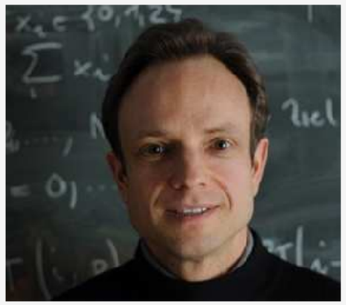
ETH Lecture 401-0674-00L Numerical Methods for Partial Differential Equations

Homework Problem Video Tutorial
Problem 2-10: Projection onto gradients

Prof. R. Hiptmair, SAM, ETH Zurich

Date: March 2, 2019



Needed: Lecture 2.7 ⇒ LF++
[In particular 2.7.4]

Also: Sect 2.4.5

$$u = \operatorname*{argmin}_{v\in H^1_0(\Omega)} J(v)\,, \quad J(v) := \int_\Omega \|\mathbf{f}(x) - \mathbf{grad}\, v(x)\|^2\,\mathrm{d}x\,. \tag{2.10.1}$$

a, Equivalent LVP

$$J(v) = \int_\Omega \|\operatorname{grad} v\|^2 - 2\mathbf{f}(x)\cdot\operatorname{grad} v + \|\mathbf{f}\|^2\,\mathrm{d}x$$

$\|\operatorname{grad} v\|^2 \rightarrow a(\cdot,\cdot)$, $2\mathbf{f}(x)\cdot\operatorname{grad} v \rightarrow \ell$

$$a(u,v) = 2\int_\Omega \operatorname{grad} u\cdot\operatorname{grad} v\,\mathrm{d}x\,,\quad \ell(v) = 2\int_\Omega \mathbf{f}\cdot\operatorname{grad} v\,\mathrm{d}x$$

$$u\in H^1_0(\Omega):\quad \int_\Omega \mathbf{grad}\, u(x)\cdot\mathbf{grad}\, v(x)\,\mathrm{d}x = \int_\Omega \mathbf{f}(x)\cdot\mathbf{grad}\, v(x)\,\mathrm{d}x \quad \forall v\in H^1_0(\Omega)\,. \tag{2.10.5}$$

b, Existence & Uniqueness

- By P.F.-ineq. $a(\cdot,\cdot)$ s.p.d.
- $\ell(\cdot)$ cont. w.r.t. $\|\cdot\|_a$ by Cauchy-Schwarz

$$|\ell(v)| = \left|\int_\Omega \mathbf{f}\cdot\operatorname{grad} v\,\mathrm{d}x\right| \le \|\mathbf{f}\|_{L^2}\cdot\|\operatorname{grad} v\|_{L^2}$$

$\le$ for $\mathbf{f}\in C^0(\Omega)$; $\|\operatorname{grad} v\|_{L^2} = \|v\|_a$

(2)

c) L^2-orthogonality of $\underline{f}_0 := \operatorname{grad} u$, $\underline{f}_1 := \underline{f} - \underline{f}_0$

$$\int_\Omega \underline{f}_0 \cdot \underline{f}_1 \, dx = 0 \quad [\leftarrow L^2(\Omega)\text{-inner product of } \underline{f}_0, \underline{f}_1]$$

$$= \int_\Omega \operatorname{grad} u \cdot (\underline{f} - \operatorname{grad} u) \, dx = 0$$

because of (2.10.5) with $v := u \in H^1_0(\Omega)$

d) For (2.10.5): $\operatorname{div} \underline{f} = 0 \Rightarrow u \equiv 0$
$\underline{f} \in C^1(\bar{\Omega})$

Hint: Green's formula for $\ell(\cdot)$

$$\ell(v) = \int_\Omega \underline{f} \cdot \operatorname{grad} v \, dx$$

$$= -\int_\Omega \operatorname{div} \underline{f} \cdot v \, dx + \int_{\partial\Omega} (\underline{f} \cdot \underline{n}) v \, dS$$

$= 0$, since $v \in H^1_0(\Omega)$

$$\triangleright \ \ell(v) = -\int_\Omega \operatorname{div} \underline{f} \cdot v \, dx$$

$\neq 0 \Leftrightarrow \operatorname{div} \underline{f} \neq 0$

For (2.10.5): $u = 0 \Leftrightarrow \ell = 0$

e) Use local midpoint rule $\forall$ integrals

$$\int_K \varphi(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \approx |K| \varphi(\boldsymbol{c}) \,, \tag{2.10.7}$$

where $\boldsymbol{c}$ is the center of gravity of the triangle K: $\boldsymbol{c} = \frac{1}{3}(\boldsymbol{a}^1 + \boldsymbol{a}^2 + \boldsymbol{a}^3)$, if K has the vertices $\boldsymbol{a}^i$, $i = 1, 2, 3$.

