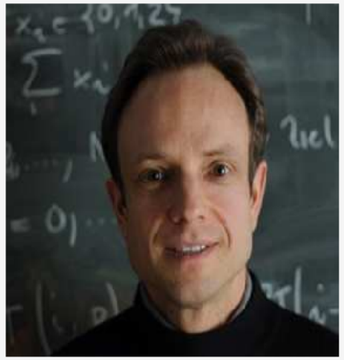
ETH Lecture 401-0674-00L Numerical Methods for Partial Differential Equations

Course Video

Section 2.4: Case Study: Triangular Linear FEM in Two Dimensions (II)

Prof. R. Hiptmair, SAM, ETH Zurich

Date: February 21, 2019



Prerequisites.

- Numerical quadrature
- C++

Dependency. Requires [Lecture → 2.5.1] and [Lecture → Section 2.2]. Familiarity with [Lecture → Section 2.3] is beneficial.

II. Finite Element Methods (FEM)

2.4. Case Study: Triangular Linear FEM in 2D

2.4.5. Computation of Galerkin Matrices

Model problem: $[\alpha \equiv 1]$ bilinear form for weak-1

$$a(u,v) := \int_\Omega \mathbf{grad}\, u \cdot \mathbf{grad}\, v \,\mathrm{d}\boldsymbol{x}\,, \quad u,v \in H^1_0(\Omega)\,.$$

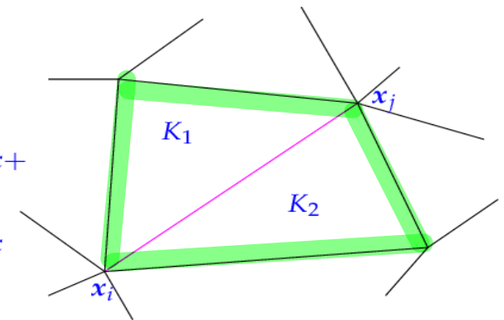
and a Galerkin discretization based on

- a triangular mesh, see Section 2.4.1, set of vertices $\{\boldsymbol{x}_i\} = \mathcal{V}(\mathcal{M})$,
- the discrete trial/test space $\mathcal{S}^0_1(\mathcal{M}) \subset H^1(\Omega)$,
- the *nodal basis* $\mathfrak{B}_h = \left\{b_h^j\right\}$ of tent functions according to (2.4.3.2).

$$(A)_{i,j} = a(b_h^j, b_h^i) = \int_\Omega \mathrm{grad}\, b_h^j \cdot \mathrm{grad}\, b_h^i \,\mathrm{d}x\,, \quad i \neq j$$

Idea: "Assembly" (add up *cell contributions*)

$$(\mathbf{A})_{ij} = \int_{K_1} \mathbf{grad}\, b^j_{h|K_1} \cdot \mathbf{grad}\, b^i_{h|K_1} \,\mathrm{d}x + \int_{K_2} \mathbf{grad}\, b^j_{h|K_2} \cdot \mathbf{grad}\, b^i_{h|K_2} \,\mathrm{d}x$$

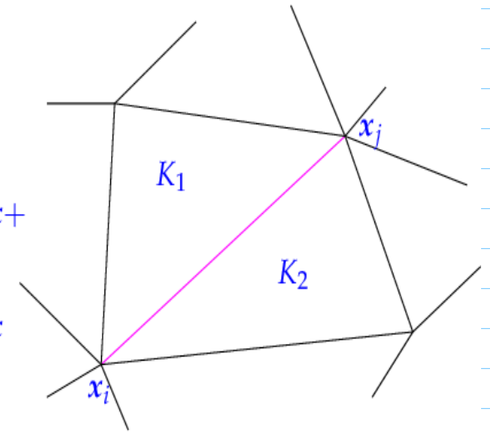


② 2.4.5.1. Local computations

Idea:

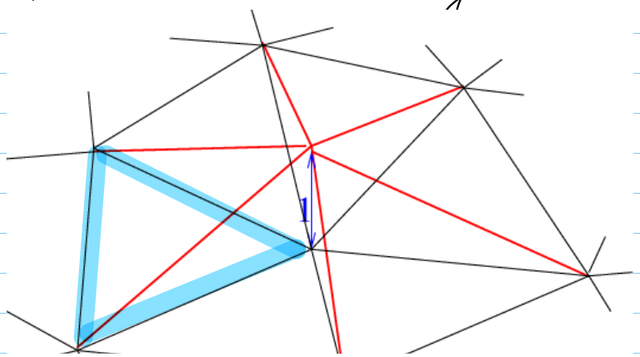
"Assembly"
(add up *cell contributions*)

$$(\mathbf{A})_{ij} = \int_{K_1} \mathbf{grad}\, b^j_{h|K_1} \cdot \mathbf{grad}\, b^i_{h|K_1}\, \mathrm{d}x + \int_{K_2} \mathbf{grad}\, b^j_{h|K_2} \cdot \mathbf{grad}\, b^i_{h|K_2}\, \mathrm{d}x$$



