9 April 2021

University of Glasgow

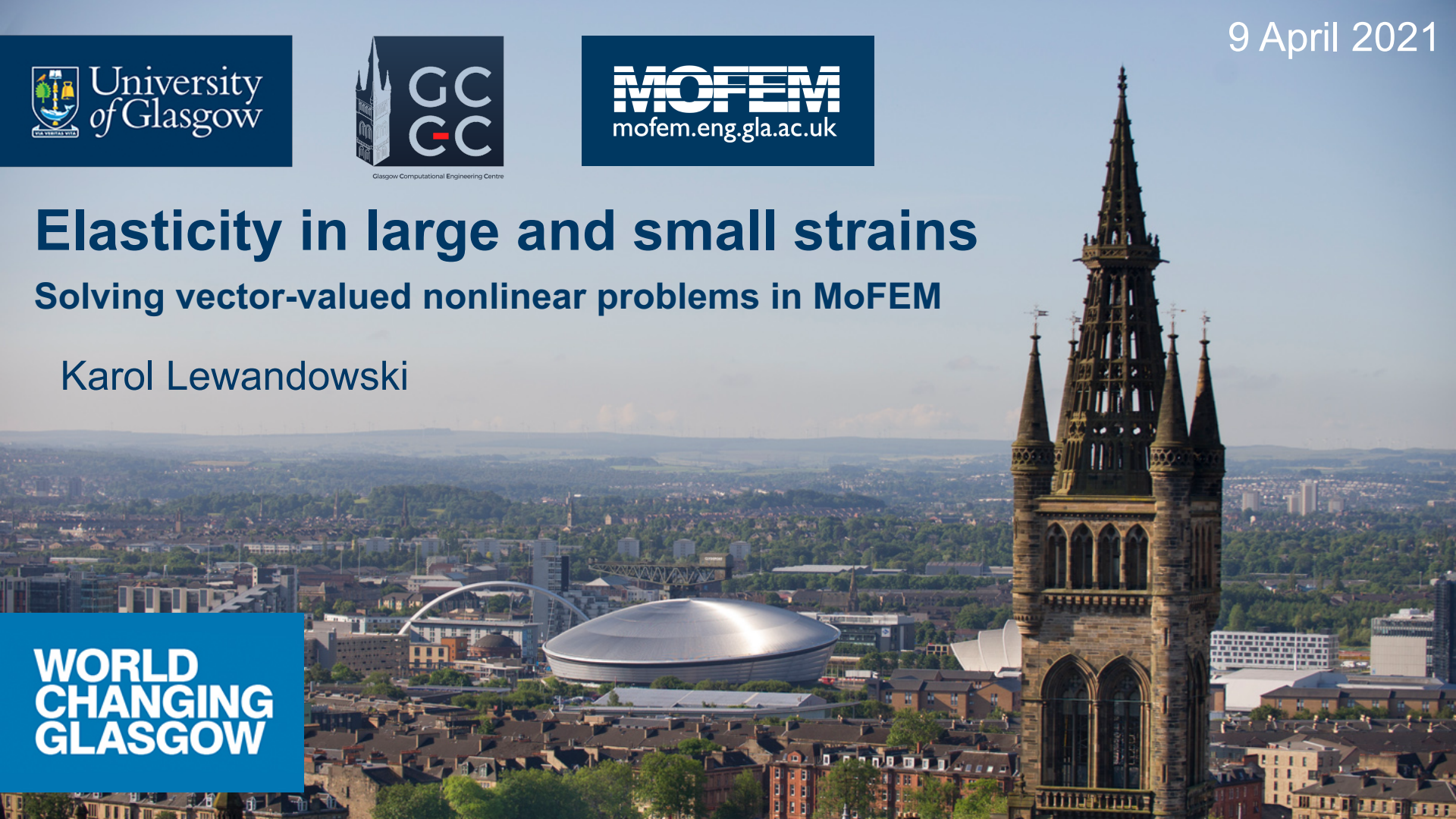
Glasgow Computational Engineering Centre

MOFEM
mofem.eng.gla.ac.uk

Elasticity in large and small strains

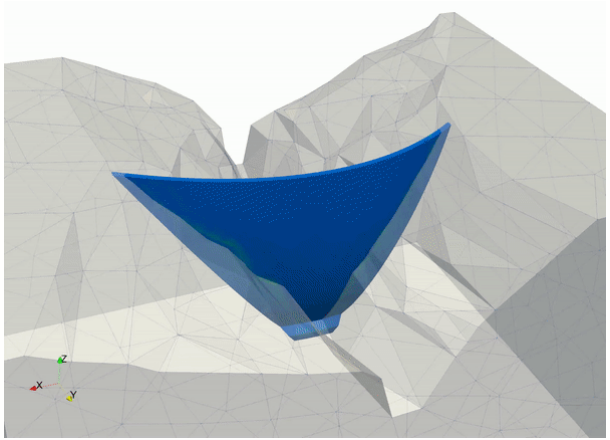
Solving vector-valued nonlinear problems in MoFEM

