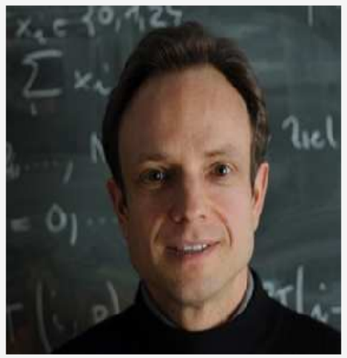
ETH Lecture 401-0674-00L Numerical Methods for Partial Differential Equations

Homework Problem Video Tutorial
Problem 3-3: Dirichlet BVP with Point-Evaluation Right-Hand-Side Functional

Prof. R. Hiptmair, SAM, ETH Zurich

Date: March 29, 2019



$$u \in H_0^1(\Omega): \quad \underbrace{\int_\Omega \mathbf{grad}\, u(x) \cdot \mathbf{grad}\, v(x)\, dx}_{a(u,v)} = \underbrace{v(0)}_{\ell(v)} \quad \forall v \in H_0^1(\Omega), \tag{3.3.1}$$

$0 \in \Omega$

↑ point-evaluation functional

BVP: $-\Delta u = \delta_0$ in Ω

$u = 0$ on $\partial\Omega$

a, Model for 2D membrane with point load

b, [Ex 1.2.3.48]

$v \to v(0)$ is not continuous on $H^1(\Omega)$ for $\Omega \subset \mathbb{R}^2$

Here: energy norm $\|\cdot\|_a = |\cdot|_{H^1(\Omega)}$

$\ell(\cdot)$ not continuous w.r.t. $\|\cdot\|_a$

(2)

c, $a(\cdot,\cdot)$ s.p.d.

↳ Galerkin matrix A s.p.d

↳ invertible

⇒ unique minimizer of $J(v) = \frac{1}{2}a(u,u) - \ell(u)$ on $S_1^0(\mathcal{M})$ exists.

d, Implement a function

```
Eigen::Vector2d GlobalInverseTria(
  Eigen::Matrix<double,2,3> vertices,
  Eigen::Vector2d x);
```

accepting the arguments

- `vertices`, a matrix whose columns store the coordinates of the vertices of a planar triangle K,
- `x`, the coordinates of a point $\boldsymbol{x}$ in the plane.

The function should return the coordinates of the pre-image $\widehat{\boldsymbol{x}}$ of $\boldsymbol{x}$ under the affine mapping $\boldsymbol{\Phi}_K : \widehat{K} \to K$, $\widehat{K}$ the usual "unit triangle".
The returned coordinates need not be located inside $\widehat{K}$, which indicates that $\boldsymbol{x} \notin K$.
For the sake of stable implementation use EIGEN's direct elimination solver for small dense matrices.

K triangle: $\Phi_K(\hat{x}) = A\hat{x} + t$, $A \in \mathbb{R}^{2,2}$ regular

$\Leftrightarrow \quad \Phi_K^{-1}(x) = A^{-1}(x - t)$

e, Implement a C++ function

```
std::pair<double, double> solveQuadraticEquation(
  double a, double b, double c);
```

that computes the real roots of the quadratic equation

$$a\xi^2 + b\xi + c = 0, \quad \triangleright \quad \xi^2 + p\xi + q = 0$$

and returns `NAN` ("Not a number", see `std::nan()`), in case none exist.

Remember that numerical stability is an issue when applying the standard formula for the roots of a parabola. Distinguish cases to achieve a cancellation-free implementation, see .

(I) $D := p^2 - 4q < 0 \Rightarrow$ no solution

(II) $\xi_1 := \begin{cases} \frac{1}{2}(-p - \sqrt{D}), & \text{if } p \geq 0 \\ \frac{1}{2}(-p + \sqrt{D}), & \text{if } p < 0 \end{cases}$

(III) $\xi_2 = q/\xi_1$

f, Implement a function

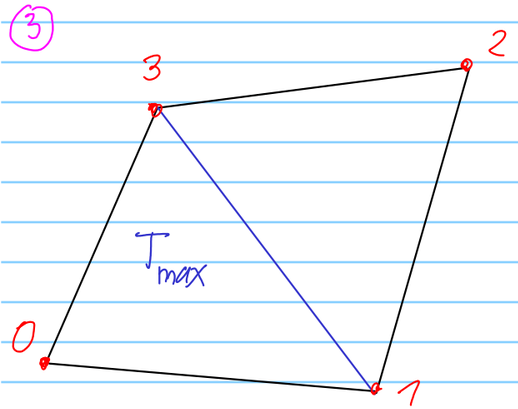
```
Eigen::Vector2d GlobalInverseQuad(
  Eigen::Matrix<double,2,4> vertices,
  Eigen::Vector2d x);
```

accepting the arguments

- `vertices`, a matrix passing the coordinates of the vertices of a planar quadrilateral K in its columns,
- `x`, the coordinates of a point $\boldsymbol{x}$ in the plane.