Implement a class **ElementMatrixProvider** compatible with the concept of **ENTITY_MATRIX_PROVIDER** as discussed in connection with [Lecture → Code 2.7.4.23] that computes the element matrices for the left-hand side bilinear form of the linear variational problem found in Sub-problem (2-10.a) based on linear Lagrangian finite elements on flat triangular cells.

C++11 code 2.10.8: Sub-problem (2-10.e): Builder class for element matrix → GITLAB

```
2 class ElementMatrixProvider {
3  private:
4   using coord_t = Eigen::Vector2d;
5
6  public:
7   Eigen::Matrix3d Eval(const lf::mesh::Entity &entity);
8   bool isActive(const lf::mesh::Entity &entity) const { return true; }
9 };
```

Comp. of elem. matrix A_K for triangle K

③

C++11 code 2.10.9: Computation of element matrix for Sub-problem (2-10.e) ➪ GITLAB

```
Eigen::Matrix3d ElementMatrixProvider::Eval(const lf::mesh::Entity &entity) {
  const lf::geometry::Geometry *geo_ptr = entity.Geometry();
  Eigen::Matrix3d loc_mat;

  /* BEGIN_SOLUTION */
  // get area of the entity
  const double area = lf::geometry::Volume(*geo_ptr);

  LF_ASSERT_MSG(lf::base::RefEl::kTria() == entity.RefEl(),
                "Function only defined for triangular cells");

  const Eigen::MatrixXd corners = lf::geometry::Corners(*geo_ptr);
  // calculate the gradients of the basis functions.
  // See [Lecture → Code 2.4.5.11], [Lecture → Code 2.4.5.13] for
     details.
  Eigen::Matrix3d grad_helper;
  grad_helper.block(0, 0, 3, 1) = Eigen::Vector3d::Ones();
  grad_helper.block(0, 1, 3, 2) = corners.transpose();
  // Matrix with gradients of the local shape functions in its columns
  const Eigen::MatrixXd grad_basis = grad_helper.inverse().block(1, 0, 2, 3);

  loc_mat = area * (grad_basis.transpose() * grad_basis);
  /* END_SOLUTION */
  return loc_mat;
}
```

$$A_K = |K| G^T G$$

f, Formula for element vector

From [Lecture → Code 2.4.5.11] we learned that it is easy to obtain a matrix $\mathbf{G} \in \mathbb{R}^{2,3}$, whose columns contain the gradients of the barycentric coordinate functions λ_ℓ, $\ell = 1,2,3$, for a triangle K.
Based on $\mathbf{G}$, find a formula for the entries of the element vector [Lecture → Def. 2.7.4.5] with respect to the finite element space $\mathcal{S}_1^0(\mathcal{M})$ for a general triangular cell K with a_K^i, $i = 1,2,3$. Employ (2.10.7). The vector field $\mathbf{f}$ and the area $|K|$ of the triangle may also be used in the expressions.

$$\ell(v) = \int_\Omega \underline{f}(x) \operatorname{grad} v(x) \, dx$$

Note: With local midpoint rule

$$(\vec{\varphi}_K)_\ell = \underline{f}(\underline{c}) \cdot \operatorname{grad} \lambda_\ell \cdot |K|$$

$$\underline{c} = \tfrac{1}{3}(a_K^1 + a_K^2 + a_K^3)$$

$$\Rightarrow \quad \vec{\varphi}_K = |K| \, G^T \cdot \underline{f}(\underline{c}) \qquad (*)$$

g, Realize a class **GradProjRhsProvider** according to the concept **ENTITY_VECTOR_PROVIDER** as introduced in [Lecture → Code 2.7.4.28].

```
template <typename FUNCTOR>
class GradProjRhsProvider {
public:
// Constructor:  initialization of functor data member
  explicit LinFEElemVecProvider(FUNCTOR f) : f_(f) {}
  virtual bool isActive(const lf::mesh::Entity & /*cell*/) { return
     true; }
  Eigen::Vector3d Eval(const lf::mesh::Entity &tria);
private:
  FUNCTOR f_; // Functor object for vector field f
};
```

Here **FUNCTOR** must provide an evaluation operator that returns an object that behaves like an **Eigen::Vector2d**.

