Block-coupled Finite Volume algorithms : A solids4Foam tutorial

Ali Shayegh

Mechanical Engineering Faculty,
Shiraz University,
Shiraz, Iran

January 19, 2021

1. Outline
2. Preface
3. My First Block-coupled Simulations
4. Theory
5. Implementation
6. Adding a Numerical Diffusion Term

Learning outcomes

- Using coupled solid models;

Learning outcomes

- Using coupled solid models;
- Coupled vs Segregated solution procedure;

Learning outcomes

- Using coupled solid models;
- Coupled vs Segregated solution procedure;
- The discretization behind coupled solid models;

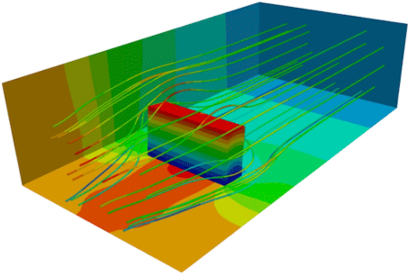
Learning outcomes

- Using coupled solid models;
- Coupled vs Segregated solution procedure;
- The discretization behind coupled solid models;
- solids4Foam source code structure;

Learning outcomes

- Using coupled solid models;
- Coupled vs Segregated solution procedure;
- The discretization behind coupled solid models;
- solids4Foam source code structure;
- Adding a numerical diffusion.

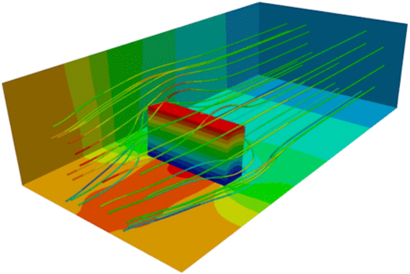
solids4Foam



By courtesy of Dr. P. Cardiff

- An OpenFOAM toolbox for solid mechanics and fluid-solid interactions

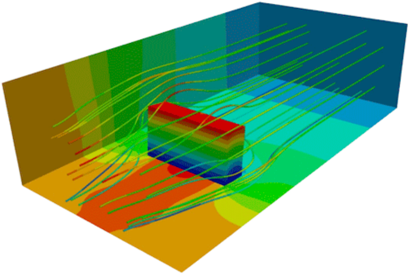
solids4Foam



By courtesy of Dr. P. Cardiff

- An OpenFOAM toolbox for solid mechanics and fluid-solid interactions
    - intuitive to *use*;

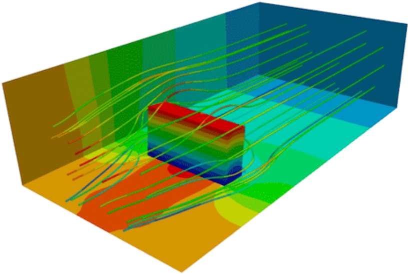
solids4Foam



By courtesy of Dr. P. Cardiff

- An OpenFOAM toolbox for solid mechanics and fluid-solid interactions
  - intuitive to *use*;
  - easy to *understand*;

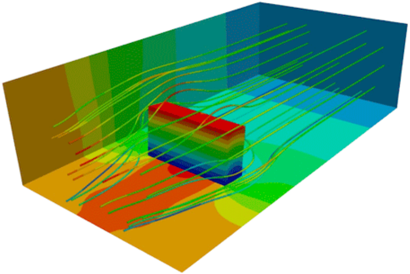
solids4Foam



By courtesy of Dr. P. Cardiff

- An OpenFOAM toolbox for solid mechanics and fluid-solid interactions
  - intuitive to *use*;
  - easy to *understand*;
  - straightforward to *maintain*;

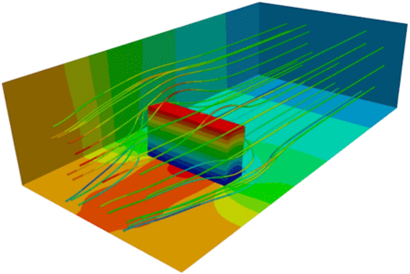
solids4Foam



By courtesy of Dr. P. Cardiff

- An OpenFOAM toolbox for solid mechanics and fluid-solid interactions
  - intuitive to *use*;
  - easy to *understand*;
  - straightforward to *maintain*;
  - uncomplicated to *extend*.

