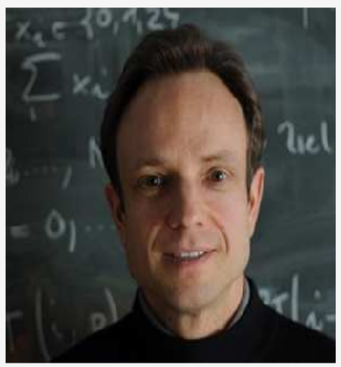
ETH Lecture 401-0674-00L Numerical Methods for Partial Differential Equations

Homework Problem Video Tutorial

Problem 2-13: Local computations for parametric Lagrangian finite elements

Prof. R. Hiptmair, SAM, ETH Zurich

Date: March 18, 2019



LVP:

$$\underbrace{\int_\Omega (\mathbf{I} + \boldsymbol{d}(\boldsymbol{x})\boldsymbol{d}(\boldsymbol{x})^\top)\,\mathbf{grad}\, u \cdot \mathbf{grad}\, v\,\mathrm{d}\boldsymbol{x} + \int_{\partial\Omega} w^2(\boldsymbol{x}) u(\boldsymbol{x})\, v(\boldsymbol{x})\,\mathrm{d}S(\boldsymbol{x})}_{\int_\Omega \frac{v(\boldsymbol{x})}{1+w(\boldsymbol{x})^2}\,\mathrm{d}\boldsymbol{x}} \quad \forall v \in H^1(\Omega)\,, \quad (2.13.1)$$

a, Strong form BVP:

I. Test with $v \in C_0^\infty(\Omega)$ + i.b.v $\Rightarrow$ PDE

$$-\operatorname{div}\left(\mathbf{I} + d(x)d(x)^T\right)\operatorname{grad} u = \frac{1}{(1+w(x))^2} \text{ in } \Omega$$

II. Test with general $v \in C^\infty(\overline{\Omega})$ + i.b.d. + use PDE

▷ Imp. b.c. $\operatorname{grad} u \cdot n + w(x)^2 u = 0$ on $\partial\Omega$

Now: $w \in S_1^0(\mathcal{M}) \leftrightarrow \vec{v} \in \mathbb{R}^N$, $N = \#\mathcal{V}(\mathcal{M})$

+ use edge-midpoint QR

(2)

b)

Implement a class **AnisotropicDiffusionElementMatrixProvider** (file ansiotropic_diffusion_element_matrix_provider.cc)

```
class AnisotropicDiffusionElementMatrixProvider {
  public:
  using elem_mat_t = Eigen::Matrix<SCALAR, Eigen::Dynamic,
    Eigen::Dynamic>;
  using ElemMat = const elem_mat_t;
  using vectorfield_t =
    std::function<Eigen::Vector2d(Eigen::Vector2d)>;

  AnisotropicDiffusionElementMatrixProvider(vectorfield_t Vf_d);
  bool isActive(const lf::mesh::Entity &) { return true; }
  ElemMat Eval(const lf::mesh::Entity &cell);
private:
  vectorfield_t Vf_d_;
};
```

compliant with LEHRFEM++'s concept of an **ENTITY_MATRIX_PROVIDER** is discussed in [Lecture → Ex. 2.7.4.29]. Its `Eval()` member function is to compute (based on (2.13.2)) the $\mathcal{S}_1^0(\mathcal{M})$-element matrix for the bilinear form $(u,v) \mapsto \int_\Omega (\mathrm{I} + d(x)d(x)^\top)\,\mathbf{grad}\,u \cdot \mathbf{grad}\,v\,\mathrm{d}x$.
Exclusively use functions of the `lf::mesh` and `lf::geometry` layer of LEHRFEM++, no built-in quadrature rules.