C++11 code 2.10.11: Solution code for Sub-problem (2-10.g) ⇢ GITLAB

```
template <typename FUNCTOR>
Eigen::Vector3d GradProjRhsProvider<FUNCTOR>::Eval(
    const lf::mesh::Entity &entity) {
  Eigen::Vector3d loc_vec;

  const lf::geometry::Geometry *geo_ptr = entity.Geometry();

  /* BEGIN_SOLUTION */
  // get area of the entity
  const double area = lf::geometry::Volume(*geo_ptr);

  LF_ASSERT_MSG(lf::base::RefEl::kTria() == entity.RefEl(),
                "Function only defined for triangular cells");

  // calculate center of mass
  const Eigen::MatrixXd corners = lf::geometry::Corners(*geo_ptr);
  const coord_t c = corners.rowwise().sum() / 3.;

  // value of the functor f at center of mass of the entity
  const Eigen::Vector2d func_value = f_(c);

  // calculate the gradients of the basis functions
  Eigen::Matrix3d grad_helper;
  grad_helper.block(0, 0, 3, 1) = Eigen::Vector3d::Ones();
  grad_helper.block(0, 1, 3, 2) = corners.transpose();
  // vector of the gradients of the basis function evaluated at c
  const Eigen::MatrixXd grad_basis = grad_helper.inverse().block(1, 0, 2, 3);

  loc_vec = area * (grad_basis.transpose() * func_value);
  /* END_SOLUTION */

  return loc_vec;
}
```

$G \in \mathbb{R}^{2,3}$

h) Taking the cue from [Lecture → Code 2.7.6.19] implement a C++ function

```
termplate <typename FUNCTOR>
Eigen::VectorXd projectOntoGradients(const lf::assemble::DofHandler
   &dofhandler,
                                     FUNCTOR f);
```

which computes the vector of nodal basis expansion coefficients of the finite element Galerkin approximation $u_h \in \mathcal{S}^0_{1,0}(\mathcal{M})$ of the solution of (2.10.1).
The argument `dofhandler` passes a LEHRFEM++ **DofHandler** object, for details refer to [Lecture → § 2.7.4.13]. Through this object the mesh can be accessed as well. The other argument `f` contains a functor object providing **f** in procedural form.

solve discrete LVP using $S^0_{1,0}(\mathcal{M})$

∇ We have to impose homogeneous Dirichlet b.c.

C++11 code 2.10.12: Function `projectionOntoGradients` for Sub-problem (2-10.h) ➙ GITLAB

```
// ASSEMBLE GLOBAL MATRIX
/* BEGIN_SOLUTION */
lf::assemble::COOMatrix<double> A(N_dofs, N_dofs);
ProjectionOntoGradients::ElementMatrixProvider my_mat_provider;
// co-dimension 0 because we locally assemble on cells
lf::assemble::AssembleMatrixLocally(0, dofh, dofh, my_mat_provider, A);
/* END_SOLUTION */
// ASSEMBLE GLOBAL RHS VECOR
/* BEGIN_SOLUTION */
Eigen::VectorXd phi(N_dofs);
phi.setZero();
ProjectionOntoGradients::GradProjRhsProvider my_vec_provider(f);
// co-dimension 0 because we locally assemble on cells
lf::assemble::AssembleVectorLocally(0, dofh, my_vec_provider, phi);
/* END_SOLUTION */
// ENFORCE HOMOGENEOUS DIRICHLET BOUNDARY CONDITIONS
/* BEGIN_SOLUTION */
// We do this by selecting the DOFs on the boundary and setting the
// values to zero.  Note, that we could also use this to set the
   boundary
// to any other value (if bc were not homogeneous)

// To select the right DOFs we need a selector:
// * for all DOFs on the boundary return true and the value on the
   boundary
// * for all other DOFs return false (and whatever as value)
const double boundary_val = 0;
auto bd_flags{lf::mesh::utils::flagEntitiesOnBoundary(dofh.Mesh(), 2)};
auto my_selector = [&dofh, &bd_flags, &boundary_val](unsigned int dof_idx)
    {
  if (bd_flags(dofh.Entity(dof_idx))) {
    return (std::pair<bool, double>(true, boundary_val));
  } else {
    // the number we return here does not matter
    return (std::pair<bool, double>(false, 42));
  }
};
// Since we know the values on the boundary we know the solution on
   these
// DOFs and we can write the Galerkin LSE in block format and solve
   only
// for the unknown coefficients.  This modification is done by the
   following
// function fix_flagged_solution_components().  We use the selector
   we have
// defined above.
// See [Lecture → Section 2.7.6] for explanations.
lf::assemble::fix_flagged_solution_components<double>(my_selector, A, phi);
/* END_SOLUTION */
```


4) Unit test using Googletest

As we saw in Sub-problem (2-10.d), for the *divergence-free* vector field

$$\mathbf{f}(\boldsymbol{x}) = \begin{bmatrix} -x_2 \\ x_1 \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad \operatorname{div} \mathbf{f} \equiv 0, \tag{2.10.13}$$

we can expect $u_h \equiv 0$, where $u_h \in \mathcal{S}^0_{1,0}(\mathcal{M})$ is the finite element Ritz-Galerkin solution of (2.10.1). Using the facilities of GOOGLETEST implement a **unit test**

```
TEST(homework_pog,div_free_test) { ... }
```

that uses $\mathbf{f}$ from (2.10.13) to test the correct implementation of the function `projectOntoGradients()` from Sub-problem (2-10.h).

