Note jotted down using Xournal:

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

Homework Problems and Projects

Prof. R. Hiptmair, SAM, ETH Zurich

Spring Term 2019

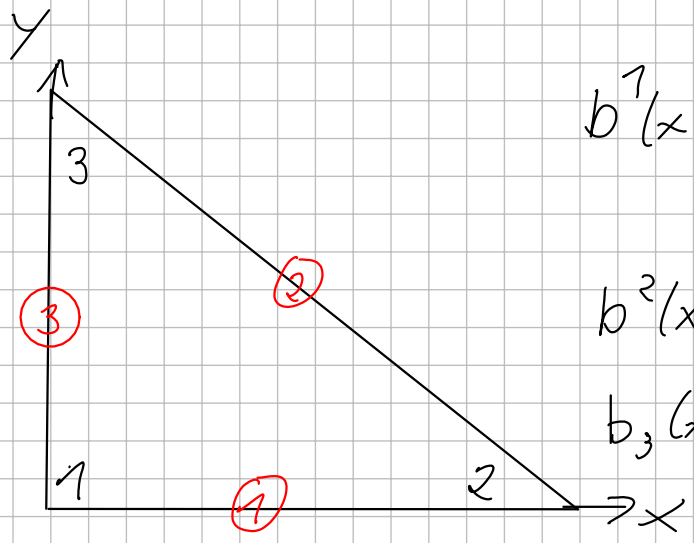
(C) Seminar für Angewandte Mathematik, ETH Zürich

[These notes refer to an old version of the homework!]

Notes for Num PDE homework problem

Non-Conforming CR-elements

Local reference shape functions



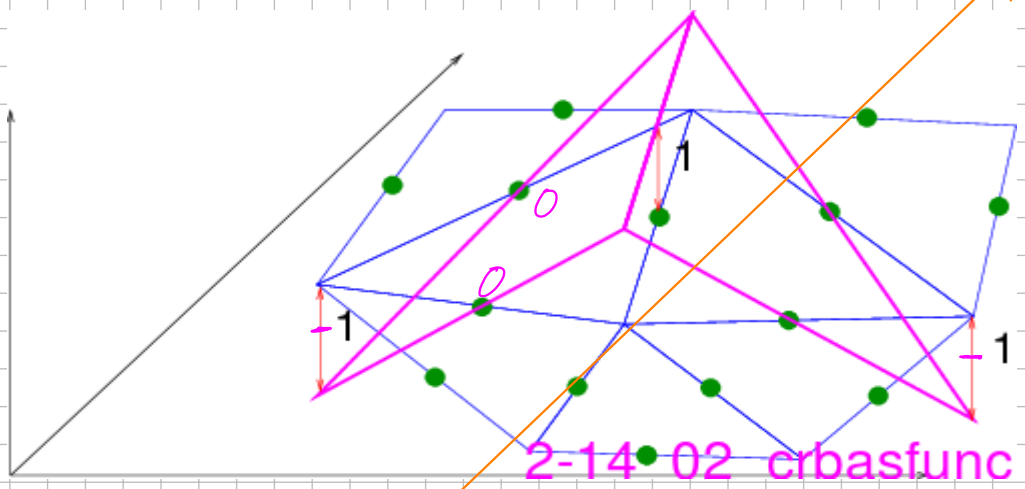
$$b^1(x) = \lambda_1 + \lambda_2 - 1 = 1 - 2y$$

$$b^2(x) = 2(x+y) - 1$$

$$b_3(x) = 1 - 2x$$

$$b^i_{h\,|K} \in \mathcal{P}_1(K) \quad \forall K \in \mathcal{M} \quad , \quad b^i_h(m_j) = \begin{cases} 1 & \text{, if } i = j \,, \\ 0 & \text{else,} \end{cases} \quad i,j \in \{1,\dots,N\} \,, \qquad (2.14.1)$$

2-14 01 eq:1



2-14 02 crbasfunc

$CR(M) =$
$\text{Span}\{b_h^j\}_{j=1}^N$

$$CR_0(\mathcal{M}) := \{v_h \in CR(\mathcal{M}) : v_h(m) = 0 \; \forall m \in \mathcal{N} \cap \partial\Omega\} \,. \qquad (2.14.2)$$

2-14 03 eq:cR0

a, Valid definition ?

▷ Affine linear function fixed by values in 3 points

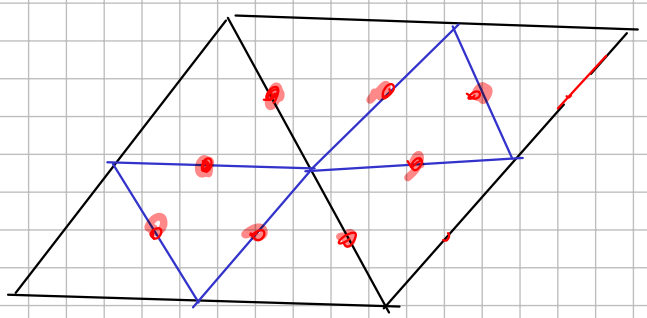
b, $\{b_h^j\}$ l.i. $\Leftarrow$ cardinal basis for $\{m_j\}_{j=1}^N$

$$\sum_{j=1}^{N} \alpha_j b_h^j(\boldsymbol{x}) = 0 \quad \forall \boldsymbol{x} \in \Omega \quad \Rightarrow \quad \alpha_j = 0 \quad \forall j \,.$$

2-14 04 lidp

c, Interpolation nodes: $\mathcal{N} = \{m_j\}$
(edge midpoints)

d, [ ]



