Scientific Computing I

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

universitätbonn

Excercise sheet 5.

Closing date **19.11.2013**.

Theoretical exercise 1. (Quadrature rules for continuous functions [5 points])

Let

$$Q_n(f) = \sum_{i=1}^{n} w_i^{(n)} f(x_i^{(n)}) , \quad n = 1, 2, \ldots$$

be a sequence of quadrature rules on $[a, b]$ with the following properties:

1. The quadrature rules converge for all polynomials, i.e., for all polynomials p we have
$$\lim_{n\to\infty} Q_n(p) = \int_a^b p(x)\mathrm{d}x . \tag{1}$$
2. There exists a constant C with
$$\sum_{i=1}^{n} |w_i^{(n)}| \leq C , \quad \forall n . \tag{2}$$

a) Show that the quadrature rules Q_n converge for all functions $f \in C[a, b]$, i.e.,

$$\lim_{n\to\infty} Q_n(f) = \int_a^b f(x)\mathrm{d}x .$$

Hint: Use the Weierstrass approximation theorem.

b) Prove that the conditions (1) and (2) are satisfied for the Gauss quadrature rules.

Theoretical exercise 2. (Weak derivatives [5 points])

Consider the function u on the domain $\Omega = (-1, 1)$ and the weak derivative $v = D^\alpha u$ with

$$\int_\Omega u(x)D^\alpha\phi(x)\mathrm{d}x = (-1)^\alpha \int_\Omega v(x)\phi(x)\mathrm{d}x \quad \forall \phi \in C_0^\infty(\Omega) . \tag{3}$$

a) Let

$$u = \begin{cases} \frac{1}{2}x^2 + x + 1 & \text{for} \quad x < 0 , \\ -\frac{1}{2}x^2 + x + 1 & \text{for} \quad x \geq 0 . \end{cases}$$

Does u have a second derivative in the strong or the weak sense? What is it?

b) Consider the Heaviside function

$$u(x) = \begin{cases} 0 & \text{for} \quad x < 0 , \\ 1 & \text{for} \quad x \geq 0 . \end{cases}$$

Show that no weak derivative Du exists!

Theoretical exercise 3. (Sobolev spaces [5 points])

Consider the function $u(x) = \sqrt{x}$ on $\Omega = (0, 1)$.
For $k \in \{0, 1, 2\}$, what is the largest $p \in \mathbb{N}$ such that $u \in W^{k,p}(\Omega)$ holds?

Theoretical exercise 4. (Norm equivalence [5 points])

Show that for $u \in W^{1,p}(\Omega)$ an equivalent norm is defined by

$$|||u|||_{1,p} = \|u\|_{L_p(\Omega)} + \|Du\|_{L_p(\Omega)} .$$

Programming exercise 1. (Reading a PDE from a file and local stiffness matrices [10 points])

The overall result of the programming exercises will be a code to solve the PDE

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

for certain (Neumann and Dirichlet) boundary conditions on $u : \mathbb{R}^2 \to \mathbb{R}$ by a triangular finite element approach. The diffusion coefficients we employ look like this:

$$A(x, y) = \begin{pmatrix} d_{11}(x, y) & 0 \\ 0 & d_{22}(x, y) \end{pmatrix}.$$

The task is simplified by assuming that the material coefficients A, c and f are constant on each element.

Tasks:

a) [5 points] First we adress the automatic PDE and mesh generation from an input file which contains information about the finite elements, the material and the boundary conditions. To this end, implement a class/struct `Material`. It contains the (constant) coefficients `d11, d22, c, f` and the necessary `get`- and `set`-routines. This class is already fully implemented in this week's code framework. Now create a class/struct `PDE`. You will need your `Mesh` and `Basis` classes/structs from the last exercises (or the corresponding sample solutions from the website). Remark: It makes sense to declare the `Mesh` class a friend of `PDE`. The `PDE` class/struct contains the following information:

   - `std::vector<double> dirichletBCs` – The values of different Dirichlet boundary conditions.
   - `std::vector<double> neumannBCs` – The values of different Neumann boundary conditions.
   - `std::vector<Material> materials` – The different material parameters which are needed.
   - `Mesh* mesh` – A pointer to the mesh that the PDE is solved on.
   - `Basis basis` – The basis which is used on the finite elements.

   Implement the following member functions:

   - `Mesh* getMesh()` – Returns the mesh-pointer
   - `void createTriMeshAndPDEFromFile(const char* filename)` – Creates a mesh and sets the mesh information, material parameters and boundary conditions according to the file `filename`. An example input file can be found on the website. The file format has to be adhered to:

– In general an input file looks like this:

```
Materials:  NumberofMaterials
Materialnr d11 d12 d21 d22 c f

Dirichlet:  NumberofDirichletBCs
DirichletBCnr value

Neumann:  NumberofNeumannBCs
NeumannBCnr value

Nodes:  NumberofNodes
Nodenr Coordinate1 Coordinate2

Edges:  NumberofEdges
Edgenr Nodenr1 Nodenr2 MidPointNodeNr bcType DirichletOrNeumannBCnr

Elements:  NumberofElements
Elementnr Edgenr1 Edgenr2 Edgenr3 Materialnr
```

– Note that `d12`=`d21`$= 0$, but they are included in the input file anyhow.
– The `Nodenr` has to begin from 1 and go up to `NumberofNodes` (internally they will be stored from 0 up to `NumberofNodes`-1), analogously for `Edgenr`, etc.
– The `MidPointNodeNr` for edges has to be specified for quadratic basis functions. For linear functions it is set to -1 for every edge.
– The `bcType` and `DirichletOrNeumannBCnr` fields for edges are optional.
– `bcType` is 1 for Dirichlet and 2 for Neumann boundary.

Test your implementation:

- Create a mesh from the file `samplePDE.txt` from the website and create the VTK file for the mesh.

b) [5 points] Enhance your `PDE` class. You will need the classes/structs `Basis` and `IntegrationRule` for this. Implement the following member functions:

- `void enableQuadrature(IntegrationRule::RuleName ruleName)` – enables numerical quadrature for the member `basis`.
- `void disableQuadrature()` – disables numerical quadrature for the member `basis`.
- `void generateLocalStiffnessMatrixAndLoadVector(int ele, double** stiffnessAndMassMatrix, double* loadVector, int matrixSize)` – Returns the matrix $S + M$ where S is the stiffness- and M the mass-matrix and a load vector for one finite element with index `ele`. `matrixSize` is the size of `loadVector` and the number of rows (or columns) of `stiffnessMatrix`.

Instruction on the last task: You have learned on the exercise sheets 2 and 3 how to calculate the mass- and stiffness matrices and load vectors on the reference triangle $\hat{T}$. The integrands are the same this time, only the integration domain changed. This time you integrate over an element T given by it's 3 corner nodes $a^1, a^2, a^3 \in \mathbb{R}^2$. Therefore you have to calculate the absolute value of the 2×2 Jacobi determinant $|\det J|$ of the linear transformation from $\hat{T}$ to T.

Then - given your material parameters `d11`, `d22`, `c` and `f` - you can call the routines you already implemented for `basis` with the `factor` variable set to

- `f` $\cdot\ |\det J|$ for the load vector,

- `c` $\cdot$ $|\det J|$ for the mass matrix,
- $|\det J|$ for the stiffness matrix.

When applying the domain transformation to the stiffness integral a simple calculation yields that you need to pass $(J^{-1})^T$ as parameter `A` to get the right result.

Test you implementation:

- Create a PDE and mesh from `samplePDE.txt`. Calculate and print out the 3×3 stiffness- (plus mass-) matrices and the load vectors for all 4 elements.

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