anisotropic diffusion tensor $\in \mathbb{R}^{2,2}$

Parametric FEM policy

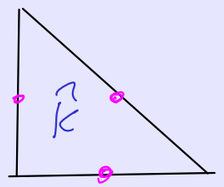
$$(\mathbf{A}_K)_{ij} = \int_{\widehat{K}} (\alpha(\Phi(\widehat{x}))(\mathrm{D}\Phi)^{-\top}\,\mathbf{grad}\,\widehat{b}^i) \cdot ((\mathrm{D}\Phi)^{-\top}\,\mathbf{grad}\,\widehat{b}^j)\,|\det \mathrm{D}\Phi|\,\mathrm{d}\widehat{x}$$
$$= \int_{\widehat{K}} ((\mathrm{D}\Phi)^{-1}\alpha(\Phi(\widehat{x}))(\mathrm{D}\Phi)^{-\top})\,\mathbf{grad}\,\widehat{b}^i \cdot \mathbf{grad}\,\widehat{b}^j\,|\det \mathrm{D}\Phi|\,\mathrm{d}\widehat{x}\,. \quad (2.8.3.11)$$

```
Eigen::MatrixXd lf::geometry::Geometry::JacobianInverseGramian(const
  Eigen::MatrixXd &local) const;
```

→ Q,R: $\int_K \varphi(x)\,\mathrm{d}x \approx \frac{|K|}{\sharp\mathcal{V}(K)} \sum_{\ell=1}^{\sharp\mathcal{V}(K)} \varphi(\boldsymbol{m}_K^\ell)\,, \quad K \in \mathcal{M}\,.$ (2.13.2)

→ 1/6 for triangle $\widehat{K}$

C++11 code 2.13.3: Sub-problem (2-13.b): Implementation of `Eval()` member function of AnisotropicDiffusionElementMatrixProvider *for triangles* → GITHUB




```
2   result = Eigen::MatrixXd::Zero(3, 3);
3   // midpoints of the edges of the reference triangle
4   Eigen::Matrix<double, 2, 3> midpoints_loc(2, 3);
5   midpoints_loc << 0.5, 0.5, 0, 0, 0.5, 0.5;
6   // obtain global midpoints
7   auto midpoints_glob = geom->Global(midpoints_loc);
8
9   // get jacobian and determinants (scaling)
10  const Eigen::MatrixXd JinvT(geom->JacobianInverseGramian(midpoints_loc));
11  const Eigen::VectorXd
        determinants(geom->IntegrationElement(midpoints_loc));
12  // local gradients
13  Eigen::Matrix<double, 2, 3> grads_loc(2, 3);
14  grads_loc << -1, 1, 0, -1, 0, 1;
15  // loop over quadrature points
16  for (int i = 0; i < 3; i++) {
17    // prepare extra factor matrix
18    Eigen::Vector2d d_x = AnisotropicDiffusionElementMatrixProvider::Vf_d_(
19        midpoints_glob.col(i));
20    Eigen::Matrix2d factor_matrix =
21        Eigen::Matrix2d::Identity(2, 2) + d_x * d_x.transpose();
22    // compute global gradients
23    Eigen::MatrixXd grad_glob(2, 3);
24    grad_glob = JinvT.block(0, 2 * i, 2, 2) * grads_loc;
25    // add contributions to element matrix
26    for (int j = 0; j < 3; j++) {
27      for (int k = 0; k < 3; k++) {
28        double tmp =
29            grad_glob.col(j).transpose() * factor_matrix *
                grad_glob.col(k);
30        result(j, k) += determinants(i) * tmp;
31      }
32    }
33  }
34  // multiply with area divided by 6 (3 quad points and RefEl has area 0.5)
35  result *= 1. / 6.;
```

$\alpha(x)$

$D\Phi^{-T}\,\mathbf{grad}\,\widehat{b}^i$

③ c/ Local computations related to finite element Galerkin right-hand-side vectors are usually farmed out to objects matching the LEHRFEM++ concept of **ENTITY_VECTOR_PROVIDER**, see [Lecture → Ex. 2.7.4.29], [Lecture → Code 2.7.4.32]. Implement a corresponding class (file `fe_source_elem_vec_provider.cc`)

```
class FESourceElemVecProvider {
 public:
  using elem_vec_t = Eigen::Matrix<SCALAR, Eigen::Dynamic, 1>;
  using ElemVec = const elem_vec_t;

  FESourceElemVecProvider(
      lf::uscalfe::UniformScalarFESpace<double> &lin_Lagr_fe_space,
      Eigen::VectorXd w_coeff_vec);
  bool isActive(const lf::mesh::Entity &) { return true; }
  ElemVec Eval(const lf::mesh::Entity &cell);
 private:
  lf::uscalfe::UniformScalarFESpace<double> &lin_Lagr_fe_space_;
  Eigen::VectorXd w_coeff_vec_;
);
```

