**Faculty of Electrical Engineering, Mathematics and Computer Science**
**Numerical Analysis, Special Topics**
**2017/2018 Take Home Assignments**

1. **Theoretical Assignment**

In this assignment, we motivate that functions with jumps are not allowed as finite element basis functions (in fact we motivate (a rigorous proof can be based on the weak derivative, see Bræss for instance) why functions with jumps are not in $H^1(\Omega)$). Although the analysis can be generalised to general domains in $\mathbb{R}^d$, we will deal with a case in a unit circle $\Omega = \{(x,y) \in \mathbb{R}^2 : x^2+y^2 < 1\}$ and with a jump along the circle $\{(x,y) \in \mathbb{R}^2 : x^2+y^2 = R^2\}$, with $0 < R < 1$. Due to the circular geometry, we use polar coordinates to deal with the problem. To this extent, we write the prescription of $u_\varepsilon : \Omega \to [0,1]$ in polar coordinates $(r,\theta)$. Let $\hat{u}_\varepsilon = \hat{u}_\varepsilon(r,\theta)$ be given by the following expression:

$$\hat{u}_\varepsilon(r,\theta) = \begin{cases} 1, & r < R-\varepsilon, \\ \dfrac{1}{2}(1-\dfrac{r-R}{\varepsilon}), & R-\varepsilon \le r \le R+\varepsilon, \\ 0, & r > R+\varepsilon, \end{cases} \tag{1}$$

where $\varepsilon > 0$ is chosen such that $R-\varepsilon \ge 0$ and $R+\varepsilon \le 1$. Further, we introduce the heaviside function $u : \Omega \to [0,1]$ by

$$u(x,y) = \begin{cases} 1, & (x,y) \in \{(x,y) \in \mathbb{R}^2 : x^2+y^2 < R^2\}, \\ 0, & (x,y) \in \{(x,y) \in \mathbb{R}^2 : x^2+y^2 > R^2\}. \end{cases} \tag{2}$$

a Assume that $f$ is differentiable. Show that

$$||\nabla f||^2 = (\frac{\partial f}{\partial r})^2 + \frac{1}{r^2}(\frac{\partial f}{\partial \theta})^2.$$

You may use $\nabla f = \frac{\partial f}{\partial r}\mathbf{e}_r + \frac{1}{r}\frac{\partial f}{\partial \theta}\mathbf{e}_\theta$, where $\mathbf{e}_r$ and $\mathbf{e}_\theta$ are unit vectors in the $r$-and $\theta$-direction, respectively.

b Show that

$$I(\varepsilon) := \int_\Omega ||\nabla u_\varepsilon||^2 d\Omega = \frac{\pi R}{\varepsilon}.$$

**Remark:** From the above relation, it is clear that for a fixed $R$, we can make $I(\varepsilon)$ arbitrarily large. Herewith it is reasonable to assert that $\int_\Omega ||\nabla u||^2 d\Omega$ does not exist. Hence functions in $H^(\Omega)$ are not allowed to have any jumps. Note that one wants to interchange the limit $\varepsilon \to 0$ and integration, which is not allowed formally because the requirements of the Lebesgue convergence theorem are not satisfied. A more rigorous proof is based on the weak derivative.

2. **Theoretical Assignment**

In this assignment, we show that, despite that functions in $H^1(\Omega)$ are not allowed to have any jumps, functions in $H^1(\Omega)$ are not necessarily continuous if the dimensionality is higher than one. To this extent, we deal with a circle $\Omega = B_\varepsilon(\mathbf{0}) = \{(x,y) \in \mathbb{R}^2 : x^2+y^2 < \varepsilon^2\}$, with $0 < \varepsilon < 2$. The circular geometry allows a treatment with polar coordinates.

a Show that the function $f(x,y) = \ln(\ln(\frac{2}{||\mathbf{x}||}))$ ($||\mathbf{x}|| := (x^2+y^2)^{\frac{1}{2}}$) is not continuous in entire $\Omega$.

