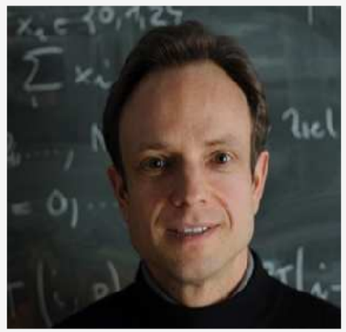
ETH Lecture 401-0674-00L Numerical Methods for Partial Differential Equations

Homework Problem Video Tutorial
Problem 2-15: Regularized Neumann Problem

Prof. R. Hiptmair, SAM, ETH Zurich

Date: March 18, 2019



[May not exactly correspond to the current version of the homework problem]

$$\underbrace{\quad}_{=:a(u,v)} \qquad \underbrace{\quad}_{=:\ell(v)}$$

$$u \in H^1(\Omega) \quad : \int_\Omega \mathbf{grad}\, u \cdot \mathbf{grad}\, v \,\mathrm{d}x = \int_\Omega fv \,\mathrm{d}x + \int_{\partial\Omega} hv \,\mathrm{d}S, \quad \forall v \in H^1(\Omega) \tag{2.15.1}$$

$f \in L^2(\Omega)$, $h \in L^2(\partial\Omega)$

a, Show (existence) & uniqueness of solutions in

$$H^1_*(\Omega) := \{v \in H^1(\Omega) : \int_\Omega v(\boldsymbol{x}) \,\mathrm{d}\boldsymbol{x} = 0\}. \tag{2.15.3}$$

Theorem 1.8.0.20. Second Poincaré-Friedrichs inequality

If $\Omega \subset \mathbb{R}^d$, $d \in \mathbb{N}$, is bounded, then

$$\exists C = C(\Omega) > 0: \quad \|u\|_0 \le C \operatorname{diam}(\Omega) \|\mathbf{grad}\, u\|_0 \quad \forall u \in H^1_*(\Omega)\,.$$

▷ $a(\cdot,\cdot)$ s.p.d. on $H^1_*(\Omega)$

▷ uniqueness of solution of

$$u \in H^1_*(\Omega) \quad : \int_\Omega \mathbf{grad}\, u \cdot \mathbf{grad}\, v \,\mathrm{d}x = \int_\Omega fv \,\mathrm{d}x + \int_{\partial\Omega} hv \,\mathrm{d}S, \quad \forall v \in H^1_*(\Omega) \tag{2.15.1}$$

Existence?

$H^1_*(\Omega)$ is a Hilbert space,

Theorem 1.3.3.6. Existence of minimizers in Hilbert spaces

On a real Hilbert space V_0 with (energy) norm $\|\cdot\|_a$ for any $\|\cdot\|_a$-bounded linear functional $\ell : V_0 \to \mathbb{R}$ the quadratic minimization problem

$$u_* = \operatorname*{argmin}_{v \in V_0} J(v)\,, \quad J(v) := \tfrac{1}{2}\|v\|_a^2 - \ell(v)\,, \tag{1.3.3.7}$$

has a unique solution.

(2) b) Let $u \in H^1_*(\Omega)$ solve

$$u \in H^1_*(\Omega) \quad : \int_\Omega \mathbf{grad}\, u \cdot \mathbf{grad}\, v \,\mathrm{d}x = \int_\Omega f v \,\mathrm{d}x + \int_{\partial\Omega} h v \,\mathrm{d}S, \quad \forall v \in H^1_*(\Omega) \qquad (2.15.1)$$

When does u solve:

$$u \in H^1(\Omega) \quad : \int_\Omega \mathbf{grad}\, u \cdot \mathbf{grad}\, v \,\mathrm{d}x = \int_\Omega f v \,\mathrm{d}x + \int_{\partial\Omega} h v \,\mathrm{d}S, \quad \forall v \in H^1(\Omega) \qquad (2.15.1)$$

(II)

Note: $H^1(\Omega) = H^1_*(\Omega) + \mathrm{Span}\{x \mapsto 1\}$

→ Need (II) to hold for $v = 1$

$$f \to f - c \quad \text{for} \quad c = \frac{1}{|\Omega|}\left(\int_\Omega f \,\mathrm{d}x + \int_{\partial\Omega} h \,\mathrm{d}S\right).$$

