Implementation in MoFEM:

A simple Poisson's problem

Hoang Nguyen,

Department of Mechanical and Construction Engineering, Northumbria University, Newcastle, UK

UKACM 2021 School

14th April 2021

MOFEM
mofem.eng.gla.ac.uk

Outline

- Goal of the presentation
- Theory
- Implementation
- Result
- Summary

SCL-1: http://mofem.eng.gla.ac.uk/mofem/html/tutorials.html

MOFEM
mofem.eng.gla.ac.uk

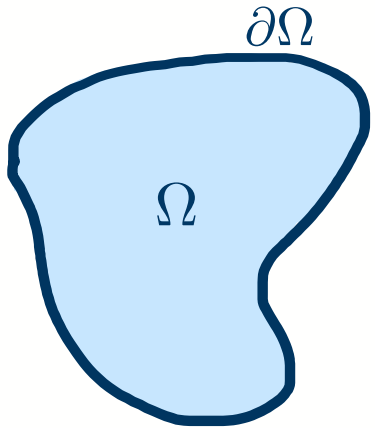
Goal of the presentation

- An introduction to implementation of a simple Poisson's problem in MoFEM
- Code structure
- Ideas of *Pipelines* and *User Data Operators (UDOs)*
- Implementation of UDOs

MOFEM
mofem.eng.gla.ac.uk

Theory

- Poisson's equation can be used to model different physical phenomena: steady-state diffusion, heat flow, electrostatics, etc.



- Strong form

$$\begin{cases} -\nabla \cdot \nabla u(\mathbf{x}) = f & \text{in } \Omega \\ \hat{u}(\mathbf{x}) = 0 & \text{on } \partial\Omega \end{cases}$$

- Weak form: Find $u \in H_0^1(\Omega)$

$$\int_\Omega \nabla \delta u \cdot \nabla u \, \mathrm{d}\Omega = \int_\Omega \delta u f \, \mathrm{d}\Omega, \quad \forall \delta u \in H_0^1(\Omega)$$

where space for test function δu is $H_0^1(\Omega) := \left\{ v \in H^1(\Omega) \mid v = 0 \text{ on } \partial\Omega \right\}$

Discretisation

- Approximation

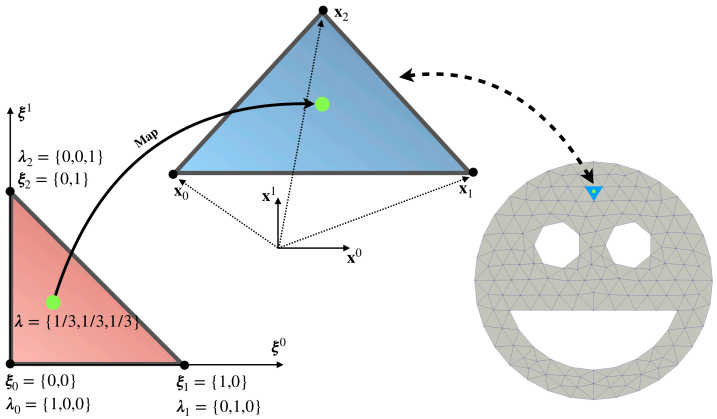
$$u \approx u^h = \sum_{j=0}^{N-1} \phi_j \bar{u}_j \qquad N \text{ : number of shape functions}$$

- Global system

$$\mathbf{KU} = \mathbf{F}$$

$$K_{ij}^e = \int_{\Omega^e} \nabla\phi_i \cdot \nabla\phi_j \, \mathrm{d}\Omega^e \approx \sum_g \nabla\phi_i(\mathbf{x}_g) \cdot \nabla\phi_j(\mathbf{x}_g) \, W_g \left\|\mathbf{J}_g^e\right\|$$

$$F_i^e = \int_{\Omega^e} \phi_i f \, \mathrm{d}\Omega^e \approx \sum_g \phi_i(\mathbf{x}_g) \, f(\mathbf{x}_g) \, W_g \left\|\mathbf{J}_g^e\right\|$$



FUN-1: Integration on finite element mesh - http://mofem.eng.gla.ac.uk/mofem/html/tutorials.html