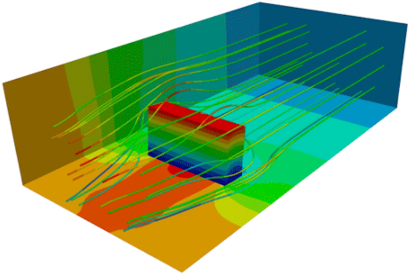
solids4Foam



By courtesy of Dr. P. Cardiff

- Installation

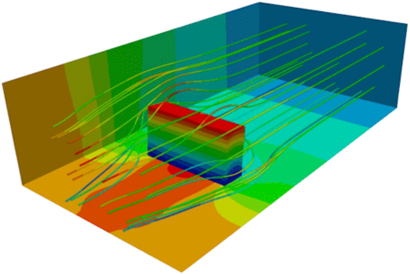
solids4Foam



By courtesy of Dr. P. Cardiff

- Installation
  - foam-extend-4.0 and 4.1 compatibility;

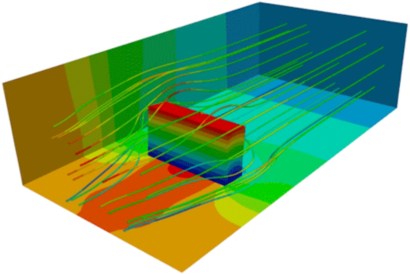
solids4Foam



By courtesy of Dr. P. Cardiff

- Installation
  - foam-extend-4.0 and 4.1 compatibility;
  - Partially compatible with ESI and Foundation forks;

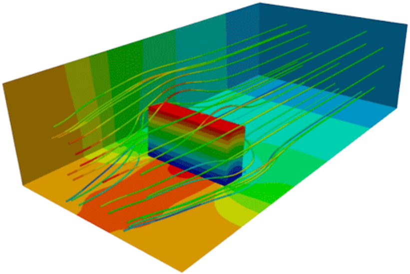
solids4Foam



By courtesy of Dr. P. Cardiff

- Installation
  - foam-extend-4.0 and 4.1 compatibility;
  - Partially compatible with ESI and Foundation forks;
  - Block-coupled solid models availability, only with foam-extend;

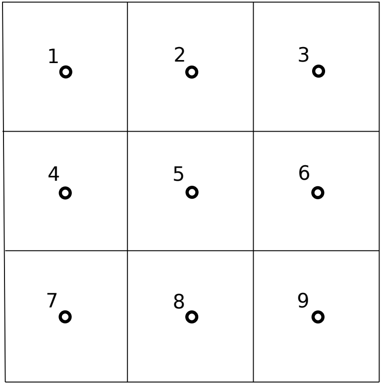
solids4Foam



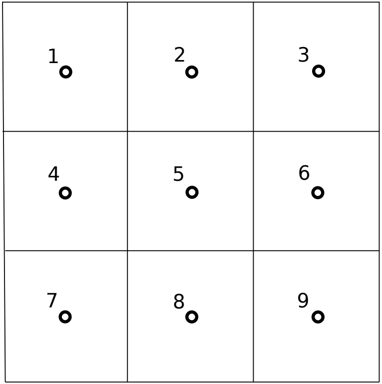
By courtesy of Dr. P. Cardiff

- Installation
  - foam-extend-4.0 and 4.1 compatibility;
  - Partially compatible with ESI and Foundation forks;
  - Block-coupled solid models availability, only with foam-extend;
  - Available instructions in bitbucket repository.

Block-Coupled in a Nutshell

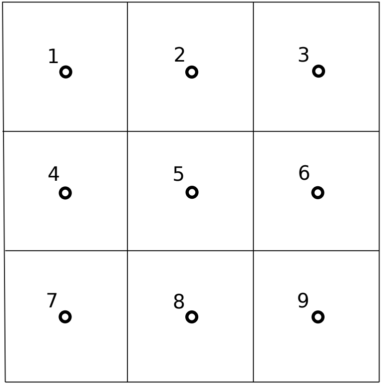


Block-Coupled in a Nutshell



$$AD = B$$

Block-Coupled in a Nutshell



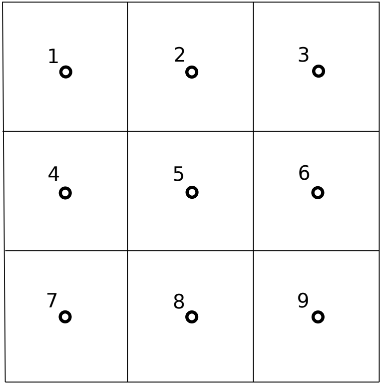
$$AD = B$$

$$A_x D_x = B_x$$

$$A_y D_y = B_y$$

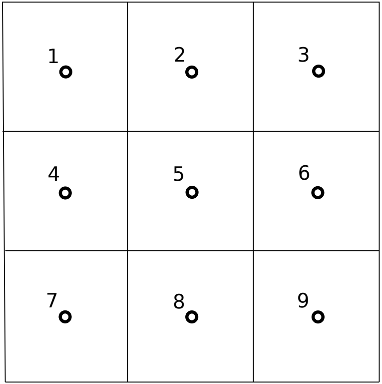
$$A_z D_z = B_z$$

Block-Coupled in a Nutshell



$$\begin{bmatrix} a_x{}^{11} & a_x{}^{12} & \cdots & a_x{}^{19} \\ a_x{}^{21} & a_x{}^{22} & \cdots & a_x{}^{29} \\ \vdots & \vdots & \ddots & \vdots \\ a_x{}^{91} & a_x{}^{92} & \cdots & a_x{}^{99} \end{bmatrix} \begin{bmatrix} D_x{}^{1} \\ D_x{}^{2} \\ \vdots \\ D_x{}^{9} \end{bmatrix} = \begin{bmatrix} B_x{}^{1} \\ B_x{}^{2} \\ \vdots \\ B_x{}^{9} \end{bmatrix}$$