The function should return the coordinates of the pre-image $\widehat{\boldsymbol{x}}$ of $\boldsymbol{x}$ under the bilinear mapping $\boldsymbol{\Phi}_K : \widehat{K} \to K$, see [Lecture → § 2.8.2.2], where $\widehat{K}$ is the unit square.
The returned coordinates need not be located inside $\widehat{K}$, which indicates that $\boldsymbol{x} \notin K$. However, there can also be cases, where $\boldsymbol{x} \notin K$ does not have a pre-image. Then the function should terminate with an error.

(3)



$$\Phi_K(\hat{x}) = t + A\hat{x} + d\hat{x}_1\hat{x}_2\,, \quad \hat{x} = \begin{bmatrix}\hat{x}_1\\ \hat{x}_2\end{bmatrix}\,, \quad t \in \mathbb{R}^2\,, \quad \mathbf{A} \in \mathbb{R}^{2,2} \text{ regular}\,. \tag{3.3.4}$$

$$A = [\,a_K^1 - a_K^0 \mid a_K^3 - a_K^0\,]$$

$$t = a_K^0$$

$$d = a_K^2 - a_K^3 - a_K^1 + a_K^0$$

$\cdot\, A^{-1} \mid A\hat{x} + t + d\hat{x}_1\hat{x}_2 = x$

$$\hat{x} + q\hat{x}_1\hat{x}_2 = y$$

[Distinguish cases : $\|q\| \approx 0$, $q_1 \approx 0$, $q_2 \approx 0$]

$\cdot\, [q_2 \; -q_1] \mid \underbrace{q_2\hat{x}_1 - q_1\hat{x}_2 = q^T y}_{\text{eliminate } x_2}$

$$\hat{x}_2 = -\frac{1}{q_1}(q_2 y_1 - q_1 y_2 - q_2\hat{x}_1)\,,$$

$$\Rightarrow \quad \underbrace{q_2\hat{x}_1^2 + (1 - q_2 y_1 + q_1 y_2)\hat{x}_1 - y_1 = 0\,.}_{\text{quadratic equation}} \tag{3.3.7}$$

pick solution $\in [0,1]$, if one exists

9. Based on the Googletest framework implement a C++ function

```
void testGlobalInverseQuad(
    const lf::mesh::Entity &quad,
    Eigen::Vector2d xh);
```

which can be used to create a **unit test** for the function `GlobalInverseQuad()`. The argument `quad` should pass an LEHRFEM++ mesh entity of topological type QUAD, the vector `xh` gives the coordinates of $\hat{x} \in \hat{K}$ to be used for testing.
Rely on the `EXPECT_NEAR()` macro to test the correctness of the result of `GlobalInverseQuad()`.

C++11 code 3.3.11: Sub-problem (3-3.g): Implementation of `testGlobalInverseQuad()`
➙ GITLAB

```
2  void testGlobalInverseQuad(const lf::mesh::Entity& quad, Eigen::Vector2d xh)
       {
3    LF_ASSERT_MSG(quad.RefEl() == lf::base::RefEl::kQuad(),
4                  "Cell must be a quadrilateral");
5    // get the coordinates of the corners of this cell
6    lf::geometry::Geometry* geo_ptr = quad.Geometry();
7    auto vertices = lf::geometry::Corners(*geo_ptr);
8    // Image of point in unit square under parametric mapping
9    Eigen::Vector2d x = geo_ptr->Global(xh);
10
11   Eigen::Vector2d xh_comp = PointEvaluationRhs::GlobalInverseQuad(vertices,
         x);
12   EXPECT_NEAR((xh - xh_comp).norm(), 0.0, 1.0E-8)
13       << "quadl " << quad << ": Mismatch xh = " << xh << ", x = " << x
14       << ", xh_comp = " << xh_comp << std::endl;
15 }
```

h) ℓ is not bounded w.r.t. $\|\cdot\|_a$

→ "Minimum of J $= -\infty$"

→ Discrete minimum → $-\infty$ when resolution of $V_{0,h}$ is increased.

j) In the file `pointEvaluation.cc` implement a class **DeltaLocalVectorAssembler** fitting LehrFEM++'s concept of an **ENTITY_VECTOR_PROVIDER** for the the computation of the element vector associated to the specific right-hand side of (3.3.1).