whose `Eval()` member functions returns the element load vector for the right-hand-side functional of (2.13.1) discretized on $\mathcal{S}_1^0(\mathcal{M})$ equipped with the customary tent-function global shape functions. The constructor accepts an object encoding the finite element space $\mathcal{S}_1^0(\mathcal{M})$, see [Lecture → § 2.8.3.27] and the basis expansion coefficient vector `w_coeff_vec` for the function w. Again, use the local quadrature rule (2.13.2).

QR: $$\int_K \varphi(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \approx \frac{|K|}{\sharp\mathcal{V}(K)} \sum_{\ell=1}^{\sharp\mathcal{V}(K)} \varphi(\boldsymbol{m}_K^\ell)\,, \quad K \in \mathcal{M}\,. \tag{2.13.2}$$

Need w at m_K^ℓ → [$w \in \mathcal{S}_1^0(\mathcal{M})$] by taking arithmetic mean

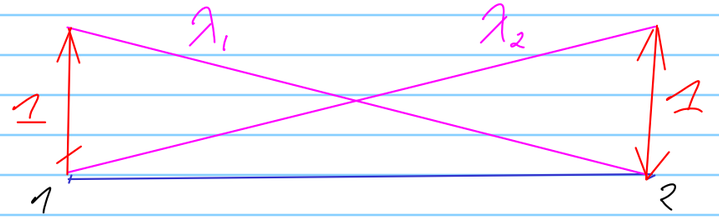
d/ Compute the $\mathcal{S}_1^0(\mathcal{M})$-element matrix spawned by the bilinear form

$$(u,v) \mapsto \int_{\partial\Omega} w^2(x)u(x)\,v(x)\,\mathrm{d}S(x)\,, \quad u,v \in H^1(\Omega)\,,$$

for a *straight* edge of $\mathcal{M}$ located on $\partial\Omega$, *cf.* [Lecture → Ex. 2.5.3.7]. Use that $w \in \mathcal{S}_1^0(\mathcal{M})$ and express the entries of the element matrix in terms of the values w_1 and w_2 of w in the endpoints of the edge.

▷ Assembly on edges :

Recall: LSFs on edges



▷ $w_{|e} = w_1\lambda_1 + w_2\lambda_2$

Lemma 2.7.5.5. Integration of powers of barycentric coordinate functions

For any non-degenerate d-simplex K with barycentric coordinate functions $\lambda_1,\ldots,\lambda_{d+1}$ and exponents $\alpha_j \in \mathbb{N}$, $j = 1,\ldots,d+1$,

$$\int_K \lambda_1^{\alpha_1}\cdot\cdots\cdot\lambda_{d+1}^{\alpha_{d+1}}\,\mathrm{d}\boldsymbol{x} = d!|K|\,\frac{\alpha_1!\alpha_2!\cdot\cdots\cdot\alpha_{d+1}!}{(\alpha_1+\alpha_2+\cdots+\alpha_{d+1}+d)!} \quad \forall\boldsymbol{\alpha} \in \mathbb{N}_0^{d+1}\,. \tag{2.7.5.6}$$

Apply for $d = 1$

$$\int_e \lambda_1^p\lambda_2^q\,\mathrm{d}\boldsymbol{x} = |e|\frac{p!q!}{(1+p+q)!}\,, \quad p,q \in \mathbb{N}_0\,.$$