C++11 code 2.10.14: Unit test for Sub-problem (2-10.i) ➙ GITLAB

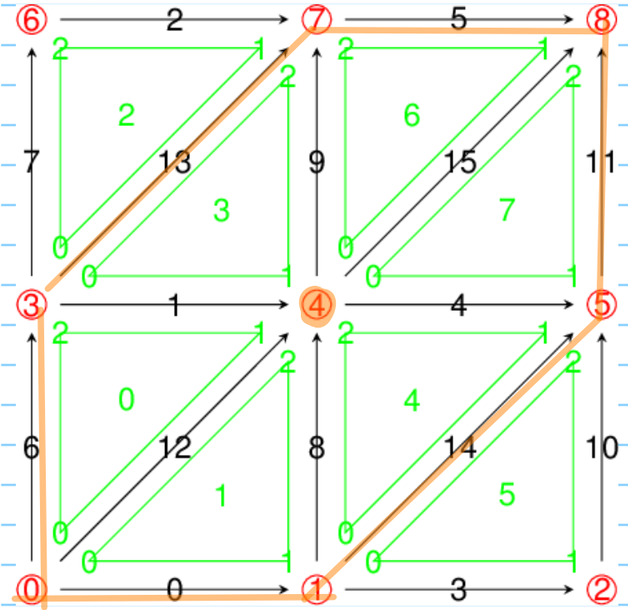
```
TEST(GradProjection, div_free_test) {
  /* BEGIN_SOLUTION */

  // Initialize all objects we need for our test case
  const auto f = [](Eigen::Vector2d x) -> Eigen::Vector2d {
    return Eigen::Vector2d(-x(1), x(0));
  };
  auto mesh_p = lf::mesh::test_utils::GenerateHybrid2DTestMesh(3);
  // DofHandler for the p.w.  linear Lagrangian finite element method
  lf::assemble::UniformFEDofHandler dofh(mesh_p,
                                         {{lf::base::RefEl::kPoint(), 1},
                                          {lf::base::RefEl::kSegment(), 0},
                                          {lf::base::RefEl::kTria(), 0}});
  // Compute solution
  const Eigen::VectorXd sol_vec =
      ProjectionOntoGradients::projectOntoGradients(dofh, f);

  // As stated in the exercise, we would expect the solution to be
     zero.
  // Hence we check every entry if its (numerically) zero.
  // The GoogleTest framework provides a function we may use:
  /*   EXPECT_NEAR(value 1, value 2, max. difference) */
  const double eps = 1e-15;
  for (std::size_t i = 0; i < sol_vec.size(); ++i) {
    EXPECT_NEAR(sol_vec[i], 0.0, eps);
    // Try testing for equality, you'll see it will fail miserably!
    /* EXPECT_EQ(sol_vec[i], 0.0); */
  }
  /* END_SOLUTION */
}
```

⑥

$\underline{f}$: $\mathbf{grad}\, u_h = \underline{f}$?

↳ If $\underline{f} \in \mathbf{grad}\, S^0_{1,0}(\mathcal{M})$

K, Unit test for $\underline{f} \in \mathbf{grad}\, S^0_{1,0}(\mathcal{M})$



$$\mathbf{f}(x) = \begin{cases} \begin{bmatrix} 2 & 0 \end{bmatrix}^\top & \text{, if } x \in K_0, \\ \begin{bmatrix} 0 & 2 \end{bmatrix}^\top & \text{, if } x \in K_1, \\ \begin{bmatrix} -2 & 2 \end{bmatrix}^\top & \text{, if } x \in K_4, \\ \begin{bmatrix} -2 & 0 \end{bmatrix}^\top & \text{, if } x \in K_7, \\ \begin{bmatrix} 0 & -2 \end{bmatrix}^\top & \text{, if } x \in K_6, \\ \begin{bmatrix} 2 & -2 \end{bmatrix}^\top & \text{, if } x \in K_3, \\ 0 & \text{elsewhere.} \end{cases} \qquad (2.10.16)$$

$\underline{f} = \mathbf{grad}\, b_h^4$

↑

Tent function at node 4.