# Scientific Computing I

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

universitätbonn

## Exercise sheet 8.

Closing date **10.12.2013**.

**Theoretical exercise 1.** (Compute FE solution by hand [5 points])

Consider the introductory example in Section 4.2.1 of the lecture. Compute the system matrix $K$ and the vector of the right hand side $\underline{f}$. Then solve $K\underline{u} = \underline{f}$ for $\underline{u}$ by hand.

**Theoretical exercise 2.** (Quasi-unifom triangulation [5 points])

Show that the following two statements are equivalent formulations to enforce quasiuniform triangulations $\{\mathcal{T}_h\}_h$ based on triangles:

1. There exists a constant $\kappa$ such that

$$\rho_T \geq \kappa^{-1} h_T \qquad \forall\, T \in \mathcal{T}_h, h > 0\,,$$

   where $\rho_T$ denotes the radius of the incircle of $T$ and $h_T$ half of the diameter of $T$.

2. There exists an angle $\delta$, such that

$$\alpha(T), \beta(T), \gamma(T) \geq \delta \qquad \forall\, T \in \mathcal{T}_h, h > 0\,,$$

   where $\alpha(T), \beta(T), \gamma(T)$ denote the interior angles of $T$.

**Theoretical exercise 3.** (Gauss's theorem for weakly differentiable functions [5 points])

We want to show that for $u \in H^2(\Omega)$ and $v \in H^1(\Omega)$, it holds that

$$\int_\Omega (-\triangle u) v \mathrm{d}x = \int_\Omega \nabla u \cdot \nabla v \mathrm{d}x - \int_{\partial\Omega} \frac{\partial u}{\partial \vec{n}} v\,, \tag{1}$$

where $\vec{n}$ is the outward-pointing normal vector. This is the Gauss divergence theorem for weakly differentiable functions.
By using a density argument and a trace theorem, it can be shown that

$$\int_\Omega \nabla \cdot \vec{u} \mathrm{d}x = \int_{\partial\Omega} \vec{u} \cdot \vec{n}$$

for $\vec{u} \in (W^{1,1}(\Omega))^n$. Your task is to complete the proof by the following two steps:

a) Show that for $v, w \in H^1(\Omega)$ and $i = 1, \ldots, n$, we have

$$\int_\Omega \left(\frac{\partial v}{\partial x_i}\right) w \mathrm{d}x = -\int_\Omega v \left(\frac{\partial w}{\partial x_i}\right) \mathrm{d}x + \int_{\partial\Omega} v w \vec{n}_i\,.$$

b) Use the result in (a) to show (1).

**The closing date for submission of the programming exercise is the 17th of December!**

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

Last week you implemented the function
`CSRMatrix* generateGlobalStiffnessMatrixAndLoadVector(double* loadVector, int loadVecSize)` using a full matrix structure. This might work for small problem instances but will fail for larger ones. This week we will consider an efficient assemblation of the global stiffness matrix.

**Tasks:**

a) [10 points] Change the function
`CSRMatrix* generateGlobalStiffnessMatrixAndLoadVector(double* loadVector, int loadVecSize)` from last week such that the assemblation of the `CSRMatrix` takes place in an efficient way this time. For $N$ basis functions you should achieve memory use and a runtime which are at most $\mathcal{O}(N \log N)$ (at least in the average case if you use algorithms with randomness). The functionality should stay the same. In fact, nothing has to be changed for the assemblation of the load vector.

One possible way would be to implement a helper-structure that first stores the non-zero entries of all local stiffness matrices (together with the information of their correct global positions) in a long array. Afterwards you can sort them according to their lexicographical order in the global stiffness matrix (e.g. by a quick sort algorithm with a corresponding lexicographic comparison of values) and finally iterate over them to add up values with the same position in the global matrix. Then you can call the `setEntries` routine of `CSRMatrix` with the results.

However, you do not have to chose this method as long as you come up with an efficient algorithm to assemble the global matrix.

Test your implementation:

- Create a PDE and mesh from `SampleGrid.txt`, uniformly refine the mesh 8 times, enable quadrature by the seven point rule and assemble the global stiffness matrix and load vector. Then use `multiplyInto` of `CSRMatrix` to multiply the stiffness matrix with the unit-vector $(1, 0, \ldots, 0)$.

  You can also try to do this with the old algorithm which should fail due to the large storage requirements.

**Feel free to use your own code or the incomplete code from the website. Note that the closing date for submission of the programming exercise is the 17th of December.**