simple linear functions on each triangle

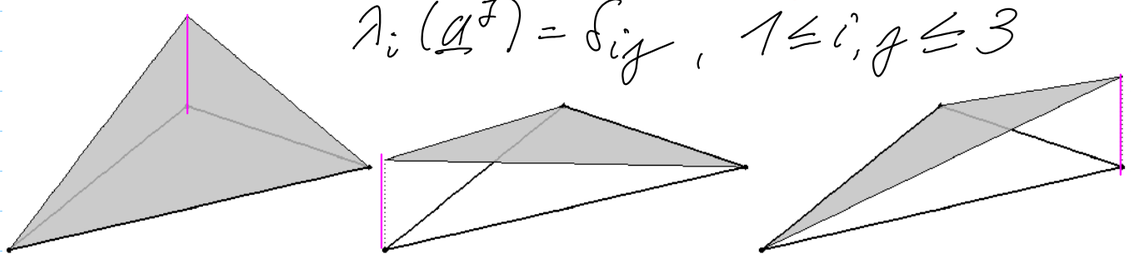
Adopt a "triangle point of view"



§2.4.5.2:

▷ Barycentric coordinate functions: λ_i

$$\lambda_i(\boldsymbol{a}^j) = \delta_{ij}, \quad 1 \le i,j \le 3$$



Restrictions $\lambda_1, \lambda_2, \lambda_3$ of p.w. linear nodal basis functions of $\mathcal{S}^0_1(\mathcal{M})$ to triangle K

Computation of grad λ_i:

① Geometric formulas for λ_i: [n^i = edge normals]

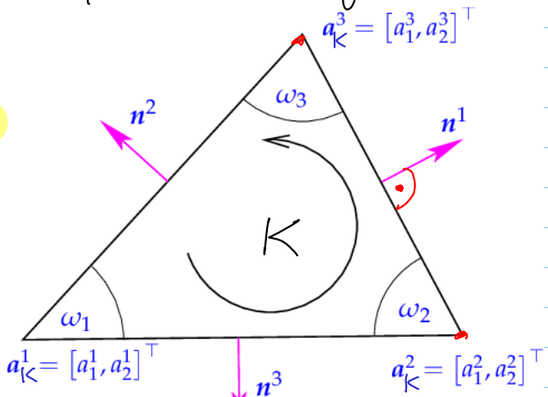
linear function

$$\lambda_1(\mathbf{x}) = \frac{1}{2|K|}(\mathbf{x}-\boldsymbol{a}^2)\cdot\begin{bmatrix} a^2_2 - a^3_2 \\ a^3_1 - a^2_1 \end{bmatrix} = -\frac{|e_1|}{2|K|}(\boldsymbol{x}-\boldsymbol{a}^2_K)\cdot\boldsymbol{n}^1,$$

$$\lambda_2(\mathbf{x}) = \frac{1}{2|K|}(\mathbf{x}-\boldsymbol{a}^3)\cdot\begin{bmatrix} a^3_2 - a^1_2 \\ a^1_1 - a^3_1 \end{bmatrix} = -\frac{|e_2|}{2|K|}(\boldsymbol{x}-\boldsymbol{a}^3_K)\cdot\boldsymbol{n}^2,$$

$$\lambda_3(\mathbf{x}) = \frac{1}{2|K|}(\mathbf{x}-\boldsymbol{a}^1)\cdot\begin{bmatrix} a^1_2 - a^2_2 \\ a^2_1 - a^1_1 \end{bmatrix} = -\frac{|e_3|}{2|K|}(\boldsymbol{x}-\boldsymbol{a}^1_K)\cdot\boldsymbol{n}^3.$$

(e_i = edge opposite vertex $\boldsymbol{a}^i_K$, see Figure for numbering scheme ▷)



$\lambda_1(x') = 1$ from formula for the distance of a point from a line (Hesse normal form)

② Algebraic formulas:

$$\lambda_i(\underline{x}) = \alpha_i + \beta_i \cdot \underline{x}, \quad \alpha_i \in \mathbb{R}, \ \beta_i = \operatorname{grad}\lambda_i \in \mathbb{R}^2$$

$$\begin{bmatrix} 1 & a^1_1 & a^1_2 \\ 1 & a^2_1 & a^2_2 \\ 1 & a^3_1 & a^3_2 \end{bmatrix} \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \beta^1_1 & \beta^2_1 & \beta^3_1 \\ \beta^1_2 & \beta^2_2 & \beta^3_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \qquad (2.4.5.10)$$

③

C++ code 2.4.5.11: Computation of gradients of barycentric coordinate functions on a triangle ⇢ GITLAB

```
Eigen::Matrix<double, 2, 3> gradbarycoordinates(const t_TriGeo& Vertices)
{
  // Argument Vertices passes the vertex positions of the triangle
  // as the rows of a 3×2-matrix, , see Code 2.4.1.3..
  // The function returns the components of the gradients as the
  // columns of a 2×3-matrix

  // Computation based on (2.4.5.10), solving for the
  // coefficients of the barycentric coordinate functions.
  Eigen::Matrix<double, 3, 3> X; // Temporary matrix
  X.block<3, 1>(0, 0) = Eigen::Vector3d::Ones();
  X.block<3, 2>(0, 1) = Vertices.transpose();
  return X.inverse().block<2, 3>(1, 0);
}
```