Karol Lewandowski



MOFEM
mofem.eng.gla.ac.uk

Elasticity problem

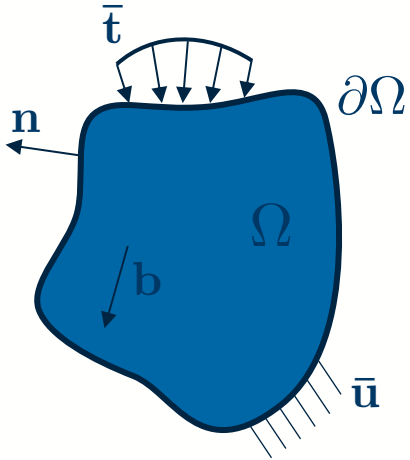


Outline

- Governing equations
- Implementation
- Result
- Large strains example
- Implementing and solving nonlinear problems
- Summary

VEC-0 http://mofem.eng.gla.ac.uk/mofem/html/tutorials.html

Governing equations



- Physical example: Deformation of an elastic body with arbitrary load

- Balance of momentum

$$\begin{cases} \nabla \cdot \boldsymbol{\sigma}\left(\mathbf{u}(\mathbf{x})\right) + \mathbf{b} = 0 & \text{in } \Omega \\ \mathbf{u}(\mathbf{x}) = \bar{\mathbf{u}} & \text{on } \partial\Omega^{\mathbf{u}} \\ \boldsymbol{\sigma} \cdot \mathbf{n} = \bar{\mathbf{t}} & \text{on } \partial\Omega^{\mathbf{t}} \end{cases}$$

- Constitutive equation

$$\boldsymbol{\sigma} = \mathbb{D} : \frac{1}{2}(\nabla \mathbf{u}^{\mathrm{T}} + \nabla \mathbf{u})$$

- Weak form – find $\mathbf{u} \in \mathbf{H}^1(\Omega)$

$$\int_\Omega \nabla \delta\mathbf{u} : \boldsymbol{\sigma}(\mathbf{u})\,\mathrm{d}\Omega = -\int_\Omega \delta\mathbf{u} \cdot \mathbf{b}\,\mathrm{d}\Omega + \int_{\partial\Omega^{\mathbf{t}}} \delta\mathbf{u} \cdot \bar{\mathbf{t}}\,\mathrm{d}S \quad \forall\, \delta\mathbf{u} \in \mathbf{H}_0^1(\Omega)$$

MOFEM
mofem.eng.gla.ac.uk

Discretisation

- Approximation

$$\mathbf{u} \approx \mathbf{u}^h = \sum_{j=0}^{N-1} \phi_j \bar{\mathbf{u}}_j \qquad \delta\mathbf{u} \approx \delta\mathbf{u}^h = \sum_{j=0}^{N-1} \phi_j \delta\bar{\mathbf{u}}_j$$

- Substitute to the weak form

- Global system

$$\mathbf{K}\mathbf{U} = \mathbf{F}$$

where

$$\mathbf{K}_{ij}^e = \int_{\Omega^e} \nabla\phi_i \mathbb{D} \nabla\phi_j \, \mathrm{d}\Omega \approx \sum_q \nabla\phi_i\left(\mathbf{x}_q\right) \mathbb{D} \nabla\phi_j\left(\mathbf{x}_q\right) w_q$$

$$\mathbf{F}_i^e = \int_{\partial\Omega^e} \phi_i \mathbf{t} \, \mathrm{d}S + \int_{\Omega^e} \phi_i \mathbf{b} \, \mathrm{d}\Omega \approx \sum_q \phi_i\left(\mathbf{x}_q\right) \mathbf{t}\left(\mathbf{x}_q\right) w_q + \sum_q \phi_i\left(\mathbf{x}_q\right) \mathbf{b}\left(\mathbf{x}_q\right) w_q$$

Implementation: Setup

```
constexpr int SPACE_DIM = 2; //< Space dimension of problem, mesh
```

```
//! [Run problem]
MoFEMErrorCode Example::runProblem() {
  MoFEMFunctionBegin;
  CHKERR readMesh();
  CHKERR setupProblem();
  CHKERR createCommonData();
  CHKERR boundaryCondition();
  CHKERR assembleSystem();
  CHKERR solveSystem();
  CHKERR outputResults();
  CHKERR checkResults();
  MoFEMFunctionReturn(0);
}
```

```
//! [Set up problem]
MoFEMErrorCode Example::setupProblem() {
  MoFEMFunctionBegin;
  Simple *simple = mField.getInterface<Simple>();
  // Add field
  CHKERR simple->addDomainField("U", H1, AINSWORTH_LEGENDRE_BASE, SPACE_DIM);
  CHKERR simple->addBoundaryField("U", H1, AINSWORTH_LEGENDRE_BASE, SPACE_DIM);
  int order = 2;
  CHKERR PetscOptionsGetInt(PETSC_NULL, "", "-order", &order, PETSC_NULL);
  CHKERR simple->setFieldOrder("U", order);
  CHKERR simple->setUp();
  MoFEMFunctionReturn(0);
}
```

