Numerische Methoden in der Strömungsmechanik October 24, 2023
OvGU WiSe 2023/24

Practical Exercises 2

In this problem sheet you learn how to set the right-hand side, boundary data and the domain. The templates are found in the folder `problem2`. The given code solves the Poisson equation

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega, \\ u &= g \quad \text{on } \partial\Omega, \end{aligned}$$

on the unit square with $f = 1$ and Dirichlet data $g = 0$ on the boundary. Compile and run the program as in Problem 1, then look at the solution in ParaView.

Problem 2.1 (Setting the right-hand side)

The equation, right-hand side and boundary data are defined in the `ProblemDescriptor`, see the file `problem.h`. For the constant right-hand side $f = 1$, we use the preimplemented class `MyRightHandSide`. For more complicated functions, we have to change the implementation of the right-hand side on our own.

The class `MyRightHandSide` can now be modified in `problem.h`. Instead of the right-hand side $f = 1$, we want to use

$$f(x, y) = \begin{cases} 1 & \text{if } x \geq 0.5, \\ 0 & \text{else.} \end{cases}$$

Implement this right-hand side in `MyRightHandSide` by changing the function `operator()`. In this function the coordinates (x, y) of the current point can be accessed through `v.x()` and `v.y()` and the right-hand side at this point is returned by the function. Compare the solution to the one obtained for constant $f = 1$.

Problem 2.2 (Setting Dirichlet boundary values)

We still work in `problem.h` file. To modify the Dirichlet boundary values, first set the corresponding pointer in `MyProblemDescriptor` from `ZeroDirichletData` to

```
GetDirichletDataPointer() = new MyDirichletData(pf);
```

Now we can specify the boundary function g in the class `MyDirichletData`. Modify this class in `problem.h` such that the boundary values are

$$g(x,y) = x(1-x).$$

As before the variable `v` can be used to access the coordinates of the current point. The Dirichlet values are not directly returned, but instead stored inside the variable `b[0]`. Compare the solution with the one obtained for homogeneous Dirichlet data.

Problem 2.3 (Mixed boundary data)

Now we want to use Dirichlet boundary values on a part Γ^d of the boundary only. On the remaining part $\Gamma^n := \partial\Omega \setminus \Gamma^d$, we want to prescribe the homogeneous Neumann boundary condition

$$\partial_n u = 0.$$

This is the so called *natural* boundary condition for the Laplace equation as

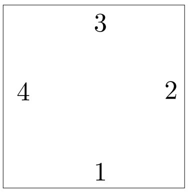
$$(\nabla u, \nabla\phi)_\Omega = (-\Delta u, \phi)_\Omega + \langle \partial_n u, \phi \rangle_{\partial\Omega}.$$

In Gascoigne, every part of the boundary has its own *color* (You learn about defining this colors in the next problem). Here, the square has four boundaries with the colors $1, 2, 3, 4$. For each color we must decide, if we want to use a Dirichlet condition or a Neumann condition.

Look at the file `gascoigne.param` where all paramters are defined. We only specify Dirichlet colors. If a boundary color does not appear in the file, it will be used as Neumann boundary. Now have a look at the `Block BoundaryManager` in the parameter file

```
//Block BoundaryManager

dirichlet      4 1 2 3 4
dirichletcomp    1  1 0
dirichletcomp    2  1 0
dirichletcomp    3  1 0
dirichletcomp    4  1 0
```



In the geometry file `square.inp` there is a “color” value assigned to each boundary line. The entries in the `Block BoundaryManager` tell Gascoigne to set Dirichlet boundary values on all lines with colors 1 to 4 (which are all boundary lines). Have a look at the chapter *”Boundary Data“* to understand the details.