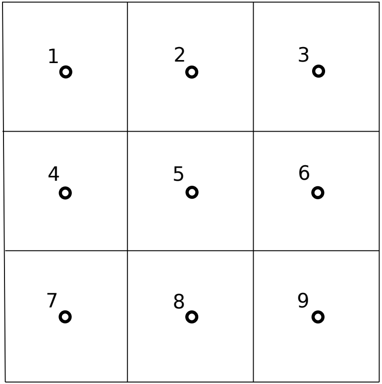
Block-Coupled in a Nutshell



$$\begin{bmatrix} {a_x}^{11} & {a_x}^{12} & \cdots & {a_x}^{19} \\ {a_x}^{21} & {a_x}^{22} & \cdots & {a_x}^{29} \\ \vdots & \vdots & \ddots & \vdots \\ {a_x}^{91} & {a_x}^{92} & \cdots & {a_x}^{99} \end{bmatrix} \begin{bmatrix} {D_x}^1 \\ {D_x}^2 \\ \vdots \\ {D_x}^9 \end{bmatrix} = \begin{bmatrix} {B_x}^1 \\ {B_x}^2 \\ \vdots \\ {B_x}^9 \end{bmatrix}$$

$$\begin{bmatrix} [A^{11}] & [A^{12}] & \cdots & [A^{19}] \\ [A^{21}] & [A^{22}] & \cdots & [A^{29}] \\ \vdots & \vdots & \ddots & \vdots \\ [A^{91}] & [A^{92}] & \cdots & [A^{99}] \end{bmatrix} \begin{bmatrix} [D^1] \\ [D^2] \\ \vdots \\ [D^9] \end{bmatrix} = \begin{bmatrix} [B^1] \\ [B^2] \\ \vdots \\ [B^9] \end{bmatrix}$$

Block-Coupled in a Nutshell



$$\begin{bmatrix} [A^{11}] & [A^{12}] & \cdots & [A^{19}] \\ [A^{21}] & [A^{22}] & \cdots & [A^{29}] \\ \vdots & \vdots & \ddots & \vdots \\ [A^{91}] & [A^{92}] & \cdots & [A^{99}] \end{bmatrix} \begin{bmatrix} [D^1] \\ [D^2] \\ \vdots \\ [D^9] \end{bmatrix} = \begin{bmatrix} [B^1] \\ [B^2] \\ \vdots \\ [B^9] \end{bmatrix}$$

$$[A^{11}] = \begin{bmatrix} {a_x}^{11} & {a_{xy}}^{11} & {a_{xz}}^{11} \\ {a_{yx}}^{11} & {a_y}^{11} & {a_{yz}}^{11} \\ {a_{zx}}^{11} & {a_{zy}}^{11} & {a_z}^{11} \end{bmatrix}, \quad [D^1] = \begin{bmatrix} {D_x}^1 \\ {D_y}^1 \\ {D_z}^1 \end{bmatrix}$$

$$[B^1] = \begin{bmatrix} {B_x}^1 \\ {B_y}^1 \\ {B_z}^1 \end{bmatrix}$$

Solid Simulation

- Run

Solid Simulation

- Run
  - coupledCantilever2D case;

Solid Simulation

- Run
  - coupledCantilever2D case;
    `$ blockMesh`

Solid Simulation

- Run
  - coupledCantilever2D case;

```
$ blockMesh
$ solids4Foam
```

Solid Simulation

- Run
  - coupledCantilever2D case;

    `$ blockMesh`

    `$ solids4Foam`
- View

Solid Simulation

- Run
  - coupledCantilever2D case;

    `$ blockMesh`

    `$ solids4Foam`
- View

    `$ paraFoam`

Solid Simulation

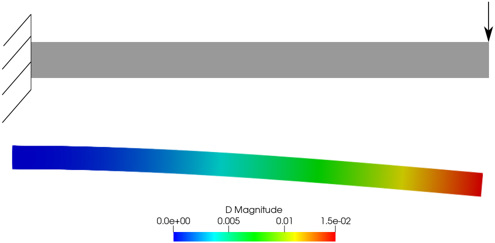
- Run
  - coupledCantilever2D case;

    `$ blockMesh`

    `$ solids4Foam`
- View

  `$ paraFoam`
  - Wrap By Vector

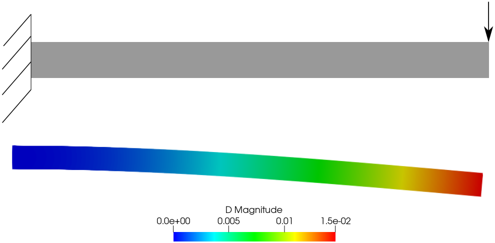
Solid Simulation



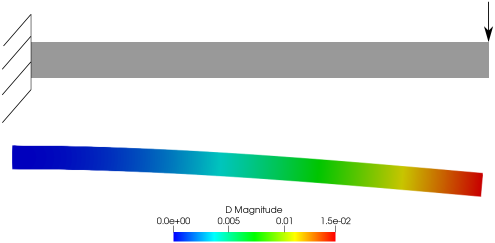
- Run
  - coupledCantilever2D case;
    `$ blockMesh`
    `$ solids4Foam`
- View
    `$ paraFoam`
  - Wrap By Vector

