Scientific Computing I

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

Exercise sheet 10.

Closing date **7.1.2014**.

Theoretical exercise 1. (Kronecker-product [7 points])

Let $A \in \mathbb{R}^{k\times l}$ and $B \in \mathbb{R}^{m\times n}$. Then, the Kronecker-product of the matrices A and B is given by

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B & \dots & a_{1l}B \\ a_{21}B & a_{22}B & \dots & a_{2l}B \\ \vdots & & \ddots & \\ a_{k1}B & a_{k2}B & \dots & a_{kl}B \end{pmatrix} \in \mathbb{R}^{km\times ln}\,.$$

Show that the following relations hold for $\alpha \in \mathbb{R}, A \in \mathbb{R}^{k\times l}, B \in \mathbb{R}^{m\times n}, C \in \mathbb{R}^{k\times l}, D \in \mathbb{R}^{l\times s}, E \in \mathbb{R}^{n\times r}$.

a) $(\alpha A) \otimes B = A \otimes (\alpha B) = \alpha(A \otimes B)$

b) $(A \otimes B)^T = A^T \otimes B^T$

c) $(A + C) \otimes B = A \otimes B + C \otimes B$

d) $(A \otimes B)(D \otimes E) = (AD) \otimes (BE)$

e) $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$

Theoretical exercise 2. (Structure of mass- and stiffness-matrices [7 points])

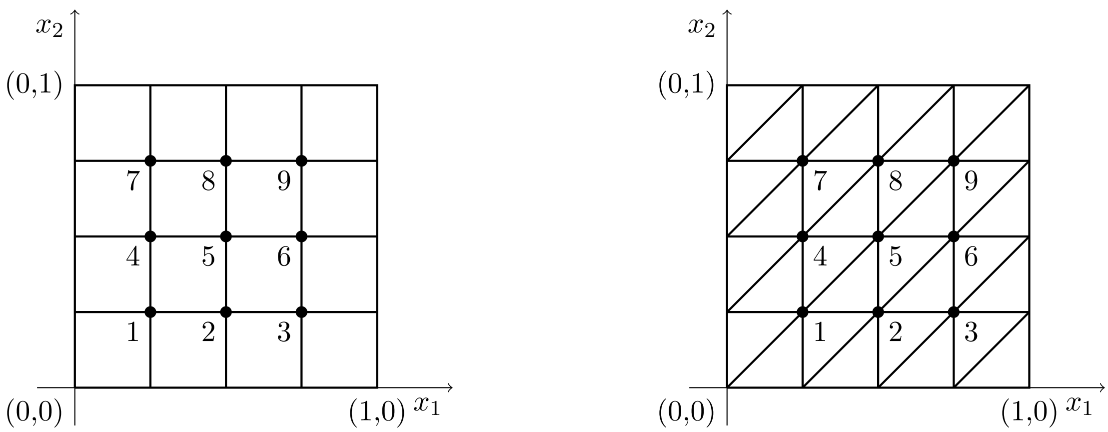

Figure 1: Triangulation for bilinear elements (left) and for linear triangle elements (right)

We are given the Poisson problem

$$-\triangle u = 1 \quad \text{on} \quad (0,1)^2$$

with homogeneous boundary conditions $u = 0 \quad \text{on} \quad \partial(0,1)^2$. See Fig. 1 for two possible triangulations of $(0,1)^2$ suited for bilinear elements and linear triangle elements. We denote the basis functions that belong to the nine inner vertices by $(\varphi_i)_{i=1}^9$.

a) Show that for the bilinear element case, the following holds: The mass matrix $M \in \mathbb{R}^{9\times 9}$ with

$$(M)_{ij} = (\varphi_i, \varphi_j)_{L_2((0,1)^2)} \quad \text{for} \quad i,j = 1,\dots,9$$

can be written as the Kronecker-product of two $\mathbb{R}^{3\times 3}$-matrices.

b) Show that for both the bilinear element case and the linear triangle element case, the following holds: The stiffness matrix $K \in \mathbb{R}^{9\times 9}$ with

$$(K)_{ij} = (\nabla\varphi_i, \nabla\varphi_j)_{L_2((0,1)^2)} \quad \text{for} \quad i,j = 1,\dots,9$$

can be written as the sum of two Kronecker-products of two $\mathbb{R}^{3\times 3}$-matrices.

c) Generalize above results to n^2 quadratic patches or $2n^2$ triangles.

Theoretical exercise 3. (Bonus: Fundamental Lemma of Calculus of Variations [5 points])

Let $\Omega \subset \mathbb{R}^n$ be an a connected and open set and $u \in L_{1,loc}(\Omega)$. Furthermore, $k \in \mathbb{N}_0$. If $D^\alpha u = 0$ for an $\alpha \in N_0^n$ with $|\alpha| = k$, then u is equal almost everywhere to a polynomial of order $k-1$.