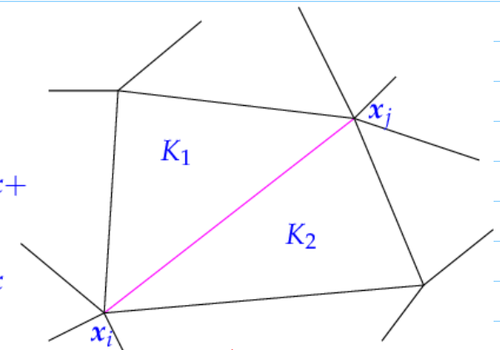
⇒ very efficient

[returns $\operatorname{grad}\lambda_i$ as columns of 2×3-matrix]

Idea:

"Assembly"
(add up *cell contributions*)

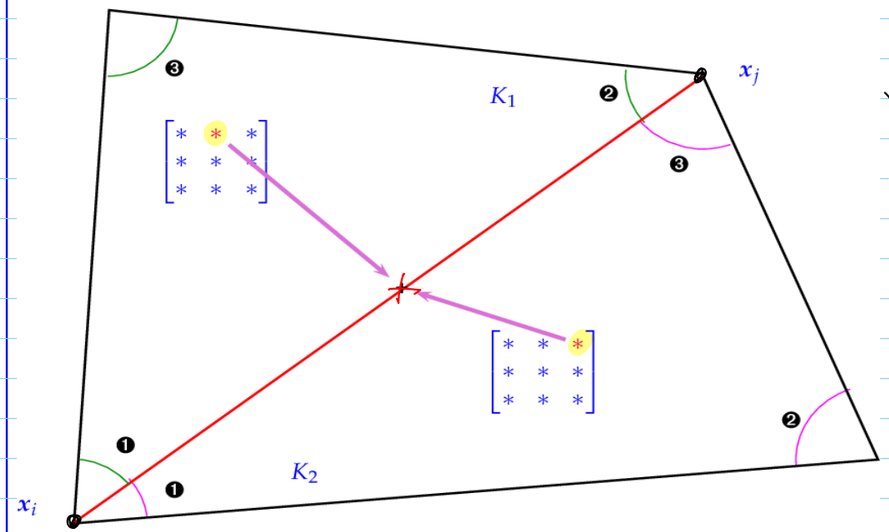
$$(\mathbf{A})_{ij} = \int_{K_1} \mathbf{grad}\, b^j_{h|K_1} \cdot \mathbf{grad}\, b^i_{h|K_1}\, \mathrm{d}x + \int_{K_2} \mathbf{grad}\, b^j_{h|K_2} \cdot \mathbf{grad}\, b^i_{h|K_2}\, \mathrm{d}x$$



▷ $(A)_{i,j}$ is a sum of entries of two element matrices

$$A_K = \left[\int_K \operatorname{grad}\lambda_l \cdot \operatorname{grad}\lambda_m \,dx\right]_{l,m=1}^{3} \in \mathbb{R}^{3,3}, \quad K \in \mathcal{M}$$

▷ $(A)_{i,j} = (A_K)_{?,?} + (A_K)_{?,?}$ $i \neq j$



$(\mathbf{A})_{ij}$ by summing entries of two element matrices

$b^i_h|_{K_1} = \lambda_1$

$b^j_h|_{K_1} = \lambda_2$

$$(A)_{i,j} = (A_{K_1})_{l,m} + (A_{K_2})_{s,t}$$

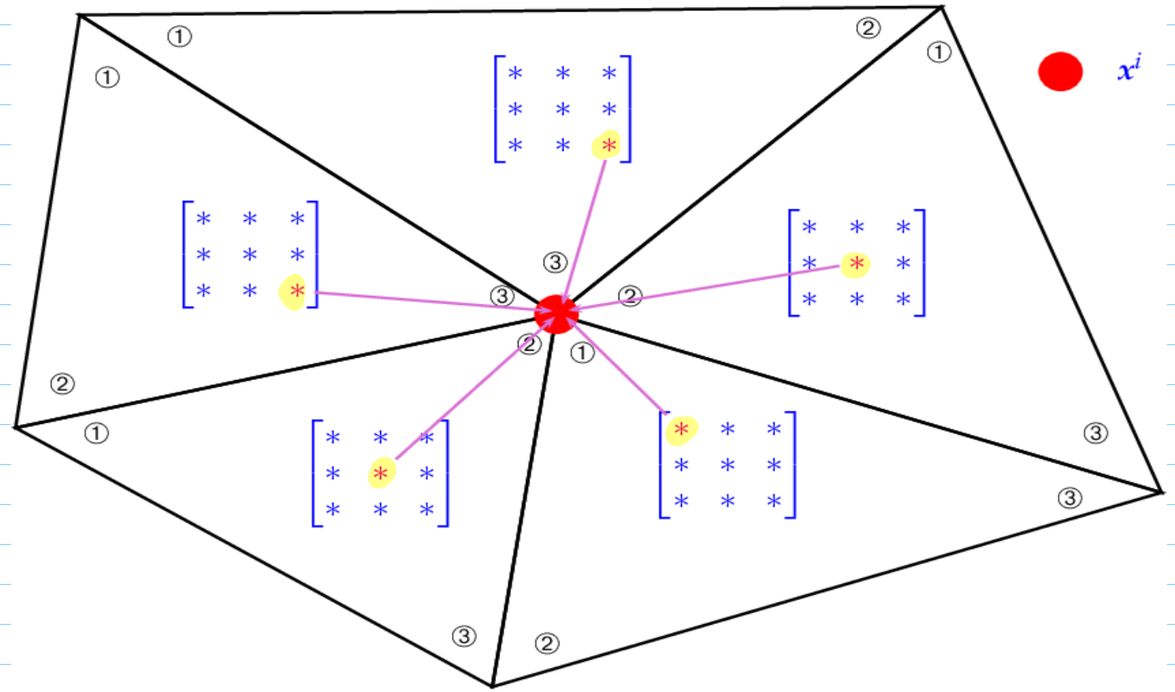
$l = s$: $a^s = x^j \Leftrightarrow \lambda_s = b^j_h|_{K_1}$