Implementation: Read input mesh

```
MoFEMErrorCode Poisson2DHomogeneous::runProgram() {
  MoFEMFunctionBegin;

  CHKERR readMesh();
  CHKERR setupProblem();
  CHKERR boundaryCondition();
  CHKERR assembleSystem();
  CHKERR setIntegrationRules();
  CHKERR solveSystem();
  CHKERR outputResults();

  MoFEMFunctionReturn(0);
}
```

```
MoFEMErrorCode Poisson2DHomogeneous::readMesh() {
  MoFEMFunctionBegin;

  CHKERR mField.getInterface(simpleInterface);
  CHKERR simpleInterface->getOptions();
  CHKERR simpleInterface->loadFile();

  MoFEMFunctionReturn(0);
}
```

- Simple interface for fast implementation
- Get file name
- Load file
- Error handling

Implementation: Setup

```
MoFEMErrorCode Poisson2DHomogeneous::runProgram() {
  MoFEMFunctionBegin;

  CHKERR readMesh();
  CHKERR setupProblem();
  CHKERR boundaryCondition();
  CHKERR assembleSystem();
  CHKERR setIntegrationRules();
  CHKERR solveSystem();
  CHKERR outputResults();

  MoFEMFunctionReturn(0);
}
```

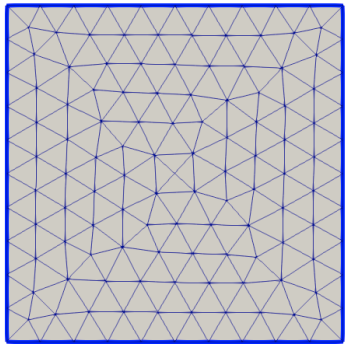
```
MoFEMErrorCode Poisson2DHomogeneous::setupProblem() {
  MoFEMFunctionBegin;

  CHKERR simpleInterface->addDomainField(domainField, H1,
                                         AINSWORTH_BERNSTEIN_BEZIER_BASE, 1);
  int oRder = 3;
  CHKERR PetscOptionsGetInt(PETSC_NULL, "", "-order", &oRder, PETSC_NULL);
  CHKERR simpleInterface->setFieldOrder(domainField, oRder);
  CHKERR simpleInterface->setUp();

  MoFEMFunctionReturn(0);
}
```

- Field name: domainField
- Space: H^1 (scalar space)
- Base: AINSWORTH_BERSTEIN_BEZIER_BASE
- #coefficients per shape function: 1 (scalar field)

Implementation: Boundary condition

```
MoFEMErrorCode Poisson2DHomogeneous::runProgram() {
  MoFEMFunctionBegin;

  CHKERR readMesh();
  CHKERR setupProblem();
  CHKERR boundaryCondition();
  CHKERR assembleSystem();
  CHKERR setIntegrationRules();
  CHKERR solveSystem();
  CHKERR outputResults();

  MoFEMFunctionReturn(0);
}
```

```
MoFEMErrorCode Poisson2DHomogeneous::boundaryCondition() {
  MoFEMFunctionBegin;

  // Get boundary edges marked in block named "BOUNDARY_CONDITION"
  Range boundary_entities;
  for (_IT_CUBITMESHSETS_BY_SET_TYPE_FOR_LOOP_(mField, BLOCKSET, it)) {
    std::string entity_name = it->getName();
    if (entity_name.compare(0, 18, "BOUNDARY_CONDITION") == 0) {
      CHKERR it->getMeshsetIdEntitiesByDimension(mField.get_moab(), 1,
                                                 boundary_entities, true);
    }
  }
  // Add vertices to boundary entities
  Range boundary_vertices;
  CHKERR mField.get_moab().get_connectivity(boundary_entities,
                                            boundary_vertices, true);
  boundary_entities.merge(boundary_vertices);

  // Remove DOFs as homogeneous boundary condition is used
  CHKERR mField.getInterface<ProblemsManager>()->removeDofsOnEntities(
      simpleInterface->getProblemName(), domainField, boundary_entities);

  MoFEMFunctionReturn(0);
}
```



Block name: BOUNDARY_CONDITION

MOAB (Mesh-Oriented datABase)

