygen (`http://www.openfoam.org/docs/cpp/`)
  - CoCoons-project (`http://www.cocoons-project.org/`)
- Get acquinted with the general code structure:
  - Study the structure of the `src`-directory
  - Try to understand 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

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.

`Sf()` 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:

```
85          //- Coefficients (not including interfaces)
86          scalarField *lowerPtr_, *diagPtr_, *upperPtr_;
```

Listing 1: `lduMatrix.H`

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

```
1           const surfaceVectorField& Sf = p.mesh().Sf();
2           const unallocLabelList& owner = mesh.owner();
3           const unallocLabelList& neighbour = mesh.neighbour();
```

Listing 2: Surface normal, owner and neighbour for each face

Matrix format in OpenFOAM III

Sparsity of matrix:

$$\mathbf{A} = A_{i,j} \tag{8}$$

- 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:

```
3     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 `lduMatrix`:
  - 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

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 I

Possible choices of format:

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

$$\mathbf{A}(y)x = a \tag{9}$$

$$\mathbf{B}(x)y = b \tag{10}$$

  - Solved together (still segregated):

$$\begin{bmatrix} \mathbf{A}(y) & 0 \\ 0 & \mathbf{B}(x) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} \tag{11}$$

  - Coupled solution:

$$\begin{bmatrix} \mathbf{A}' & \mathbf{A}_y \\ \mathbf{B}_x & \mathbf{B}' \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} \tag{12}$$

  - 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):

$$\mathbf{C}z = c \tag{13}$$

$$\mathbf{C} = C_{i,j} = \begin{bmatrix} c_{a,a} & c_{a,b} \\ c_{b,a} & c_{b,b} \end{bmatrix}_{i,j} \tag{14}$$

$$c = c_i = \begin{bmatrix} s_a & s_b \end{bmatrix}_i^\top \tag{15}$$

$$z = z_i = \begin{bmatrix} x & y \end{bmatrix}_i^\top \tag{16}$$

  - 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 I

- 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):
  - `http://www.cplusplus.com/doc/tutorial/templates/`
  - 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")

Studying `blockCoupledScalarTransportFoam` I

- Example solver in OpenFOAM-1.6-ext: `blockCoupledScalarTransportFoam`
- Theory behind solver: coupled two phase heat transfer[1]:

$$\nabla \cdot (\mathbf{U}\, T) - \nabla(D \nabla \cdot T) = \alpha(T_s - T) \tag{17}$$

$$-\nabla(D T_s \nabla \cdot D_s) = \alpha(T - T_s) \tag{18}$$

- Velocity field $\mathbf{U}$ 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 transfered and
- how the system is solved

Run the `blockCoupledSwirlTest`

[1] Henrik Rusche and Hrvoje Jasak. *Implicit solution techniques for coupled multi-field problems – Block Solution, Coupled Matrices*. June 2010; Ivor Clifford. *Block-Coupled Simulations Using OpenFOAM*. June 2011.

Pressure-velocity coupling

Implicit model

- Navier-Stokes, incompressible, steady-state:

$$\nabla \cdot (\mathbf{U}) = 0 \tag{19}$$

$$\nabla \cdot (\mathbf{U}\mathbf{U}) - \nabla(\nu \nabla \mathbf{U}) = -\frac{1}{\rho}\nabla p \tag{20}$$

- Semi-discretized form:

$$\sum_{\text{faces}} \mathbf{U}_\text{f} \cdot \mathbf{S}_\text{f} = 0 \tag{21}$$

$$\sum_{\text{faces}} \left[\mathbf{U}\mathbf{U} - \nu \nabla \mathbf{U}\right]_\text{f} \cdot \mathbf{S}_\text{f} = -\sum_{\text{faces}} P_\text{f} \mathbf{S}_\text{f} \tag{22}$$

- Modified pressure:

$$\frac{p}{\rho} = P \tag{23}$$

- Rhie-Chow in continuity equation:

$$\sum_{\text{faces}} \left[\overline{\mathbf{U}_\text{f}} - \overline{\mathbf{D}_\text{f}} \left(\nabla P_\text{f} - \overline{\nabla P_\text{f}}\right)\right] \cdot \mathbf{S}_\text{f} = 0 \tag{24}$$

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>`

$$[A][x] = [b] \tag{25}$$

Study the Programmers guide to find the available:

- fvm discretizations
- fvc discretizations

Pressure-velocity discretization

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

```
1    fvm::div(U)
2  - fvm::laplacian(D,p)
3  ==
4  - fvc::div(D*fvc::grad(p))
```