$$\mathbf{B}_e = \begin{bmatrix} \int_e (w_1\lambda_1 + w_2\lambda_2)^2\lambda_1^2\,\mathrm{d}S & \int_e (w_1\lambda_1 + w_2\lambda_2)^2\lambda_1\lambda_2\,\mathrm{d}S \\ \int_e (w_1\lambda_1 + w_2\lambda_2)^2\lambda_1\lambda_2\,\mathrm{d}S & \int_e (w_1\lambda_1 + w_2\lambda_2)^2\lambda_2^2\,\mathrm{d}S \end{bmatrix}$$
$$= w_1^2 \cdot \begin{bmatrix} \int_e \lambda_1^4\,\mathrm{d}S & \int_e \lambda_1^3\lambda_2\,\mathrm{d}S \\ \int_e \lambda_1^3\lambda_2\,\mathrm{d}S & \int_e \lambda_1^2\lambda_2^2\,\mathrm{d}S \end{bmatrix} +$$
$$2w_1w_2 \cdot \begin{bmatrix} \int_e \lambda_1^3\lambda_2\,\mathrm{d}S & \int_e \lambda_1^2\lambda_2^2\,\mathrm{d}S \\ \int_e \lambda_1^2\lambda_2^2\,\mathrm{d}S & \int_e \lambda_1^1\lambda_2^3\,\mathrm{d}S \end{bmatrix} +$$
$$w_2^2 \cdot \begin{bmatrix} \int_e \lambda_1^2\lambda_2^2\,\mathrm{d}S & \int_e \lambda_1\lambda_2^3\,\mathrm{d}S \\ \int_e \lambda_1\lambda_2^3\,\mathrm{d}S & \int_e \lambda_2^4\,\mathrm{d}S \end{bmatrix}. \tag{2.13.7}$$

$$\mathbf{B}_e = \frac{|e|}{120}\left(w_1^2 \begin{bmatrix} 24 & 6 \\ 6 & 4 \end{bmatrix} + w_1w_2 \begin{bmatrix} 6 & 4 \\ 4 & 6 \end{bmatrix} + w_2^2 \begin{bmatrix} 4 & 6 \\ 6 & 24 \end{bmatrix} \right). \tag{2.13.8}$$

e, Implement a class (file `impedance_boundary_edge_matrix_provider.cc`)

```
class ImpedanceBoundaryEdgeMatrixProvider {
 public:
  using elem_mat_t = Eigen::Matrix<double, Eigen::Dynamic,
     Eigen::Dynamic>;
  using ElemMat = const elem_mat_t;

  ImpedanceBoundaryEdgeMatrixProvider(
    lf::uscalfe::UniformScalarFESpace<double> &lin_Lagr_fe_space,
    Eigen::VectorXd w_coeff_vec);
  bool isActive(const lf::mesh::Entity &);
  ElemMat Eval(const lf::mesh::Entity &cell);
 private:
  lf::uscalfe::UniformScalarFESpace<double> &lin_Lagr_fe_space_;
  Eigen::VectorXd w_coeff_vec_;
  std::shared_ptr<lf::mesh::utils::CodimMeshDataSet<bool>> bd_flags_;
};
```

fitting LEHRFEM++'s concept of **ENTITY_MATRIX_PROVIDER** that computes the edge element matrix derived in Sub-problem (2-13.d) using the result of `lf::geometry::Volume` as a substitute for the length $|e|$ of the edge. The arguments for the constructor are the same as those for the constructor of the class **FESourceElemVecProvider** from Sub-problem (2-13.c).
Note that this time you also have to supply a non-trivial `isActive()` method that returns true, when given an edge on the boundary.

C++11 code 2.13.9: Sub-problem (2-13.e): Constructor of ImpedanceBoundaryEdgeMatrixProvider ➪ GITHUB

```
ImpedanceBoundaryEdgeMatrixProvider::ImpedanceBoundaryEdgeMatrixProvider(
    std::shared_ptr<lf::uscalfe::UniformScalarFESpace<double>>
        lin_Lagr_fe_space,
    Eigen::VectorXd w_coeff_vec) {
  ImpedanceBoundaryEdgeMatrixProvider::lin_Lagr_fe_space_ =
      lin_Lagr_fe_space;
  ImpedanceBoundaryEdgeMatrixProvider::w_coeff_vec_ = w_coeff_vec;
  // initialize boundary flags for isActive() function
  auto mesh = lin_Lagr_fe_space->Mesh();
  ImpedanceBoundaryEdgeMatrixProvider::bd_flags_ =
      std::make_shared<lf::mesh::utils::CodimMeshDataSet<bool>>(
          lf::mesh::utils::flagEntitiesOnBoundary(mesh, 1));
}
```