b Show that the $L^2$-norm of $f(x,y) = \ln(\ln(\frac{2}{||\mathbf{x}||}))$ is finite, that is show that

$$\int_\Omega f^2(x,y)d\Omega < \infty.$$

*Hint: Use polar coordinates and use the substitution* $u = \ln(\frac{2}{r})$*, where* $r = \sqrt{x^2+y^2}$*, and use a converging mayorant inspired by* $\ln(u) < u$ *for* $u > 1$.

c Show that the function $f(x,y) = \ln(\ln(\frac{2}{||\mathbf{x}||}))$ satisfies

$$\int_\Omega ||\nabla f(x,y)||^2 d\Omega = \frac{2\pi}{\ln(\frac{2}{\varepsilon})}, \text{ for } 0 < \varepsilon < 2.$$

*Hint: Use the substitution* $u = \ln(\frac{2}{r})$.

**Remark:** After having answered these questions, it can be concluded in spite of $\ln(\ln(\frac{2}{||\mathbf{x}||})) \notin C^0(\Omega)$, we have $\ln(\ln(\frac{2}{||\mathbf{x}||})) \in H^1(\Omega)$. Hence functions in $H^1(\Omega)$ are not necessarily continuous if the dimensionality is higher than one.

3. **Programming Assignment**

Consider the following convection-diffusion problem:

$$-D\frac{d^2u}{dx^2} + v\frac{du}{dx} = 1, \qquad u(0) = 1, \;\; D\frac{du}{dx}(1) + \kappa(u(1) - u^\infty) = 0. \tag{3}$$

Here $D$, $\kappa$, $u^\infty$ and $v$ are positive constants. The interval $[0,1]$ is divided into $n$ elements (where $n$ is a given positive integer), such that $e_i = [x_i, x_{i+1}]$, for $i \in \{1,\ldots,n\}$. So element $e_i$ has end points (also called vertices) $x_i$ and $x_{i+1}$, where we require $x_1 = 0$ and $x_n = 1$ and $h = 1/(n-1)$. Hence there are $n$ gridpoints and $n-1$ unknowns (or degrees of freedom) because the solution is known on $x_0$. In this lab assignment, the participant develops a finite-element code for 1D in Matlab from scratch for the case of an non-homogeneous essential (Dirichlet) boundary condition, as well as a non-homogeneous Robin condition. The treatment is formal in terms of topology, element matrices and vectors such that the student gets the idea of how finite-element packages are constructed, also regarding the implementation of an essential boundary condition. Once the mesh and topology have been adapted to multi-dimensional problems, then it is relatively straightforward to adjust the code to higher dimensional problems.

**Assignment 1** *Determine the exact solution to the boundary value problem.* ◊

**Assignment 2** *Derive a weak form of the above problem (see equation (3)), where the order of the spatial derivative is minimised. Take care of the boundary conditions.* ◊

We are going to solve this differential equation by the use of Galerkin's Finite Element method.

**Assignment 3** *Write the Galerkin formulation of the weak form as derived in the previous assignment for a general number of elements given by n (hence* $x_n = 1$*). Give the Galerkin equations, that is, the linear system in terms of*

$$S\underline{u} = \underline{f}, \tag{4}$$

*all expressed in the basis-functions,* $f(x)$*,* $\lambda$ *and D. Make a distinction between the first equation (at* $x = 0 = x_1$*) and the remaining equations. Take special care of the second equation in terms of the essential boundary condition.* ◊

**Assignment 4** *Write a matlab routine, called* GenerateMesh.m *that generates an equidistant distribution of meshpoints over the interval* $[0,1]$*, where* $x_1 = 0$ *and* $x_n = 1$ *and* $h = \frac{1}{n-1}$*. You may use x = linspace(0,1,n).* ◊

Further, we need to know which vertices belong to a certain element $i$.

