DEPARTMENT MATHEMATIK
UND INFORMATIK
DER UNIVERSITÄT ZU KÖLN

Sommersemester 2019

Prof. Dr. A. Klawonn
J. Knepper, M. Sc.

17. April 2019

3. Übung zu Wissenschaftliches Rechnen II

Aufgabe 1: (6 Bonuspunkte)
Beweisen Sie, dass für $u, v \in \left(\mathcal{C}^2(\overline{\Omega})\right)^3$ die Greensche Formel

$$\int_\Omega \varepsilon(u) : \varepsilon(v)\,\mathrm{d}x = -\int_\Omega v \cdot \operatorname{div}(\varepsilon(u))\,\mathrm{d}x + \int_{\partial\Omega} v \cdot \varepsilon(u) n\,\mathrm{d}s$$

gilt.

Aufgabe 2: (13 Punkte)
Es sei zur schwachen Formulierung

$$\underbrace{2\mu(\varepsilon(u),\varepsilon(v))_{L^2(\Omega)} + \lambda(\operatorname{div}(u),\operatorname{div}(v))_{L^2(\Omega)}}_{=:a(u,v)} = \underbrace{(f,v)_{L^2(\Omega)} + (g,v)_{L^2(\Gamma_N)}}_{=:L(v)} \quad \forall v \in H^1_{\Gamma_D}(\Omega),$$

$$H^1_{\Gamma_D}(\Omega) := \{v \in (H^1(\Omega))^3 : v|_{\Gamma_D} = 0\},$$

eines *St. Venant-Kirchhoff*-Materials (hier: vollständig linearisierte Elastizität) eine Lösung $u \in H^1_{\Gamma_D}(\Omega)$ gegeben, welche zudem in $(C^2(\Omega))^3 \cap (C^1(\Omega \cup \Gamma_N))^3 \cap (C^0(\overline{\Omega}))^3$ liegt. Zeigen Sie, dass u auch

$$\begin{aligned} -2\mu \operatorname{div}(\varepsilon(u)) - \lambda\nabla(\operatorname{div}(u)) &= f \text{ in } \Omega, \\ u &= 0 \text{ auf } \Gamma_D, \\ \sigma(u)\mathbf{n} &= g \text{ auf } \Gamma_N, \\ \sigma(u) &= 2\mu\,\varepsilon(u) + \lambda \operatorname{tr}(\varepsilon(u))I, \\ \varepsilon(u) &= \tfrac{1}{2}\left(\nabla u + (\nabla u)^T\right), \end{aligned}$$

erfüllt.

Hinweis: Zeigen Sie zunächst

$$\int_\Omega \operatorname{div}(u)\operatorname{div}(v)\,\mathrm{d}x = \int_{\partial\Omega} \operatorname{div}(u) v \cdot n\,\mathrm{d}s - \int_\Omega \nabla \operatorname{div}(u) \cdot v\,\mathrm{d}x.$$

Wenden Sie anschließend Aufgabe 1 an. Mit Hilfe des Fundamentallemmas der Variationsrechnung lässt sich die erste Zeile der DGL zeigen.

Programmieraufgabe 1: (15 Punkte)
Auf einem Gebiet $\Omega \subset \mathbb{R}^2$ mit

$$\Omega_D := \{(x,y) \in \partial\Omega : (x = 0) \vee (y = 0)\}$$

soll für $u\colon \mathbb{R}^2 \to \mathbb{R}^2$ die folgende Differentialgleichung mit FEniCS approximativ gelöst werden[1]:

$$\begin{aligned} -\Delta u + c_1 \cdot \left\langle \nabla u, \begin{pmatrix}1\\1\end{pmatrix} \right\rangle + c_2 \cdot u &= \begin{pmatrix}1\\1\end{pmatrix}, && \text{in } \Omega, \\ u &= \begin{pmatrix}0\\0\end{pmatrix}, && \text{auf } \partial\Omega_D, \\ (\nabla u)\cdot \mathbf{n} &= g, && \text{auf } \partial\Omega \setminus \partial\Omega_D, \end{aligned}$$

wobei

$$\begin{aligned} c_1(x,y) &= 100\sin(6\pi x)^2, \\ c_2(x,y) &= 1000\sin(6\pi y)^2, \\ g(x,y) &= (g_1, g_2)^T, \\ g_i(x,y) &= 0.1\sin(2\pi(x-y)). \end{aligned}$$