Prove the Lemma stated above with the hints you got in your tutoring session.

The closing date for submission of the programming exercise is the 14th of January!

Programming exercise 1. (Incorporation of boundary conditions and solving a PDE [20 points])

After assembling the full stiffness matrices over the last weeks we now have to incorporate the Dirichlet and Neumann boundary conditions. In a final step we will then put all pieces together to result with a Finite Element program.

Tasks:

a) [10 points] Implement the member function `void incorporateBoundaryConditions(CSRMatrix* stiffnessMatrix, double* loadVector)` of the class `PDE`. As parameters you pass a pointer to the already assembled stiffness matrix and the load vector.

First, iterate over the `neumannEdgeIndices` of `Mesh` which lie on the finest level, i.e. for which `isRefined` is `false`. For every corresponding edge e you have to compute the Jacobi determinant J of the linear transformation from the "interval" $[0,1]$ on the x-axis (i.e. y-coordinate is 0) to the edge e such that $(0,0)$ is mapped to the first edge-node and $(1,0)$ is mapped to the other one. Therefore $|J|$ is just the length of the edge e. Then call `calcBoundaryIntegral` of `Basis` for every node of e (with the corresponding local node index as parameter) with $|J| \cdot a$ as `factor` and add this value at the correct position of the load vector. Here, a is the value of the Neumann boundary for the edge e.

Now, set $g := 10^{30} \cdot n \cdot M$ with n being the number of nodes in the mesh and M being the largest absolute value of an entry of the stiffness matrix. Then, iterate over the `DirichletEdgeIndices` of `Mesh` which lie on the finest level. For every node index i of a node of the corresponding edge e add g to the diagonal entry K_{ii} of the stiffness matrix and add $g \cdot b$ to the corresponding load vector position. Here, b is the value of the Dirichlet boundary for the edge e.

Remark: For your convenience the member functions `addToDiagonal(int i, double value)` and `getMaxAbsEntry()` have been added to the implementation of the `CSRMatrix` class.

b) [10 points] Complete your finite element program: Implement the member function `void solvePDEFromFile(const char* infilename, int numMeshRefinements, int maxIt, double eps, const char* outfilename)` of `PDE`. The function should read the file specified by `infilename`, refine the created mesh `numMeshRefinements` times, assemble the global stiffness-matrix and load vector, incorporate the boundary conditions, solve the resulting system with a CG-algorithm for CSR-matrices (exit after `maxIt` iterations or if the relative error is below `eps`) and write the result to a VTK-file called `outfilename`.

Here, the VTK-file should like this:

```
# vtk DataFile Version 3.0
PDE solution
ASCII
DATASET UNSTRUCTURED_GRID
POINTS N double
coord0 coord1 0.0
coord2 coord3 0.0
⋮
CELLS E X
L node1 node2 node3 node4 node5 node6
L node7 node8 node9 node10 node11 node12
⋮
CELL_TYPES E
T
T
⋮
POINT_DATA N
SCALARS u(x,y) double 1
LOOKUP_TABLE default
s1
s2
⋮
```

Here, N denotes the number of nodes. `coord0` and `coord1` denote the x- and y-coordinates of the first node. The other nodes follow analogously. E denotes the number of elements. L denotes the number of local basis functions (i.e. 3 for the linear and 6 for the quadratic Lagrange basis) and $X = E \cdot (L+1)$. `node1` denotes the index of the first node of the first element. `node4 node5 node6` and `node10 node11 node12` are optional and have only to be given if the corresponding element is quadratic. The other elements follow analogously. The number T is 5 for linear elements and 22 for quadratic elements (this is a VTK-internal numbering). It has to appear E times. `s1` is the value of the solution of the PDE at node 1. The other node values follow analogously.

Test your implementation:

- a) Enable quadrature (use the seven point rule) and call your function `void solvePDEFromFile(const char* infilename, int numMeshRefinements, int maxIt, double eps, const char* outfilename)` for the file `SampleGrid.txt`. This resembles the Poisson problem

  $$\Delta u = 1$$

  on $[0,1]^2$ with $u = 0$ on $\partial[0,1]^2$ discretized by quadratic Lagrange elements. The mesh should be refined six times. The CG-algorithm should be called with parameters `maxIt`$= 10^6$ and `eps`$= 10^{-20}$. Visualize the result (e.g. by paraview).

- b) To compare the diagonally preconditioned CG-algorithm to a standard CG algorithm run the same routine for a non-preconditioned CG algorithm (you can just set all entries of `diag` to 1 instead of calling `genDiag` in `pcCG` of `CSRMatrix`). Compare the number of iterations.

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 14th of January.