Chair for Computational Analysis of Technical Systems

RWTH AACHEN UNIVERSITY

Isogeometric Analysis

Wintersemester 15/16

Exercise Sheet 7
Poisson Problem

The main objective of this exercise will be to solve a Poisson problem on a NURBS patch with homogeneous boundary conditions. More specifically we are going to solve the weak form of

$$-\Delta u(\boldsymbol{x}) = s(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in \Omega \tag{1}$$

$$u|_{\partial\Omega} = 0\,, \tag{2}$$

with

$$s(\boldsymbol{x}) = \frac{8 - 9\cdot\sqrt{x^2+y^2}}{x^2+y^2}\cdot \sin(2\cdot\arctan(y/x)) \tag{3}$$

on the domain $\Omega \subset \mathbb{R}^2$. In our specific example, Ω will be represented by the NURBS geometry that is generated with `generate_testnurb`.

The plumbing work of the last exercises now allows us to easily incorporate different PDEs. We just need to implement

- an `element_matrix` function named `blk_dudv` that returns $\boldsymbol{A}^{(i)}_{A,B}$, which is in this case given by

$$\boldsymbol{A}^{(i)}_{A,B} = \mathbf{1}_{\texttt{ndof}\times\texttt{ndof}} \int_{\tilde{\Omega}^i} \nabla_{x,y} R_A(\xi,\eta)\cdot\nabla_{x,y} R_B(\xi,\eta)\,|\det JF(\xi,\eta)|\,d\xi d\eta \tag{4}$$

$$\approx \mathbf{1}_{\texttt{ndof}\times\texttt{ndof}} \sum_{j=1}^{\texttt{nquad}} \underbrace{\nabla_{x,y} R_A(\boldsymbol{\xi}_j)}_{=\texttt{dV(:,j)}} \cdot \underbrace{\nabla_{x,y} R_B(\boldsymbol{\xi}_j)}_{=\texttt{dU(:,j)}} \underbrace{|\det JF(\boldsymbol{\xi}_j)|}_{=\texttt{jac\_det(j)}} \cdot w_j\,, \tag{5}$$

  where $\boldsymbol{\xi}_j$ and w_j are the quadrature points resp. weights.

- a `rhs_function` named `rhs_testfun` that returns

$$\boldsymbol{s}^{(i)}_A = \mathbf{1}_{\texttt{ndof}} \int_{\tilde{\Omega}^i} s(\boldsymbol{F}(\xi,\eta)) R_A(\xi,\eta)\,d\xi d\eta \tag{6}$$

$$\approx \mathbf{1}_{\texttt{ndof}} \sum_{j=1}^{\texttt{nquad}} s(\underbrace{\boldsymbol{F}(\boldsymbol{\xi}_j)}_{=\texttt{F(:,j)}}) \underbrace{R_A(\boldsymbol{\xi}_j)}_{=\texttt{V(j)}}\,|\det JF(\boldsymbol{\xi}_j)|\cdot w_j\,. \tag{7}$$

The two functions should have the following design:

```
function matloc = blk_dudv(ndof, quad_points, quad_weight, jac_det, ...
    F, U, dU, V, dV)

matloc = zeros(ndof, ndof);
nquad = size(quad_weight,2);

for idof=1:ndof
    for j=1:nquad
        matloc(idof,idof) = matloc(idof,idof) + YOUR_CODE;
    end
end

end
```

```
function rhsloc = rhs_testfun(ndof, quad_points, quad_weight, ...
    jac_det, F, V, dV)

rhsloc = zeros(ndof, 1);
nquad = size(quad_points,2);

for idof=1:ndof
    for j=1:nquad
        rhsloc(idof) = rhsloc(idof) + YOUR_CODE;
    end
end

end
```

Although both functions have been implemented for general `ndof`, we will restrict ourselves to the `ndof=1` case, since u is a scalar field. In principle we now have everything at hand to assemble the matrix and the right-hand side, but so far we have not incorporated the homogeneous Dirichlet boundary conditions. Since clamped NURBS (open knot vector) have an interpolatory behavior at boundaries, we just have to set the control variables $\boldsymbol{u}_B$ to zero that correspond to the boundary control points, as well as skip the equations corresponding to the weighting function $\boldsymbol{v}_B$ with the same index B.