**Assignment 5** *Write a routine, called* GenerateTopology.m*, that generates a two dimensional array, called* elmat*, which contains the indices of the vertices of each element, that is*

$$\begin{array}{l} elmat(i,1) = i \\ elmat(i,2) = i+1 \end{array} \text{, for } i \in \{1,\ldots,n-1\}. \tag{5}$$

$\Diamond$

Next we compute the element matrix $S_{elem}$. In this case, the matrix is the same for each element (because the grid spacing is the same everywhere), that is, if we consider element $e_i$. First we treat the internal line elements, and subsequently, we assess the boundary conditions (boundary point element at $x = 1$ and the Dirichlet boundary condition at $x = 0$).

**Assignment 6** *Derive the element matrix, $S_{elem}$ over a generic internal line element $e_i$ on paper.* $\Diamond$

**Assignment 7** *Write a matlab routine, called* GenerateElementMatrix.m*, in which $S_{elem}$ ($2 \times 2$-matrix) is generated.* $\Diamond$

Subsequently, we are going to sum the connections of the vertices in each element matrix, over all the elements. The result is an $n$-by-$n$ matrix, called $S$.

**Assignment 8** *Write a matlab routine, called* AssembleMatrix.m*, that performs this summation, such that S is first initialised as a zero n-by-n matrix and perform*

$$S(elmat(i,k), elmat(i,l)) = S(elmat(i,k), elmat(i,l)) + S_{elem}(k,l), \tag{6}$$

*for $k,l \in \{1,2\}$ over all elements $i \in \{1,\ldots,n-1\}$. Note that GenerateElementMatrix.m needs to be called for each element. Note that we have $n-1$ elements.* $\Diamond$

Next, we treat the boundary element at $x = 1$.

**Assignment 9** *Compute the boundary element matrix, $S_{belem}$, at $x = 1 = be_n$ (just one entry).* $\Diamond$

**Assignment 10** *Write a matlab routine, called* GenerateBoundaryElementMatrix.m*, in which $S_{belem}$ ($1 \times 1$-matrix in one dimension) is generated (just one entry, so you can use a scalar here).* $\Diamond$

**Assignment 11** *We incorporate the Robin boundary condition at $x = 1$ into the code.*
*Extend the matlab routine* AssembleMatrix.m *with the following lines on the bottom:*

$$\begin{array}{l} GenerateBoundaryElementMatrix \\ \\ S(n,n) = S(n,n) + S_{belem}; \end{array} \tag{7}$$

$\Diamond$

Now, you developed a routine for the assembly of the large matrix $S$ from the element matrices $S_{elem}$ for each element. This procedure is common for the construction of the large discretisation matrices needed in Finite Element methods. The procedure, using the array *elmat* looks a bit overdone and complicated. However, this approach facilitates the application to multi-dimensional problems. The next step is to generate a large right-hand side vector using the similar principles. First, we need the element vector. We start with the internal line-elements.

**Assignment 12** *Derive the element vector over a generic internal line-element $e_i$ on paper.* $\Diamond$

To assemble the right-hand side vector, we proceed as follows

**Assignment 13** *Implementation of the right-hand vector:*

*a Write a matlab routine, called* GenerateElementVector.m*, that gives the vector $f_{elem}$ (column vector of length 2). Here $f_{elem}(1)$ and $f_{elem}(2)$ respectively provide information about node $i$ and node $i+1$, which are the vertices of element $e_i$. This is needed for all elements. Use $f(x) = 1$ here.*

*b Write a matlab routine, called* AssembleVector.m*, that performs the following summation after setting $f = zeros(n,1)$ by:*

$$f(elmat(i,k)) = f(elmat(i,k)) + f_{elem}(k), \tag{8}$$

*for $k \in \{1,2\}$ over all elements $i \in \{1,\ldots,n-1\}$.*

$\Diamond$

Next, we implement the natural boundary condition at $x = 1$ into the right-hand side vector.