↳ $\ell(\{x \mapsto 1\}) = 0$

C++11 code 2.15.5: Implementation of `getGalerkinLSE`

```
template <typename FUNCT_F, typename FUNCT_H>
std::pair<Eigen::SparseMatrix<double>, Eigen::VectorXd> getGalerkinLSE(
    const std::shared_ptr<lf::uscalfe::ScalarUniformFESpace<double>>
        fe_space,
    const FUNCT_F &f, const FUNCT_H &h) {
  const lf::assemble::DofHandler &dofh{fe_space->LocGlobMap()};

  const std::size_t N_dofs = dofh.NoDofs();
  // Right-hand-side vector; don't forget to set to zero initially!
  Eigen::VectorXd rhs_vec(N_dofs);
  rhs_vec.setZero();

  // ASSEMBLE MATRIX
  lf::assemble::COOMatrix<double> A_aux(N_dofs, N_dofs);
  lf::uscalfe::LinearFELaplaceElementMatrix my_mat_provider{};
  // co-dimension 0 because we locally assemble on cells
  lf::assemble::AssembleMatrixLocally(0, dofh, dofh, my_mat_provider, A_aux);

  // ASSEMBLE GLOBAL RHS VECOR
  // Volume part v ↦ ∫_Ω f v dx of the rhs linear
  // functional
  lf::uscalfe::ScalarLoadElementVectorProvider my_vec_provider(fe_space, f);
  // co-dimension 0 because we locally assemble on cells
  lf::assemble::AssembleVectorLocally(0, dofh, my_vec_provider, rhs_vec);

  // Boundary part v ↦ ∫_∂Ω h v dS of the rhs
  // linear functional.  Only edges on the boundary should be included
    in
  // assembly, which is achieved by passing a suitable selector
    predicate to the
  // builder object through a lambda function
  auto bd_edges{lf::mesh::utils::flagEntitiesOnBoundary(dofh.Mesh(), 1)};
  lf::uscalfe::ScalarLoadEdgeVectorProvider my_vec_provider_edge(
      fe_space, h, [&bd_edges](const lf::mesh::Entity &edge) -> bool {
        return bd_edges(edge);
      });
  // co-dimension 1 because we locally assemble on edges !
  lf::assemble::AssembleVectorLocally(1, dofh, my_vec_provider_edge,
    rhs_vec);

  return std::make_pair(A_aux.makeSparse(), rhs_vec);
}
```

③

c, Drop first d.o.f. from $S_1^0(\mathcal{M})$:

↪ Setting corresp. exp. coeff $= 0$

C++11 code 2.15.7: Sub-problem (2-15.c): Setting a single d.o.f. to zero ⇢ GITHUB

```
auto selector = [](lf::base::glb_idx_t idx) -> std::pair<bool, double> {
  if (idx == 0) {
    return {true, 0.0};  // fix first d.o.f. to zero
  } else {
    return {false, 42.0};  // keep all others
  }
};
lf::assemble::fix_flagged_solution_components(selector, A_aux, rhs_vec);
```

(modified)

full Galerkin matrix ▷ $\begin{bmatrix} a_{11} & * \\ * & \widetilde{\mathbf{A}} \end{bmatrix} \begin{bmatrix} \mu_1 \\ \widetilde{\mu} \end{bmatrix} = \begin{bmatrix} \varphi_1 \\ \widetilde{\boldsymbol{\varphi}} \end{bmatrix}$,

into

Modified system ▷ $\begin{bmatrix} a_{11} & \mathbf{0}^\top \\ \mathbf{0} & \widetilde{\mathbf{A}} \end{bmatrix} \begin{bmatrix} \mu_1 \\ \widetilde{\mu} \end{bmatrix} = \begin{bmatrix} 0 \\ \widetilde{\boldsymbol{\varphi}} \end{bmatrix}$.

d, Why regular?

$\mathcal{N}(A) = \operatorname{Span}\{1\}$. If $\mu_1 = 0$ the only "constant vector" remaining $= 0$.

e, Use $H^1_*(\Omega)$ in FE context.

Bad news: $H^1_*(\Omega) \cap S_1^0(\mathcal{M})$ does not have a localized basis

Idea: Augment LVP by constraint

$u \in H^1(\Omega)$ ↪ Lagrange multiplier $\lambda \in \mathbb{R}$

$$\begin{cases} \int_\Omega \mathbf{grad}\, u \cdot \mathbf{grad}\, v \,\mathrm{d}x + \lambda \int_\Omega v \,\mathrm{d}x = \int_\Omega f v \,\mathrm{d}x + \int_{\partial\Omega} h v \,\mathrm{d}S \quad \forall v \in H^1(\Omega)\,, \\ \int_\Omega u \,\mathrm{d}x = 0\,. \end{cases} \tag{2.15.8}$$

↳ constraint

$$u \in H^1_*(\Omega) \;:\; \int_\Omega \mathbf{grad}\, u \cdot \mathbf{grad}\, v \,\mathrm{d}x = \int_\Omega f v \,\mathrm{d}x + \int_{\partial\Omega} h v \,\mathrm{d}S, \quad \forall v \in H^1_*(\Omega) \tag{2.15.1}$$

- u solves (8) ⇒ u solves (1):

  ↳ $u \in H^1_*(\Omega)$ by [redacted], $v \in H^1_*(\Omega) \Rightarrow \lambda$ can be dropped

- u solves (1) ⇒ u solves (8):

  [redacted] v, $\lambda = \frac{1}{|\Omega|}\left(\int_\Omega f \,\mathrm{d}x + \int_{\partial\Omega} h \,\mathrm{d}S\right)$