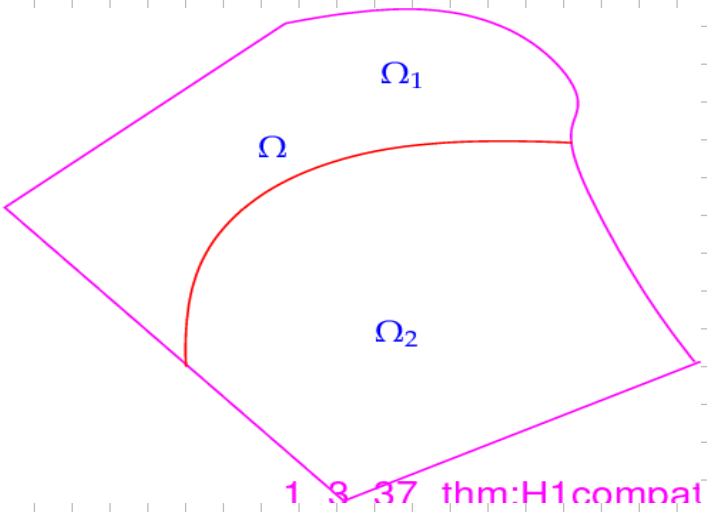
• $\in$ functions in $CR(\mathcal{M}^1)$ continuous here, but not functions in $CR(\mathcal{M})$

e) $CR(\mathcal{M}) \not\subset C^0(\overline{\Omega})$, but p.w. polynomial

▷ $CR(\mathcal{M}) \not\subset H^1(\Omega)$

Theorem 1.3.4.23. Compatibility conditions for piecewise smooth functions in $H^1(\Omega)$

Let Ω be partitioned *into sub-domains Ω_1 and Ω_2. A function u that is continuously differentiable in both sub-domains and continuous up to their boundary, belongs to $H^1(\Omega)$, if and only if u is continuous on Ω.*



1 8 37 thm:H1compat

f) $\operatorname{supp} b_h^j = \bigcup \{K \in \mathcal{M} : m_j \in \overline{K}\}$

[Two triangles adjacent to edge j]

g) $\{b_h^j\}$ valid FE global shape functions ?

Third main ingredient of FEM: locally supported basis functions

(see Section 2.2 for role of bases in Galerkin discretization)

Basis functions $b_h^1, \ldots, b_h^N$ for a finite element trial/test space $V_{0,h}$ built on a mesh $\mathcal{M}$ *must* satisfy:

($\mathfrak{B}_1$) $\mathfrak{B}_h := \{b_h^1, \ldots, b_h^N\}$ is basis of $V_{0,h}$ ➢ $N = \dim V_{0,h}$,

($\mathfrak{B}_2$) each b_h^i is associated with a single geometric entity (cell/edge/face/vertex) of $\mathcal{M}$,

($\mathfrak{B}_3$) $\operatorname{supp}(b_h^i) = \bigcup\{\overline{K}: K \in \mathcal{M}, p \subset \overline{K}\}$, if b_h^i associated with cell/edge/face/vertex p.

2 5 13 par:febasfn

1. follows by the definition of $CR(\mathcal{M})$ and Sub-problem (2-14.b). These two combined imply that the set of functions defined in (2.14.1) are a basis for an N-dimensional finite element space $CR(\mathcal{M}) \subset L^2(\Omega)$.
2. holds as each b_h^i is associated to a single edge.
3. As we saw in Sub-problem (2-14.f), if b_h^i is associated with the edge $\mathbf{e}$, then $\operatorname{supp}(b_h^i) = \bigcup\{\bar{K} : K \in \mathcal{M}, \mathbf{e} \subset \bar{K}\}$. Moreover, $\#\{K \in \mathcal{M} : K \subset \operatorname{supp}(b_h^i)\} = 2$.

2-14 07 shpfn

h) Give the C++ code for the initialization of a LEHRFEM++ **lf::assemble::DofHandler** object suitable for the finite element space $\mathcal{CR}(\mathcal{M})$, if a pointer to a **lf::mesh::Mesh** object describing the triangular mesh $\mathcal{M}$ is available in the object `mesh_p`.

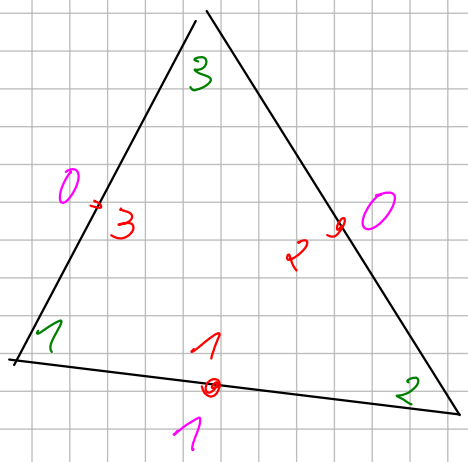
2-14 08 sp:4x

SOLUTION of (2-14.i):

```
lf::assemble::UniformFEDofHandler dof_handler(
      mesh_p, {{lf::base::RefEl::kPoint(), 0},
               {lf::base::RefEl::kSegment(), 1},
               {lf::base::RefEl::kTria(), 0},
               {lf::base::RefEl::kQuad(), 0}});
```

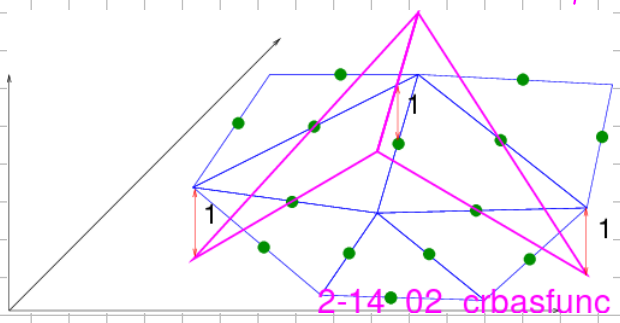
2-14-9-1:s4x

i) Local shape functions b_K^i, $i = 1,2,3$ as linear combinations of barycentric coordinate functions:



$$b_K^1(a^1) = 1$$
$$b_K^1(a^2) = 1$$
$$b_K^1(a^3) = -1$$

14_02



2-14-02 crbasfunc

$$\triangleright \quad b_K^1 = \lambda_1 + \lambda_2 - \lambda_3 \underset{\sum \lambda_i = 1}{=} 1 - 2\lambda_3$$

$$b_K^i = \lambda_i + \lambda_{i+1} - \lambda_{i+2} \quad \text{[cyclic numbering]}$$

j)

