Solving PDEs with OpenFOAM

- The PDEs we wish to solve involve derivatives of tensor fields with respect to time and space
- The PDEs must be discretized in time and space before we solve them

- We will start by having a look at algebra of tensors in OpenFOAM at a single point
- We will then have a look at how to generate tensor fields from tensors
- Finally we will see how to discretize PDEs and how to set boundary conditions using high-level coding in OpenFOAM

Basic tensor classes in OpenFOAM

- Pre-defined classes for tensors of rank 0-3, but may be extended indefinitely

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

Example:

A tensor $T = \begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{bmatrix}$ is defined line-by-line:

```
tensor T( 11, 12, 13, 21, 22, 23, 31, 32, 33);

Info << "Txz = " << T.xz() << endl;
```

Outputs to the screen:

```
Txz = 13
```

Algebraic tensor operations in OpenFOAM

- Tensor operations operate on the entire tensor entity instead of a series of operations on its components
- The OpenFOAM syntax closely mimics the syntax used in written mathematics, using descriptive functions or symbolic operators

Examples:

Operation	Comment	Mathematical description	Description in OpenFOAM
Addition		$\mathbf{a} + \mathbf{b}$	a + b
Outer product	Rank $\mathbf{a}, \mathbf{b} \geq 1$	$\mathbf{ab}$	a * b
Inner product	Rank $\mathbf{a}, \mathbf{b} \geq 1$	$\mathbf{a} \cdot \mathbf{b}$	a & b
Cross product	Rank $\mathbf{a}, \mathbf{b} = 1$	$\mathbf{a} \times \mathbf{b}$	a ^ b
Operations exclusive to tensors of rank 2			
Transpose		T^T	T.T()
Diagonal		$\mathrm{diag}\mathbf{T}$	diag(T)
Determinant		$\det\mathbf{T}$	det(T)
Operations exclusive to scalars			
Positive (boolean)		$\mathrm{s} \geq 0$	pos(s)
Hyperbolic arc sine		asinh s	asinh(s)

Outer product between two vectors in OpenFOAM 1.4.1

See src/OpenFOAM/primitives/Tensor/TensorI.H (lines 351-362):

```
//- Outer-product between two vectors
template <class Cmpt>
inline typename outerProduct<Vector<Cmpt>, Vector<Cmpt> >::type
operator*(const Vector<Cmpt>& v1, const Vector<Cmpt>& v2)
{
    return Tensor<Cmpt>
    (
        v1.x()*v2.x(), v1.x()*v2.y(), v1.x()*v2.z(),
        v1.y()*v2.x(), v1.y()*v2.y(), v1.y()*v2.z(),
        v1.z()*v2.x(), v1.z()*v2.y(), v1.z()*v2.z()
    );
}
```

Dimensional units in OpenFOAM

- OpenFOAM checks the dimensional consistency

Declaration of a tensor with dimensions:

```
dimensionedTensor sigma
    (
        "sigma",
        dimensionSet( 1, -1, -2, 0, 0, 0, 0),
        tensor( 1e6, 0, 0, 0, 1e6, 0, 0, 0, 1e6)
    );
```

The values of dimensionSet correspond to the powers of each SI unit:

No.	Property	Unit	Symbol
1	Mass	kilogram	kg
2	Length	metre	m
3	Time	second	s
4	Temperature	Kelvin	K
5	Quantity	moles	mol
6	Current	ampere	A
7	Luminous intensity	candela	cd

sigma then has the dimension $[kg/ms^2]$

Construction of a tensor field in OpenFOAM

- A tensor field is a list of tensors
- The use of typedef in OpenFOAM yields readable type definitions: scalarField, vectorField, tensorField, symmTensorField, ...
- Algebraic operations can be performed between different fields, and between a field and a single tensor, e.g. Field U, scalar 2.0: U = 2.0 * U;

Discretization of a tensor field in OpenFOAM

- FVM (Finite Volume Method) or FEM (Finite Element Method)
- No limitations on the number of faces bounding each cell
- No restriction on the alignment of each face
- The mesh class polyMesh can be used to construct a polyhedral mesh using the minimum information required
- The fvMesh class extends the polyMesh class to include additional data needed for the FV discretization
- The geometricField class relates a tensor field to an fvMesh (can also be typedef volField, surfaceField, pointField)
- A geometricField inherits all the tensor algebra of its corresponding field, has dimension checking, and can be subjected to specific discretization procedures

Equation discretization in OpenFOAM

- Converts the PDEs into a set of linear algebraic equations, **Ax=b**, where **x** and **b** are volFields (geometricFields). **A** is an fvMatrix, which is created by a discretization of a geometricField and inherits the algebra of its corresponding field, and it supports many of the standard algebraic matrix operations
- The fvm (Finite Volume Method) and fvc (Finite Volume Calculus) classes contain static functions for the differential operators, and discretize any geometricField. fvm returns an fvMatrix, and fvc returns a geometricField.

