Scientific Computing I

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

universitätbonn

Exercise sheet 7.

Closing date **3.12.2013**.

Theoretical exercise 1. (Weak form of the biharmonic equation [5 points])

Consider the biharmonic equation: Find u such that

$$\begin{aligned} \triangle(\triangle u) &= f \qquad \forall x \in \Omega \,, \\ u &= 0 \qquad \forall x \in \partial\Omega \,, \\ \frac{\partial u}{\partial n} &= 0 \qquad \forall x \in \partial\Omega \,. \end{aligned}$$

It can be written in weak form: Find $u \in H_0^2(\Omega) = \{u \in H^2(\Omega) : u|_{\partial\Omega} = 0, \frac{\partial u}{\partial n}|_{\partial\Omega} = 0\}$ such that

$$a(u,v) = F(v) \qquad \forall v \in H_0^2(\Omega) \,.$$

a) Determine the symmetric bilinear form $a(\cdot,\cdot)$.

b) Show that $a(\cdot,\cdot)$ is bounded on $H_0^2(\Omega)$.

c) Bonus question: What do you need to also show coercivity?

Theoretical exercise 2. (Weak form of the magnetostatics equation [5 points])

The equation reads: Find $\vec{u}$ such that

$$\begin{aligned} -\operatorname{curl}(\operatorname{curl}\vec{u}) + \kappa \cdot \vec{u} &= \vec{f} \qquad \forall x \in \Omega \,, \\ (\vec{u} \times \vec{n}) \times \vec{n} &= 0 \qquad \forall x \in \partial\Omega \,, \end{aligned} \tag{1}$$

where $\vec{n}$ is the outer normal vector. Now let

$$\begin{aligned} H(\operatorname{curl},\Omega) &= \{\vec{u} \in (L_2(\Omega))^3 : (\nabla \times \vec{u}) \in (L_2(\Omega))^3\} \,, \\ H_0(\operatorname{curl},\Omega) &= \{\vec{u} \in H(\operatorname{curl},\Omega) : (\vec{u} \times \vec{n}) \times \vec{n} = 0 \quad \text{on} \quad \partial\Omega\} \,. \end{aligned}$$

Our weak formulation of (1) reads: Find $\vec{u} \in H_0(\operatorname{curl},\Omega)$ such that

$$a(\vec{u},\vec{v}) = F(\vec{v}) \qquad \forall \vec{v} \in H_0(\operatorname{curl},\Omega) \,.$$

a) Determine the symmetric bilinear form $a(\cdot,\cdot)$.

b) Show that $a(\cdot,\cdot)$ is bounded on $H(\operatorname{curl},\Omega)$.

c) Show that $a(\cdot,\cdot)$ is coercive.

Theoretical exercise 3. (Weak form of Lamé's equation [5 points])

The stationary case of Lamé's equation in strong form reads

$$\begin{aligned} -(\lambda+\mu)\operatorname{grad}\operatorname{div}\vec{u} - \lambda\Delta\vec{u} &= \vec{f} \quad \text{in} \quad \Omega \,, \\ \vec{u} &= 0 \quad \text{on} \quad \partial\Omega \end{aligned}$$

for $\lambda, \mu > 0$ and can be written in weak form for $\vec{u} \in \left(H_0^1(\Omega)\right)^3$ with

$$a(\vec{u},\vec{v}) = F(\vec{v}) \quad \forall \vec{v} \in \left(H_0^1(\Omega)\right)^3 \,.$$

a) Determine the symmetric bilinear form $a(\cdot,\cdot)$.

b) Show that $a(\cdot,\cdot)$ is bounded on $\left(H_0^1(\Omega)\right)^3$.

c) Bonus question: Show that $a(\cdot,\cdot)$ is coercive.