Definition 2.8.3.1. Parametric finite elements

A finite element space on a mesh $\mathcal{M}$ is called parametric, if there exist a few reference elements $\widehat{K}_1, \ldots, \widehat{K}_R$, $R \in \mathbb{N}$, numbers $Q_R \in \mathbb{N}$, and functions $\widehat{b}_r^i \in C^0(\widehat{K})$, $i = 1, \ldots, Q$, $r = 1, \ldots, R$, such that

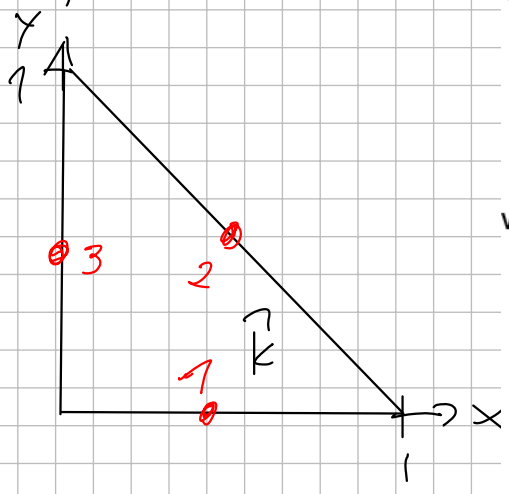
$$\forall K \in \mathcal{M}: \quad \exists r \in \{1, \ldots, R\}, \text{ bijection } \boldsymbol{\Phi}_K : \widehat{K} \mapsto K: \quad \widehat{b}_r^i = \boldsymbol{\Phi}_K^* b_K^i, \quad i = 1, \ldots, Q \,,$$

where $\{b_K^1, \ldots, b_K^Q\}$ is the set of local shape functions on K.

2_8_24_def:parfe

- Reference element $\hat{K} \hat{=}$ unit triangle
- Here $\Phi_K \hat{=}$ unique affine mapping $\Phi_K : \hat{K} \to K$ $K \in \mathcal{M}$
- $\hat{b}^i = \Phi_K^* b_K^i$, because this is true for λ_i & b_K^i are linear comb. of λ_i

k) Explicit formulas for $\hat{b}^i$, $i = 1, 2, 3$



with gradients

$$b^1_{h|K} = 1 - 2\lambda_3 = 1 - 2y$$
$$b^2_{h|K} = 1 - 2\lambda_1 = 2(x+y) - 1,$$
$$b^3_{h|K} = 1 - 2\lambda_2 = 1 - 2x,$$

$$\mathbf{grad}\, b^1_{h|K} = -2\,\mathbf{grad}\,\lambda_3 = \begin{bmatrix} 0 \\ -2 \end{bmatrix},$$
$$\mathbf{grad}\, b^2_{h|K} = -2\,\mathbf{grad}\,\lambda_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix},$$
$$\mathbf{grad}\, b^3_{h|K} = -2\,\mathbf{grad}\,\lambda_2 = \begin{bmatrix} -2 \\ 0 \end{bmatrix}.$$

2-14 11 5sol

l) Element mass matrix

$$M_K = \left[\int_K b_K^i b_K^j \, dx \right]_{i,j=1}^3$$

$a_K(u,v) := \int_K u(x)v(x)\,\mathrm{d}x.$

$$\begin{aligned}
\left(a_K(b^i_{h|K}, b^j_{h|K})\right)_{i,j} &= \left(a_K(1 - 2\lambda_{\mathrm{opp}(i)}, 1 - 2\lambda_{\mathrm{opp}(j)})\right)_{i,j} \\
&= \left(\int_K 1 \,\mathrm{d}x - 2\int_K (\lambda_{\mathrm{opp}(i)} + \lambda_{\mathrm{opp}(j)})\,\mathrm{d}x + 4\int_K \lambda_{\mathrm{opp}(i)}\lambda_{\mathrm{opp}(j)}\,\mathrm{d}x \right)_{i,j} \\
&= \left(|K| - 2\cdot 2\frac{|K|}{3}\right)\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} + \frac{|K|}{3}\begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix} \\
&= \frac{|K|}{3}\mathbf{I}\,.
\end{aligned}$$

2-14 13 sol6

BVPs

$$-\Delta u + \gamma(x)u = f(x) \quad \text{in } \Omega \qquad \begin{cases} \mathbf{grad}\, u \cdot \boldsymbol{n} &= 0 \\ u &= 0 \end{cases} \quad \text{on } \partial\Omega\,, \tag{2.14.4}$$

2-14 14 eq:bvp

▷ Broken-up bilinear form

$$a_{\mathcal{M}}(u_h, v_h) + \int_\Omega c(x)u_h(x)v_h(x)\,\mathrm{d}x = \int_\Omega f(x)v_h(x)\,\mathrm{d}x \quad \forall v_h \in \mathcal{CR}_0(\mathcal{M})\,, \tag{2.14.5}$$

with $a_{\mathcal{M}}(u_h, v_h) := \sum_{K\in\mathcal{M}} \int_K \mathbf{grad}\, u(x) \cdot \mathbf{grad}\, v(x)\,\mathrm{d}x\,.$ (2.14.6)