$m = t$: $a^t = x^i \Leftrightarrow \lambda_t = b^i_h|_{K_1}$

Diagonal entries

$$(A)_{i,i} = \sum_{K:\, x^i \in \partial K} \int_K \operatorname{grad} b^i_{h|K} \cdot \operatorname{grad} b^i_{h|K}\, dx$$

λ_* ↓



$(\mathbf{A})_{ii}$ by summing diagonal entries of element matrices of adjacent triangles

$$(A)_{i,i} = \sum_{K:\, x^i \in \partial K} (A_K)_{\ell_K,\ell_K}, \quad \ell_K : a^{\ell_K} = x^i$$

Remark: Computation of A_K

$\operatorname{grad}\lambda_i \equiv \text{const} \Rightarrow A_K = |K| \left[\operatorname{grad}\lambda_i \cdot \operatorname{grad}\lambda_j\right]_{i,j=1}^{3}$

$= |K|\, X^T X$

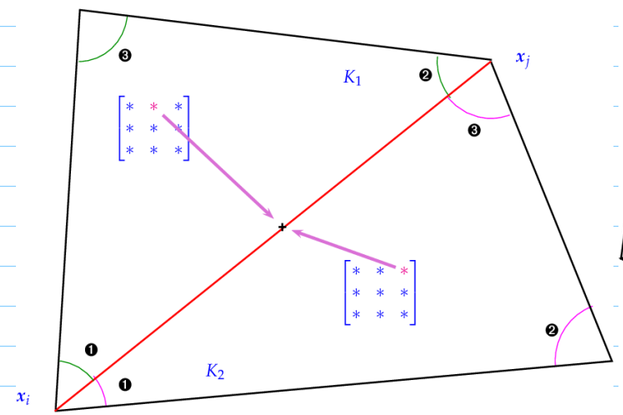
$X = [\operatorname{grad}\lambda_1 \mid \operatorname{grad}\lambda_2 \mid \operatorname{grad}\lambda_3] \in \mathbb{R}^{2,3}$

C++ code 2.4.5.13: Computation of element matrix for $-\Delta$ on a triangle and for linear Lagrangian finite elements ↪ GITLAB

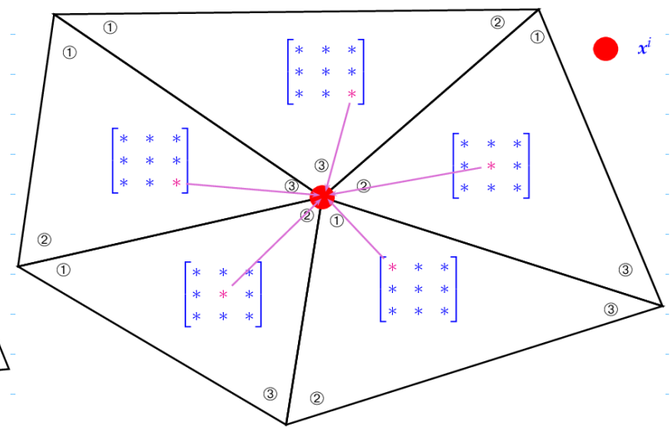
```
Eigen::Matrix3d ElementMatrix_Lapl_LFE(const t_TriGeo& V)
{
  // Argument V same as Vertices in Code 2.4.5.11.
  // The function returns the 3×3 element matrix as a fixed size
  // EIGEN matrix.

  // Evaluate (2.4.5.1), exploiting that the gradients are constant.
  // First compute the area of triangle by determinant formula
  const double area = 0.5*std::abs(
  (V(1,0)-V(0,0))*(V(2,1)-V(1,1))-(V(2,0)-V(1,0))*(V(1,1)-V(0,1)));
  // Get gradients of barycentric coordinate functions,
  // see Code 2.4.5.11
  const Eigen::Matrix<double,2,3> X = gradbarycoordinates(V);
  // Compute inner products of gradients through matrix multiplication
  return area*X.transpose()*X;
}
```