MOFEM
mofem.eng.gla.ac.uk

Implementation: Elasticity tensor

```
//! [Run problem]
MoFEMErrorCode Example::runProblem() {
  MoFEMFunctionBegin;
  CHKERR readMesh();
  CHKERR setupProblem();
  CHKERR createCommonData();
  CHKERR boundaryCondition();
  CHKERR assembleSystem();
  CHKERR solveSystem();
  CHKERR outputResults();
  CHKERR checkResults();
  MoFEMFunctionReturn(0);
}
```

```
constexpr auto t_kd = FTensor::Kronecker_Delta_symmetric<int>();
constexpr double A =
        (SPACE_DIM == 2) ? 2 * shear_modulus_G /
                                (bulk_modulus_K + (4. / 3.) * shear_modulus_G) : 1;
    auto t_D = getFTensor4DdgFromMat<SPACE_DIM, SPACE_DIM, 0>(*matDPtr);

    t_D(i, j, k, l) = 2 * shear_modulus_G * ((t_kd(i, k) ^ t_kd(j, l)) / 4.) +
                      A * (bulk_modulus_K - (2. / 3.) * shear_modulus_G) *
                          t_kd(i, j) * t_kd(k, l);
```

$$\mathbb{D}_{ijkl} = G\left[\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}\right] + A(K - \frac{2}{3}G)\left[\delta_{ij}\delta_{kl}\right]$$

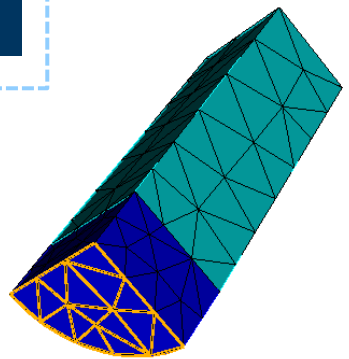
where for 3D or plane strain: $A = 1$

plane stress: $A = \dfrac{2G}{K + \frac{4}{3}G}$

Implementation: Dirichlet BC

```
//! [Run problem]
MoFEMErrorCode Example::runProblem() {
  MoFEMFunctionBegin;
  CHKERR readMesh();
  CHKERR setupProblem();
  CHKERR createCommonData();
  CHKERR boundaryCondition();
  CHKERR assembleSystem();
  CHKERR solveSystem();
  CHKERR outputResults();
  CHKERR checkResults();
  MoFEMFunctionReturn(0);
}
```

```
CHKERR remove_ents(fix_disp("FIX_X"), 0, 0);
CHKERR remove_ents(fix_disp("FIX_Y"), 1, 1);
CHKERR remove_ents(fix_disp("FIX_Z"), 2, 2);
CHKERR remove_ents(fix_disp("FIX_ALL"), 0, 3);
```



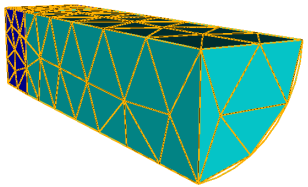
MOFEM
mofem.eng.gla.ac.uk

Implementation: Neumann BC

```
//! [Run problem]
MoFEMErrorCode Example::runProblem() {
  MoFEMFunctionBegin;
  CHKERR readMesh();
  CHKERR setupProblem();
  CHKERR createCommonData();
  CHKERR boundaryCondition();
  CHKERR assembleSystem();
  CHKERR solveSystem();
  CHKERR outputResults();
  CHKERR checkResults();
  MoFEMFunctionReturn(0);
}
```

```
auto set_body_force = [&]() {
  FTensor::Index<'i', SPACE_DIM> i;
  MoFEMFunctionBegin;
  auto t_force = getFTensor1FromMat<SPACE_DIM, 0>(*bodyForceMatPtr);
  t_force(i) = 0;
  t_force(1) = -1;
  MoFEMFunctionReturn(0);
};
```

```
pipeline_mng->getOpDomainRhsPipeline().push_back(new OpBodyForce(
    "U", bodyForceMatPtr, [](double, double, double) { return 1.;
}));
```



$$\mathbf{F}_i^e = \int_{\Omega^e} \phi_i \mathbf{b} \, \mathrm{d}V$$

Implementation: Assembly

Pushing UDOs into pipelines

```
//! [Run problem]
MoFEMErrorCode Example::runProblem() {
  MoFEMFunctionBegin;
  CHKERR readMesh();
  CHKERR setupProblem();
  CHKERR createCommonData();
  CHKERR boundaryCondition();
  CHKERR assembleSystem();
  CHKERR solveSystem();
  CHKERR outputResults();
  CHKERR checkResults();
  MoFEMFunctionReturn(0);
}
```

```
using OpK = FormsIntegrators<DomainEleOp>::Assembly<PETSC>::BiLinearForm<
    GAUSS>::OpGradSymTensorGrad<1, SPACE_DIM, SPACE_DIM, 0>;
```

```
if (SPACE_DIM == 2) {
   pipeline_mng->getOpDomainLhsPipeline().push_back(
       new OpCalculateInvJacForFace(invJac));
   pipeline_mng->getOpDomainLhsPipeline().push_back(
       new OpSetInvJacH1ForFace(invJac));
 }
 pipeline_mng->getOpDomainLhsPipeline().push_back(new OpK("U", "U", matDPtr));
```

$$\mathbf{K}_{ij}^{e} = \int_{\Omega^e} \nabla \phi_i \mathbb{D} \nabla \phi_j \, \mathrm{d}\Omega$$