SCALABLE INTERFACES FOR GEOMETRY AND MESH BASED APPLICATIONS

Implementation: Assembly

```
MoFEMErrorCode Poisson2DHomogeneous::runProgram() {
  MoFEMFunctionBegin;

  CHKERR readMesh();
  CHKERR setupProblem();
  CHKERR boundaryCondition();
  CHKERR assembleSystem();
  CHKERR setIntegrationRules();
  CHKERR solveSystem();
  CHKERR outputResults();

  MoFEMFunctionReturn(0);
}
```

```
MoFEMErrorCode Poisson2DHomogeneous::assembleSystem() {
  MoFEMFunctionBegin;

  auto pipeline_mng = mField.getInterface<PipelineManager>();

  { // Push operators to the Pipeline for calculating LHS
    pipeline_mng->getOpDomainLhsPipeline().push_back(
        new OpCalculateInvJacForFace(invJac));
    pipeline_mng->getOpDomainLhsPipeline().push_back(
        new OpSetInvJacH1ForFace(invJac));

    pipeline_mng->getOpDomainLhsPipeline().push_back(
        new OpDomainLhsMatrixK(domainField, domainField));
  }

  { // Push operators to the Pipeline for calculating RHS
    pipeline_mng->getOpDomainRhsPipeline().push_back(
        new OpDomainRhsVectorF(domainField));
  }

  MoFEMFunctionReturn(0);
}
```

Pipelines UDOs

LhsPipeline

InvJac

LhsMat

$$K^e_{ij} = \int_{\Omega^e} \nabla\phi_i \cdot \nabla\phi_j \, \mathrm{d}\Omega^e$$

RhsPipeline

RhsVec

$$F^e_i = \int_{\Omega^e} \phi_i f \, \mathrm{d}\Omega^e$$

- User Data Operators (UDOs)
- Pipelines

Implementation: LHS pipeline

```
MoFEMErrorCode Poisson2DHomogeneous::assembleSystem() {
  MoFEMFunctionBegin;

  auto pipeline_mng = mField.getInterface<PipelineManager>();

  { // Push operators to the Pipeline for calculating LHS
    pipeline_mng->getOpDomainLhsPipeline().push_back(
        new OpCalculateInvJacForFace(invJac));
    pipeline_mng->getOpDomainLhsPipeline().push_back(
        new OpSetInvJacH1ForFace(invJac));

    pipeline_mng->getOpDomainLhsPipeline().push_back(
        new OpDomainLhsMatrixK(domainField, domainField));
  }

  { // Push operators to the Pipeline for calculating RHS
    pipeline_mng->getOpDomainRhsPipeline().push_back(
        new OpDomainRhsVectorF(domainField));
  }

  MoFEMFunctionReturn(0);
}
```

$$J^{-1} = \frac{\partial \xi_i}{\partial X_j}$$

$$(\nabla \phi)_j = \frac{\partial \phi}{\partial \xi_i} \frac{\partial \xi_i}{\partial X_j}$$

ξ_i : Parent coordinates

X_j : Global coordinates

MOFEM
mofem.eng.gla.ac.uk

Implementation: LHS pipeline

```
MoFEMErrorCode Poisson2DHomogeneous::assembleSystem() {
  MoFEMFunctionBegin;

  auto pipeline_mng = mField.getInterface<PipelineManager>();

  { // Push operators to the Pipeline for calculating LHS
    pipeline_mng->getOpDomainLhsPipeline().push_back(
        new OpCalculateInvJacForFace(invJac));
    pipeline_mng->getOpDomainLhsPipeline().push_back(
        new OpSetInvJacH1ForFace(invJac));

    pipeline_mng->getOpDomainLhsPipeline().push_back(
        new OpDomainLhsMatrixK(domainField, domainField));
  }

  { // Push operators to the Pipeline for calculating RHS
    pipeline_mng->getOpDomainRhsPipeline().push_back(
        new OpDomainRhsVectorF(domainField));
  }

  MoFEMFunctionReturn(0);
}
```