2-14 15 eq:vc

m) $a_{\mathcal{M}}(v_h, v_h) > 0 \quad \forall v_h \in \mathcal{CR}_0(\mathcal{M}) \setminus \{0\}$

- $$a_{\mathcal{M}}(v_h, v_h) = \sum_{K\in\mathcal{M}} \int_K |\mathbf{grad}\, v_h(x)|^2\,\mathrm{d}x \geq 0.$$

2-14 17 amesh

- $a_{\mathcal{M}}(v_h, v_h) = 0 \Rightarrow v_h|_K \equiv \text{const} \quad \forall K \in \mathcal{M}$

$\Downarrow$

same value $\forall m \in \mathcal{N}$

$m \in \partial\Omega \Rightarrow v_h(m) = 0 \Rightarrow v_h \equiv 0$

n) :

Let $\mathbf{A}_K$, $K \in \mathcal{M}$ a triangle, be the element matrix for the bilinear form $(u, v) \mapsto \int_\Omega \mathbf{grad}\, u \cdot \mathbf{grad}\, v\,\mathrm{d}x$ and the Lagrangian finite element space $\mathcal{S}_1^0(\mathcal{M})$ equipped with the standard "tent-function basis". Write $\mathbf{A}_K^{\mathcal{CR}}$ for the element matrix spawned by the bilinear form $a_{\mathcal{M}}$, the finite element space $\mathcal{CR}(\mathcal{M})$ and the basis introduced in (2.14.1). Which *congruence transformation* converts $\mathbf{A}_K$ into $\mathbf{A}_K^{\mathcal{CR}}$?

2-14 18 sp:6a

Lemma 2.2.3.2. Effect of change of basis on Galerkin matrix

Consider (2.2.1.1) *and two bases of* $V_{0,h}$,

$$\mathfrak{B}_h := \{b_h^1, \ldots, b_h^N\} \quad , \quad \widetilde{\mathfrak{B}}_h := \{\widetilde{b}_h^1, \ldots, \widetilde{b}_h^N\}\,,$$

related by the basis transformation matrix $\mathbf{S}$ *according to*

$$\widetilde{b}_h^j = \sum_{k=1}^N s_{jk} b_h^k \quad \text{with} \quad \mathbf{S} = [s_{jk}]_{j,k=1}^N \in \mathbb{R}^{N,N} \text{ regular.} \tag{2.2.3.3}$$

Then the Galerkin matrices $\mathbf{A}, \widetilde{\mathbf{A}} \in \mathbb{R}^{N,N}$, *the right hand side vectors* $\vec{\boldsymbol{\varphi}}, \widetilde{\vec{\boldsymbol{\varphi}}} \in \mathbb{R}^N$, *and the coefficient vectors* $\vec{\boldsymbol{\mu}}, \widetilde{\vec{\boldsymbol{\mu}}} \in \mathbb{R}^N$, *respectively, satisfy*

$$\widetilde{\mathbf{A}} = \mathbf{S}\mathbf{A}\mathbf{S}^\top \quad , \quad \widetilde{\vec{\boldsymbol{\varphi}}} = \mathbf{S}\vec{\boldsymbol{\varphi}} \quad , \quad \widetilde{\vec{\boldsymbol{\mu}}} = \mathbf{S}^{-\top}\vec{\boldsymbol{\mu}}\,. \tag{2.2.3.4}$$

2_2_13_lem:bascho

Apply to a mesh with a single cell K.
[Note: Same localized bilinear form]

$$\mathbf{A}_K := \left[\int_K \mathbf{grad}\,\lambda_j \cdot \mathbf{grad}\,\lambda_i \,\mathrm{d}x\right]_{i,j=1,2,3}$$

$$\mathbf{A}_K^{\mathcal{CR}} := \left[\mathsf{a}_{\mathcal{M},K}(b_K^j, b_K^i)\right]_{i,j=1,2,3}.$$

Basis transformation:

$$b_K^1 = \lambda_1 + \lambda_2 - \lambda_3$$
$$b_K^2 = \lambda_2 + \lambda_3 - \lambda_1$$
$$b_K^3 = \lambda_1 + \lambda_3 - \lambda_2$$

$$\mathbf{S} = \begin{bmatrix} 1 & 1 & -1 \\ -1 & 1 & 1 \\ 1 & -1 & 1 \end{bmatrix}.$$

2-14 21 hint 2-14 22 S

$$\mathbf{A}_K^{\mathcal{CR}} = \mathbf{S}\,\mathbf{A}_K^{\mathcal{L}}\,\mathbf{S}^T$$

0)

Give the formula for the element matrix for the finite element space $\mathcal{CR}(\mathcal{M})$ equipped with the basis introduced in (2.14.1), a triangular cell $K \in \mathcal{M}$, and for the bilinear form $\mathsf{a}_{\mathcal{M}}$ from (2.14.7) evaluated approximately by means of (2.14.9).
You can use the symbol $\mathbf{A}_K^{\mathcal{CR}}$ for the element matrix computed in Sub-problem (2-14.n). 2-14 24 sp:7

$$\mathsf{a}_{\mathcal{M}}(u_h, v_h) + \int_\Omega c(x) u_h(x) v_h(x)\,\mathrm{d}x = \int_\Omega f(x) v_h(x)\,\mathrm{d}x \quad \forall v_h \in \mathcal{CR}_0(\mathcal{M}), \tag{2.14.6}$$

with $$\mathsf{a}_{\mathcal{M}}(u_h, v_h) := \sum_{K\in\mathcal{M}} \int_K \mathbf{grad}\,u(x) \cdot \mathbf{grad}\,v(x)\,\mathrm{d}x. \tag{2.14.7}$$