2.4.5.2 Assembly of full Galerkin Matrix



$(\mathbf{A})_{ij}$ by summing entries of two element matrices



$(\mathbf{A})_{ii}$ by summing diagonal entries of element matrices of adjacent triangles

▷ "collect approach"

⑤

Pseudocode 2.4.5.16: Vertex-centered assembly of Galerkin matrix for linear finite elements

```
foreach e ∈ E(M)        (✎ notation: E(M) ≙ set of edges of M)
    (i,j) ≙ vertex numbers of endpoints of e
    (A)_{i,j} ← 0, (A)_{j,i} ← 0,
    foreach triangle K adjacent to e
        find local numbers l,m ∈ {1,2,3} of endpoints of e
        (A)_{i,j} ← (A)_{i,j} + (A_K)_{l,m}     → Fig. 68, A_K from (2.4.5.8)
        (A)_{j,i} ← (A)_{j,i} + (A_K)_{m,l}     → Fig. 68, A_K from (2.4.5.8)
    endfor
endfor
```

⇒ Off-diagonal entries

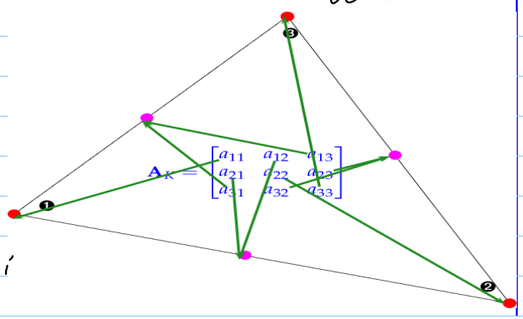
```
foreach v ∈ V(M)
    j ≙ number of vertex v
    (A)_{j,j} ← 0
    foreach triangle K adjacent to v
        l ≙ local number of v in K
        (A)_{j,j} ← (A)_{j,j} + (A_K)_{l,l}     → Fig. 69, A_K from (2.4.5.8)
    endfor
endfor
```

⇒ Diagonal entries

Problem:
- Edges not available in data structure
- Triangles adjacent to nodes not available

Alternative dual view:

distribute & add entries of A_K to corresponding entries of A:



▷ Cell-oriented implementation (§2.4.5.17)

Pseudocode 2.4.5.19: Assembly of finite element Galerkin matrix for linear finite elements

```
SparseMatrix A ∈ R^{N,N}, N := ♯V(M)   (number of vertices)
A := O;
for i = 1 to N do
    K ← mesh.getVtCoords(i)
    A_K ← getElementMatrix(K);
    for k = 1 to 3 do
        for j = 1 to 3 do
            A(i : x^i = a_K^k, ℓ : x^ℓ = a_K^j) += A_K(k,j);   ⇔ "Distribute & add"
        endfor
    endfor endfor
```

↑ index mapping (*)

Computational effort: $O(M)$, if index remapping has $O(1)$ cost!

(*) Task of a d.o.f. mapper / d.o.f. handler

Data structure: $\texttt{dofh} \in \mathbb{N}^{\sharp\mathcal{M},3}$: local→global *index mapping array*: "d.o.f. mapper"

$\texttt{dofh}(k,l)$ = global number of vertex l of k-th cell $\in \{1,\dots,N\}$

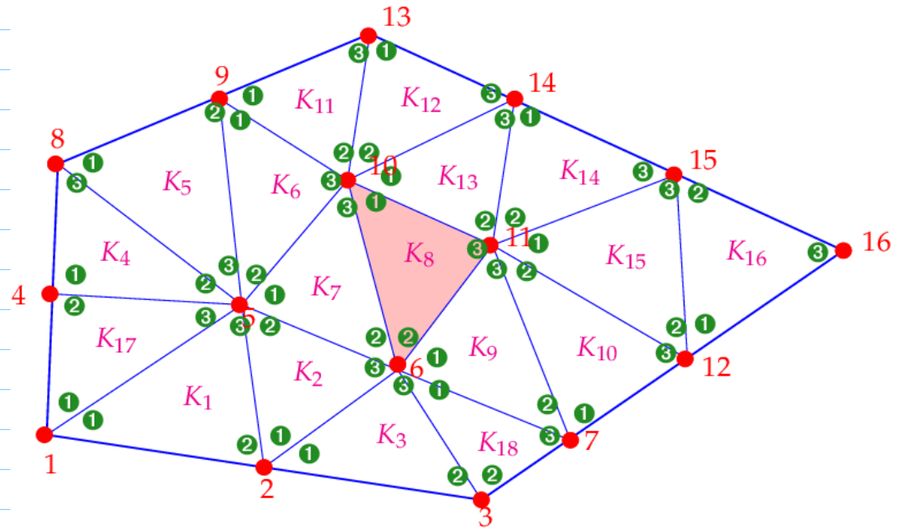
$x_{\texttt{dofh}(k,l)} = a^l$ when a^1, a^2, a^3 are the vertices of K_k, (2.4.5.21)

for $l \in \{1,2,3\}$, $k \in \{1,\dots,M\}$, $M := \sharp\mathcal{M}$, $N := \sharp\mathcal{V}(\mathcal{M})$ ("mathematical indexing"!).