$$K_{ij}^e = \int_{\Omega^e} \nabla\phi_i \cdot \nabla\phi_j$$

$$\approx \sum_g \nabla\phi_i\left(\mathbf{x}_g\right) \cdot \nabla\phi_j\left(\mathbf{x}_g\right) W_g \left\|\mathbf{J}_g^e\right\|$$

```
struct OpDomainLhsMatrixK : public OpFaceEle {
public:
  OpDomainLhsMatrixK(std::string row_field_name, std::string col_field_name)
      : OpFaceEle(row_field_name, col_field_name, OpFaceEle::OPROWCOL) {
    sYmm = true;
  }
  ...
}
```

```
      // get element area
      const double area = getMeasure();
      // get number of integration points
      const int nb_integration_points = getGaussPts().size2();
      // get integration weights
      auto t_w = getFTensor0IntegrationWeight();
      // get derivatives of base functions on row
      auto t_row_diff_base = row_data.getFTensor1DiffN<2>();
      for (int gg = 0; gg != nb_integration_points; gg++) {
        const double a = t_w * area;
        for (int rr = 0; rr != nb_row_dofs; ++rr) {
          // get derivatives of base functions on column
          auto t_col_diff_base = col_data.getFTensor1DiffN<2>(gg, 0);
          for (int cc = 0; cc != nb_col_dofs; cc++) {
            locLhs(rr, cc) += t_row_diff_base(i) * t_col_diff_base(i) * a;
            ++t_col_diff_base;
          }
          ++t_row_diff_base;
        }
        ++t_w;
      }
```

MOFEM
mofem.eng.gla.ac.uk

Implementation: RHS pipeline

```
MoFEMErrorCode Poisson2DHomogeneous::assembleSystem() {
  MoFEMFunctionBegin;

  auto pipeline_mng = mField.getInterface<PipelineManager>();

  { // Push operators to the Pipeline for calculating LHS
    pipeline_mng->getOpDomainLhsPipeline().push_back(
        new OpCalculateInvJacForFace(invJac));
    pipeline_mng->getOpDomainLhsPipeline().push_back(
        new OpSetInvJacH1ForFace(invJac));

    pipeline_mng->getOpDomainLhsPipeline().push_back(
        new OpDomainLhsMatrixK(domainField, domainField));
  }

  { // Push operators to the Pipeline for calculating RHS
    pipeline_mng->getOpDomainRhsPipeline().push_back(
        new OpDomainRhsVectorF(domainField));
  }

  MoFEMFunctionReturn(0);
}
```

```
struct OpDomainRhsVectorF : public OpFaceEle {
public:
  OpDomainRhsVectorF(std::string field_name)
      : OpFaceEle(field_name, OpFaceEle::OPROW) {}
  ...
}
```

```
      // get element area
      const double area = getMeasure();
      // get number of integration points
      const int nb_integration_points = getGaussPts().size2();
      // get integration weights
      auto t_w = getFTensor0IntegrationWeight();
      // get base function
      auto t_base = data.getFTensor0N();

      for (int gg = 0; gg != nb_integration_points; gg++) {
        const double a = t_w * area;
        for (int rr = 0; rr != nb_dofs; rr++) {
          locRhs[rr] += t_base * body_source * a;
          ++t_base;
        }
        ++t_w;
      }
```

$$F_i^e = \int_{\Omega^e} \phi_i f$$

$$\approx \sum_g \phi_i(\mathbf{x}_g) f(\mathbf{x}_g) W_g \left\| \mathbf{J}_g^e \right\|$$

Implementation: Integration rules

```
MoFEMErrorCode Poisson2DHomogeneous::runProgram() {
  MoFEMFunctionBegin;

  CHKERR readMesh();
  CHKERR setupProblem();
  CHKERR boundaryCondition();
  CHKERR assembleSystem();
  CHKERR setIntegrationRules();
  CHKERR solveSystem();
  CHKERR outputResults();

  MoFEMFunctionReturn(0);
}
```

```
MoFEMErrorCode Poisson2DHomogeneous::setIntegrationRules() {
  MoFEMFunctionBegin;

  auto rule_lhs = [](int, int, int p) -> int { return 2 * (p - 1); };
  auto rule_rhs = [](int, int, int p) -> int { return 2 * p; };

  auto pipeline_mng = mField.getInterface<PipelineManager>();
  CHKERR pipeline_mng->setDomainLhsIntegrationRule(rule_lhs);
  CHKERR pipeline_mng->setDomainRhsIntegrationRule(rule_rhs);

  MoFEMFunctionReturn(0);
}
```

