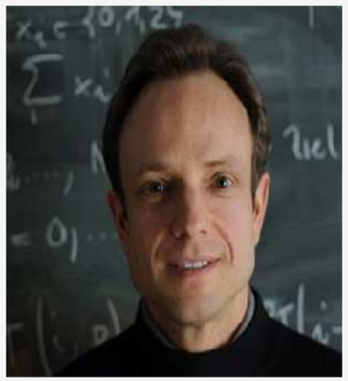
ETH Lecture 401-0674-00L Numerical Methods for Partial Differential Equations

Homework Problem Video Tutorial

Problem 3-4: A Boundary Value Problem Modelling Stationary Heat Conduction

Prof. R. Hiptmair, SAM, ETH Zurich

Date: March 29, 2019



c) Remember the use of offset functions for imposing $\neq 0$ Dirichlet boundary conditions

For $S_1^0(\mathcal{M})$: $\mu_{0,h} = \sum_{x \in \mathcal{N}_h \cap \partial\Omega} g(x)\, b_h^x$ (supported near $\partial\Omega$)

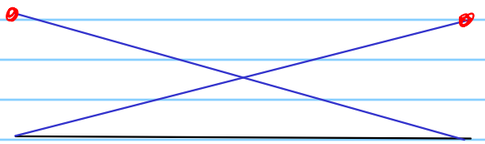
double

Study the documentation of the LehrFEM++ function `lf::uscalfe::InitEssentialCondition` and explain the purpose of its template parameters and arguments.

```
template <typename SCALAR, typename EDGESELECTOR, typename FUNCTION>
std::vector<std::pair<bool, SCALAR>>
   InitEssentialConditionFromFunction(
     const lf::assemble::DofHandler &dofh,
     const ScalarReferenceFiniteElement<SCALAR> &fe_spec_edge,
     EDGESELECTOR &&esscondflag, FUNCTION &&g);
```

predicate for selection of relevant edge

$S_1^0(\mathcal{M})$-LSF for edge:



(2) d_1

C++ code 3.4.1: Sub-problem (3-4.d): Implementation of `solveTemperatureDistribution()`, more copiously commented version ⇢ GITLAB

```
double solveTemperatureDistribution(
  // Dirichlet data
    std::shared_ptr<const lf::mesh::Mesh> mesh_p) {
  auto bc = [](Eigen::Vector2d x) -> double {
    return x[1] <= 0 ? 1 - x[1] : 0; };
  lf::uscalfe::MeshFunctionGlobal mf_bc{bc};
  auto fe_space =
      std::make_shared<lf::uscalfe::FeSpaceLagrangeO1<double>>(mesh_p);
  const lf::mesh::Mesh& mesh{*(fe_space->Mesh())};
  const lf::assemble::DofHandler& dofh{fe_space->LocGlobMap()};
  const size_type N_dofs(dofh.NoDofs());
  // Matrix in triplet format holding Galerkin matrix, zero initially.
  lf::assemble::COOMatrix<double> A(N_dofs, N_dofs);
  // Element matrix builder for the negative Laplacian
  lf::uscalfe::LinearFELaplaceElementMatrix elmat_builder{};
  // Cell-oriented assembly
  lf::assemble::AssembleMatrixLocally(0, dofh, dofh, elmat_builder, A);
  // Right-hand-side vector
  Eigen::Matrix<double, Eigen::Dynamic, 1> phi(N_dofs); phi.setZero();
  // Impose Dirichlet boundary conditions
  std::shared_ptr<const lf::uscalfe::ScalarReferenceFiniteElement<double>>
      rsf_edge_p =
          fe_space->ShapeFunctionLayout(lf::base::RefEl::kSegment());
  auto bd_flags{lf::mesh::utils::flagEntitiesOnBoundary(fe_space->Mesh(),
      1)};
  auto ess_bdc_flags_values{lf::uscalfe::InitEssentialConditionFromFunction(
      dofh, *rsf_edge_p,
      [&bd_flags](const lf::mesh::Entity& edge) -> bool {
        return (bd_flags(edge)); }, mf_bc)};
  lf::assemble::fix_flagged_solution_components<double>(
      [&ess_bdc_flags_values](glb_idx_t gdof_idx) {
        return ess_bdc_flags_values[gdof_idx]; },  A, phi);
  // Solve linear system of equations
  Eigen::SparseMatrix<double> A_crs = A.makeSparse();
  Eigen::SparseLU<Eigen::SparseMatrix<double>> solver;
  solver.compute(A_crs);
  LF_VERIFY_MSG(solver.info() == Eigen::Success, "LU decomposition failed");
  Eigen::VectorXd sol_vec = solver.solve(phi);
  LF_VERIFY_MSG(solver.info() == Eigen::Success, "Solving LSE failed");
  // Compute H^1(Ω)-semi-norms
  lf::uscalfe::MeshFunctionL2GradientDifference loc_comp(
      fe_space, lf::uscalfe::MeshFunctionConstant(Eigen::Vector2d(0.0, 0.0)),
      2);
  return lf::uscalfe::NormOfDifference(dofh, loc_comp, sol_vec);
}
```

$\rightarrow V_{0,h} = S_1^0(\mathcal{M})$

$a(u,v) = \int_\Omega \operatorname{grad} u \cdot \operatorname{grad} v \, dx$

$\rightarrow -\Delta$

$\rightarrow \vec{\varphi} = 0$

Detailed Description

Computing the element matrix for the (negative) Laplacian and linear finite elements.