Note: For TriaMesh2D data structure

dofh(k,l) = Mesh.Elements(k-1,l-1)

EXAMPLE 2.4.5.22 (Index mapping by d.o.f. mapper)



dofh:

K_1	1	2	5
K_2	2	5	6
K_3	2	3	6
K_4	4	5	8
K_5	8	9	5
K_6	9	5	10
K_7	5	6	10
K_8	10	6	11
K_9	6	7	11
K_{10}	7	11	12
K_{11}	9	10	13
K_{12}	13	10	14
K_{13}	10	11	14
K_{14}	14	11	15
K_{15}	11	12	15
K_{16}	12	15	16
K_{17}	1	4	5
K_{18}	6	3	7

In Fig. 71, for cell K_8: element matrix $\mathbf{A}_K$ contributes to `A([10 6 11],[10 6 11])` ❑

Pseudocode 2.4.5.23: Assembly of finite element Galerkin matrix for linear finite elements

SparseMatrix $\mathbf{A} \in \mathbb{R}^{N,N}$, $N := \sharp\mathcal{V}(\mathcal{M})$ (number of vertices)
$\mathbf{A} := \mathbf{O}$;
for $i = 1$ **to** N **do**
  $K \leftarrow$ `mesh.getVtCoords`(i)
  $\mathbf{A}_K \leftarrow$ **getElementMatrix**(K);
  for $k = 1$ **to** 3 **do**
    for $j = 1$ **to** 3 **do**
      $\mathbf{A}$`(dofh(i,k),dofh(i,j))` $+=$ $\mathbf{A}_K(k,j)$;
    endfor
  endfor endfor

C++ code 2.4.5.24: Cell-oriented assembly of Galerkin matrix for linear finite elements on a triangular mesh ↪ GITLAB

```
1  // Functor referencing a function for the computation of the element
2  // matrices like ElementMatrix_Lapl_LFE from Code 2.4.5.13.
3  typedef function<Eigen::Matrix3d(const t_TriGeo &)> LocalMatrixHandle_t;
4
5  Eigen::SparseMatrix<double>
6  assembleGalMatLFE(const TriaMesh2D& Mesh,
7                    const LocalMatrixHandle_t& getElementMatrix) {
8    // Fetch the number of vertices
9    int N = Mesh.Coordinates.rows();
10   // Fetch the number of elements/cells, see § 2.4.1.1
11   int M = Mesh.Elements.rows();
12   // Create empty sparse Galerkin matrix A
13   Eigen::SparseMatrix<double> A(N,N);
14   // Loop over elements and "distribute" local contributions
15   for (int i = 0; i < M; i++) {
16     // Get local→global index mapping for current element, cf. (2.4.5.21)
17     Eigen::Vector3i dofhk = Mesh.Elements.row(i);
18     t_TriGeo Vertices;
19     // Extract vertices of current element, see § 2.4.1.1
20     for (int j = 0; j < 3; j++)
21     Vertices.col(j) = Mesh.Coordinates.row(dofhk(j)).transpose();
22     // Compute 3×3 element matrix A_K
23     Eigen::Matrix3d Ak = getElementMatrix(Vertices);
24     // Add local contribution to Galerkin matrix
25     for (int j = 0; j < 3; j++)
26       for (int k = 0; k < 3; k++)
27         A.coeffRef(dofhk(j),dofhk(k)) += Ak(j, k);
28   }
29   // Convert into CRS format, see .
30   A.makeCompressed();
31   return A;
32 }
```

Inefficient initialization of a sparse matrix

Better: use triplet format

⑦ 2.4.6. Computation of rhs Vector $\vec{\varphi}$

Entry: $(\vec{\varphi})_j = \varphi_j = \int f(x)\, b_h^j(x)\, dx$

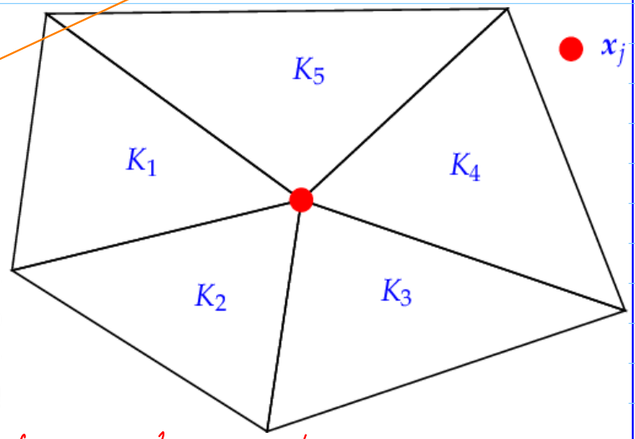
$$= \sum_{K : x^j \in K} \int_K f(x)\, b_h^j(x)\, dx \qquad (2.4.6.2)$$

→ local contributions

Idea: "Assembly"

$$(\vec{\varphi})_j = \sum_{l=1}^{N_j} \int_{K_l} f(x)\, b_h^j|_{K_l}(x)\, dx\,,$$

where $K_1, \ldots, K_{N_j}$ are the triangles adjacent to node x_j. No other matter, because the integration can be confined to $\mathrm{supp}(b_h^j)$!