- Possible to set different integration rules for different operators

Implementation: Solve

```cpp
MoFEMErrorCode Poisson2DHomogeneous::runProgram() {
  MoFEMFunctionBegin;

  CHKERR readMesh();
  CHKERR setupProblem();
  CHKERR boundaryCondition();
  CHKERR assembleSystem();
  CHKERR setIntegrationRules();
  CHKERR solveSystem();
  CHKERR outputResults();

  MoFEMFunctionReturn(0);
}
```

```cpp
MoFEMErrorCode Poisson2DHomogeneous::solveSystem() {
  MoFEMFunctionBegin;

  auto pipeline_mng = mField.getInterface<PipelineManager>();
  auto ksp_solver = pipeline_mng->createKSP();
  CHKERR KSPSetFromOptions(ksp_solver);
  CHKERR KSPSetUp(ksp_solver);
  // Create RHS and solution vectors
  auto dm = simpleInterface->getDM();
  auto D = smartCreateDMVector(dm);
  auto F = smartVectorDuplicate(D);
  // Setup KSP solver
  CHKERR KSPSolve(ksp_solver, F, D);
  // Scatter result data on the mesh
  CHKERR VecGhostUpdateBegin(D, INSERT_VALUES, SCATTER_FORWARD);
  CHKERR VecGhostUpdateEnd(D, INSERT_VALUES, SCATTER_FORWARD);
  CHKERR DMoFEMMeshToLocalVector(dm, D, INSERT_VALUES, SCATTER_REVERSE);

  MoFEMFunctionReturn(0);
}
```

MOFEM
mofem.eng.gla.ac.uk

Implementation: Output results

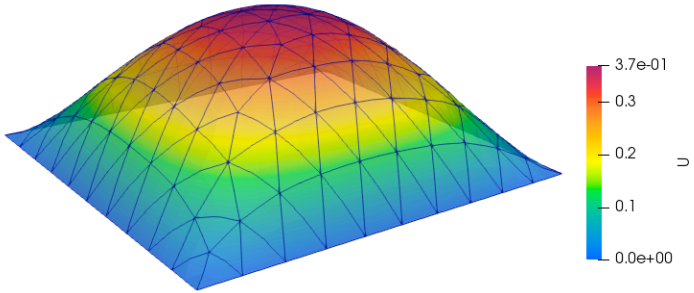
```
MoFEMErrorCode Poisson2DHomogeneous::runProgram() {
  MoFEMFunctionBegin;

  CHKERR readMesh();
  CHKERR setupProblem();
  CHKERR boundaryCondition();
  CHKERR assembleSystem();
  CHKERR setIntegrationRules();
  CHKERR solveSystem();
  CHKERR outputResults();

  MoFEMFunctionReturn(0);
}
```

```
MoFEMErrorCode Poisson2DHomogeneous::outputResults() {
  MoFEMFunctionBegin;

  auto pipeline_mng = mField.getInterface<PipelineManager>();
  pipeline_mng->getDomainLhsFE().reset();

  auto post_proc_fe = boost::make_shared<PostProcEle>(mField);
  post_proc_fe->generateReferenceElementMesh();
  post_proc_fe->addFieldValuesPostProc(domainField);
  pipeline_mng->getDomainRhsFE() = post_proc_fe;
  CHKERR pipeline_mng->loopFiniteElements();
  CHKERR post_proc_fe->writeFile("out_result.h5m");

  MoFEMFunctionReturn(0);
}
```

MOFEM
mofem.eng.gla.ac.uk

Result

- Compile code - *CMake*
- Input
  - Mesh - *Gmsh/Cubit*
  - Fixed boundary
  - Constant source term
- Convert output format - *mbconvert*
- Visualise result - *Paraview*