```
1    fvm::div(phi,U)
2  + turbulence->divDevReff(U)
3  ==
4  - fvm::grad(p)
```

Problem:

- Implicit `div` and `grad` not generally desired $\rightarrow$ implementations not existing

Implementing the pressure-velocity coupling I

Solution vector:

$$x^P = x_l^P = \begin{bmatrix} u^P \\ v^P \\ w^P \\ P^P \end{bmatrix}_l \tag{26}$$

```
117 // Block vector field for the pressure and velocity field to be solved for
118 volVector4Field pU
119 (
120     IOobject
121     (
122         "pU",
123         runTime.timeName(),
124         mesh,
125         IOobject::NO_READ,
126         IOobject::NO_WRITE
127     ),
128     mesh,
129     dimensionedVector4(word(),dimless,vector4::zero)
130 );
```

Implementing the pressure-velocity coupling II

```
132 // Insert the pressure and velocity internal fields in to the volVector2Field
133 {
134     vector4Field blockX = pU.internalField();
135
136     // Separately add the three velocity components
137     for (int i=0; i<3;i++)
138     {
139         tmp<scalarField> tf = U.internalField().component(i);
140         scalarField& f = tf();
141         blockMatrixTools::blockInsert(i,f,blockX);
142     }
143
144     // Pressure is the 2nd component
145     scalarField& f = p.internalField();
146     blockMatrixTools::blockInsert(3,f,blockX);
147 }
```

Implementing the pressure-velocity coupling III

Equation system to be formed:

$$\mathbf{A}^P x^P + \sum_{\mathrm{F}\in\{\mathrm{N}\}} \mathbf{A}^F x^F = b^P \tag{27}$$

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

Construct block matrix:

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

Retrieve fields:

```
191             // Diagonal is set separately
192             Field<tensor4>& d = B.diag().asSquare();
193
194             // Off-diagonal also as square
195             Field<tensor4>& u = B.upper().asSquare();
196             Field<tensor4>& l = B.lower().asSquare();
```

Source:

```
198             // Source term for the block matrix
199             Field<vector4> s(mesh.nCells(), vector4::zero);
```

Discretizing the momentum equation I

LHS: Turbulence is introduced by calling the `divDivReff(U)`

```
182             tmp<fvVectorMatrix> UEqnLHS
183             (
184                 fvm::div(phi,U)
185               + turbulence->divDevReff(U)
186             );
```

Retrieve matrix coefficients:

```
202             tmp<scalarField> tdiag = UEqnLHS().D();
203             scalarField& diag = tdiag();
204             scalarField& upper = UEqnLHS().upper();
205             scalarField& lower = UEqnLHS().lower();
```

Add boundary contribution:

```
211             // Add source boundary contribution
212             vectorField& source = UEqnLHS().source();
213             UEqnLHS().addBoundarySource(source, false);
```

Discretizing the momentum equation II

Considering RHS as separate problem:

$$\sum_{\text{faces}} P_{\text{f}} \mathbf{S}_{\text{f}} = 0 \tag{29}$$

Interpolation weights:

```
218             // Interpolation scheme for the pressure weights
219             tmp<surfaceInterpolationScheme<scalar> >
220             tinterpScheme_
221             (
222                 surfaceInterpolationScheme<scalar>::New
223                 (
224                     p.mesh(),
225                     p.mesh().interpolationScheme("grad(p)")
226                 )
227             );
```

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

$$w_N = 1 - w_P \tag{30}$$

Discretizing the momentum equation III

Equivalent to matrix fields:

```
229             // Pressure gradient contributions — corresponds to an implicit
230             // gradient operator
231             tmp<vectorField> tpUv = tmp<vectorField>
232                 (
233                     new vectorField(upper.size(),pTraits<vector>::zero)
234                 );
235             vectorField& pUv = tpUv();
236             tmp<vectorField> tpLv = tmp<vectorField>
237                 (
238                     new vectorField(lower.size(),pTraits<vector>::zero)
239                 );
240             vectorField& pLv = tpLv();
241             tmp<vectorField> tpSv = tmp<vectorField>
242                 (
243                     new vectorField(source.size(),pTraits<vector>::zero)
244                 );
245             vectorField& pSv = tpSv();
246             tmp<vectorField> tpDv = tmp<vectorField>
247                 (
248                     new vectorField(diag.size(),pTraits<vector>::zero)
249                 );
250             vectorField& pDv = tpDv();
```

Discretizing the momentum equation IV

Calcualte elements:

```
256             for(int i=0;i<owner.size();i++)
257             {
258                 int o = owner[i];
259                 int n = neighbour[i];
260                 scalar w = weights.internalField()[i];
261                 vector s = Sf[i];
262
263                 pDv[o]+=s*w;
264                 pDv[n]-=s*(1-w);
265                 pLv[i]=-s*w;
266                 pUv[i]=s*(1-w);
267
268             }
```

Boundary contribution:

```
271             p.boundaryField().updateCoeffs();
272             forAll(p.boundaryField(),patchI)
273             {
274                 // Present fvPatchField
275                 fvPatchField<scalar> & fv = p.boundaryField()[patchI];
276
277                 // Retrieve the weights for the boundary
278                 const fvsPatchScalarField& pw = weights.boundaryField()[patchI];
279
280                 // Contributions from the boundary coefficients
281                 tmp<Field<scalar> > tic = fv.valueInternalCoeffs(pw);
282                 Field<scalar>& ic = tic();
283                 tmp<Field<scalar> > tbc = fv.valueBoundaryCoeffs(pw);
```

Discretizing the momentum equation V

```
284                 Field<scalar>& bc = tbc();
285
286                 // Get the fvPatch only
287                 const fvPatch& patch = fv.patch();
288
289                 // Surface normals for this patch
290                 tmp<Field<vector> > tsn = patch.Sf();
291                 Field<vector> sn = tsn();
292
293                 // Manually add the contributions from the boundary
294                 // This what happens with addBoundaryDiag, addBoundarySource
295                 forAll(fv,facei)
296                 {
297                     label c = patch.faceCells()[facei];
298
299                     pDv[c]+=ic[facei]*sn[facei];
300                     pSv[c]-=bc[facei]*sn[facei];
301                 }
302             }
```

Discretizing the momentum equation VI

$a_{u,u}$, $a_{v,v}$, $a_{w,w}$, $a_{p,u}$, $a_{p,v}$, $a_{p,w}$:

```
317             forAll(d,i)
318             {
319                 d[i](0,0) = diag[i];
320                 d[i](1,1) = diag[i];
321                 d[i](2,2) = diag[i];
322 
323                 d[i](0,3) = pDv[i].x();
324                 d[i](1,3) = pDv[i].y();
325                 d[i](2,3) = pDv[i].z();
326             }
327             forAll(l,i)
328             {
329                 l[i](0,0) = lower[i];
330                 l[i](1,1) = lower[i];
331                 l[i](2,2) = lower[i];
332 
333                 l[i](0,3) = pLv[i].x();
334                 l[i](1,3) = pLv[i].y();
335                 l[i](2,3) = pLv[i].z();
336             }
337             forAll(u,i)
338             {
339                 u[i](0,0) = upper[i];
340                 u[i](1,1) = upper[i];
341                 u[i](2,2) = upper[i];
342 
343                 u[i](0,3) = pUv[i].x();
344                 u[i](1,3) = pUv[i].y();
345                 u[i](2,3) = pUv[i].z();
346             }
347             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 I

One implicit and one explicitcontribution:

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

```
442             tmp<volVectorField> texp = fvc::grad(p);
443             volVectorField& exp = texp();
444             tmp<volVectorField> texp2 = exp/A;
445             volVectorField exp2 = texp2();
446
447             tmp<fvScalarMatrix> MEqnLHSp
448             (
449                 -fvm::laplacian(1/A,p)
450                 ==
451                 -fvc::div(exp2)
452             );
```

```
454             // Add the boundary contributions
455             scalarField& pMdiag = MEqnLHSp().diag();
456             scalarField& pMupper = MEqnLHSp().upper();
457             scalarField& pMlower = MEqnLHSp().lower();
458
459             // Add diagonal boundary contribution
460             MEqnLHSp().addBoundaryDiag(pMdiag,0);
461
462             // Add source boundary contribution
463             scalarField& pMsource = MEqnLHSp().source();
464             MEqnLHSp().addBoundarySource(pMsource, false);
```