**Assignment 14** *Derive the boundary element vector, $f_{belem}$, for the boundary $x = 1 = be_n$ (Note that in this simple one-dimensional case this vector has a length one).* $\Diamond$

**Assignment 15** *Write a matlab routine, called* GenerateBoundaryElementVector.m*, in which $f_{belem}$ (vector of length one, hence you can use a scalar) is generated.* $\Diamond$

**Assignment 16** *We incorporate the Robin boundary condition at $x = 1$ into the code. Extend the matlab routine AssembleVector.m with the following lines on the bottom:*

$$\begin{array}{l} GenerateBoundaryElementVector \\ \\ f(n) = f(n) + f_{belem}; \end{array} \tag{9}$$

$\Diamond$

We are almost done ..., next we implement the Dirichlet boundary condition at $x = 0$.

**Assignment 17** *The essential boundary condition $u(0) = 1$ is implemented by adjusting the set of equations such that the first equation becomes $u_1 = 1$, to this extent, we set $S_{11} = 1$ and $f(1) = 1$ as the first equation and $S(1,:) = 0$ (the remaining elements of the first row are set to zero instead of deleting the entire row as in the book). Furthermore, one likes to preserve symmetry of the discretisation matrix and to this extent, the entries of the first column of S, except $S(1,1)$ are set zero as well. However, before we do this, it follows from the Galerkin equations (see Assignment 2) that the second term of the right-hand side, that is, $f(2)$, needs to be adjusted to*

$$f(2) \longrightarrow f(2) - S(2,1) * 1. \tag{10}$$

*This needs to be done before the first row and first column of the matrix S are adjusted. We proceed as follows. Write a matlab routine* ProcessEssentialBoundaryConditions.m *that contains the following statements:*

$$\begin{array}{l} f(2) = f(2) - S(2,1) * 1; \\ \\ S(:,1) = 0;\ S(1,:) = 0;\ S(1,1) = 1; \\ \\ f(1) = 1; \end{array} \tag{11}$$

$\Diamond$

**Assignment 18** *Run the assembly routines and the routine* ProcessEssentialBoundaryConditions.m *to get the matrix S and vector f for* $n = 100$. ◊

**Assignment 19** *Write the main program that gives the finite-element solution. Call the main program* femsolve1d.m. *The program femsolve1d.m should consist of*

*clear all*
*GenerateMesh*
*GenerateTopology*
*AssembleMatrix*
*AssembleVector*
*ProcessEssentialBoundaryConditions*
$u = S \backslash f$
*plot(x,u)* ◊

Now, you wrote the backbone of a simple Finite Element program for a one dimensional model problem where a 'complicated' Robin boundary condition, as well as an essential boundary condition has been implemented. The discretisation matrix and right-hand side vector have been constructed.

**Assignment 20** *Compute the Finite Element solution u for* $v = 1$, $D = 1$, $K = 1$, $u^\infty = 0$ *and* $n = 100$. *Plot the solution. Is this what you would expect? Compare your solution for various number of gridnodes to the exact solution.* ◊

**Assignment 21** *Compute the Finite Element solution u for* $v = 1$, $D = 1$, $u^\infty = 0$ *and* $n = 1000$. *Use several curves for different K:* $K = 1,\ 10,\ 100,\ 1000$. *Plot the solution. Is this what you would expect?* ◊

**Assignment 22** *Choose* $D = 10^{-6}$, $v = 1$, $u^\infty = 0$, $K = 1000$. *Perform some experiments with several values of n (*$n = 10,\ 100,\ 1000$*). Plot the solutions for the various numbers of gridnodes in one plot and compare your solution to the exact solution. Explain what you see. Is this what you expect?* ◊

**Assignment 23** *Next, implement the SUPG method to tackle the spurious oscillations. Use the same parametric values as the previous assignment and compare the solution at various resolutions (*$n = 10,\ 100,\ 1000$*). You need to derive the new weak form, new Galerkin equations and new element matrix.* ◊

You just wrote a simple finite-element code in one dimension and with two different boundary conditions, in such a way that an extension to two- and three dimensional Finite Element programs is rather straightforward. Further, you considered convection–diffusion and a SUPG method. All you need to know is, which mesh points are vertices of each element. The latter distribution is commonly called the *topology* of the elements. In higher dimensions, the treatment of the boundary needs to be a little more formal than in the current treatment.