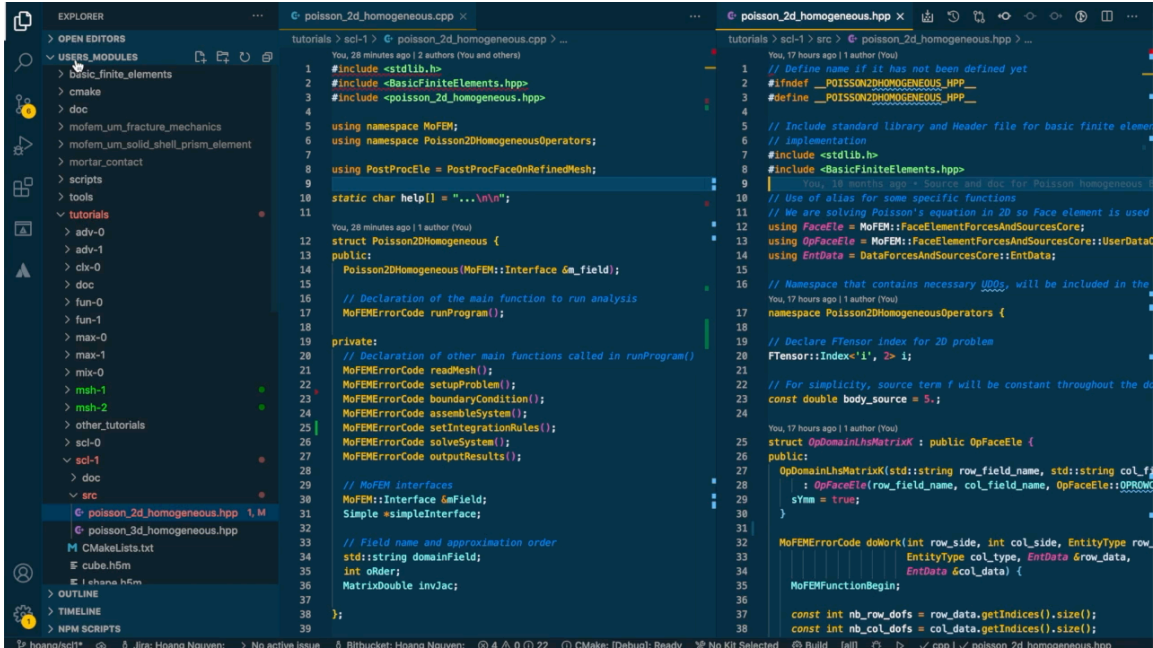
CMake

Gmsh

coreform Cubit™

SIGMA
SCALABLE INTERFACES FOR GEOMETRY AND MESH BASED APPLICATIONS

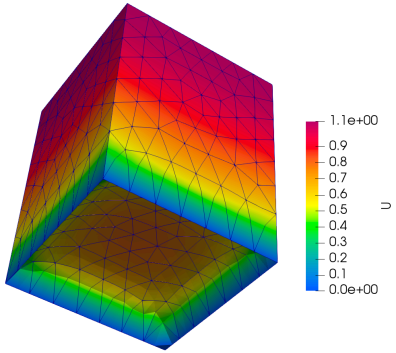
ParaView



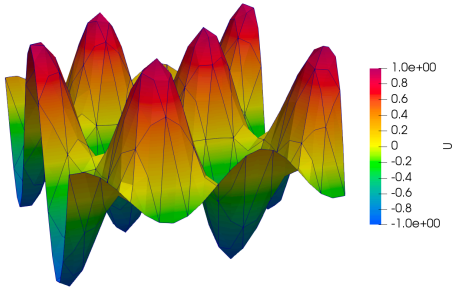
MOFEM
mofem.eng.gla.ac.uk

Result

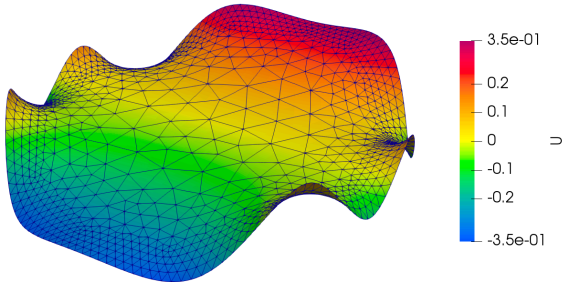
- Minimal effort to extend to 3D
- Extension is also possible for nonhomogeneous boundary conditions, nonlinear, time-dependent problems



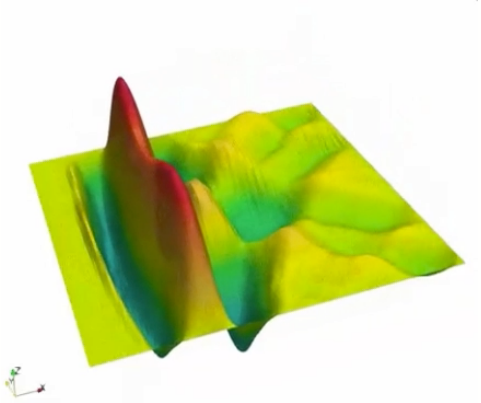
Poisson's problem in 3D



Nonhomogeneous BC



Nonlinear minimal surface equation



Wave equation

SCL-1 to SCL-7:
http://mofem.eng.gla.ac.uk/mofem/html/tutorials.html

MOFEM
mofem.eng.gla.ac.uk

Summary

- A general structure of a program developed in MoFEM
- Implementation of *User Data Operators*
- ’Push’ *UDOs* to *Pipelines*
- Postprocessing