2-14 15 eq:vc

Local quadrature formula

$$\int_K \psi(x)\,\mathrm{d}x \approx \frac{|K|}{3} \sum_{\ell=1}^{3} \psi(m_K^\ell), \quad K \in \mathcal{M}, \tag{2.14.9}$$

2-14 23 eq:emp

$$\mathbf{A}_K = \mathbf{A}_K^{\mathcal{CR}} + \frac{|K|}{3} \begin{bmatrix} \gamma(m_K^1) & & \\ & \gamma(m_K^2) & \\ & & \gamma(m_K^3) \end{bmatrix}.$$

2-14 25 VAdiag

↑ diagonal due to cardinal basis property

p) Give the formula for the element right-hand-side vector (element load vector) $\vec{\varphi}_K$, $K \in \mathcal{M}$, for the variational problem (2.14.6) discretized by means of $\mathcal{CR}(\mathcal{M})$. Since the source function f is available only in procedural form, use the quadrature formula (2.14.9) locally.

2-14 26 sp:7a

By cardinal basis property:

$$\vec{\varphi}_K[j] = \int_K f(\boldsymbol{x}) b_K^j = \tfrac{1}{3}|K| \sum_{l=1}^{3} f(\boldsymbol{m}^l) b_h^j(\boldsymbol{m}^l) = \tfrac{1}{3}|K| f(\boldsymbol{m}^l)\delta_{jl},$$

2-14 27 phiK

$$\Rightarrow \quad \vec{\varphi}_K = \frac{1}{3}|K| \begin{bmatrix} f(\boldsymbol{m}^1) \\ f(\boldsymbol{m}^2) \\ f(\boldsymbol{m}^3) \end{bmatrix}.$$

2-14 28 phiKvec

q)

Implement the following member functions of **CRReferenceFiniteElement**:

- `base::RefEl CRReferenceFiniteElement::RefEl() const;` = TRIA
  `unsigned int CRReferenceFiniteElement::Degree() const;` 1

  `size_type CRReferenceFiniteElement::NumRefShapeFunctions() const;` = 3
  `size_type CRReferenceFiniteElement::NumRefShapeFunctions(dim_t codim) const;` = 1 for codim = 1 (edges)
  `size_type CRReferenceFiniteElement::NumRefShapeFunctions(dim_t codim, sub_idx_t subidx) const;`

- `Eigen::Matrix<double, Eigen::Dynamic, Eigen::Dynamic> CRReferenceFiniteElement::EvalReferenceShapeFunctions(const Eigen::MatrixXd& refcoords) const;` 2-14 29 srfe

$$\hat{b}^1(\hat{x}) = 1 - 2y, \quad \hat{x} = \begin{bmatrix} x \\ y \end{bmatrix}$$

⋮

[2-14_11]
[Also for +)]

$$\hat{b}^1 = 1 - 2\lambda_3 = 1 - 2y$$
$$\hat{b}^2 = 1 - 2\lambda_1 = 2(x+y) - 1,$$
$$\hat{b}^3 = 1 - 2\lambda_2 = 1 - 2x,$$

with gradients

$$\mathbf{grad}\,\hat{b}^1 = -2\,\mathbf{grad}\,\lambda_3 = \begin{bmatrix} 0 \\ -2 \end{bmatrix},$$
$$\mathbf{grad}\,\hat{b}^2 = -2\,\mathbf{grad}\,\lambda_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix},$$
$$\mathbf{grad}\,\hat{b}^3 = -2\,\mathbf{grad}\,\lambda_2 = \begin{bmatrix} -2 \\ 0 \end{bmatrix}.$$

r)

Implement the member function

```
Eigen::Matrix<double, Eigen::Dynamic, Eigen::Dynamic>
  CRReferenceFiniteElement::GradientsReferenceShapeFunctions(
    const Eigen::MatrixXd& refcoords) const;
```

Look up the web page ⇢ (documentation) to find the specification for this method.

2-14 33 sp:9h

Return value: Gradients in rows of a matrix

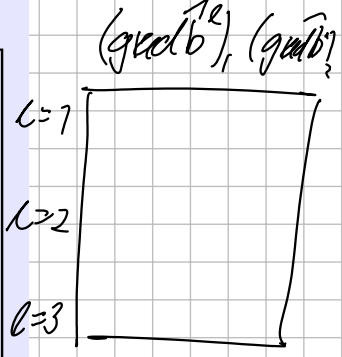
C++ code 2.14.14: Sub-problem (2-14.r): `GradientsReferenceShapeFunctions()` for CRReferenceFiniteElement ⇢ GITLAB

```
Eigen::Matrix<scalar_type, Eigen::Dynamic, Eigen::Dynamic>
CRReferenceFiniteElement::GradientsReferenceShapeFunctions(
    const Eigen::MatrixXd& refcoords) const {
  const auto num_points = static_cast<size_type>(refcoords.cols());
  Eigen::MatrixXd grad_ref_shape_functions(3, 2 * num_points);

  // TODO: task 2-14.r)
  /* BEGIN_SOLUTION */
  grad_ref_shape_functions.row(0) = (Eigen::Vector2d() << 0, -2)
                                    .finished()
                                    .transpose()
                                    .replicate(1, num_points);
  grad_ref_shape_functions.row(1) =
      2. * Eigen::VectorXd::Ones(2 * num_points).transpose();
  grad_ref_shape_functions.row(2) = (Eigen::Vector2d() << -2, 0)
                                    .finished()
                                    .transpose()
                                    .replicate(1, num_points);
  /* END_SOLUTION */

  return grad_ref_shape_functions;
}
```

▷ Gradients are constant



2-14 34 cpp:refel

s)

Implement the following member functions of **CRReferenceFiniteElement**:

- ```
  Eigen::MatrixXd CRReferenceFiniteElement::EvaluationNodes()
    const;
  ```
  → midpoints of edges
- ```
  Eigen::Matrix<double, 1, Eigen::Dynamic>
    CRReferenceFiniteElement::NodalValuesToDofs(
      const Eigen::Matrix<SCALAR, 1, Eigen::Dynamic>& nodvals)
      const;
  ```
  → identity mapping

For explanations see [Lecture → § 2.8.3.24] and ⇢ (documentation). Note that in LEHRFEM++ an "evaluation node" corresponds to a local interpolation node. The local interpolation nodes for $\mathcal{RM}(\mathcal{M})$ are just the midpoints of the edges of a triangle.