▷ Assemble $\vec{\varphi}$ from element vectors

$$\vec{\varphi}_K = \left[\int_K f(x)\, \lambda_j(x)\, dx \right]_{j=1}^{3}$$

↑ barycentric coord. fct.

C++ code 2.4.6.8: Cell-oriented assembly of right hand side vector for linear finite elements, see (2.4.6.6) ↪ GITLAB

```
1  typedef function<double(const Eigen::Vector2d&)> FHandle_t;
2  typedef function<Eigen::Vector3d(const t_TriGeo &,FHandle_t)>
       LocalVectorHandle_t;
3
4  Eigen::VectorXd assemLoad_LFE(const TriaMesh2D &Mesh,
5  const LocalVectorHandle_t &getElementVector,
6  const FHandle_t &FHandle)
7  {
8    // Obtain the number of vertices and cells (elements)
9    int N = Mesh.Coordinates.rows();
10   int M = Mesh.Elements.rows();
11   // Initialize right hand side vector with zero.
12   Eigen::VectorXd phi = Eigen::VectorXd::Zero(N);
13
14   // Loop over elements and "distribute" local contributions
15   for (int i = 0; i < M; i++) {
16     // get local→global index mapping for current element,
17     // cf.  (2.4.5.21)
18     Eigen::Vector3i dofhk = Mesh.Elements.row(i);
19     t_TriGeo Vertices;
20     // Extract geometry of current element, see § 2.4.1.1
21     for (int j = 0; j < 3; j++)
22       Vertices.row(j) = Mesh.Coordinates.row(dofhk(j));
23     //compute element right hand side vector
24     Eigen::Vector3d philoc = getElementVector(Vertices, FHandle);
25     //add contributions to global load vector
26     for (int j = 0; j < 3; j++)
27       phi(dofhk(j)) += philoc(j);   ← distribute & add
28   }
29   return phi;
30 }
```

▷ Cost $O(M)$ for $M \to \infty$

§2.4.6.9 (Numerical quadrature for assembly of r.h.s. vector)

How to compute

$$\vec{\varphi}_k = \left[\int_K f(x)\, \lambda_j(x)\, dx \right]_{j=1}^{3} \quad ?$$

↑ given in procedural

▷ Numerical quadrature mandatory

Example: 2D trapezoidal rule

Idea: 2D trapezoidal rule

for triangle K with vertices $\boldsymbol{a}^1, \boldsymbol{a}^2, \boldsymbol{a}^3$

$$\int_K f(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} \approx \frac{|K|}{3}(f(\boldsymbol{a}^1) + f(\boldsymbol{a}^2) + f(\boldsymbol{a}^3)) \,. \tag{2.4.6.10}$$

$\hat{=}$ integration of linear interpolant $\sum_{i=1}^{3} f(\boldsymbol{a}^i)\lambda_i$ of f.

$$\int_K f(x)\, \lambda_j(x)\, dx = \frac{|K|}{3} f(a^j)$$

► element (load) vector: $\vec{\boldsymbol{\varphi}}_K := \left[\ell_K(b_h^{j(i)}|_K)\right]_{i=1}^{3} = [\ell_K(\lambda_i)]_{i=1}^{3} \approx \frac{|K|}{3}\begin{bmatrix} f(\boldsymbol{a}^1) \\ f(\boldsymbol{a}^2) \\ f(\boldsymbol{a}^3) \end{bmatrix}$, (2.4.6.11)

(9) Review questions 2.4.6.13

[Answer in "closed-book mode"]

A: Explain the mathematics underlying the following code

C++ code 2.4.5.11: Computation of gradients of barycentric coordinate functions on a triangle ➙ GITLAB

```
Eigen::Matrix<double, 2, 3> gradbarycoordinates(const TriGeo_t& vertices) {
  Eigen::Matrix<double, 3, 3> X;
  // Argument vertices passes the vertex positions of the triangle
  // as the rows of a 3×2-matrix, , see
  // Code 2.4.1.3.  The function returns the components of the
  // gradients as the columns of a 2×3-matrix

  // Computation based on (2.4.5.10), solving for the
  // coefficients of the barycentric coordinate functions.
  X.block<3, 1>(0, 0) = Eigen::Vector3d::Ones();
  X.block<3, 2>(0, 1) = vertices.transpose();
  return X.inverse().block<2, 3>(1, 0);
}
```

C++11 code 2.4.1.2: Class handling planar triangular mesh ➙ GITLAB

```
// Matrix containing vertex coordinates of a triangle
using TriGeo_t = Eigen::Matrix<double, 2, 3>;
struct TriaMesh2D {
  // Constructor:  reads mesh data from file, whose name is passed
  TriaMesh2D(const std::string&); //
  virtual ~TriaMesh2D(void) {}

  // Retrieve coordinates of vertices of a triangles as rows of a 3x2
     matrix
  TriGeo_t getVtCoords(std::size_t) const;

  // Data members describing geometry and topolgy
  Eigen::Matrix<double, Eigen::Dynamic, 2> Coordinates;
  Eigen::Matrix<int, Eigen::Dynamic, 3> Elements;
};
```

B:

[Computing gradients of barycentric coordinate functions] Justify, why the following code computes an element matrix for $-\Delta$ on a triangle.