Implementation: Solve

```
//! [Run problem]
MoFEMErrorCode Example::runProblem() {
  MoFEMFunctionBegin;
  CHKERR readMesh();
  CHKERR setupProblem();
  CHKERR createCommonData();
  CHKERR boundaryCondition();
  CHKERR assembleSystem();
  CHKERR solveSystem();
  CHKERR outputResults();
  CHKERR checkResults();
  MoFEMFunctionReturn(0);
}
```

```
//! [Solve]
MoFEMErrorCode Example::solveSystem() {
  MoFEMFunctionBegin;
  auto *simple = mField.getInterface<Simple>();
  auto *pipeline_mng = mField.getInterface<PipelineManager>();
  auto solver = pipeline_mng->createKSP();
  CHKERR KSPSetFromOptions(solver);
  CHKERR KSPSetUp(solver);

  auto dm = simple->getDM();
  auto D = smartCreateDMVector(dm);
  auto F = smartVectorDuplicate(D);

  CHKERR KSPSolve(solver, F, D);
  CHKERR VecGhostUpdateBegin(D, INSERT_VALUES, SCATTER_FORWARD);
  CHKERR VecGhostUpdateEnd(D, INSERT_VALUES, SCATTER_FORWARD);
  CHKERR DMoFEMMeshToLocalVector(dm, D, INSERT_VALUES,
      SCATTER_REVERSE);
  MoFEMFunctionReturn(0);
}
```

This snippet will be the similar for all linear problems

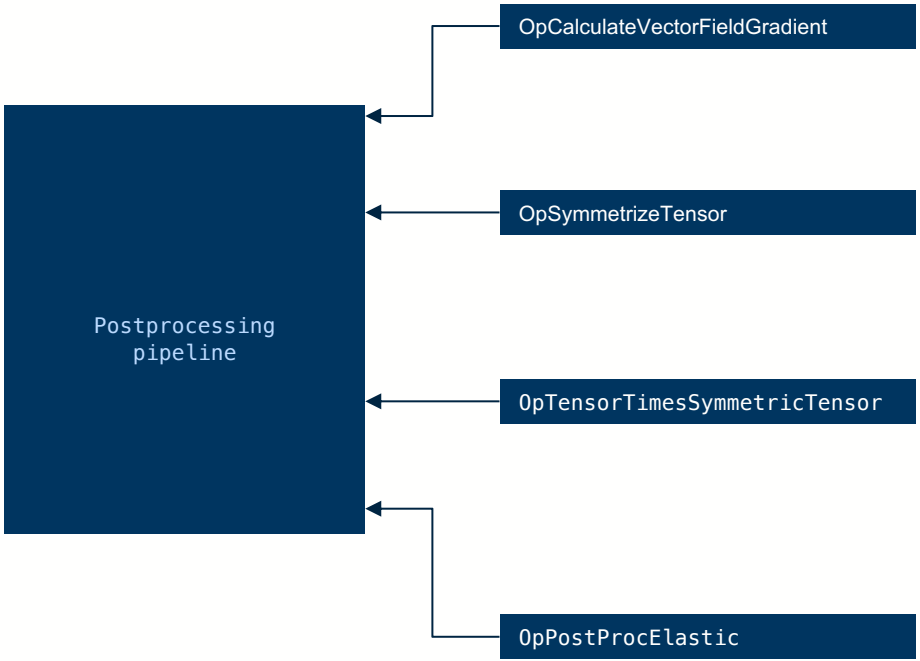
MOFEM
mofem.eng.gla.ac.uk

Implementation: Postprocessing

```
//! [Run problem]
MoFEMErrorCode Example::runProblem() {
  MoFEMFunctionBegin;
  CHKERR readMesh();
  CHKERR setupProblem();
  CHKERR createCommonData();
  CHKERR boundaryCondition();
  CHKERR assembleSystem();
  CHKERR solveSystem();
  CHKERR outputResults();
  CHKERR checkResults();
  MoFEMFunctionReturn(0);
}
```

```
pipeline_mng->getDomainLhsFE().reset();
auto post_proc_fe = boost::make_shared<PostProcEle>(mField);
post_proc_fe->generateReferenceElementMesh();
post_proc_fe->getOpPtrVector().push_back(
    new OpCalculateVectorFieldGradient<SPACE_DIM, SPACE_DIM>("U",
                                                             matGradPtr));
post_proc_fe->getOpPtrVector().push_back(
    new OpSymmetrizeTensor<SPACE_DIM>("U", matGradPtr, matStrainPtr));
post_proc_fe->getOpPtrVector().push_back(
    new OpTensorTimesSymmetricTensor<SPACE_DIM, SPACE_DIM>(
        "U", matStrainPtr, matStressPtr, matDPtr));
post_proc_fe->getOpPtrVector().push_back(new OpPostProcElastic<SPACE_DIM>(
    "U", post_proc_fe->postProcMesh, post_proc_fe->mapGaussPts, matStrainPtr,
    matStressPtr));
post_proc_fe->addFieldValuesPostProc("U");
pipeline_mng->getDomainRhsFE() = post_proc_fe;
CHKERR pipeline_mng->loopFiniteElements();
CHKERR post_proc_fe->writeFile("out_elastic.h5m");
```