Discretizing the continuity equation II

Need implicit divergence scheme:

```
348             // Again an implicit version not existing, now the div operator
349             tmp<surfaceInterpolationScheme<scalar> >
350             UtinterpScheme_
351             (
352                 surfaceInterpolationScheme<scalar>::New
353                 (
354                     U.mesh(),
355                     U.mesh().interpolationScheme("div(U)(implicit)")
356                 )
357             );
358
359
360             // 1) Setup diagonal, source, upper and lower
361             tmp<vectorField> tMUpper = tmp<vectorField>
362                 (new vectorField(upper.size(),pTraits<vector>::zero));
363             vectorField& MUpper = tMUpper();
364
365             tmp<vectorField> tMLower = tmp<vectorField>
366                 (new vectorField(lower.size(),pTraits<vector>::zero));
367             vectorField& MLower = tMLower();
368
369             tmp<vectorField> tMDiag = tmp<vectorField>
370                 (new vectorField(diag.size(),pTraits<vector>::zero));
371             vectorField& MDiag = tMDiag();
372
373             tmp<vectorField> tMSource = tmp<vectorField>
374                 (
375                     new vectorField
376                     (
377                         source.component(0)().size(),pTraits<vector>::zero
378                     )
```

Discretizing the continuity equation III

```
379                 );
380             vectorField& MSource = tMSource();
381
382             // 2) Use interpolation weights to assemble the contributions
383             tmp<surfaceScalarField> tMweights =
384                 UtinterpScheme_().weights(mag(U));
385             const surfaceScalarField& Mweights = tMweights();
386
387             for(int i=0;i<owner.size();i++)
388             {
389                 int o = owner[i];
390                 int n = neighbour[i];
391                 scalar w = Mweights.internalField()[i];
392                 vector s = Sf[i];
393
394                 MDiag[o]+=s*w;
395                 MDiag[n]-=s*(1-w);
396                 MLower[i]=-s*w;
397                 MUpper[i]=s*(1-w);
398             }
399
400             // Get boundary condition contributions for the pressure grad(P)
401             U.boundaryField().updateCoeffs();
402             forAll(U.boundaryField(),patchI)
403             {
404                 // Present fvPatchField
405                 fvPatchField<vector> & fv = U.boundaryField()[patchI];
406
407                 // Retrieve the weights for the boundary
408                 const fvsPatchScalarField& Mw =
409                     Mweights.boundaryField()[patchI];
410
411                 // Contributions from the boundary coefficients
```

Discretizing the continuity equation IV

```
412                 tmp<Field<vector> > tic = fv.valueInternalCoeffs(Mw);
413                 Field<vector>& ic = tic();
414                 tmp<Field<vector> > tbc = fv.valueBoundaryCoeffs(Mw);
415                 Field<vector>& bc = tbc();
416
417                 // Get the fvPatch only
418                 const fvPatch& patch = fv.patch();
419
420                 // Surface normals for this patch
421                 tmp<Field<vector> > tsn = patch.Sf();
422                 Field<vector> sn = tsn();
423
424                 // Manually add the contributions from the boundary
425                 // This what happens with addBoundaryDiag, addBoundarySource
426                 forAll(fv,facei)
427                 {
428                     label c = patch.faceCells()[facei];
429
430                     MDiag[c]+=cmptMultiply(ic[facei],sn[facei]);
431                     MSource[c]-=cmptMultiply(bc[facei],sn[facei]);
432                 }
433             }
```

Discretizing the continuity equation V

$a_{u,p}$, $a_{v,p}$, $a_{w,p}$, $a_{p,p}$:

```
469             forAll(d,i)
470             {
471                 d[i](3,0) = MDiag[i].x();
472                 d[i](3,1) = MDiag[i].y();
473                 d[i](3,2) = MDiag[i].z();
474                 d[i](3,3) = pMdiag[i];
475             }
476             forAll(l,i)
477             {
478                 l[i](3,0) = MLower[i].x();
479                 l[i](3,1) = MLower[i].y();
480                 l[i](3,2) = MLower[i].z();
481                 l[i](3,3) = pMlower[i];
482             }
483             forAll(u,i)
484             {
485                 u[i](3,0) = MUpper[i].x();
486                 u[i](3,1) = MUpper[i].y();
487                 u[i](3,2) = MUpper[i].z();
488                 u[i](3,3) = pMupper[i];
489             }
490             forAll(s,i)
491             {
492                 s[i](3) =  MSource[i].x()
493                           +MSource[i].y()
494                           +MSource[i].z()
495                           +pMsource[i];
496             }
```

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

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 currents commits using log:

```
git log
```

Git - Hands on II

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 brougt together.

Git - Hands on III

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:

```
1 # Skip all the files ending with .o (object files)
2 *.o
3
4 # Skip all dependency files
5 *.dep
```

- When looking at the status of the repository the above files will be ignored.