  → testing with constants ok

(4)

f, Galerkin matrix for

$$\begin{cases} \int_\Omega \mathbf{grad}\, u \cdot \mathbf{grad}\, v \,\mathrm{d}\boldsymbol{x} + \lambda \int_\Omega v \,\mathrm{d}\boldsymbol{x} = \int_\Omega f v \,\mathrm{d}\boldsymbol{x} + \int_{\partial\Omega} h v \,\mathrm{d}S \quad \forall v \in H^1(\Omega)\,, \\ \int_\Omega u \,\mathrm{d}\boldsymbol{x} = 0\,. \end{cases} \tag{2.15.8}$$

replace w/ $\mathcal{S}_1^0(\mathcal{M})$

Basis $\{b_h^1, \ldots, b_h^N, 1\}$ for Galerkin disc.

$$\tilde{\mathbf{A}}\vec{\tilde{\mu}} = \vec{\tilde{\varphi}}, \tag{2.15.10}$$

with the Galerkin matrix and right hand side vector augmented as follows:

$$\tilde{\mathbf{A}} := \begin{bmatrix} \mathbf{A} & \mathbf{c} \\ \mathbf{c}^T & 0 \end{bmatrix} \in \mathbb{R}^{N+1,N+1},$$

$$\vec{\tilde{\mu}} = \begin{bmatrix} \mu \\ \lambda \end{bmatrix} \in \mathbb{R}^{N+1}, \quad \vec{\tilde{\varphi}} = \begin{bmatrix} \varphi \\ 0 \end{bmatrix} \in \mathbb{R}^{N+1},$$

and where $\mathbf{c} \in \mathbb{R}^N$ has the components

$$(\mathbf{c})_j = \int_\Omega 1 \cdot b_h^j(\boldsymbol{x}) \,\mathrm{d}\boldsymbol{x}\,. \tag{2.15.11}$$

g,

Based on LEHRFEM++'s function `lf::assemble::AssembleVectorLocally()` implement a routine (`regNeumann.h`)

```
Eigen::VectorXd assembleVector_c(const lf::assemble::DofHandler
    &dofh);
```

that computes the vector $\mathbf{c} \in \mathbb{R}^N$ from (2.15.9). The `dofh` argument passes the local-to-global index mapping and mesh information for the finite element space $\mathcal{S}_1^0(\mathcal{M})$.
As an auxliary class that provides the **ENTITY_VECTOR_PROVIDER** implement

```
class VecHelper {
 public:
  explicit VecHelper() {}
  bool isActive(const lf::mesh::Entity &entity) const;
  Eigen::Vector3d Eval(const lf::mesh::Entity &entity);
};
```

$$c_j = \frac{1}{3} \sum_{K:\mathbf{p}_j \in K} |K|\,, \tag{2.15.13}$$

↳ element vector $|K|/3 \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

C++11 code 2.15.14: VecHelper ⇝ GITHUB

```
class VecHelper {
 public:
  explicit VecHelper() {}
  bool isActive(const lf::mesh::Entity &entity) const { return true; }
  Eigen::Vector3d Eval(const lf::mesh::Entity &entity) {
    LF_ASSERT_MSG(lf::base::RefEl::kTria() == entity.RefEl(),
                  "Function only defined for triangular cells");
    Eigen::Vector3d result;
    /* BEGIN_SOLUTION */
    // Obtain shape information for the cell
    const lf::geometry::Geometry *geo_ptr = entity.Geometry();
    // Fetch area
    const double area = lf::geometry::Volume(*geo_ptr);
    // Initialize element vector |K|/3*[1,1,1]^T
    result = (area / 3.0) * Eigen::Vector3d::Ones();
    /* END_SOLUTION */
    return result;
  }
};
```

h,

Modify the function `getGalerkinLSE` from Code 2.15.5 into

```
template <typename FUNCT_F, typename FUNCT_H>
std::pair<Eigen::SparseMatrix<double>, Eigen::VectorXd>
  getGalerkinLSE_augment(
    const std::shared_ptr<lf::uscalfe::UniformScalarFESpace<double>>
      fe_space,
    const FUNCT_F &f, const FUNCT_H &h);
```

such that it returns the Galerkin matrix and right hand side vector of (2.15.8) discretized by means of piecewise linear Lagrangian finite elements.

(5)

Augmented Galerkin matrix $\begin{bmatrix} A & C \\ C^T & 0 \end{bmatrix}$

C++11 code 2.15.16: Sub-problem (2-15.h): Excerpt of `getGalerkinLSE_augment()`
➙ GITHUB

```
2  // Calculate vector c
3  Eigen::VectorXd c = assembleVector_c(dofh);
4  // Add c to the matrix
5  for (int i = 0; i < c.size(); i++) {
6    // add triplets for vector c
7    A_aux.AddToEntry(N_dofs - 1, i, c(i));
8    A_aux.AddToEntry(i, N_dofs - 1, c(i));
9  }
```