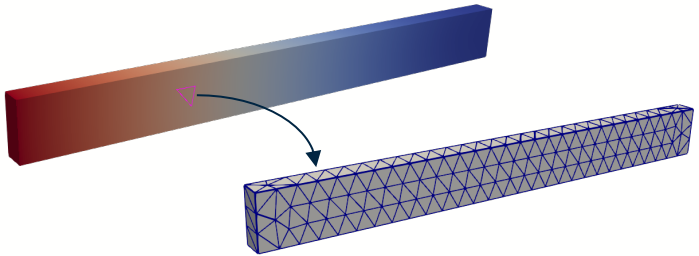
Implementation: Postprocessing



$$\nabla \mathbf{u}^h = \sum_{m=0}^{N-1} \nabla \phi_m \bar{\mathbf{u}}_m$$

$$\boldsymbol{\varepsilon} = \frac{1}{2}(\nabla \mathbf{u}^{\mathrm{T}} + \nabla \mathbf{u})$$

$$\boldsymbol{\sigma} = \mathbb{D} : \boldsymbol{\varepsilon}$$



MOFEM
mofem.eng.gla.ac.uk

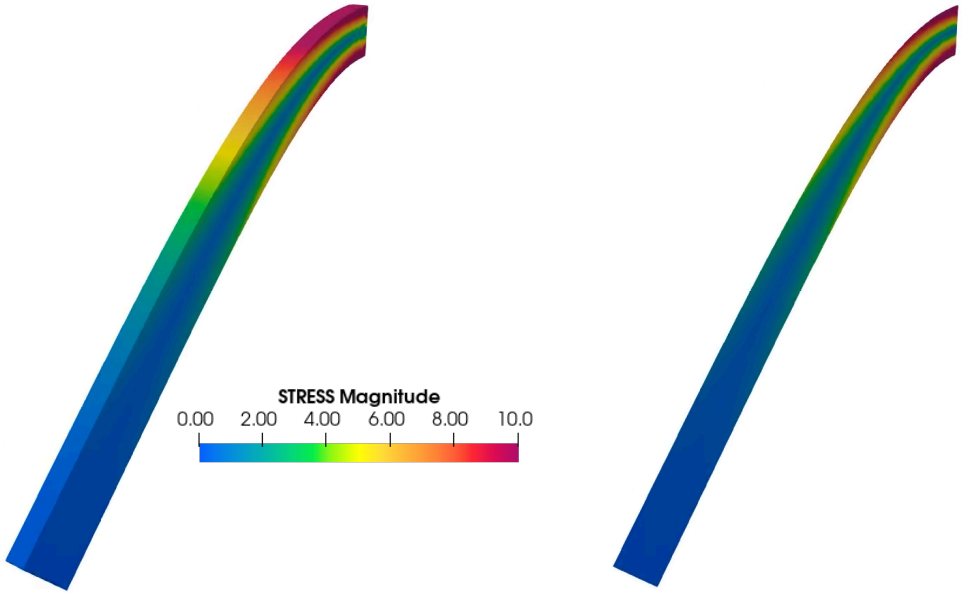
Results: 2D and 3D

```
constexpr int SPACE_DIM = 3; //< Space dimension of problem, mesh
```

Execute program:

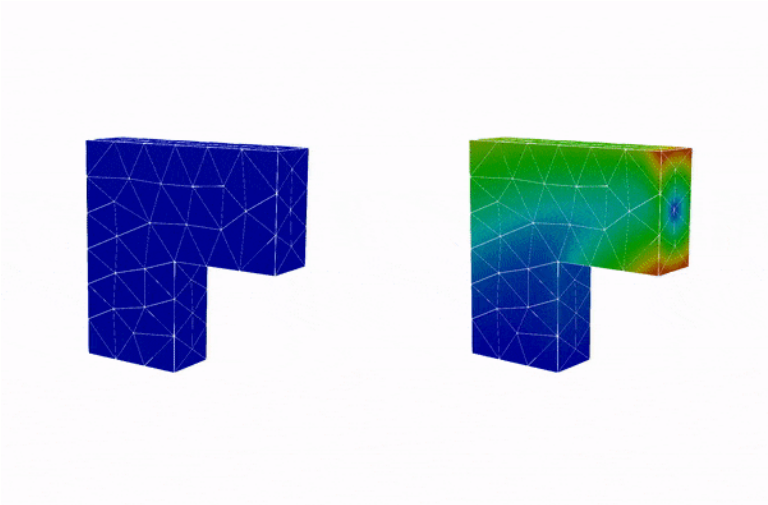
```
./elastic -file_name beam_3D.cub -order 2
```

- Cantilever beam
- Uniform body force
- *mbconvert* tool
- *Paraview WarpByVector filter*



MOFEM
mofem.eng.gla.ac.uk

Nonlinear problem



Small strain theory cannot deal with large rotations!

Introduce Hencky measures [1]:

- Logarithmic strain

$$\mathbf{E} := \frac{1}{2}\ln\left(\mathbf{F}^{\mathrm{T}}\mathbf{F}\right)$$

- Work conjugate stress:

$$\mathbf{T} = \mathbb{D} : \mathbf{E}$$

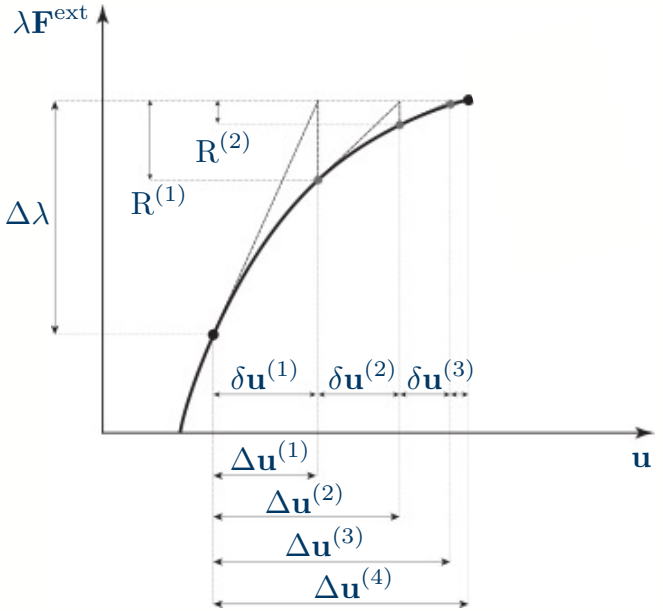
- Geometric postprocessor:

$$\mathbf{P} = \mathbf{F}\mathbf{T} : 2\partial_{\mathbf{C}}\mathbf{E}$$

[1] C. Miehe et. al (2002), *Computer Methods in Applied Mechanics and Engineering*,191(47-48):5383–5425, 2002