2-14 35 sp:9c

t)

Implement a class **CRFeSpace** compliant with LEHRFEM++'s interface **lf::uscalfe::ScalarUniformFESpace** (and derived from it) for the finite element space $\mathcal{CR}(\mathcal{M})$. This interface is presented in [Lecture → § 2.8.3.27].

Take the cue from the implementation of **lf::uscalfe::FeSpaceLagrangeO1**, but note that you need only supply the specification for local shape functions on triangles, none for QUADs and EDGEs (pass **nullptr** for those).

2-14 38 sp:9x

C++ code 2.14.17: Sub-problem (2-14.t): Implementation of CRFeSpace, ↪ GITLAB

```
class CRFeSpace : public lf::uscalfe::UniformScalarFESpace<scalar_type> {
 public:
  /**
   * @brief no default constructors
   */
  CRFeSpace() = delete;
  CRFeSpace(const CRFeSpace &) = delete;
  CRFeSpace(CRFeSpace &&) noexcept = default;
  CRFeSpace &operator=(const CRFeSpace &) = delete;
  CRFeSpace &operator=(CRFeSpace &&) noexcept = default;

  /**
   * @brief main constructor that sets up the local-to-global index mapping
   * @param mesh_ptr shared pointer to underlying mesh (immutable)
   */
  explicit CRFeSpace(std::shared_ptr<const lf::mesh::Mesh> mesh_ptr)
      : lf::uscalfe::UniformScalarFESpace<scalar_type>(
            std::move(mesh_ptr),
            std::make_shared<CRReferenceFiniteElement>(),
            nullptr, nullptr) {
    // TODO: task 2-14.t)
    /* BEGIN_SOLUTION */
    /* END_SOLUTION */
  }

  ~CRFeSpace() override = default;
};
```

nothing to do

2-14 39 cpp:crfes

Implement a C++ function

```
template <typename GAMMA_COEFF, typename F_FUNCTOR>
  Eigen::VectorXd solveCRNeumannBVP(std::shared_ptr<CRFeSpace>
     fe_space,
                                    GAMMA_COEFF &&gamma
                                    F_FUNCTOR &&f);
```

that solves the discrete variational problem (2.14.6) with $\mathcal{CR}_0(\mathcal{M})$ replaced with $\mathcal{CR}(\mathcal{M})$ (no boundary conditions enforced on trial and test space any more) and returns the basis expansion coefficient vector of the solution $u_h \in \mathcal{CR}(\mathcal{M})$. The argument `gamma` passes a functor (with evaluations operator

```
double operator () (Eigen::Vector2d x) const;
```

for the coefficient function $\gamma \in C^0(\overline{\Omega})$ that is supposed to be uniformly positive.
Of course, your implementation should rely on the assembly functions in the LEHRFEM++ module `lf::assemble`. Moreover, you can make use of the LEHRFEM++ classes **lf::uscalfe::ReactionDiffusionElementMatrixProvider** ([Lecture → Ex. 2.8.3.28]) and **lf::uscalfe::ScalarLoadElementVectorProvider**, which can serve as builder helper classes for element matrices and element vectors. Note that functions are passed as **lf::uscalfe::MeshFunctionGlobal** objects to the constructors of these classes!
Examining [Lecture → Code 2.8.3.30] convince yourself that the `Eval()` member functions of theses classes will smoothly work in the case of $\mathcal{CR}(\mathcal{M})$ provided that the finite element space contains the right **lf::uscalfe::ScalarReferenceFiniteElement** object.

2-14 40 sp:12a

$$a_{\mathcal{M}}(u_h, v_h) + \int_\Omega \gamma(x) u_h(x) v_h(x)\,\mathrm{d}x = \int_\Omega f(x) v_h(x)\,\mathrm{d}x \quad \forall v_h \in \mathcal{CR}_0(\mathcal{M})\,, \tag{2.14.6}$$

$$\text{with} \quad a_{\mathcal{M}}(u_h, v_h) := \sum_{K\in\mathcal{M}} \int_K \mathbf{grad}\, u(x) \cdot \mathbf{grad}\, v(x)\,\mathrm{d}x\,. \tag{2.14.7}$$