Programming exercise 1. ((Easy) assemblation of the global stiffness matrix [10 points])

Reminder: We want to solve

$$\nabla \cdot (D(x,y)\nabla u(x,y)) + c(x,y) \cdot u(x,y) = f(x,y)$$

on some domain Ω with piecewise (i.e. elementwise) constant D, c and f (and with certain boundary conditions).
Discretizing u by

$$u(x,y) = \sum_{i=1}^{N} \alpha_i \phi_i(x,y)$$

for some basis $\phi_i, i = 1, \ldots, N$ and using the same basis for the test function space, the weak formulation (when omitting boundary integrals) reads

$$\begin{aligned} &\sum_{j=1}^{N} \alpha_j \int_\Omega (\nabla\phi_i(x,y))^T D(x,y) \nabla\phi_j(x,y) \mathrm{d}x\mathrm{d}y \; + \sum_{j=1}^{N} \alpha_j \int_\Omega c(x,y)\phi_i(x,y)\phi_j(x,y)\mathrm{d}x\mathrm{d}y \\ = \; &\int_\Omega f(x,y)\phi_i(x,y)\mathrm{d}x\mathrm{d}y \quad \forall i = 1, \ldots, N. \end{aligned}$$

Therefore, the system $K\vec{\alpha} = \vec{f}$ has to be solved with matrix entries

$$K_{ij} = \int_\Omega (\nabla\phi_i(x,y))^T D(x,y) \nabla\phi_j(x,y) \mathrm{d}x\mathrm{d}y \; + \int_\Omega c(x,y)\phi_i(x,y)\phi_j(x,y)\mathrm{d}x\mathrm{d}y$$

and vector entries

$$\vec{f_i} = \int_\Omega f(x,y)\phi_i(x,y)\mathrm{d}x\mathrm{d}y.$$

Here, K is called global stiffness (plus mass-) matrix and $\vec{f}$ is called global load vector. As you will see in the lecture, you can decompose the integrals into sums of local integrals on the elements, i.e.

$$K_{ij} = \sum_{T \in \text{Elements}} \int_T (\nabla\phi_i(x,y))^T D(x,y) \nabla\phi_j(x,y) \mathrm{d}x\mathrm{d}y + \int_T c(x,y)\phi_i(x,y)\phi_j(x,y)\mathrm{d}x\mathrm{d}y$$

and

$$\vec{f_i} = \sum_{T \in \text{Elements}} \int_T f(x,y)\phi_i(x,y)\mathrm{d}x\mathrm{d}y.$$

Therefore you can assemble the global stiffness matrix and load vector by summing up over all finite elements and calculating the local stiffness matrices and load vectors there. However, you need to identify the global index $i \in \{1, \ldots, N\}$ that a local basis function belongs to.

Tasks:

a) [10 points] Enhance the `PDE` class by the method `CSRMatrix* generateGlobalStiffnessMatrixAndLoadVector(double* loadVector, int loadVecSize)`. It assembles the global stiffness matrix as `CSRMatrix` (see exercise sheet 1) and returns a pointer to it. It also assembles the global load vector and stores it at the memory of size `loadVecSize` pointed at by `loadVector`.

For this you just have to iterate over all elements in the mesh, call `generateLocalStiffnessMatrixAndLoadVector` and add the results for the current element to the corresponding entries in the global stiffness matrix and load vector.

Note: As the number of (global) basis functions equals the number of nodes in the mesh, you can just identify a node with a basis function. Therefore the function `getNodesToElement` from the class `Mesh` is very useful to build a mapping from local basis function indices to global ones.

For simplification: You are allowed to create the global stiffness matrix as full $N \times N$ matrix (e.g. by using a $N \times N$ array) and then convert it to CSRMatrix format, e.g. by the `setEntries` function. We will consider a more efficient method which avoids this step next week.

Test your implementation:

- Create a PDE and mesh from `SampleGrid.txt`, enable quadrature by the seven point rule and assemble the global stiffness matrix and load vector. Then call the `printMatrix` routine of `CSRMatrix`.

Feel free to use your own code or the incomplete code from the website.