MOFEM
mofem.eng.gla.ac.uk

Nonlinear problem



The solution to a nonlinear problem will be obtained in increments:

$$\bar{\mathbf{u}}_{n+1} = \bar{\mathbf{u}}_n + \Delta\bar{\mathbf{u}}$$

System of previous linear equations is restated as:

$$\mathbf{F}^{\text{int}}(\bar{\mathbf{u}}) - \mathbf{F}^{\text{ext}} = \mathbf{0}$$

$$\mathbf{F}^{\text{int}}\left(\bar{\mathbf{u}}_0 + \delta\bar{\mathbf{u}}\right) = \mathbf{F}^{\text{int}}\left(\bar{\mathbf{u}}_0\right) + \left[\frac{\partial \mathbf{F}(\bar{\mathbf{u}})}{\partial \bar{\mathbf{u}}}\right] \cdot \delta\bar{\mathbf{u}}$$

$$\left[K_T\right]_{\bar{\mathbf{u}}_0} \delta\bar{\mathbf{u}} = \Delta\lambda \mathbf{F}^{\text{ext}} - \mathbf{F}^{\text{int}}(\bar{\mathbf{u}})$$

Implementation: Assemble

Pushing UDOs into pipelines

```
//! [Run problem]
MoFEMErrorCode Example::runProblem() {
  MoFEMFunctionBegin;
  CHKERR readMesh();
  CHKERR setupProblem();
  CHKERR createCommonData();
  CHKERR boundaryCondition();
  CHKERR assembleSystem();
  CHKERR solveSystem();
  CHKERR outputResults();
  CHKERR checkResults();
  MoFEMFunctionReturn(0);
}
```

Operators push both to RHS and LHS

```
auto add_domain_base_ops = [&](auto &pipeline) {
  MoFEMFunctionBegin;

  pipeline.push_back(new OpCalculateVectorFieldGradient<SPACE_DIM,
      SPACE_DIM>("U", matGradPtr));

  pipeline.push_back(
      new OpCalculateEigenVals<SPACE_DIM>("U",
      commonHenckyDataPtr));
  pipeline.push_back(
      new OpCalculateLogC<SPACE_DIM>("U", commonHenckyDataPtr));
  pipeline.push_back(
      new OpCalculateLogC_dC<SPACE_DIM>("U", commonHenckyDataPtr));
  pipeline.push_back(
      new OpCalculateHenckyStress<SPACE_DIM>("U",
      commonHenckyDataPtr));
  pipeline.push_back(
      new OpCalculatePiolaStress<SPACE_DIM>("U",
      commonHenckyDataPtr));

  MoFEMFunctionReturn(0);
};
```

MOFEM
mofem.eng.gla.ac.uk

Implementation: Assemble

```
F(i, j) = t_grad(i, j) + t_kd(i, j);
C(i, j) = F(k, i) ^ F(k, j);

// rare case when two eigen values are equal
auto nb_uniq = get_uniq_nb(eig);
if (DIM == 3 && nb_uniq == 2)
  sort_eigen_vals(eig, eigen_vec);

auto logC = EigenMatrix::getMat(eig, eigen_vec, f);

T(i, j) = t_D(i, j, k, l) * logC(k, l);

auto dlogC_dC = EigenMatrix::getDiffMat(eig, eigen_vec, f, d_f, nb_uniq);
dlogC_dC(i, j, k, l) *= 2;

S(k, l) = T(i, j) * dlogC_dC(i, j, k, l);
P(i, l) = F(i, k) * S(k, l);

FTensor::Tensor4<double, DIM, DIM, DIM, DIM> dC_dF;
dC_dF(i, j, k, l) = (t_kd(i, l) * F(k, j)) + (t_kd(j, l) * F(k, i));

auto TL = EigenMatrix::getDiffDiffMat(eig, eigen_vec, f, d_f, dd_f, T,
                                      nb_uniq);
TL(i, j, k, l) *= 4;
FTensor::Ddg<double, DIM, DIM> P_D_P_plus_TL;
P_D_P_plus_TL(i, j, k, l) =
      TL(i, j, k, l) +
      (dlogC_dC(i, j, o, p) * t_D(o, p, m, n)) * dlogC_dC(m, n, k, l);
P_D_P_plus_TL(i, j, k, l) *= 0.5;
dP_dF(i, j, m, n) = t_kd(i, m) * (t_kd(k, n) * S(k, j));
dP_dF(i, j, m, n) +=
      F(i, k) * (P_D_P_plus_TL(k, j, o, p) * dC_dF(o, p, m, n));
```