In the code this is accomplished by filling an array with the indices B of the equations that should be skipped by the linear solver. Combining all these considerations, the Poisson problem can be solved using the following code:

```
function exercise7()

% we want to simulate on 25 elements
refinement_level=5;
ref_nurb = nurb_knot_refinement(generate_testnurb(),refinement_level -1);

% degrees of freedom per shape function
ndof =1;
[mat, rhs] = assemble_matrix_2d(false, ndof, ref_nurb, @blk_dudv, @rhs_testfun);
```

```
n = ref_nurb.number(1);
m = ref_nurb.number(2);

all_dofs = [1 : n*m];

% equation numbers that correspond to weighting functions with support on
% the boundary with eta=0
boundary = YOUR_CODE;
% boundary with eta=1
boundary = union(boundary,YOUR_CODE);
% boundary with xi=0
boundary = union(boundary,YOUR_CODE);
% boundary with xi=1
boundary = union(boundary,YOUR_CODE);

% subtract the boundary set from all degrees
inner_dofs = setdiff(all_dofs,boundary);

% initialize solution vector to zero, including the boundary terms
solution = zeros(n*m,1);

% solve the linear system only for the inner degrees
solution(inner_dofs) = mat(inner_dofs,inner_dofs) \ rhs(inner_dofs);

% reshape the solution into a coefficients array
coeffs = reshape(solution,ndof,n,m);
% remember that the nurb_eval routine needs the coefficients premultiplied
% with the weights
for i=1:ndof
    coeffs(i,:,:) = coeffs(i,:,:).*ref_nurb.coeffs(4,:,:);
end

% show a surface plot of the solution
draw_nurb_surf(ref_nurb, [30 30], ...
    @(xi,eta) nurb_eval(ref_nurb, coeffs, ndof, [xi; eta]));

end
```

The final result should then look similar to Figure 1.

Contact:
Florian Zwicke, M.Sc. · zwicke@cats.rwth-aachen.de

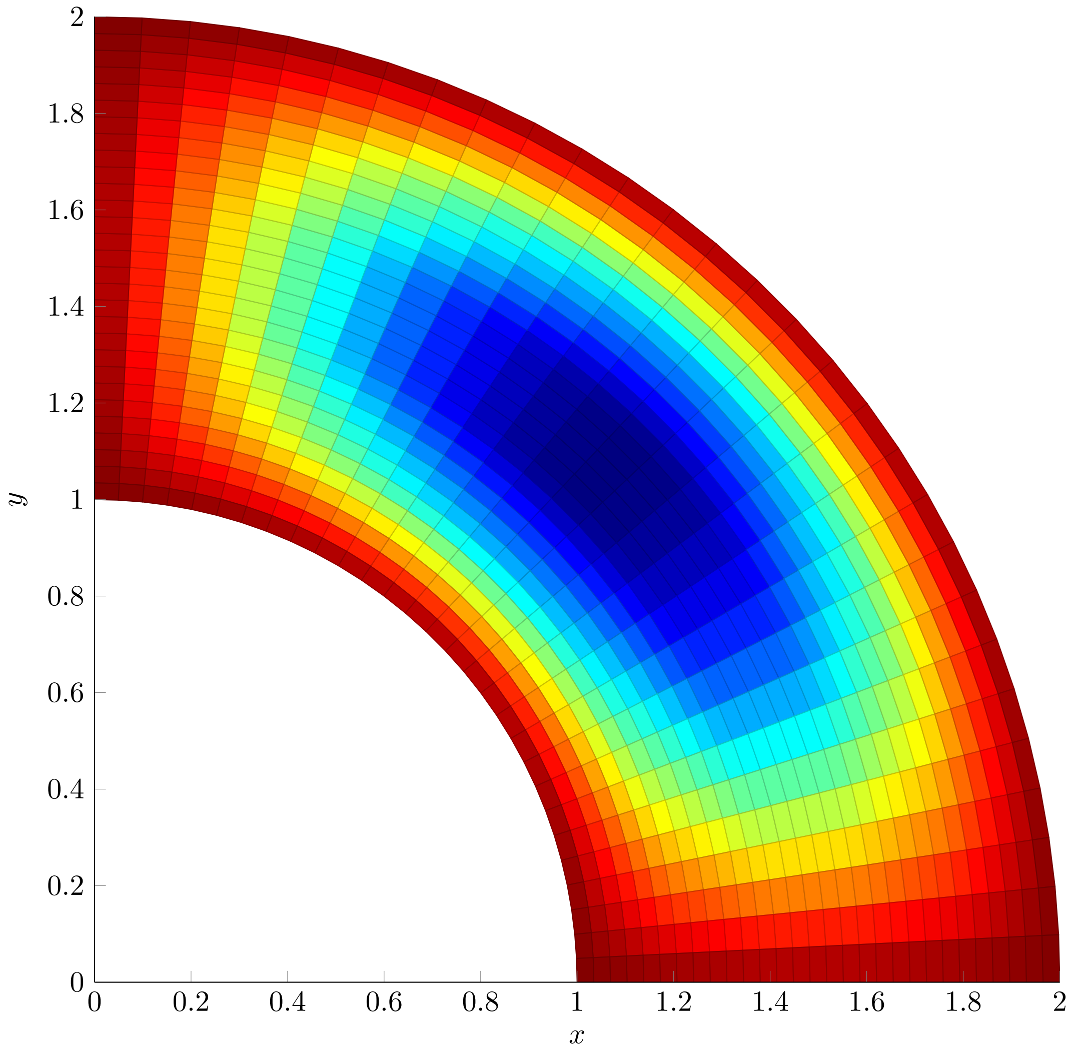

Figure 1: Surface plot of the solution of the Poisson Problem.