Scientific Computing I

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

universitätbonn

Exercise sheet 6.

Closing date **26.11.2013**.

Theoretical exercise 1. (Weak formulation [7 points])

Consider the PDE

$$-\triangle u + \lambda^2 u = 0$$

on the domain $\Omega = (0,1)^2$ with homogeneous Neumann boundary conditions on $\partial\Omega \setminus \{1\} \times (0,1)$, i.e.

$$\nabla u(x) \cdot n(x) = 0 \quad \text{for} \quad x \in \partial\Omega \setminus \{1\} \times (0,1)\,,$$

where $n(x)$ denotes the outer normal vector. State the weak formulation of the PDE given the remaining boundary condition

a) $u(x) = 1 \quad \text{for} \quad x \in \{1\} \times (0,1)$,

b) $\nabla u(x) \cdot n(x) = 1 \quad \text{for} \quad x \in \{1\} \times (0,1)$,

c) $\nabla u(x) \cdot n(x) = u(x) \quad \text{for} \quad x \in \{1\} \times (0,1)$.

Theoretical exercise 2. (Upper and lower bounds [7 points])

Consider the weak formulation: Find $u \in H^1_{\Gamma_1,0}(\Omega)$ such that

$$a(u,v) = F(v) \quad \forall v \in H^1_{\Gamma_1,0}(\Omega)$$

with

$$a(u,v) = \int_\Omega \nabla v(x) \cdot D(x) \nabla u(x) \mathrm{d}x + \int_\Omega \left(b(x) \cdot \nabla u(x) + c(x)u(x)\right) v(x) \mathrm{d}x + \alpha \int_{\Gamma_3} u(x)v(x) \mathrm{d}S$$

and

$$F(v) = \int_\Omega f(x)v(x)\mathrm{d}x + \int_{\Gamma_2} g_2(x)v(x)\mathrm{d}S + \alpha \int_{\Gamma_3} g_3(x)v(x)\mathrm{d}S$$

under the Assumptions 3.1 and 3.2.

a) Prove the boundedness of $a(\cdot,\cdot)$, see part (a) of Lemma 3.12, by showing that there exists a constant γ with

$$\int_\Omega \left(b(x) \cdot \nabla u(x) + c(x)u(x)\right) v(x)\mathrm{d}x \leq \gamma \|u\|_1 \|v\|_1 \quad \forall u \in H^1(\Omega), v \in H^1(\Omega)$$

and setting $\beta = \|D\|_{L_\infty} + c_E^2 c_T^2 + \gamma$.

b) Prove the coercivity of $a(\cdot,\cdot)$, see part (b) of Lemma 3.12, for the case

- $c(x) \geq c_0 > 0$ almost everywhere, and for
- $\alpha > 0$ and $\mathrm{meas}(\Gamma_3) > 0$.

c) Show that $F(\cdot)$ is bounded on $H^1(\Omega)$, see part (c) of Lemma 3.12.

Programming exercise 1. (Uniform mesh refinement [10 points])

The creation of large meshes by hand is long-winded. To create a fine mesh from a coarse one, you will implement a uniform mesh refinement routine this week. To this end, a triangle element T is divided into four smaller triangle elements T_1, T_2, T_3 and T_4 by creating new edges which connect the old edge midpoints P_{01}, P_{12} and P_{02} of T.

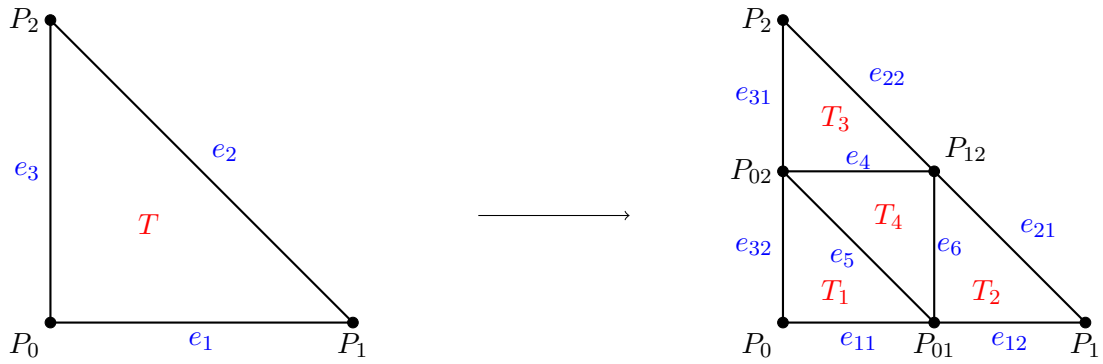

Figure 1: Refinement of the triangle T

Tasks:

Enhance the `Mesh` class by three new methods (variable-names refer to the code from the website)

a) [5 points] Implement the member function `void refineEdge(int edg, int* newEdge1, int* newEdge2, int* newNode)`. It refines the edge with index `edg` by dividing it at the mid point and returns the indices of the two new edges `newEdge1`, `newEdge2` and the index of the mid point node of the old edge `newNode`. For example, if the edge e_1 from the left side of Figure 1 is to be refined, then the edges e_{11} and e_{12} and the node P_{01} have to be created. Their indices are then stored at the memory pointed at by `newEdge1`, `newEdge2` and `newNode`. The procedure of refinement is as follows:

- Check if the edge with index `edg` has already been refined. If this is the case just return the corresponding indices for the two finer edges and the mid point node. This has already been implemented in the incomplete code on the website.
- Create the mid point node of the old edge if it does not already exist (i.e. if `midPointNodeIndex` of `edges[edg]` is -1, this corresponds to the linear Lagrange basis) and add it to the `nodes`-vector.
- Create the two new edges and add them to the `edges`-vector. Copy the boundary condition type and the boundary condition of the old edge to the new edges. When using a linear Lagrange basis (i.e. if `midPointNodeIndex` of `edges[edg]` was -1 in the beginning) you can pass -1 as `midPointNodeIndex` for the two new edges. When using a quadratic Lagrange basis you have to create the mid points of the two new edges, add them to the `nodes`-vector and pass their indices to the two new edges.
- If the old edge had Dirichlet boundary condition, you need to add the edge indices of the two new edges to `DirichletEdgeIndices` (analogously for Neumann boundary conditions).

- Refresh the information for the old edge: Set the `refined` variable to `true` and set the `refinedEdge` indices accordingly.

b) [5 points] Implement the member function `void refineElement(int ele)`. It refines the element with index `ele` by dividing it into four new elements as shown in Figure 1, i.e. you have to create T_1, T_2, T_3 and T_4 from T. In the code, T has to be overwritten by one of the new elements. The procedure is as follows:

- Check if the mid points of the edges e_4, e_5 and e_6 have to be created (i.e. check if the quadratic Lagrange basis is used on the element). This is already implemented in the incomplete code on the website.
- Call `refineEdge` for the edges e_1, e_2 and e_3.
- Create the edges e_4, e_5 and e_6 (and their mid points if they are needed, i.e. if `midPointsofNewEdgesNeeded` is `true`) and add them to the `edges`-vector. Note that these (inner) edges do not have any boundary conditions.
- Create the elements T_1, T_2, T_3 (see Figure 1), copy the material information and basis type of the old element to the new ones and add the elements to the `elements`-vector. Here, the `shareNode` function might be helpful.
- Create the element T_4, copy the material information and basis type of the old element to the new one and overwrite the memory for T in the `elements`-vector with this new element.

Finally write a member function `void Mesh::uniformRefine(int numRefines)` which refines every element `numRefines` times. This function has already been implemented in the incomplete code on the website.

Test your implementation:

- Create a mesh from the file `SampleGrid.txt` by using the method `createTriMeshAndPDEFromFile` from the `PDE` class. Refine it four times and write the resulting mesh to a VTK-file by the `createVTKfile` method of the `Mesh` class.

Feel free to use your own code or the incomplete code from the website. If you use the code from the website make yourself familiar with the design of the `Element`,`Edge` and `Node` classes there.

The following has not been implemented during the exercises but has been added to the incomplete code on the website for your convenience: The variable `basisType` has been added to the `Element` class in the code of the website. Therefore, also the constructors of `Element` have changed. The `basisType` is set by the `createTriMeshAndPDEFromFile` method of `PDE` when creating the mesh. If you use your own code you might need to add this information to get access to the basis type on the elements when refining them.