Strain measures:

$$\mathbf{F} = \nabla \mathbf{u} + \mathbf{I} \quad \mathbf{C} = \mathbf{F}^{\mathrm{T}}\mathbf{F}$$

Compute tensor function using MoFEM function:

$$\mathbf{E} = \frac{1}{2}\ln \mathbf{C} \quad \mathbf{T} = \mathbb{D} : \mathbf{E}$$

and derivatives:

$$\mathbf{S} = \mathbf{T} : \mathbb{P}_{\mathrm{L}} \quad \mathbb{P}_{\mathrm{L}} := 2\partial_{\mathbf{C}}\mathbf{E}$$

$$\mathbb{L}_{\mathrm{L}} = 4\partial_{\mathbf{C}\mathbf{C}}\mathbf{E}$$

Tangent operator for LHS:

$$\mathbb{C}^{\mathrm{ep}}_{\mathrm{L}} = \mathbb{P}_{\mathrm{L}} : \mathbb{D} : \mathbb{P}_{\mathrm{L}} + \mathbf{T} : \mathbb{L}_{\mathrm{L}}$$

$$\mathbb{C}^{\mathrm{ep}}_{\mathrm{N}} = \mathbb{I}\mathbf{S} + \mathbf{F}\mathbb{C}^{\mathrm{ep}}_{\mathrm{L}} : \partial_{\mathbf{C}}\mathbf{F}$$

Implementation: Assemble

Pushing UDOs into pipelines

```
//! [Run problem]
MoFEMErrorCode Example::runProblem() {
  MoFEMFunctionBegin;
  CHKERR readMesh();
  CHKERR setupProblem();
  CHKERR createCommonData();
  CHKERR boundaryCondition();
  CHKERR assembleSystem();
  CHKERR solveSystem();
  CHKERR outputResults();
  CHKERR checkResults();
  MoFEMFunctionReturn(0);
}
```

```
auto add_domain_ops_lhs = [&](auto &pipeline) {
  MoFEMFunctionBegin;

  pipeline.push_back(
      new OpHenckyTangent<SPACE_DIM>("U", commonHenckyDataPtr));
  pipeline.push_back(new OpK("U", "U", commonHenckyDataPtr->getMatTangent()));

  MoFEMFunctionReturn(0);
};
```

$$\mathbf{K}^e_{ij} = \int_{\Omega^e} \nabla\phi_i \mathbb{C}^{\mathrm{ep}}_{\mathrm{N}} \nabla\phi_j \,\mathrm{d}\Omega$$

```
auto add_domain_ops_rhs = [&](auto &pipeline) {
  MoFEMFunctionBegin;
  // Calculate internal forece
  pipeline.push_back(new OpInternalForce(
      "U", commonHenckyDataPtr->getMatFirstPiolaStress()));

  MoFEMFunctionReturn(0);
};
```

$$\mathbf{F}^e_i = \int_{\Omega^e} \nabla\phi_i \cdot \mathbf{P} \,\mathrm{d}\Omega$$

Implementation: Solve

```
//! [Run problem]
MoFEMErrorCode Example::runProblem() {
  MoFEMFunctionBegin;
  CHKERR readMesh();
  CHKERR setupProblem();
  CHKERR createCommonData();
  CHKERR boundaryCondition();
  CHKERR assembleSystem();
  CHKERR solveSystem();
  CHKERR outputResults();
  CHKERR checkResults();
  MoFEMFunctionReturn(0);
}
```

```
auto dm = simple->getDM();
auto ts = pipeline_mng->createTS();

 // Add monitor to time solver
auto monitor_ptr = boost::make_shared<Monitor>(dm, post_proc_fe);
CHKERR DMMoFEMTSSetMonitor(dm, ts, simple->getDomainFEName(), null_fe,
       null_fe, monitor_ptr);

auto D = smartCreateDMVector(simple->getDM());

CHKERR TSSetSolution(ts, D);
CHKERR TSSetFromOptions(ts);

CHKERR TSSolve(ts, NULL);
```

This snippet will be the similar for all nonlinear problems

$$[K_T]_{\bar{\mathbf{u}}_0}\,\delta\bar{\mathbf{u}} = \Delta\lambda\mathbf{F}^{\text{ext}} - \mathbf{F}^{\text{int}}(\bar{\mathbf{u}})$$

LHS pipeline

RHS pipeline

Results: Nonlinear 3D problem

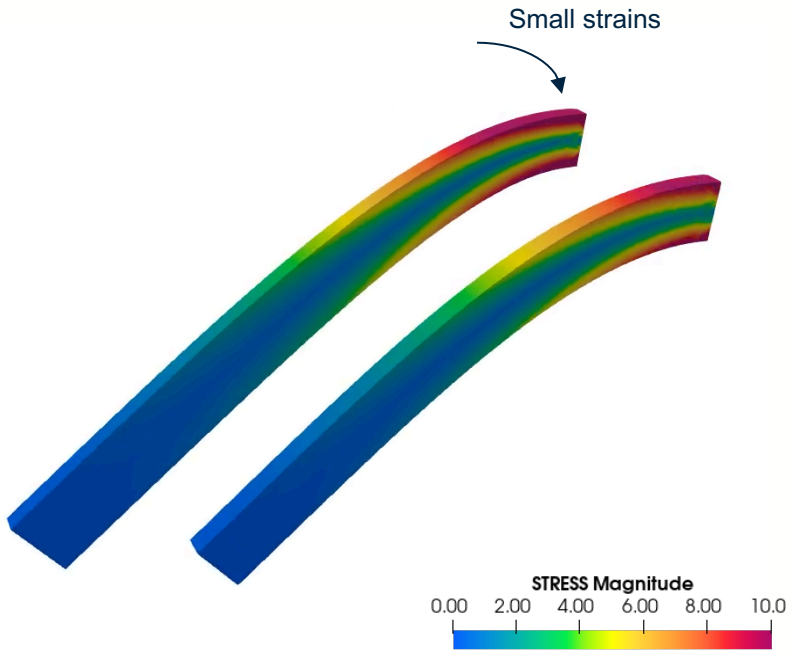
```
constexpr int SPACE_DIM = 3; //< Space dimension of problem, mesh
```