- Lösen die obige DGL als vektorwertiges Problem in FEniCS (siehe Hinweise).
- Das Gitter wird auf der Übungswebseite zur Verfügung gestellt[2].
- Sofern `a` die Bilinearform, `L` das lineare Funktional, `bc` der Operator der Randbedingungen und `V` der zugehörige Finite-Elemente-Raum ist, lösen Sie das Problem mit

```
A, b = assemble_system(a, L, bc)
solver = KrylovSolver('gmres','hypre_amg')
solver.parameters["monitor_convergence"] = True
solver.parameters['absolute_tolerance'] = 1e-10
solver.parameters['relative_tolerance'] = 1e-7
solver.parameters['maximum_iterations'] = 100
solver.set_operator(A)
u = Function(V)
solver.solve(u.vector(),b)
```

- Berechnen Sie die Differenz $|u_1 - u_2|$ der Lösung und geben Sie den maximalen Fehler aus (Sie können den Fehler mit Python oder auch in ParaView mit dem *Calculator*-Filter bestimmen).

Hinweise:

- Schauen Sie sich zur Herleitung der Variationsformulierung die Bilinearform zum Stokes-Problem $a(u,v) = \int_\Omega \nabla u : \nabla v \,\mathrm{d}x$ an (*Numerik partieller DGL*, SS2018).
- Für die Implementierung hilft u.a. das Beispiel zur linearen Elastizität (Übungswebseite).
- Wir nutzen für $u = (u_1, \ldots, u_d)^T$ die Konvention

$$\nabla u := \text{Jacobimatrix}(u) = \begin{pmatrix} \nabla u_1 \\ \nabla u_2 \\ \vdots \\ \nabla u_d \end{pmatrix}.$$

[1] Vektorwertig: $\langle \nabla u, \mathbf{1}\rangle := (\nabla u)\mathbf{1}$. Skalarwertig: $\langle \nabla u, \mathbf{1}\rangle := \sum_i \frac{\partial u_i}{\partial x_i}$.

[2] Unter Ubuntu kann man mit `dolfin-convert 'gitter.msh' 'gitter.xml'` ein Gmsh-Gitter in das FEniCS-XML-Format konvertieren (FEniCS-Standard-Installation). Ein in FEniCS erzeugtes Gitter `mesh` kann man mit `File('mesh.xml') << mesh` exportieren.

Manchmal wird der Gradient auch als die transponierte Jacobimatrix definiert, weshalb in FEniCS `nabla_grad` existiert; siehe UFL-Manual und auch S. 58f in *Solving PDEs in Python – The FEniCS Tutorial I*. Es gilt mit $A = (a_1, \ldots, a_d) := \nabla u$, $B = (b_1, \ldots, b_d) := \nabla v$, $u, v : \mathbb{R}^d \to \mathbb{R}^d$:

Operator	Definition in FEniCS
`grad(u)`	∇u
`nabla_grad(u)`	$(\nabla u)^T$
`dot(A,B)`	AB
`inner(A,B)`	$A : B = \mathrm{tr}(A^T B) = \sum_{i,j} A_{ij} B_{ij} = \sum_i a_i^T b_i$

Es gilt `inner(grad(u),grad(v))=inner(nabla_grad(u),nabla_grad(v))`, denn

$$\mathrm{tr}(A^T B) = \sum_{i,j} A_{ij} B_{ij} = \sum_{i,j} (A^T)_{ij} (B^T)_{ij} = \mathrm{tr}(AB^T).$$

Abgabe: Bis Mittwoch, 24. April 2019, 14:00 Uhr, im entsprechenden Kasten in Raum 3.01 des Mathematischen Instituts.