Solid Simulation

- Boundary conditions



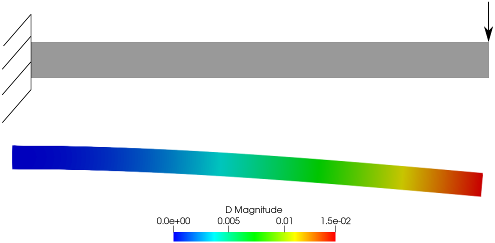
Solid Simulation



- Boundary conditions

```
left
{
    type  blockFixedDisplacement;
    value uniform (0 0 0);
}
```

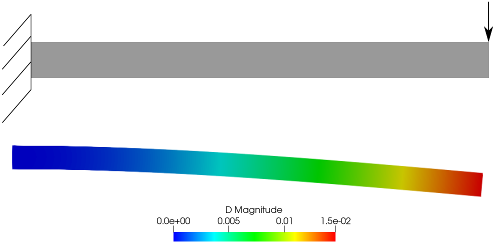
Solid Simulation



- Boundary conditions

```
left
{
    type  blockFixedDisplacement;
    value uniform (0 0 0);
}


right
{
    type            blockSolidTraction;
    traction        uniform ( 0 -1e6 0 );
    pressure        uniform 0;
    value           uniform (0 0 0);
}
```

Solid Simulation



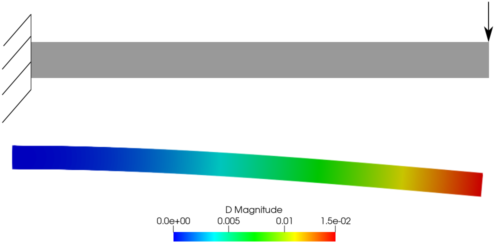
- Boundary conditions

```
left
{
    type  blockFixedDisplacement;
    value uniform (0 0 0);
}


right
{
    type            blockSolidTraction;
    traction        uniform ( 0 -1e6 0 );
    pressure        uniform 0;
    value           uniform (0 0 0);
}
```

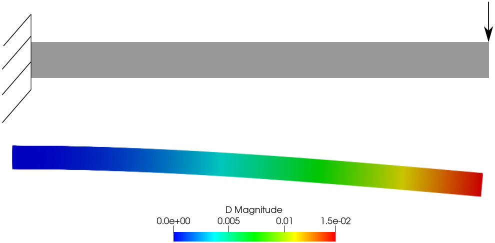
applied traction = `traction` − `n` ∗ `pressure`

Solid Simulation

- fvSchemes



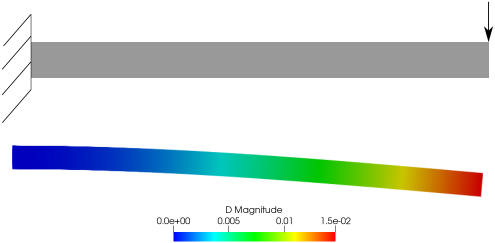
Solid Simulation



- fvSchemes

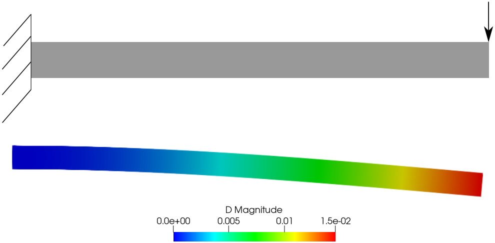
```
laplacianSchemes
{
  default
    none;
  fvmBlockLaplacian(D)
    pointGaussLeastSquaresLaplacian;
  fvmBlockLaplacianTranspose(D)
    pointGaussLeastSquaresLaplacianTranspose;
  fvmBlockLaplacianTrace(D)
    pointGaussLeastSquaresLaplacianTrace;
}
```

Solid Simulation



- fvSolution

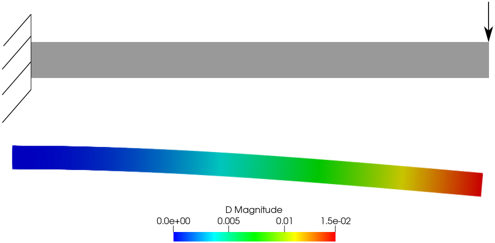
Solid Simulation



- fvSolution

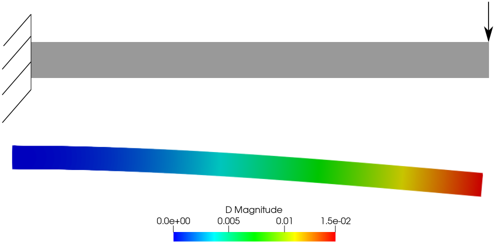
```
solvers
{
    blockD
    {
        // Direct solver
        solver     EigenSparseLU;
    }
}
```