Git - Information and software

Some documentation:

- Git - Documentation: `http://git-scm.com/doc` (entire book available at: `https://github.s3.amazonaws.com/media/progit.en.pdf`)
- 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

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

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

- Strongly typed language: change of type requires explicit conversion

```
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'
```

Python syntax I

- Commented lines start with "#"
- Loops and conditional statements controlled by indentation

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

- Three important data types:
  - Lists:

```
1  >>> foo = [1,"a"]
2  >>> bar = [1, 2, 3, 4]
3  >>> print foo[0]
4  1
5  >>> print bar[:]
6  [1, 2, 3, 4]
7  >>> print bar[1:2]
8  [2]
9  >>> print bar[-1]
10 4
11 >>> bar.append(4)
12 >>> print bar
13 [1, 2, 3, 4, 4]
```

Python syntax II

- Tuples:

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

- Dictionaries:

```
>>> test = {}
>>> test['value']=4
>>> test['name']="test"
>>> print test
{'name': 'test', 'value': 4}
```

Python modules I

Auxiliary code can be included from modules. Examples:

- `os`: Operating system interface. Example:

```
1 import os
2
3 # Run a command
4 os.system("run command")
```

- `shutil`: High-level file operations

```
1 import shutil
2
3 # Copy some files
4 shutil.copytree('template','runfolder')
```

Case study: Running a set of simulations I

- 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

```
#!/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"%p)
        os.system("sed -i 's/CYCLETYPE/%s/' system/fvSolution"%c)
        os.system("mpirun -np 8 steadyNavalFoam -parallel > log.steadyNavalFoam")

        os.chdir('..')
```

Case study: Extract convergence results I

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

```
#!/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"%p)
        os.system("sed -i 's/CYCLETYPE/%s/' system/fvSolution"%c)
        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=%f, clocktime=%f"%(c,p,exectime,clocktime)
        os.chdir('..')
```

Case study: Setting up large cases I

```
#!/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-1.6-ext-git/run/krjPbe/2D/meshes/box/coarse/moderator.blockMesh'
FIELDS = '/home/klas/OpenFOAM/klas-1.6-ext-git/run/krjPbe/2D/meshes/box/coarse/0'

##############################################################################
#########################   OPTIONS   ########################################
##############################################################################

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


##############################################################################
#########################   CLEAN UP   #######################################
##############################################################################

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

Case study: Setting up large cases II

```
34
35 ##############################################################################
36 ############################    SETUP    #####################################
37 ##############################################################################
38
39 if options.setup:
40     shutil.copy(MESH,'constant/polyMesh/blockMeshDict')
41
42     p = subprocess.Popen(['blockMesh'],\
43         stdout=subprocess.PIPE, stderr=subprocess.PIPE)
44     out, error = p.communicate()
45
46     if error:
47         print bcolors.FAIL + "ERROR: blockMesh failing" + bcolors.ENDC
48         print bcolors.ENDC + "ERROR MESSAGE: %s"%error + bcolors.ENDC
49
50     try:
51         shutil.rmtree('0')
52     except OSError:
53         pass
54
55     shutil.copytree(FIELDS,'0')
```

Plotting with Python - matplotlib