2-14 15 eq:vc

C++ code 2.14.18: Sub-problem (2-14.u): Implementation of `solveCRNeumannBVP()`, → GITLAB

```
template <typename GAMMA_COEFF, typename F_FUNCTOR>
Eigen::VectorXd solveCRNeumannBVP(std::shared_ptr<CRFeSpace> fe_space,
                                  GAMMA_COEFF &&gamma, F_FUNCTOR &&f) {
  Eigen::VectorXd sol;
  /* BEGIN_SOLUTION */
  // Obtain local to global index mapping for shape functions
  const lf::assemble::DofHandler &dof_handler{fe_space->LocGlobMap()};
  const size_type num_dofs = dof_handler.NoDofs();
  // Prepare coefficient and source functions as MeshFunction
  lf::uscalfe::MeshFunctionGlobal mf_one{
      [](Eigen::Vector2d x) -> double { return 1.; }};
  lf::uscalfe::MeshFunctionGlobal mf_gamma{gamma};
  lf::uscalfe::MeshFunctionGlobal mf_f{f};
  // Sparse Galerkin matrix in triplet format
  lf::assemble::COOMatrix<scalar_type> A(num_dofs, num_dofs);
  // Initialize ELEMENT_MATRIX_PROVIDER object
  lf::uscalfe::ReactionDiffusionElementMatrixProvider<
      scalar_type, decltype(mf_one), decltype(mf_gamma)>
      element_matrix_builder(fe_space, mf_one, mf_gamma);
  // Fill Galerkin matrix (create array of triplets)
  lf::assemble::AssembleMatrixLocally(0, dof_handler, dof_handler,
                                      element_matrix_builder, A);
  // Right-hand-side vector; do not forget to set to zero!
  Eigen::Matrix<double, Eigen::Dynamic, 1> phi(num_dofs);
  phi.setZero();
  // Initialize ELEMENT_VECTOR_PROVIDER object
  lf::uscalfe::ScalarLoadElementVectorProvider<double, decltype(mf_f)>
      load_vector_builder(fe_space, mf_f);
  // Fill right-hand-side vector (cell oriented assembly)
  lf::assemble::AssembleVectorLocally(0, dof_handler, load_vector_builder,
     phi);
  // Set up Galerkin matrix in CRS format
  Eigen::SparseMatrix<scalar_type> A_crs = A.makeSparse();
  // ... and solve the linear system of equations by Gaussian elimination
  Eigen::SparseLU<Eigen::SparseMatrix<scalar_type>> solver;
  solver.compute(A_crs);
  sol = solver.solve(phi);
  /* END_SOLUTION */
  return sol;
}
```

2-14 41 cpp:crneu

Based on LEHRFEM++'s assembly facilities and the classes **lf::uscalfe::ReactionDiffusionElementMatri** and **lf::uscalfe::ScalarLoadElementVectorProvider** create a C++ function

```
template <typename GAMMA_COEFF, typename F_FUNCTOR>
Eigen::VectorXd solveCRDirichletBVP(std::shared_ptr<CRFeSpace>
   fe_space,
                                    GAMMA_COEFF &&gamma,
                                    F_FUNCTOR &&f);
```

that solves (2.14.6) with trial and test space $\mathcal{CR}_0(\mathcal{M})$, which is supposed to provide a non-conforming finite element discretization for the a Dirichlet boundary value problem with homogeneous Dirichlet boundary conditions. The arguments and return value are the same as in Sub-problem (2-14.u).

To solve this problem remember how to enforce Dirichlet boundary conditions in LEHRFEM++: [Lecture → **??**].

2-14 42 sp:13

[C-11_43]

wj

We now want to compute the $L^2(\Omega)$-norm of the finite element discretization error $u_h - u$ of the $\mathcal{CR}_0(\mathcal{M})$ non-conforming finite element solution of the homogeneous Dirichlet problem (2.14.5), where $u \in H^1(\Omega)$ stands for its exact solution.
For this purpose, write the code for a lfpp-based C++ function

```
template <typename FUNCTION>
double computeCRL2Error((std::shared_ptr<CRFeSpace> fe_space,
                        const Eigen::VectorXd &mu,
          FUNCTION && u)
```

which takes as input arguments

- a **CRFeSpace** object for the finite element space $\mathcal{CR}(\mathcal{M})$,
- a column vector `mu` supplying the basis expansion coefficients a finite element function $u_h(x) = \sum_{j=0}^{N-1} \mu_j b_h^j(x) \in \mathcal{CR}(\mathcal{M})$, and
- a functor object `u`, which allows access to the exact solution u through a point-evaluation operator with signature **double operator**`(Eigen::Vector2d x)` **const**.

and returns the $L^2(\Omega)$-norm of the $\mathcal{CR}_0$ finite element discretization error: $\|u_h - u\|_{L^2(\Omega)}$.
Compute the norm directly in a cell oriented fashion using a loop over the cells and the local quadrature formula (2.14.10).

2-14 44 sn:15

↳ edge-midpoint rule

$$\int_K \psi(x)\,\mathrm{d}x \approx \frac{|K|}{3} \sum_{\ell=1}^{3} \psi(m_K^\ell)\,, \quad K \in \mathcal{M}\,, \tag{2.14.10}$$

2-14 23 eq:emp

C++ code 2.14.20: Sub-problem (2-14.w): Implementation of computeCRL2Error, → GITLAB

```
template <typename FUNCTION>
double computeCRL2Error(std::shared_ptr<CRFeSpace> fe_space,
                        const Eigen::VectorXd &mu, FUNCTION &&u) {
  double l2_error = 0.;
  /* BEGIN_SOLUTION */
  // Obtain local-to-global map and current mesh object
  const lf::assemble::DofHandler &dof_handler{fe_space->LocGlobMap()};
  auto mesh_ptr = fe_space->Mesh();

  // Loop over all cells of the mesh (entities of co-dimension 0)
  for (const lf::mesh::Entity &cell : mesh_ptr->Entities(0)) {
    const size_type num_nodes = cell.RefEl().NumNodes();
    LF_ASSERT_MSG(num_nodes == 3, "Only meaningful for triangles!");
    // Obtain pointer to shape information for cell
    lf::geometry::Geometry &cell_geom{*cell.Geometry()};
    // 2x3- matrix with corner coordinates in its columns
    const Eigen::MatrixXd vertices{lf::geometry::Corners(cell_geom)};
    // clang-format off
    // 2x3-matrix of midpoint coordinates
    auto midpoints{vertices *
      (Eigen::Matrix<double, 3, 3>(3,3) <<
                    0.5, 0.0, 0.5,
                    0.5, 0.5, 0.0,
                    0.0, 0.5, 0.5)
                .finished()};
    // clang-format on
    // Obtain global indices of local shape functions
    const lf::base::RandomAccessRange<const gdof_idx_t> cell_dof_idx(
        dof_handler.GlobalDofIndices(cell));

    // Sum contributions of quadrature nodes
    double local_sum = 0.;
    // The CR interpolation nodes are the midpoints and so the exact
    // solution needs to be evaluated at the same points
    for (int loc_idx = 0; loc_idx < num_nodes; ++loc_idx) {
      local_sum +=
          std::pow(mu[cell_dof_idx[loc_idx]] - u(midpoints.col(loc_idx)), 2);
    }
    // Sum cell contributions
    l2_error += lf::geometry::Volume(cell_geom) * (local_sum / num_nodes);
  }
  /* END_SOLUTION */
  return std::sqrt(l2_error);
}
```