Solid Simulation



- constant

Solid Simulation



- constant
  - physicsProperties
  - solidProperties
  - mechanicalProperties

Solid Simulation

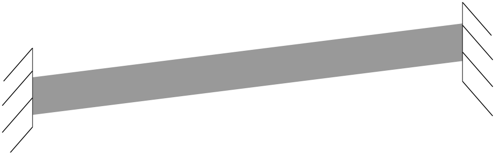
- Modified case

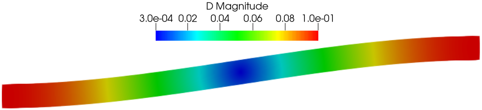
Solid Simulation

- Modified case

```
right
{
    type            blockFixedDisplacement;
    value           uniform (0 0.1 0);
}
    left
{
    type            blockFixedDisplacement;
    value           uniform (0 -0.1 0);
}
```

Solid Simulation

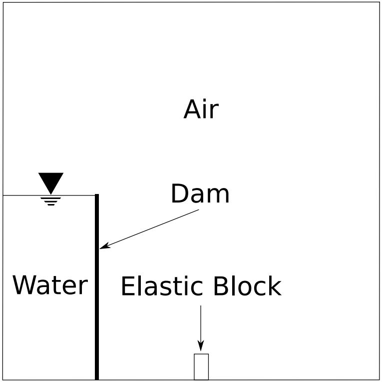




- Modified case

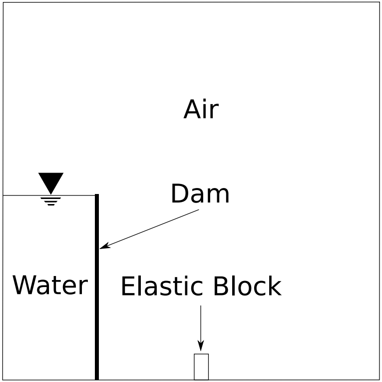
```
right
{
    type            blockFixedDisplacement;
    value           uniform (0 0.1 0);
}
    left
{
    type            blockFixedDisplacement;
    value           uniform (0 -0.1 0);
}
```

Fluid-Solid Interaction



- fsiProperties

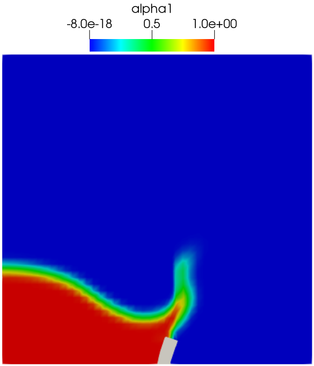
Fluid-Solid Interaction

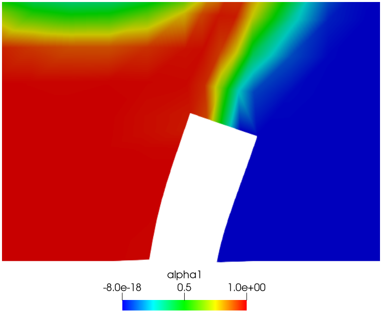


- fsiProperties

```
AitkenCoeffs
{
    solidPatch interface;

    fluidPatch interface;

    outerCorrTolerance 1e-6;

    nOuterCorr 20;
}
```

Fluid-Solid Interaction





Governing Equations

$$\int_\Omega \frac{\partial^2(\rho \mathsf{u})}{\partial t^2} \mathrm{d}V = \oint_\Gamma \mathsf{n}.\boldsymbol{\sigma} \mathrm{d}S + \int_\Omega \rho \mathsf{f} \mathrm{d}V \tag{1}$$

Governing Equations

$$\int_{\Omega} \frac{\partial^2(\rho \mathrm{u})}{\partial t^2} \mathrm{d}V = \oint_{\Gamma} \mathrm{n}.\boldsymbol{\sigma} \mathrm{d}S + \int_{\Omega} \rho \mathrm{f} \mathrm{d}V \tag{1}$$

$$\boldsymbol{\sigma} = \sigma_{ij} = \mu \partial_i u_j + \mu \partial_j u_i + \lambda \delta_{ij} \partial_k u_k \tag{2}$$

Governing Equations

$$\int_{\Omega} \frac{\partial^2(\rho \mathsf{u})}{\partial t^2} \mathrm{d}V = \oint_{\Gamma} \mathsf{n}.\boldsymbol{\sigma} \mathrm{d}S + \int_{\Omega} \rho \mathrm{f} \mathrm{d}V \tag{1}$$

$$\boldsymbol{\sigma} = \sigma_{ij} = \mu \partial_i u_j + \mu \partial_j u_i + \lambda \delta_{ij} \partial_k u_k \tag{2}$$

$$\oint_{\Gamma} \mathsf{n}.\boldsymbol{\sigma} \mathrm{d}S = \oint_{\Gamma} \mathsf{n}.[\mu \nabla \mathsf{u} + \mu (\nabla \mathsf{u})^{\mathrm{T}} + \lambda \mathsf{I} tr(\nabla \mathsf{u})] \mathrm{d}\Gamma \tag{3}$$

