Scientific Computing I

Winter Semester 2013 / 2014
Prof. Dr. Beuchler
Bastian Bohn and Alexander Hullmann

Excercise sheet 4.

Closing date **12.11.2013**.

Theoretical exercise 1. (Eigendecomposition of the Laplacian finite difference matrix [5 points])

Let L_h be the operator matrix that stems from the discretization of $-\Delta u = f$ on a regular grid with mesh-width $h = \frac{1}{n}$ on $(0,1)^2$ using the 5-point-stencil.

a) Show that $u^{\nu\mu}(x,y) = \sin(\nu\pi x)\sin(\mu\pi y)$ with $(x,y) \in \Omega_h$ and $1 \leq \nu, \mu \leq n-1$ are the $(n-1)^2$ eigenvectors of L_h.

b) Show that the corresponding eigenvalues are

$$\lambda_{\nu\mu} = 4h^{-2}\left(\sin^2\left(\frac{\nu\pi h}{2}\right) + \sin^2\left(\frac{\mu\pi h}{2}\right)\right)$$

für $1 \leq \nu, \mu \leq n-1$.

c) Prove the following two results:

$$\begin{aligned} \|L_h^{-1}\|_2 &= \frac{1}{8}h^2\sin^{-2}\left(\frac{\pi h}{2}\right) < \frac{1}{8} \\ \|L_h\|_2 &= 8h^{-2}\cos^2\left(\frac{\pi h}{2}\right) < 8h^{-2} \end{aligned}$$

Hint for (a) and (b): Set $u^{\nu\mu}(x,y) = u^\nu(x) \cdot u^\mu(y)$. That way, the eigenvectors can be written as the tensor product of one-dimensional functions, and we only need to consider the one-dimensional case.

Theoretical exercise 2. (Finite difference stencil [5 points])

Show that the stencil

$$h^{-2}\begin{bmatrix} -1/4 & 0 & 1/4 \\ 0 & 0 & 0 \\ 1/4 & 0 & -1/4 \end{bmatrix}$$

is a discretization of $\frac{\partial^2}{\partial x \partial y}$ with central differences and prove that it is consistent of order 2.

Theoretical exercise 3. (Limited regularity on non-convex domains [5 points])

We consider the domain

$$\Omega = \{(r,\phi) : r \in (0,1), \phi \in (0, \tfrac{3}{2}\pi)\}$$

and the boundary value problem

$$\begin{aligned} -\Delta u &= 0 & \text{on} \quad & \Omega\,, & (1) \\ u &= \sin(\tfrac{2}{3}\phi) & \text{on} \quad & (r,\phi) \in \partial\Omega\,. & (2) \end{aligned}$$

a) Express the Laplacian operator in polar coordinates (r, ϕ) with

$$\begin{aligned} x &= r\cos\phi \\ y &= r\sin\phi \end{aligned}$$

and compute the solution u that satisfies (1) and (2) using a product ansatz in ϕ and r.

b) Show that if $\frac{\partial u}{\partial x}$, $\frac{\partial u}{\partial y}$ were bounded, the same would hold for $\frac{\partial u}{\partial r}$. Thereby conclude that $u \notin C^1(\bar{\Omega})$.

Programming exercise 1. (Data structure for meshes [10 points])

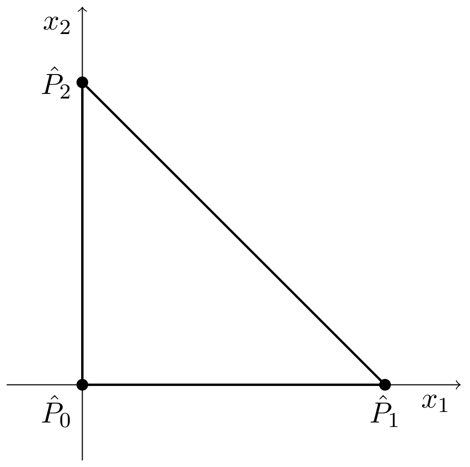
This week you will implement a class `Mesh` for your finite element project.

A two-dimensional "mesh" is a decomposition of a domain into simple subdomains, e.g. quadrangles or triangles. For the implementation, we will use triangles as subdomains. Each triangle is called an "element" and consists of exactly 3 "edges". Each edge is represented by 2 "nodes" for the linear Lagrange basis and by 3 "nodes" for the quadratic Lagrange basis. Each node is a two-dimensional point defined by its coordinates in $\mathbb{R}^2$. Note that certain "admissibility" assumptions have to hold, e.g. for two elements E_1 and E_2

- $E_1 \cap E_2 = \emptyset$ or
- $E_1 \cap E_2 = e$ for an edge e or
- $E_1 \cap E_2 = n$ for a node n.

More on these assumptions and the possible structures of admissible meshes will be introduced in the lecture during the next weeks.

Please stay consistent with the numbering of nodes in the reference triangle when you index element nodes, i.e. for quadratic elements node $\hat{P}_3$ lies on the edge given by the nodes $\hat{P}_0$ and $\hat{P}_1$ and so on.