4. **Theoretical Assignment**

Let $\mathbf{u} = [u_x\ u_y]^T$, then we consider the following Stokes' flow problem in a rectangle $\Omega$ with non-overlapping boundary segments $\Gamma_1$, $\Gamma_2$ and $\Gamma_3$:

$$\begin{aligned} -\tfrac{1}{Re}\Delta\mathbf{u} + \nabla p &= \mathbf{f}(x,y), && \text{in } \Omega, \\ \nabla\cdot\mathbf{u} &= 0, && \text{in } \Omega, \\ \mathbf{u} &= \mathbf{u}_0, && \text{on } \partial\Omega. \end{aligned} \tag{12}$$

a Show that the solution to the above problem is unique in $\mathbf{u}$ and that the pressure $p$ is determined up to a constant. Hint: Use contraposition. Consider the difference between two different solutions in both PDEs. Use integration by parts. It might be easier to write out the vectorial components. (A real existence and uniqueness proof can be based on the LBB-condition).

b Next we consider the continuous penalisation method, applied to the above system, which reads as

$$\begin{aligned} -\tfrac{1}{Re}\Delta\mathbf{u}^\varepsilon + \nabla p^\varepsilon &= \mathbf{f}(x,y), && \text{in } \Omega, \\ \nabla\cdot\mathbf{u}^\varepsilon + \varepsilon p^\varepsilon &= 0, && \text{in } \Omega, \\ \mathbf{u}^\varepsilon &= \mathbf{u}_0, && \text{on } \partial\Omega. \end{aligned} \tag{13}$$

Show that $\mathbf{u}-\mathbf{u}^\varepsilon$, $p-p^\varepsilon$ satisfies

$$\begin{aligned} -\tfrac{1}{Re}\Delta(\mathbf{u}-\mathbf{u}^\varepsilon) + \nabla(p-p^\varepsilon) &= \mathbf{0}, && \text{in } \Omega, \\ \nabla\cdot(\mathbf{u}-\mathbf{u}^\varepsilon) - \varepsilon p^\varepsilon &= 0, && \text{in } \Omega, \\ \mathbf{u}-\mathbf{u}^\varepsilon &= \mathbf{0}, && \text{on } \partial\Omega. \end{aligned} \tag{14}$$

c Show that that equation (14) implies

$$\int_\Omega ||\nabla(u-u_x^\varepsilon)||^2 + ||\nabla(u-u_y^\varepsilon)||^2 d\Omega + \varepsilon\int_\Omega (p^\varepsilon)^2 d\Omega = \varepsilon\int_\Omega p p^\varepsilon d\Omega. \tag{15}$$

d Poincaré's (Lax-Friedrichs) Inequality says that under the present circumstances, there is an $\alpha > 0$ such that

$$\int_\Omega ||\nabla u||^2 d\Omega \geq \alpha\int_\Omega u^2 d\Omega.$$

Use this assertion, with $|ab| \leq \frac{1}{2}(a^2+b^2)$, to finally assert that $\mathbf{u}^\varepsilon \to \mathbf{u}$ and $\nabla p^\varepsilon \to \nabla p$ as $\varepsilon \to 0$.

Now you have demonstrated that the penalty method is consistent, in the sense that the numerical solution pair $(\mathbf{u}^\varepsilon, p^\varepsilon) \to (\mathbf{u}, p)$ as $\varepsilon \to 0$ (converges to the solution of the original Stokes problem).

5. **Programming Assignment**

We consider Biot's model for poro-elasticity. This model is used for the simulation of the interaction between fluid flow in a porous medium and the mechanical deformation of the medium. The problem is notorious for its saddle-point nature, like Stokes's flow model. We consider the following partial differential equations:

$$\begin{aligned} -\mu\Delta u - (\lambda+\mu)\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) + \frac{\partial p}{\partial x} &= 0, && \text{in } \Omega,\ t>0, \\ -\mu\Delta v - (\lambda+\mu)\frac{\partial}{\partial y}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) + \frac{\partial p}{\partial y} &= 0, && \text{in } \Omega,\ t>0, \\ \frac{\partial}{\partial t}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) - \frac{\kappa}{\mu}\Delta p &= f(x,y), && \text{in } \Omega,\ t>0. \end{aligned} \tag{16}$$