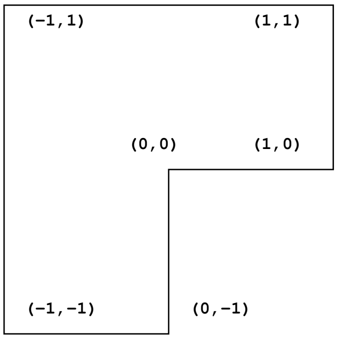
Now change the parameter file, such that Dirichlet data is given only on the left and right parts of the boundary. These parts are indicated by the color values 2 and 4. On the top and bottom part, marked by the color values 1 and 3, we want to prescribe homogeneous Neumann data. In Paraview use the option `Filters->Warp By Scalar`. Press the button `3D` and you can have a 3d-look at your solution. How does the solution and its normal derivative $\vec{n} \cdot \nabla u$ behave at the boundary. Also try to choose other sides for Neumann.

What happens if you use Neumann boundary values on all parts of the boundary? For no Dirichlet values at the whole boundary, the `//Block BoundaryManager` has to be modified to

```
//Block BoundaryManager

dirichlet 0
```

Problem 2.4 (Modifying the computational domain)



In this exercise, we want to solve the Poisson problem on the L-shaped domain Ω_L. The geometry of the domain is specified inside an `.inp`-file, which consists of vertices, lines and quadrilaterals in a text format. Before creating the new domain, have a look at the geometry file for the unit square, `square.inp`, and read the chapter *"Definition of coarse meshes"* in the script to understand its entries.

Create a geometry file `lshape.inp` for the L-shaped domain and solve the Poisson problem with constant right-hand side and homogeneous Dirichlet data on Ω_L. Furthermore, change `gridname` in the `Block Mesh` of the parameter file to `lshape.inp`.

Hint: You will need three quadrilaterals to define the coarsest grid of the L-shaped domain and therefore eight vertices in total. You don't have to list inner lines in the geometry file, but the boundary lines have to be specified as they contain the information which boundary color the line belongs to.

Hint: You can open `.inp`-files in ParaView to visually check if the geometry is correct. Open the file as usual with File → Open. Change Surface to Wireframe in the toolbar. ParaView will color the lines according to their boundary color. Note that this visual check does not guarantee that the .inp-file is indeed a correct geometry file for Gascoigne.

Troubleshooting:

- Make sure that the numbers in the first line of your `.inp`-file are correct. Remember that the second number in that line is the *sum* of the number of boundary lines and quadrilaterals.
- Make sure that the numbering of the vertices (and lines, quads) is consecutive.

- Make sure that the vertices of all quads are specified in a counter-clockwise order.
- Make sure that the boundary colors in the parameter file and the MyDirichletData class matches that of the geometry file.
- No not list internal lines, only lines at the boundary of the domain are given in the `inp`-file.

Problem 2.5 (Extra)

Gascoigne can be used to create curved domains as well. Here, we want to modify the L-shaped domain from Exercise 2.4 in such a way that the lower-right part of the domain between the vertices (0,-1) and (1,0) is curved, i.e. it lies on a circle with radius 1 and midpoint (1,-1).

Therefore, specify a seperate color (e.g. 80) for the two edges of the L-shaped domain that you want to "curve".

The class `MyMeshAgent` is already prepared in the file `problem.h`. Add the following lines to its constructor

```
MyMeshAgent() : MeshAgent()
{
  double r = 1.;
  Vertex2d mp(1., -1.);
  D.BasicInit(mp, r);
  AddShape(80, &D);
}
```

Here, the first 3 lines define the circle that you want to project the edge to. The last line contains the color of the edges you want to project (here: color 80) and the object that describes the circle.

Then, we need to add the mesh agent in `main.cc` to `LocalLoop`. This was already done in the Problem 1. Add the following code to `main.cc` and use `LocalLoop` instead of `StdLoop` :

```
class LocalLoop : public StdLoop {
  public:
  void BasicInit(const ParamFile& paramfile,
                 const ProblemContainer* PC,
                 const FunctionalContainer* FC) {
    GetMeshAgentPointer() = new MyMeshAgent;
    StdLoop::BasicInit(paramfile, PC, FC);
  }
};
```

Run the program again on this curved domain. Do not forget to set the boundary values on the new boundary color in `gascoigne.param`.