Governing Equations

$$\int_{\Omega} \frac{\partial^2(\rho \mathsf{u})}{\partial t^2} \mathrm{d}V = \oint_{\Gamma} \mathsf{n}.\boldsymbol{\sigma} \mathrm{d}S + \int_{\Omega} \rho \mathrm{f} \mathrm{d}V \tag{1}$$

$$\boldsymbol{\sigma} = \sigma_{ij} = \mu \partial_i u_j + \mu \partial_j u_i + \lambda \delta_{ij} \partial_k u_k \tag{2}$$

$$\oint_{\Gamma} \mathsf{n}.\boldsymbol{\sigma} \mathrm{d}S = \oint_{\Gamma} \mathsf{n}.[\mu \nabla \mathsf{u} + \mu (\nabla \mathsf{u})^{\mathrm{T}} + \lambda \mathsf{I} tr(\nabla \mathsf{u})] \mathrm{d}\Gamma \tag{3}$$

$$\oint_{\Gamma} \mu \mathsf{n}.\nabla \mathsf{u} \ \mathrm{d}\Gamma = \oint_{\Gamma} [\mu \mathsf{n}.\nabla \mathsf{u}_n + \mu \mathsf{n}.\nabla \mathsf{u}_t] \mathrm{d}\Gamma \tag{4}$$

$$\oint_{\Gamma} \mu \mathsf{n}.(\nabla \mathsf{u})^T \mathrm{d}\Gamma = \oint_{\Gamma} [\mu \mathsf{n}.\nabla \mathsf{u}_n + + \mu \nabla_t u_n] \mathrm{d}\Gamma \tag{5}$$

$$\oint_{\Gamma} \lambda \mathsf{n}.\mathsf{I} tr(\nabla \mathsf{u}) \mathrm{d}\Gamma = \oint_{\Gamma} [\lambda \mathsf{n}.\nabla \mathsf{u}_n + \lambda \mathsf{n} tr(\nabla_t \mathsf{u}_t)] \mathrm{d}\Gamma \tag{6}$$

$$\mathsf{u} = \mathsf{u}_n + \mathsf{u}_t \ , \ \mathsf{u}_n = \mathsf{nn.u} \ , \ u_n = \mathsf{n.u}$$

Coupled vs Segregated

(a) More calculations per time step, but totally more time efficient;

Coupled vs Segregated

(a) More calculations per time step, but totally more time efficient;

(b) Fast convergence rates;

Coupled vs Segregated

(a) More calculations per time step, but totally more time efficient;
(b) Fast convergence rates;
(c) Greater time step, while the same accuracy;

solids4Foam Structure

```
solids4foam-release/
├── Allwclean
├── Allwmake
├── applications
├── bitbucket-pipelines.yml
├── filesToReplaceInOF
├── README.md
├── src
├── ThirdParty
└── tutorials
```

solids4Foam.C

```
#include "fvCFD.H"
#include "physicsModel.H"
int main(int argc, char *argv[])
{
    #   include "setRootCase.H"
    #   include "createTime.H"
    #   include "solids4FoamWriteHeader.H"
    autoPtr<physicsModel> physics = physicsModel::New(runTime);

    while (runTime.run())
    {
        physics().setDeltaT(runTime);

        runTime++;
        physics().evolve();
        if (runTime.outputTime())
        {
            physics().writeFields(runTime);
        }
    }
    return(0);
}
```

evolve()

blockM D = blockB

```
bool coupledUnsLinGeomLinearElasticSolid::evolve()
{
    // Create source vector for block matrix
    vectorField blockB(solutionVec_.size(), vector::zero);

    // Create block system
    BlockLduMatrix<vector> blockM(extendedMesh_);

    // Grab block diagonal and set it to zero
    Field<tensor>& d = blockM.diag().asSquare();
    d = tensor::zero;

    // Grab linear off-diagonal and set it to zero
    Field<tensor>& l = blockM.lower().asSquare();
    Field<tensor>& u = blockM.upper().asSquare();
    u = tensor::zero;
    l = tensor::zero;
```

Continued ...

evolve()

$$\oint_{\Gamma} \mathsf{n}.\boldsymbol{\sigma} \mathrm{d}S = \oint_{\Gamma} \mathsf{n}.[\mu \nabla \mathsf{u} + \mu (\nabla \mathsf{u})^{\mathrm{T}} + \lambda \mathsf{I} tr(\nabla \mathsf{u})] \mathrm{d}\Gamma$$

```
// Laplacian
// non-orthogonal correction is treated implicitly
BlockLduMatrix<vector> blockMatLap =
BlockFvm::laplacian(extendedMesh_, muf_, D(), blockB);

// Laplacian transpose == div(mu*gradU.T())
BlockLduMatrix<vector> blockMatLapTran =
BlockFvm::laplacianTranspose(extendedMesh_, muf_, D(), blockB);

// Laplacian trace == div(lambda*I*tr(gradU))
BlockLduMatrix<vector> blockMatLapTrac =
BlockFvm::laplacianTrace(extendedMesh_, lambdaf_, D(), blockB);
```