C++ code 2.4.5.13: Computation of element matrix for $-\Delta$ on a triangle and for linear Lagrangian finite elements ➙ GITLAB

```
Eigen::Matrix3d ElementMatrix_Lapl_LFE(const TriGeo_t& V) {
  // Argument V same as vertices in Code 2.4.5.11.
  // The function returns the 3×3 element matrix as a fixed size
  // EIGEN matrix.

  // Evaluate (2.4.5.1), exploiting that the gradients are constant.
  // First compute the area of triangle by determinant formula
  double area = 0.5 * std::abs((V(0, 1) - V(0, 0)) * (V(1, 2) - V(1, 1)) -
                               (V(0, 2) - V(0, 1)) * (V(1, 1) - V(1, 0)));
  // Compute gradients of barycentric coordinate functions, see
  // ??
  Eigen::Matrix<double, 2, 3> X = gradbarycoordinates(V);
  // compute inner products of gradients through matrix multiplication
  return area * X.transpose() * X;
}
```

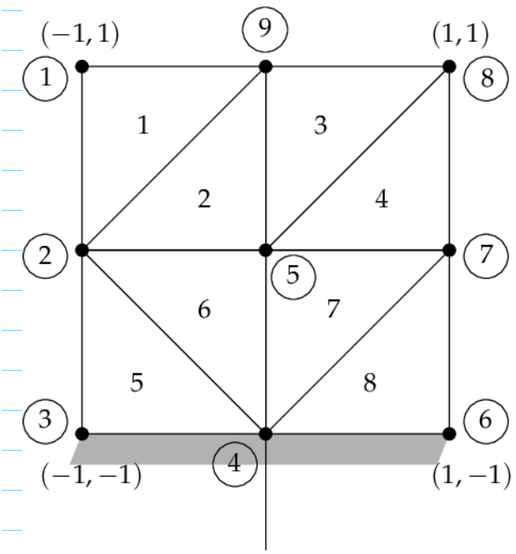
C:

[Taking in account zero Dirichlet boundary conditions] We are provided with an **TriaMesh2D** object describing a planar triangulation $\mathcal{M}$ of a polygon Ω with N nodes and `std::vector<bool> bdflags;` where `bdflags[k] == true`, if the node with number k is located on $\partial\Omega$. Outline how one has to modify Code 2.4.5.24 (this you may look up in the lecture document)so that it assembles a Galerkin matrix w.r.t. the trial/test space $\mathcal{S}^0_{1,0}(\mathcal{M})$.

D:

Write $\mathbf{A}_K$ for the element matrix for linear finite elements, the bilinear form $\mathrm{a}(u,v) := \int_\Omega \mathbf{grad}\, u \cdot \mathbf{grad}\, v \,\mathrm{d}x$, and a triangle K. We use numerical quadrature based on the 2D trapezoidal rule to compute the element matrix $\mathbf{B}_K$ for the bilinear form $\widetilde{b}(u,v) := \int_\Omega \sigma(x)\, \mathbf{grad}\, u \cdot \mathbf{grad}\, v \,\mathrm{d}x$, $\sigma \in C^0(\overline{\Omega})$. How can $\mathbf{B}_K$ be computed from $\mathbf{A}_K$?

E:



Compute the Galerkin matrix for the bilinear form

$$\mathrm{a}(u,v) = \int_\Omega \mathbf{grad}\, u \cdot \mathbf{grad}\, v \,\mathrm{d}x\,,$$

$u,v \in H^1(\Omega)$, $\Omega =]-1,1[^2$, and the trial/test space $\mathcal{S}_1^0(\mathcal{M})$, $\mathcal{M}$ shown in Fig. 84, using the tent function bases numbered according to the numbering of the nodes of the mesh as indicated in Fig. 84.

Hint. All triangles of the mesh are congruent. You may also use the formula

$$\mathbf{A}_K = \frac{1}{2}\begin{bmatrix} \cot\omega_3 + \cot\omega_2 & -\cot\omega_3 & -\cot\omega_2 \\ -\cot\omega_3 & \cot\omega_3 + \cot\omega_1 & -\cot\omega_1 \\ -\cot\omega_2 & -\cot\omega_1 & \cot\omega_2 + \cot\omega_1 \end{bmatrix} \tag{2.4.5.8}$$

giving the $\mathcal{S}_1^0(\mathcal{M})$-element matrix for $-\Delta$ on a triangle with angles ω_i, $i = 1,2,3$.

F:

The cells of a triangular mesh $\mathcal{M}$ are generated by connecting all n corners of a regular polygon with diameter 2 with its center. On this mesh we consider the finite element space $\mathcal{S}_1^0(\mathcal{M})$ and the corresponding Galerkin matrix for the bilinear form

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

where Ω is the interior of the polygon. We assume that the standard tent function basis of $\mathcal{S}_1^0(\mathcal{M})$ is used and that their numbering starts with the central node and then continues counterclockwise through the corners of the polygon.

Hint. The formula Eq. (2.4.5.8) can be used.

This list of review questions may not be complete. Additional review questions may be provided in the lecture document.