```
1  #!/usr/bin/python
2
3  import matplotlib.pyplot as plt
4  import numpy as np
5
6  x = np.linspace(0,1)
7  y = np.linspace(0,2)
8  y = y**2
9
10 plt.figure()
11 plt.plot(x,y)
12 plt.title('Test of matplotlib')
13 plt.xlabel('x')
14 plt.ylabel('y')
15 plt.savefig('Test.pdf',format='pdf')
```

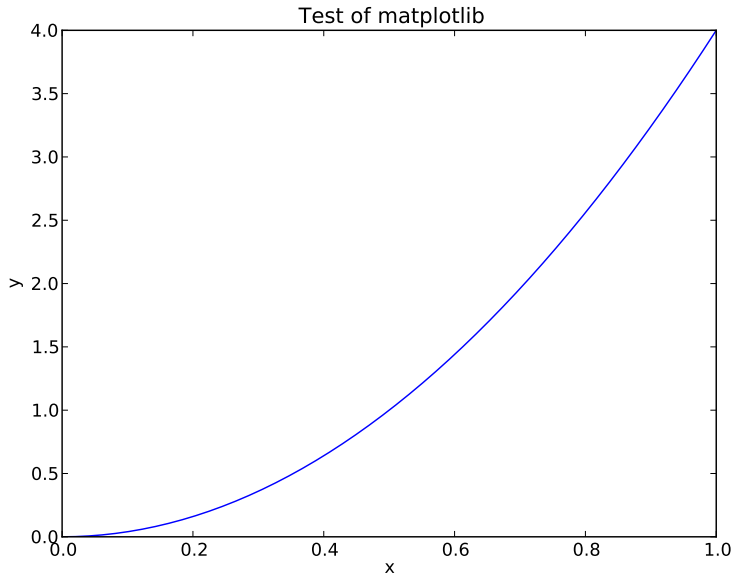


Figure: Example plot from matplotlib

More on plotting

- matplotlib (`http://matplotlib.org/`):
  - Plotting package with MATLAB equivalent syntax
  - Primarily 2D plots
- MayaVi2 (`http://code.enthought.com/projects/mayavi/`):
  - 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

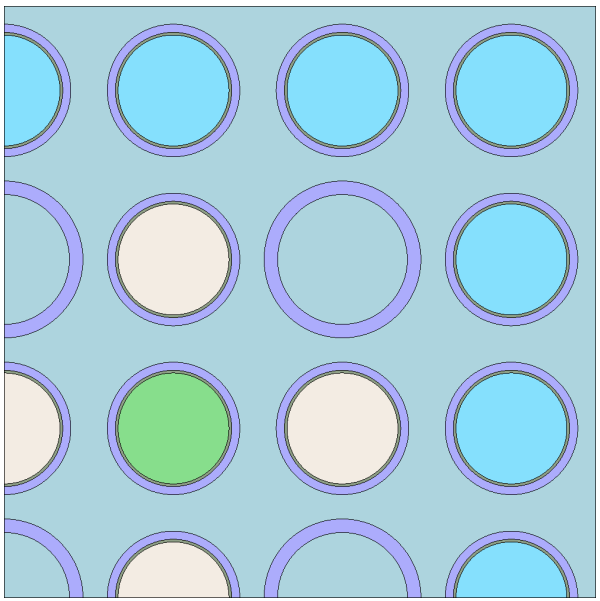
- `logging`: Flexible logging which could be used also for modules.
- `optparse`: Parser for command line options. Example from `http://docs.python.org/2/library/optparse.html`:

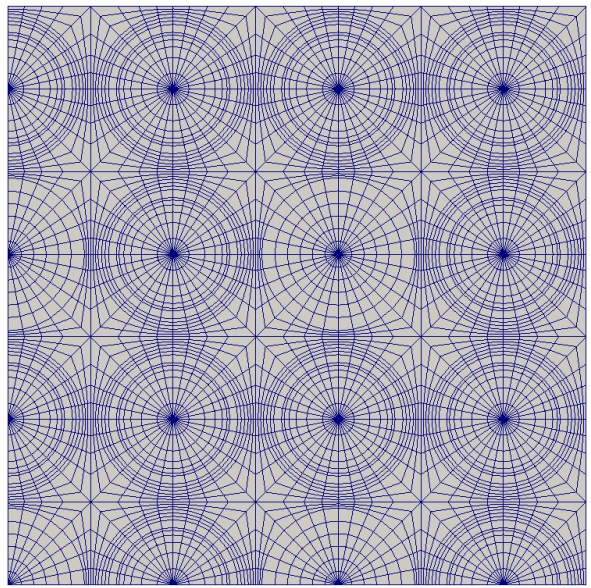
```
1  from optparse import OptionParser
2  [...]
3  parser = OptionParser()
4  parser.add_option("-f", "--file", dest="filename",
5                    help="write report to FILE", metavar="FILE")
6  parser.add_option("-q", "--quiet",
7                    action="store_false", dest="verbose", default=True,
8                    help="don't print status messages to stdout")
9
10 (options, args) = parser.parse_args()
```

- `numpy`: Scientific computing with Python. Information `http://wiki.scipy.org/Tentative_NumPy_Tutorial`
  - Array and matrix operations
  - Linear algebra

Case study: Meshing with Python I

- 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 (16x4+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

```
30
31 [pin0]
32 type: FuelPin
33 fue_ro: 0.41
34 fue_ri: 0.12
35 fue_dr: 4
36 ....
```

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 2 5 6 7 7 ) ( 1 1 1 ) simpleGrading (1.000000 1.000000 1.000000 )
    hex ( 0 2 10 10 5 7 13 13 ) ( 1 1 1 ) simpleGrading (1.000000 1.000000 1.000000 )
    hex ( 0 10 16 16 5 13 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 VI

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 librray written in Python