Continued ...

evolve()

$$\oint_{\Gamma} \mu \mathbf{n}.\nabla \mathbf{u} \, d\Gamma = \oint_{\Gamma} [\mu \mathbf{n}.\nabla \mathbf{u}_n + \mu \mathbf{n}.\nabla \mathbf{u}_t] d\Gamma$$

$$\oint_{\Gamma} \mathbf{n}.\boldsymbol{\sigma} dS = \oint_{\Gamma} \mathbf{n}.[\mu \nabla \mathbf{u} + \mu (\nabla \mathbf{u})^{\mathrm{T}} + \lambda \mathbf{I} tr(\nabla \mathbf{u})] d\Gamma$$

```
// Laplacian
// non-orthogonal correction is treated implicitly
BlockLduMatrix<vector> blockMatLap =
BlockFvm::laplacian(extendedMesh_, muf_, D(), blockB);

// Laplacian transpose == div(mu*gradU.T())
BlockLduMatrix<vector> blockMatLapTran =
BlockFvm::laplacianTranspose(extendedMesh_, muf_, D(), blockB);

// Laplacian trace == div(lambda*I*tr(gradU))
BlockLduMatrix<vector> blockMatLapTrac =
BlockFvm::laplacianTrace(extendedMesh_, lambdaf_, D(), blockB);
```

Continued ...

BlockFvmDivSigma.C

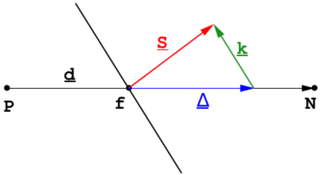
```
namespace BlockFvm
{
    tmp<BlockLduMatrix<vector> >
    laplacian
    (
        const solidPolyMesh& solidMesh,
        const surfaceScalarField& muf,
        GeometricField<vector, fvPatchField, volMesh>& U,
        Field<vector>& blockB
    )
    {
        return fv::blockLaplacian::New
        (
            U.mesh(),
            U.mesh().schemesDict().laplacianScheme
            (
            "fvmBlockLaplacian(" + U.name() + ')'
        )
        )().fvmBlockLaplacian(solidMesh, muf, U, blockB);
    }
}
```

pointGaussLsBlockLaplacianScheme.C

fvmBlockLaplacian()

```
{
    tmp<BlockLduMatrix<vector> > tBlockM
    (
        new BlockLduMatrix<vector>(solidMesh)
    );
    BlockLduMatrix<vector>& blockM = tBlockM();
    // Insert coeffs due to normal derivative terms
    insertCoeffsNorm(solidMesh, muf, U, blockB, blockM);

    // Insert coeffs due to tangential derivative terms from the non-orthogonal
    // corrections
    if (!U.mesh().orthogonal())
    {
        insertCoeffsTang(solidMesh, muf, U, blockB, blockM);
    }
    return tBlockM;
}
```

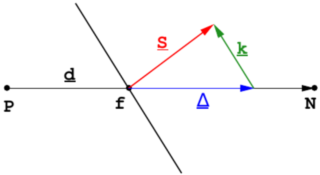
pointGaussLsBlockLaplacianScheme.C



insertCoeffsNorm()

```
// Normal derivative terms
const tensor coeff = I*faceMu*faceMagSf*faceDeltaCoeff;

d[own] -= coeff;
d[nei] -= coeff;

const label varI = fvMap[faceI];
u[varI] += coeff;
l[varI] += coeff;
```

pointGaussLsBlockLaplacianScheme.C



$$\oint_{\Gamma} \mu n_i \partial_i u_j \mathrm{d}S = \sum_f (\mu S_i \partial_i u_j)_f$$

$$S_i \partial_i u_j = S_i (\partial_i u_j)^{\mathrm{exp}} + \underbrace{[S \frac{d_j}{d_i n_i} \frac{u_{j_N} - u_{j_P}}{d} - \Delta_i (\partial_i u_j)^{\mathrm{exp}}]}_{\text{Stabilisation term}}$$

`insertCoeffsNorm()`

```
// Normal derivative terms
const tensor coeff = I*faceMu*faceMagSf*faceDeltaCoeff;

d[own] -= coeff;
d[nei] -= coeff;

const label varI = fvMap[faceI];
u[varI] += coeff;
l[varI] += coeff;
```

evolve()

```
    // Add diagonal contributions
    d += blockMatLap.diag().asSquare();
    d += blockMatLapTran.diag().asSquare();
    d += blockMatLapTrac.diag().asSquare();

    // Add off-diagonal contributions
    u += blockMatLap.upper().asSquare();
    u += blockMatLapTran.upper().asSquare();
    u += blockMatLapTrac.upper().asSquare();
    l += blockMatLap.lower().asSquare();
    l += blockMatLapTran.lower().asSquare();
    l += blockMatLapTrac.lower().asSquare();
```