The core task is to complete the code for the `Eval()` method of the **DeltaLocalVectorAssembler** class. The method should check whether $\mathbf{0}$ belongs to the passed cell and then compute the corresponding values using linear Lagrangian finite elements.

$\ell(v_h) = v_h(0)$

- Find $K \in \mathcal{M}$: $0 \in \overline{K}$ ←
- Element vector :

$\vec{\varphi}_K = \left[b_K^\ell(0) \right]_\ell = \left[\hat{b}^\ell(\Phi_K^{-1}(0)) \right]_\ell$

↑ for parametric FEM

GlobalInverse * ([0 0]) ↓

⚠ Ensure that exactly one such K is used

k) Write a C++ function (in the file `norms.cc`)

```
double computeL2normLinearFE(
    const lf::assemble::DofHandler &dofh,
    const Eigen::VectorXd &mu);
```

that approximately computes $\|u_h\|_{L^2(\Omega)}$ for a finite element function $u_h \in \mathcal{S}_1^0(\mathcal{M})$, $\mathcal{M}$ a hybrid mesh with straight edges.

One option: $\|v_h\|^2_{L^2(\Omega)} = \vec{\nu}^T \mathbf{M} \vec{\nu}$

↑ Galerkin mass matrix : $\left[a(v,u) = \int_\Omega u(x)v(x)\,dx \right]$

l) function (in the file `norms.cc`)

```
double computeH1seminormLinearFE(
  const lf::assemble::DofHandler &dofh,
  const Eigen::VectorXd &mu);
```

that approximately computes $|u_h|_{H^1(\Omega)}$ for a finite element function $u_h \in \mathcal{S}_1^0(\mathcal{M})$, $\mathcal{M}$ a hybrid mesh with straight edges. The argument `dofh` gives the local→global index mapping for $\mathcal{S}_1^0(\mathcal{M})$, whereas `mu` passes the basis expansion coefficients of u_h.

One option: $|v|^2_{H^1(\Omega)} = \vec{\nu}^T \mathbf{A} \vec{\nu}$

↑ Galerkin stiffness matrix $\left[(u,v) \to \int_\Omega \operatorname{grad} u \cdot \operatorname{grad} v\,dx \right]$

(5)

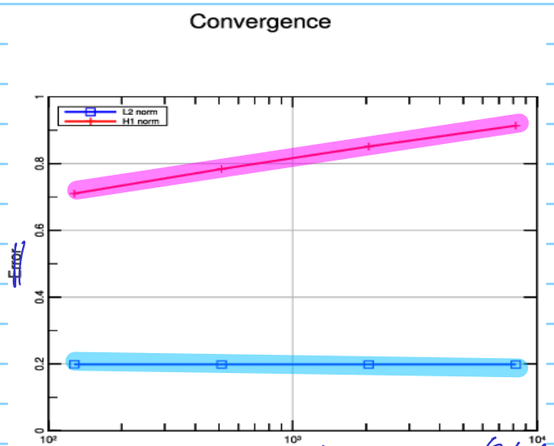
m)

Copying the relevant parts of the function `lecturedemoDirichlet()` from `examples/lecturedemos/lecturedemoassemble.cc` ➡ GITHUB create a function

```
std::pair<double,double>
  normsSolutionPointLoadDirichletBVP(
    const lf::mesh::Mesh &mesh);
```

that solves (3.3.1) on the domain covered by the mesh stored in `mesh` and by means of lowest-order Lagrangian finite elements. It should return the $L^2(\Omega)$-norm and $H^1(\Omega)$-seminorm of the finite element solution $u_h \in \mathcal{S}_1^0(\mathcal{M})$.

n)



Norms

$N = \dim \mathcal{S}_1^0(\mathcal{M})$

L^2-norm: "stable"

H^1-seminorm: "mild blow-up"

$|u_h|_{H^1(\Omega)} \to \infty$ for $K \to \infty$

o) $a(u,v) = \ell(v) \quad \forall v \in V_0$

$J(u) = \frac{1}{2}a(u,u) - \ell(u) = -\frac{1}{2}a(u,u)$

p) Here: $-\frac{1}{2}a(u_h,u_h) = -\frac{1}{2}|u_h|^2_{H^1(\Omega)}$

$\to -\infty$ for $K \to \infty$