f, Unit test by comparing with LF++ built-in PROVIDER classes

```
template <typename SCALAR, typename DIFF_COEFF, typename
    REACTION_COEFF>
 ReactionDiffusionElementMatrixProvider<SCALAR, DIFF_COEFF,
     REACTION_COEFF>::
  ReactionDiffusionElementMatrixProvider(
      std::shared_ptr<UniformScalarFESpace<SCALAR>> fe_space,
      DIFF_COEFF alpha, REACTION_COEFF gamma,
      quad_rule_collection_t qr_collection);

using quad_rule_collection_t = std::map<lf::base::RefEl,
    lf::quad::QuadRule>;
```

for $a(u,v) = \int_\Omega \alpha(x)\,\mathbf{grad}\,u \cdot \mathbf{grad}\,v + \gamma(x)\,u\,v\,\mathrm{d}x, \quad u,v \in H^1(\Omega),$ (2.8.3.29)

by $\gamma \equiv 0$

(5) Pass $\alpha(x)$ → MeshFunction

C++11 code 2.13.12: Sub-problem (2-13.f): Unit test for AnisotropicDiffusionElementMatrixProvider ⇒ GITHUB

```
TEST(ParametricElementMatrices, TestGalerkin) {
  /* SOLUTION_BEGIN */
  // use test mesh to set up fe space
  auto mesh = lf::mesh::test_utils::GenerateHybrid2DTestMesh(0, 1);
  auto fe_space =
      std::make_shared<lf::uscalfe::FeSpaceLagrangeO1<double>>(mesh);
  const lf::assemble::DofHandler &dofh{fe_space->LocGlobMap()};
  const lf::base::size_type N_dofs(dofh.NoDofs());

  // compute galerkin matrix for d(x) = (0,0)' using the implemented class
  lf::assemble::COOMatrix<double> A(N_dofs, N_dofs);
  auto d = [](Eigen::Vector2d x) -> Eigen::Vector2d {
    return Eigen::Vector2d::Zero();
  };
  auto elmat_builder =
      ParametricElementMatrices::AnisotropicDiffusionElementMatrixProvider(d);
  lf::assemble::AssembleMatrixLocally(0, dofh, dofh, elmat_builder, A);
  // compute galerkin matrix using ReactionDiffusionElementMatrixProvider
  lf::assemble::COOMatrix<double> B(N_dofs, N_dofs);
  auto identity = [](Eigen::Vector2d x) -> double { return 1.; };
  lf::uscalfe::MeshFunctionGlobal mf_identity{identity};
  auto zero = [](Eigen::Vector2d x) -> double { return 0.; };
  lf::uscalfe::MeshFunctionGlobal mf_zero{zero};
  // set up quadrature rule to be able to compare
  std::map<lf::base::RefEl, lf::quad::QuadRule> quad_rules{
      {lf::base::RefEl::kTria(), lf::quad::make_TriaQR_EdgeMidpointRule()},
      {lf::base::RefEl::kQuad(), lf::quad::make_QuadQR_EdgeMidpointRule()}};

  lf::uscalfe::ReactionDiffusionElementMatrixProvider<
      double, decltype(mf_identity), decltype(mf_zero)>
      elmat_builder_org(fe_space, mf_identity, mf_zero, quad_rules);
  lf::assemble::AssembleMatrixLocally(0, dofh, dofh, elmat_builder_org, B);

  auto A_crs = A.makeSparse();
  auto B_crs = B.makeSparse();
  // compare results (floating point comparison!)
  for (int i = 0; i < N_dofs; i++) {
    for (int j = 0; j < N_dofs; j++) {
      ASSERT_NEAR(A_crs.coeff(i, j), B_crs.coeff(i, j), 0.001);
    }
  }
  /* SOLUTION_END */
}
```

↑ floating point comparison

g, Test for r.h.s. vector

- ∇ Test conducted for linear $w \in S_1^0(\mathcal{M})$
  - o
- ∇ Conduct the test for a mesh with only affine cells
  - o