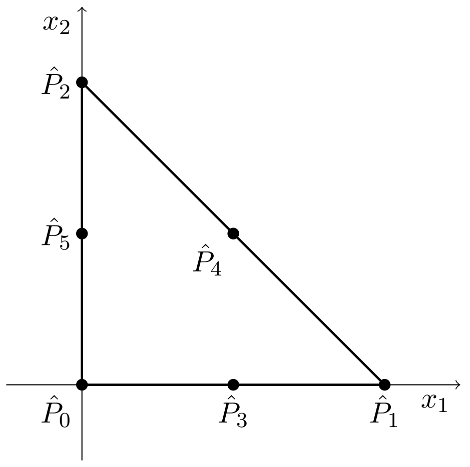


Figure 1: Reference triangles for linear (left) and quadratic (right) polynomials (numbering beginning with index 0)

Tasks:

a) [4 points] Implement the following classes/structs

- `Element` – represents a triangle element. It contains the three indices of the edges that belong to the element and an integer number which represents the material inside the element (this will be relevant later).
- `Edge` – represents an edge. It contains the two indices of the nodes at the end of the edge, one index for the node at the midpoint and two integer numbers which represent the boundary condition and the boundary condition type (this will be relevant later). For an edge-refinement algorithm which will be implemented in a few weeks, we also need a boolean variable which indicates if the edge has already been refined and also two integers indices for the two finer edges that result from an edge-refinement.
- `Node` – represents a node (point) in the mesh. It contains the coordinates in $\mathbb{R}^2$ where the node lies.

Implement the necessary constructors and `set`- and `get`-routines for the member variables for each of these classes.

b) [6 points] Implement the class/struct `Mesh`. It contains the following member variables:

- `std::vector<Element> elements` – the elements of the mesh.
- `std::vector<Edge> edges` – the edges of the mesh.
- `std::vector<Node> nodes` – the nodes of the mesh.
- `std::vector<int> neumannEdgeIndices` – this will be used for boundary condition treatment of PDEs in a few weeks.
- `std::vector<int> dirichletEdgeIndices` – this will be used for boundary condition treatment of PDEs in a few weeks.

(You can also use `Element*` instead of `std::vector<Element>` for example. However, you will have to pay attention to explicit memory allocation.)

Implement the following member functions for the `Mesh` class:

- `void getNumElements()` – returns the number of elements in the mesh.
- `void getNumEdges()` – returns the number of edges in the mesh.
- `void getNumNodes()` – returns the number of nodes in the mesh.
- `void getNodesToElement(int ele, int* n, int size)` – fills the array `n` of size `size` with the node indices of the element with index `ele`. For linear Lagrangian basis, `n` must have size 3. For quadratic Lagrangian basis, `n` must have size 6. Please stay consistent with the node numbering from Figure 1.
- `int shareNode(int edg1, int edg2)` – if the edges with indices `edg1` and `edg2` share a node (i.e. if they touch each other) the index of the corresponding node is returned. If the edges do not share a node the value -1 is returned.
- `void createVTKFile(const char* filename)` – Create a VTK-file (see e.g. `http://www.vtk.org/VTK/img/file-formats.pdf`) to visualize the mesh. It is stored as `filename`. The VTK-file looks like this:

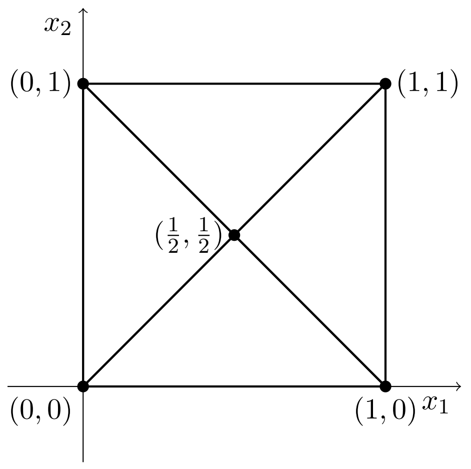
```
# vtk DataFile Version 3.0
vtk output
ASCII
DATASET POLYDATA
POINTS x float
```

```
coord0 coord1 0.0
coord2 coord3 0.0
⋮
POLYGONS y z
3 ind0 ind1 ind2
3 ind3 ind4 ind5
⋮
```

where `x` denotes the number of nodes. `coord0` and `coord1` are the coordinates of node 0, `coord2` and `coord3` are the coordinates of node 1 and so on. `y` is the number of elements and `z` $= 4 \cdot$`y`. `ind0`, `ind1` and `ind2` are the indices (beginning from 0) of the nodes in the corners of the first element. `ind3`, `ind4` and `ind5` are the indices of the nodes in the corners of the second element and so on.

Test your implementation:

- Construct the following mesh by hand and create a VTK file for it. You can visualize the VTK-file using e.g. VTK or ParaView (both open source software). You have to choose "Surface with Edges" in Paraview.


x_2
$(0,1)$
$(1,1)$
$(\frac{1}{2},\frac{1}{2})$
$(0,0)$
$(1,0)$
x_1