Examples:

Term description	Implicit/explicit	Mathematical expression	fvm::/fvc:: functions
Laplacian	Implicit/Explicit	$\nabla \cdot \Gamma \nabla \phi$	laplacian(Gamma,phi)
Time derivative	Implicit/Explicit	$\partial \phi / \partial t$	ddt(phi)
		$\partial \rho \phi / \partial t$	ddt(rho, phi)
Convection	Implicit/Explicit	$\nabla \cdot (\psi)$	div(psi, scheme)
		$\nabla \cdot (\psi \phi)$	div(psi, phi, word)
			div(psi, phi)
Source	Implicit	$\rho \phi$	Sp(rho, phi)
	Implicit/Explicit		SuSp(rho, phi)

ϕ: vol<type>Field, ρ: scalar, volScalarField, ψ: surfaceScalarField

Example

The equation

$$\frac{\partial \rho \vec{U}}{\partial t} + \nabla \cdot \phi \vec{U} - \nabla \cdot \mu \nabla \vec{U} = -\nabla p$$

has the OpenFOAM representation

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

Example: laplacianFoam, the source code

Solves $\partial T/\partial t - \nabla \cdot k \nabla T = 0$

```
#include "fvCFD.H"  // Include the class definitions
int main(int argc, char *argv[])
{
#   include "setRootCase.H" // Set the correct path
#   include "createTime.H" // Create the time
#   include "createMesh.H" // Create the mesh
#   include "createFields.H" // Temperature field T and diffusivity DT
    for (runTime++; !runTime.end(); runTime++) // Time loop
    {
#   include "readSIMPLEControls.H" // Read solution controls
        for (int nonOrth=0; nonOrth$<$=nNonOrthCorr; nonOrth++)
        {
            solve( fvm::ddt(T) - fvm::laplacian(DT, T) ); // Solve eq.
        }
#   include "write.H" // Write out results at specified time instances}
    }
    return(0); // End with 'ok' signal
}
```

Example: laplacianFoam, discretization and boundary conditions

Discretization:
dictionary fvSchemes, read from file:

```
ddtSchemes
{
    default Euler;
}

laplacianSchemes
{
    default             none;
    laplacian(DT,T)     Gauss linear corrected;
}
```

Boundary conditions:
Part of class volScalarField object T, read from file:

```
boundaryField{
    patch1{ type zeroGradient;}
    patch2{ type fixedValue; value uniform 273;}}
```

Applications in OpenFOAM

- An application in OpenFOAM is a high-level code using the OpenFOAM libraries
- Applications are categorized into Solvers and Utilities. Solvers solve specific problems in continuum mechanics, Utilities perform tasks involving data manipulation

Examples of precompiled solvers:

Category	Application	Description
Solver	potentialFoam	Simple potential flow solver which can be used to generate starting fi
Solver	simpleFoam	Steady-state solver for incompressible, turbulent flow of non-Newton
Solver	turbFoam	Transient solver for incompressible turbulent flow
Solver	sonicTurbFoam	Transient solver for trans-sonic/supersonic turbulent flow of a compr
Solver	lesInterFoam	Solver for 2 incompressible fluids capturing the interface, using a rur
Solver	dnsFoam	Direct numerical simulation solver for boxes of isotropic turbulence
Solver	dieselEngineFoam	Diesel engine spray and combustion code
Solver	buoyantFoam	Transient solver for buoyant, turbulent flow of compressible fluids fo
Solver	electroStatic Foam	Solver for electrostatics
Solver	stressedFoam	Transient/steady-state solver of linear-elastic small-strain deformati
Solver	financialFoam	Solves the Black-Scholes equation to price commodities

Applications in OpenFOAM (continued)

Examples of precompiled utilities:

Category	Application	Description
Utility	mapFields	Maps volume fields from one mesh to another, reading and interpola
Utility	blockMesh	Mesh generator
Utility	fluentMeshToFoam	Converts a Fluent mesh to OpenFOAM format
Utility	checkMesh	Checks validity of a mesh
Utility	renumberMesh	Renumbers the cell list in order to reduce the bandwidth, reading an
Utility	foamToEnsight	Translates OpenFOAM data to Ensight format
Utility	Lambda2	Calculates and writes the second largest eigenvalue of the sum of the
Utility	checkYPlus	Calculates and reports y^+ for all wall patches, for each time in a data
Utility	decomposePar	Automatically decompose a mesh and fields for a case for parallel ex

Etc., etc. ...