Continued ...

evolve()

$$\int_{\Omega} \frac{\partial^2(\rho \mathsf{u})}{\partial t^2} \mathrm{d}V = \oint_{\Gamma} \mathsf{n}.\boldsymbol{\sigma} \mathrm{d}S + \int_{\Omega} \rho \mathsf{f} \mathrm{d}V$$

`blockM D = blockB`

```
    extendedMesh_.insertBoundaryConditions
    (
        blockM, blockB, muf_, lambdaf_, D()
    );

    // Add terms temporal and gravity terms to the block matrix and source
    extendedMesh_.addFvMatrix
    (
        blockM,
        blockB,
        rho()*fvm::d2dt2(D()) - rho()*g(),
        true
    );
```

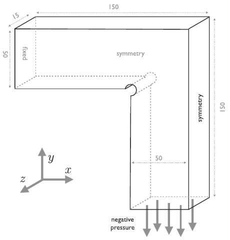
Continued ...

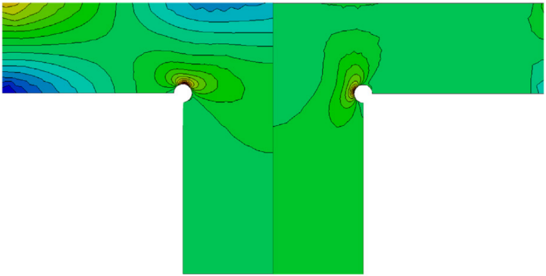
evolve()

blockM D = blockB

```
        solverPerfD =
            BlockLduSolver<vector>::New
            (
                D().name(),
                blockM,
                mesh().solutionDict().solver("blockD")
            )->solve(solutionVec_, blockB);
        return true;
    }
```

Cardiff et al. (2016): Oscillations





linGeomSolid.C

$$\int_{\Omega} \frac{\partial^2(\rho \mathrm{u})}{\partial t^2} \mathrm{d}V = \oint_{\Gamma} \mathrm{n}.\boldsymbol{\sigma} \mathrm{d}S + \int_{\Omega} \rho \mathrm{f} \mathrm{d}V$$

```
  rho*fvm::d2dt2(u)
 -fvm::laplacian(k,u)
 +fvc::laplacian(k,u)
 -fvc::div(sigma)
 +fvc::div(k*grad(u))
 -fvc::laplacian(k,u)
==
  rho*f
```

linGeomSolid.C

$$\int_{\Omega} \frac{\partial^2(\rho \mathrm{u})}{\partial t^2} \mathrm{d}V = \oint_{\Gamma} \mathrm{n}.\boldsymbol{\sigma} \mathrm{d}S + \int_{\Omega} \rho \mathrm{f} \mathrm{d}V$$

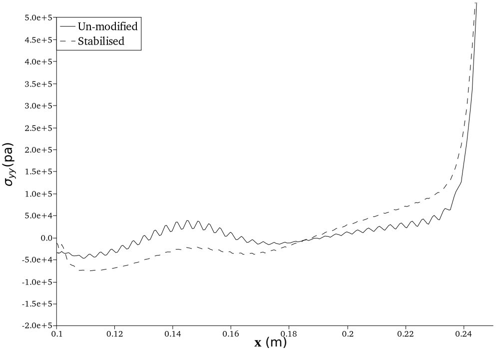
```
  rho*fvm::d2dt2(u)
 -fvm::laplacian(k,u)
 +fvc::laplacian(k,u)
 -fvc::div(sigma)
 +fvc::div(k*grad(u))
 -fvc::laplacian(k,u)
==
  rho*f
```

```
    // Linear momentum equation total displacement form
    fvVectorMatrix DDEqn
    (
          rho()*fvm::d2dt2(DD())
        + rho()*fvc::d2dt2(D().oldTime())
       == fvm::laplacian(impKf_, DD(), "laplacian(DDD,DD)")
        - fvc::laplacian(impKf_, DD(), "laplacian(DDD,DD)")
        + fvc::div(sigma(), "div(sigma)")
        + rho()*g()
        + mechanical().RhieChowCorrection(DD(), gradDD())
    );
```

coupledStabilised.C

```
bool coupledStabilised::evolve()
{
    for (int i = 0; i<3; i++){    // Added
        extendedMesh_.addFvMatrix
        (
            blockM,
            blockB,
            rho()*fvm::d2dt2(D()) - rho()*g()
            - mechanical().RhieChowCorrection(D(), gradD()),    // Added
            true
        );
    }    // Added
return true;
}
```

Results



Un-modified
Stabilised
σ_{yy}(pa)
x (m)
5.0e+5
4.5e+5
4.0e+5
3.5e+5
3.0e+5
2.5e+5
2.0e+5
1.5e+5
1.0e+5
5.0e+4
0.0
-5.0e+4
-1.0e+5
-1.5e+5
-2.0e+5
0.1
0.12
0.14
0.16
0.18
0.2
0.22
0.24

Thank you!