2-14 45 cpp:computeL2Error

Empirically we study the behavior of the $L^2(\Omega)$-norm of the $\mathcal{CR}_0(\mathcal{M})$-discretization error for the 2nd-order elliptic boundary value problem with a known solution u, with the data

$$\begin{aligned} f(\boldsymbol{x}) &= (2\pi^2 + x_1 x_2)\sin(\pi x_1)\sin(\pi x_2) \\ c(\boldsymbol{x}) &= x_1 x_2 \,, \\ u(\boldsymbol{x}) &= \sin(\pi x_1)\sin(\pi x_2) \,, \end{aligned} \qquad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} . \tag{2.14.21}$$

Code a C++ function

```
double L2errorCRDiscretizationDirichletBVP(std::string filename);
```

that reads a 2D triangular mesh $\mathcal{M}$ from a `.msh`-file with name `filename`, computes the $\mathcal{CR}_0(\mathcal{M})$-finite element solution u_h of the homogeneous Dirichlet boundary value problem (**??**) on it using the data from (2.14.21), and returns the $\|u - u_h\|_{L^2(\Omega)}$.

2-14 46 sp:14

C++ code 2.14.22: Sub-problem (2-14.x): Implementation of `L2errorCRDiscretizationDirichletBVP()` ➙ GITLAB

```
double L2errorCRDiscretizationDirichletBVP(const std::string &filename) {
  double l2_error;

  // TODO: task 2-14.x)
  /* BEGIN_SOLUTION */
  // Right-hand-side source function
  auto f = [](Eigen::Vector2d x) -> double {
    return (2. * M_PI * M_PI + x.prod()) * std::sin(M_PI * x(0)) *
           std::sin(M_PI * x(1));
  };
  // Reaction coefficient
  auto gamma = [](Eigen::Vector2d x) -> double { return x.prod(); };
  // Analytic solution
  auto u = [](Eigen::Vector2d x) -> double {
    return std::sin(M_PI * x(0)) * std::sin(M_PI * x(1));
  };

  // Read mesh from file
  auto mesh_factory = std::make_unique<lf::mesh::hybrid2d::MeshFactory>(2);
  const lf::io::GmshReader reader(std::move(mesh_factory), filename);

  // Build CR FE space
  auto fe_space = std::make_shared<CRFeSpace>(reader.mesh());

  // Solve homogeneous Dirichlet problem
  Eigen::VectorXd mu = solveCRDirichletBVP(fe_space, gamma, f);
  // Compute L2 norm of error
  l2_error = computeCRL2Error(fe_space, mu, u);
  /* END_SOLUTION */

  return l2_error;
}
```

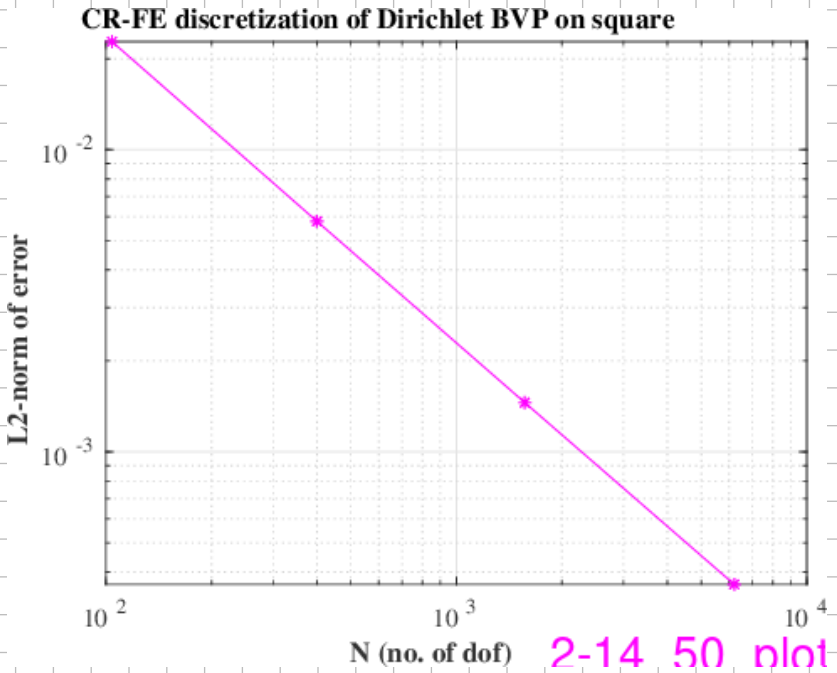
2-14 47 cpp:testbvp

4, Tabulate L^2-error for test meshes

$N = \dim \mathcal{RC}_0(\mathcal{M})$	104	400	1568	6208
$\|u - u_h\|_{L^2(\Omega)}$	0.0227969	0.00579489	0.00145469	0.000364046

2-14 48 errtab

2, cvg.



2-14 50 plot

▷ Alg. cvg. rate = 1