Interdisziplinäres Zentrum
für Wissenschaftliches Rechnen (IWR)
Ruprecht–Karls–Universität Heidelberg

Prof. Dr. Vincent Heuveline
Dr. Philipp Gerstner

Programming Sheet 3

04.12.2020 – 11.12.2020

Finite Elements (Winter Term 2020/2021)

Important Notes:
The programming exercises have to be solved within the provided C++ framework. Please add detailled comments to your code. We only evaluate codes, which compile without any bugs and do not exhibit run time errors. *We do not debug your code!* You can compile your code by opening a terminal within the Code directory and typing `make`.
Those parts of the code that should be modified in this exercise are marked with `TODO`.
After you have put your solution into the provided framework, create an zip or tar archive and upload it to moodle. In addition to your code, this archive should contain a PDF file, in which the terminal output (and plots) created by your program are stated.
We test your solution on Ubuntu 20.04 or Linux Mint XFCE. If you want to ensure that your solution compiles successfully on our computers, you should use one of these OS (possibly installing it on a virtual machine, e.g. VirtualBox). Further, you should install the package `build-essential` (`sudo apt install build-essential`) for compiling C++ codes on Debian systems (includes Ubuntu and Linux Mint).

Exercise 1 (Programming). Boundary Values (5+5=10 Points)
In this exercise, we consider the handling of boundary conditions, in particular those of so-called *Dirichlet* type:

$$u_h(x) = g(x) \text{ for } x \in \Gamma_D$$

where $\Gamma_D \subset \partial\Omega$ denotes the Dirichlet part of the boundary and $u_h \in V_h$ is a Finite Element function. The underlying idea is the following: first, one determines all Dirichlet vertices $\Gamma_{D,h} := \{j \in \{1, \ldots N\} \colon a_j \in \Gamma_D\}$. Then, one sets the corresponding entries in the coefficient vector of u_h (see programming sheet 2) according to $u_j := g(a_j)$ for $j \in \Gamma_{D,h}$.

a. Implement an algorithm to determine the nodes in the grid, which lie on the boundary, in the function `void GRID::compute_boundary_flag()` in the file `grid.cc`. This function shall fill the vector `std::vector<bool> boundary_flag_`, which the class `GRID` has as member variable, in the following way: For each node in the grid `boundary_flag_` contains an entry, i.e. its length is `num_nodes()`. If a node is on the boundary, the corresponding component in `boundary_flag_` shall be set to `true`, otherwise to `false`.

The `main` is implemented, such that `boundary_flag_` is written to a `FE_VEC`, which is visualized, i.e. you can check the correctness of your algorithm in ParaView. The data set is called "Boundary flag".

b. Write an algorithm to find those nodes on the boundary, where Dirichlet boundary conditions are prescribed, and to compute the corresponding values in the function

```
void GRID::compute_dirichlet_nodes_and_values(
        void (*function_to_call)(GRID&, const std::vector<int>&, std::vector<double>&),
        std::vector<int> &dirichlet_nodes,
        std::vector<double> &dirichlet_val
                                       );
```

in the file `grid.cc`. To this end, `dirichlet_nodes` shall contain the indices of the Dirichlet nodes and `dirichlet_val` the corresponding values, i.e. these vectors have length "number of Dirichlet nodes" and **not** `num_nodes()`. (The values are set in the `main` function via the `setValues` function of `FE_VEC`.) To achieve this, proceed in the following way:

(a) Iterate over all triangles in the grid.

(b) Check for each edge in the current triangle, if **both** nodes are on the boundary of the grid (this is why you have computed `boundary_flag_` in exercise **a.**).

(c) If yes, create `std::vector<int> bd_pts` of length 2, which contains the indices of the nodes at the ends of the current edge, as well as `std::vector<double> bd_vals` of length 0 (i.e. it is **empty**) and evaluate the function `function_to_call`, which has been passed as function pointer, in the following way:

```
(*function_to_call)(*this, bd_pts, bd_vals);
```

(An example for such a function, which is used in the current `main` function, can be found in `exercise_sheet_3.h`) This function works according to the principle, that *either* both nodes are Dirichlet nodes - the output vector with the `double` values has length 2 - *or* both nodes are no Dirichlet nodes and the output vector is **empty**.

(d) If the vector `std::vector<double> bd_vals` is **not empty any more**, i.e. both nodes at the ends of the current edge are Dirichlet nodes, add the indices and values of the nodes to `dirichlet_nodes` and `dirichlet_val`.

You can check the correctness of your algorithm in this exercise via ParaView, too: the given function in `exercise_sheet_3.h` prescribes on the "upper" ($y = 1$) and "right" ($x = 1$) boundary the function

$$g(x, y) = \cos(x)\cos(y).$$

The data set is called "Dirichlet BC".