The main purpose of this class is to compute the element matrix for the Laplacian on affine triangles or bilinearly mapped quadrilaterals. These element matrices are provided by the `Eval()` method.

Note

the `Eval()` method will always return a *reference* to a 4x4 matrix also for triangles. In this case the last row and column must be ignored.

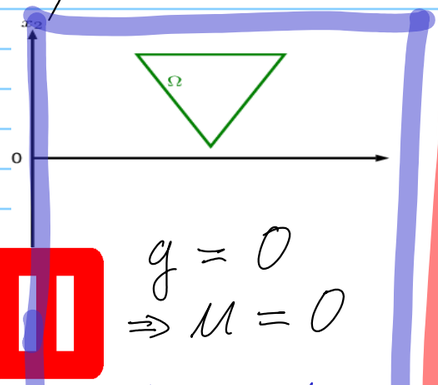
This class complies with the requirements for the type `ENTITY_MATRIX_PROVIDER` given as a template parameter to define an incarnation of the function AssembleMatrixLocally().

BVP strong form:

$$-\Delta u = 0$$

$$u = g, \quad g(x) = \begin{cases} 0, & x_2 > 0 \\ 1 - x_2, & x_2 \le 0 \end{cases}$$

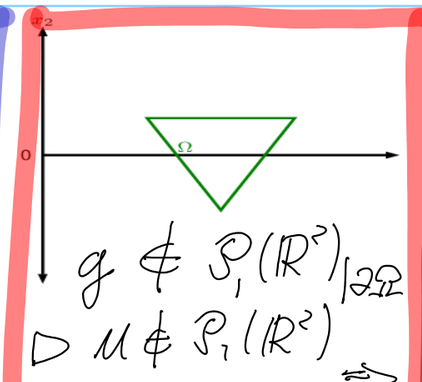
e_1



$g = 0$

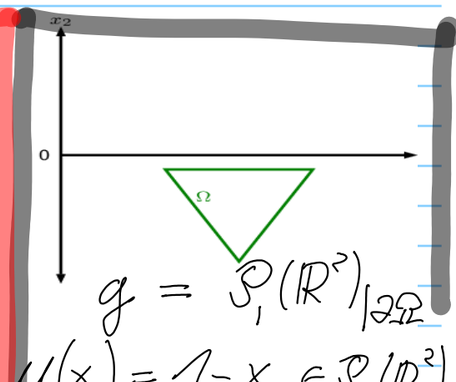
$\Rightarrow u = 0$

$u_h = u$



$g \notin \mathcal{P}_1(\mathbb{R}^2)|_{\partial\Omega}$

$\triangleright\ u \notin \mathcal{P}_1(\mathbb{R}^2)$



$g = \mathcal{P}_1(\mathbb{R}^2)|_{\partial\Omega}$

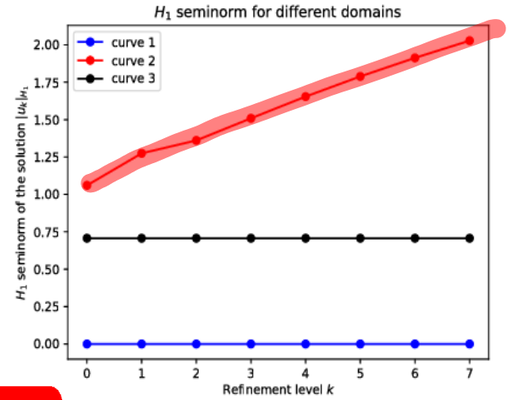
$\Rightarrow u(x) = 1 - x_2 \in \mathcal{P}_1(\mathbb{R}^2)$

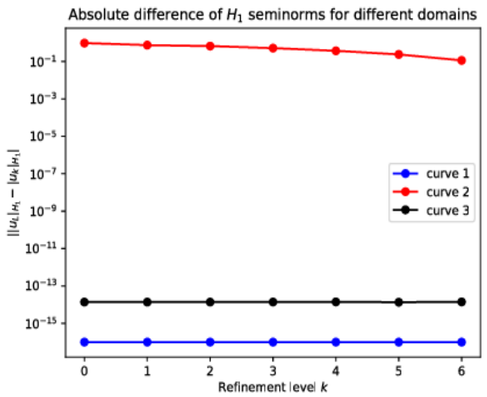
$u_h = u$

Note: $\Delta w = 0 \quad \forall w \in \mathcal{P}_1(\mathbb{R}^2)$

③ f)

For the three different domains shown introduced in Sub-problem (3-4.e), we compute the finite element solutions u_k on sequences of regularly refined triangular meshes $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_L$ and plot $k \mapsto |u_k|_{H^1(\Omega)}$ in Fig. 23 and $k \mapsto |u_k|_{H^1(\Omega)} - |u_L|_{H^1(\Omega)}$ in Fig. 24, $k = 1, \ldots, L-1$.





$|u_k|_1 \to \infty$ for $k \to \infty$

[$|u_k|_1 \approx C + k$]

g) For the middle domain: g discontinuous

⇓

There is no $u \in H^1(\Omega)$: $u|_{\partial\Omega} = g$

▷ No (weak) solution of BVP exists.

FEM sees (for $k \to \infty$) an "ever steeper" g