Execute program:

```
./nonlinear_elastic -file_name beam_3D.cub -order 2 -ts_dt 0.05
```

Program output for each load step $\Delta\lambda$:

```
[0] <inform> [petsc]    0 SNES Function norm 1.639129230990e-04
[0] <inform> [petsc]    1 SNES Function norm 1.459143968364e-02
[0] <inform> [petsc]    2 SNES Function norm 1.716350367415e-05
[0] <inform> [petsc]    3 SNES Function norm 9.827788747448e-07
[0] <inform> [petsc]    4 SNES Function norm 1.108872284616e-12
[0] <inform> [petsc] 10 TS dt 0.05 time 0.5
```



Verify consistency of tangent matrix

Test Jacobian matrix using finite difference method:

```
./nonlinear_elastic -file_name beam_2D.cub -order 2 -ts_dt 0.05
–snes_test_jacobian
```

$$\mathbf{K}^e_{ij} \approx \frac{\mathbf{F}^e_i(\bar{\mathbf{u}} + h\mathbf{e}_j) - \mathbf{F}^e_i(\bar{\mathbf{u}})}{h}$$

Absolute and relative error:

```
[0] <inform> [petsc]     ---------- Testing Jacobian -------------
[0] <inform> [petsc]     ||J - Jfd||_F/||J||_F = 5.22467e-07, ||J - Jfd||_F
= 0.00128756
```

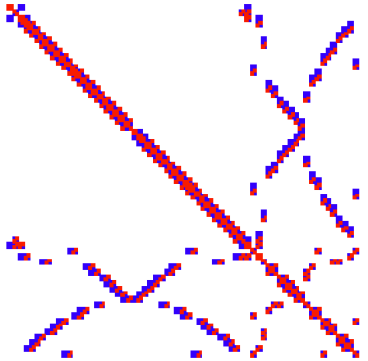
Plotting all entries for implemented matrix:

```
–snes_test_jacobian –snes_test_jacobian_display
```

```
[0] <inform> [petsc]     Hand-coded minus finite-difference Jacobian with tolerance 1e-05 ----------
[0] <inform> [petsc]   Mat Object: 1 MPI processes
[0] <inform> [petsc]     type: mpiaij
[0] <inform> [petsc] row 0: (0, 7.66377e-05)  (1, 4.4597e-05)  (4, 5.94409e-05)
[0] <inform> [petsc] row 1: (0, 4.66387e-05)  (1, 6.35757e-05)  (89, 4.46589e-05)
[0] <inform> [petsc] row 2: (2, 7.66377e-05)  (3, 4.45971e-05)  (86, 5.94409e-05)
[0] <inform> [petsc] row 3: (2, -4.66387e-05)  (3, -6.35735e-05)  (89, -4.46588e-05)
```

Visualising tangent matrix errors:

```
-snes_compare_explicit_draw -draw_save
```

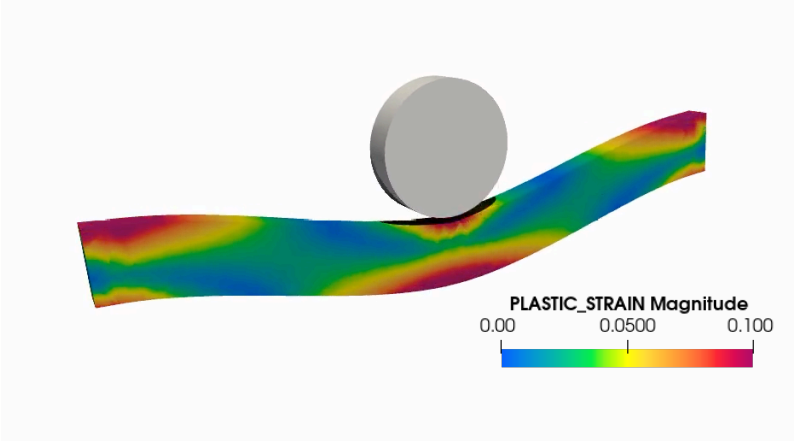


MOFEM
mofem.eng.gla.ac.uk

Summary

- Implementation of a vector valued problem
- Using operators from MoFEM library
- Solving nonlinear elasticity
- Tools for debugging tangent matrix

- Modular structure can be easily extended with:
  - Contact
  - *Plasticity*
  - *…*



Source code:

VEC-0 and VEC-2: http://mofem.eng.gla.ac.uk/mofem/html/tutorials.html


University of Glasgow
GCCC
Glasgow Computational Engineering Centre
MOFEM
mofem.eng.gla.ac.uk
Thank you for your attention!
WORLD CHANGING GLASGOW