We consider $\Omega = (0,8) \times (0,8)$. The boundary conditions are given by

$$\begin{aligned} p &= 0, && \text{on } \partial\Omega \setminus (\Gamma_1 \cup \Gamma_3), \\ \mathbf{n} \cdot \nabla p &= 0, && \text{on } \Gamma_1 \cup \Gamma_2, \\ \sigma\mathbf{n} &= -\sigma_0, && \text{on } \Gamma_1, \\ \sigma\mathbf{n} &= 0, && \text{on } \Gamma_2, \\ \sigma\mathbf{n} + \kappa\mathbf{u} &= \mathbf{0}, && \text{on } \partial\Omega \setminus (\Gamma_1 \cup \Gamma_2), \end{aligned} \tag{17}$$

where

$$\Gamma_1 = \{(x,y) \in \partial\Omega | |x| \leq 0.8,\ y = 8\},$$

$$\Gamma_2 = \{(x,y) \in \partial\Omega | |x| > 0.8,\ y = 8\},$$

$$\Gamma_3 = \{(x,y) \in \partial\Omega | y = 0\}.$$

and

$$\sigma = \lambda tr(\varepsilon) I + 2\mu\varepsilon = \begin{pmatrix} (\lambda + 2\mu)\frac{\partial u}{\partial x} + \lambda\frac{\partial v}{\partial y} & \mu(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}) \\ \mu(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}) & (\lambda + 2\mu)\frac{\partial v}{\partial y} + \lambda\frac{\partial u}{\partial x} \end{pmatrix}.$$

a Derive the weak formulation.

b Derive the Galerkin equations in the form

$$\sum_j S_{ij}^{uu} u_j + \sum_j S_{ij}^{uv} u_j + \sum_j S_{ij}^{up} p_j = f_j^u,$$

for the three PDE's and write the matrices $S^{uu}$, etc, in terms of integrals over $\Omega$ and its boundary regarding the basis functions.

c Derive the internal element matrices for all the PDEs.

d Derive the boundary element matrices for all the PDEs.

e Subsequently, we consider a stabilisation, which changes the last PDE into

$$\frac{\partial}{\partial t}(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}) - \frac{\kappa}{\mu}\Delta p - \beta\Delta(\frac{\partial p}{\partial t}) = f(x,y), \qquad \text{in } \Omega,\ t > 0, \tag{18}$$

with $\beta = \frac{\Delta x^2 + \Delta y^2}{4(\lambda + 2\mu)}$. Repeat parts a–d.

We use the programs supplied on http://ta.twi.tudelft.nl/users/vermolen/SpecialTopics. All input values are listed in InputParameters.m. SpecialTopicsMesh.m, SpecialTopicsComp.m, SpecialTopicsPost.m are, respectively, used for the generation of the mesh, execution of the calculation program, and the postprocessing.

f Use the files from Program 1 (based on linear elements) to simulate the problem over the first five time-steps, use $\Delta t = 1,\ 10^{-2},\ 10^{-4},\ 10^{-6},\ 10^{-8}$. What do you observe? Is this what you expect?

g Use the files from Program 2 (based on Taylor-Hood elements) to simulate the problem over the first five time-steps, use $\Delta t = 1,\ 10^{-2},\ 10^{-4},\ 10^{-6},\ 10^{-8}$. What do you observe? Is this what you expect?

h Adjust the files in Program 1 to incorporate the stabilisation with $\beta = \frac{\Delta x^2 + \Delta y^2}{4(\lambda + 2\mu)}$. The adjustments can be made in the files *GenerateRightStiffnessMatrix.m*, *GenerateLeftStiffnessMatrix.m*, *GenerateLeftBoundaryElementMassMatrix.m* and *GenerateRightBoundaryElementMassMatrix.m*. Do the same as parts f–g.

i Write a report of about two pages of your findings of in this assignment.