1 09.12.2020 Form Optimierung

1.1 Formulierung des Problems

Die Formoptimierung ist ein Optimierungsproblem mit PDG als Nebenbedingungen. Es hat die folgende Gestalt:

$$\begin{aligned} \min \quad & J(v,u) \\ s.t. \quad & ... \end{aligned}$$

Wobei $J(v,u)$ unsere Zielfunktion ist, die aus mehreren Termen besteht, v ist ein Phasenfeld und u die zugehoerige Loesung der linearen Elastizitaet bzw. minimiert die Elastische Energie. Ist die Loesung u eindeutig, koennen wir u als $u(v)$ ausdruecken und $J(v,u)$ auf eine Variable reduzieren:

$$\hat{J}(v) = J(v, u(v))$$

Man nennt v auch Control/ Steuerungs Variable, u den associated State, $\hat{J}$ das reduzierte Funktional und das Problem allgemein Optimale Steuerung.
Konkret haben wir (Vergleiche Kapitel 1):

$$E(u,v) = \underbrace{\int_\Omega \chi(v)(\sigma(u) : \varepsilon(u))\, dx}_{\text{gespeicherte Elastische Energie}} - \underbrace{\int_{\partial\Omega} g \cdot u\, ds}_{\text{Nachgiebigkeit}}$$

Statt nur der Charakterristischen Funktion, bedient man sich dem Hard-soft-approach. Dies hat den Vorteil, dass u, wo kein Material ist, nicht beliebig ist:

$$E(u,v) = \int_\Omega ((1-\delta)\chi(v) + \delta)\sigma(u) : \varepsilon(u)\, dx \;+\; \int_{\partial\Omega} g \cdot u\, ds$$

Ausserdem soll $v = 1$ auf $\Gamma \subseteq \partial\Omega$ und $g > 0$ auf Γ sein. Nun kommen wir zum Ziel Funktional:

$$J(u,v) := \int_{\partial\Omega} g \cdot u ds \;+\; \alpha \int_\Omega \frac{1}{2}(\epsilon|\nabla v|^2 + \frac{1}{\epsilon}\psi(v))dx \;+\; \beta \int_\Omega \chi(v) dx$$

Insgesamt sieht unser Problem (P) wie folgt aus:

$$\begin{aligned} \min_{u,v} \quad & J(u,v) && (1) \\ \text{s.t.} \quad & u = min_{u'} E(u',v) && (2) \\ & v \in V_{\text{admissible}} && (3) \end{aligned}$$

1.2 LÃ¶sen in Fenics

1.2.1 Dolfin-Adjoint

Wir werden Dolfin-Adjoint (PyAdjoint) brauchen. Um dies zu installieren geht man wie folgt vor (http://www.dolfin-adjoint.org/en/latest/download/):

- In Anaconda: conda install dolfin-adjoint
- In Docker: Image quay.io/dolfinadjoint/pyadjoint
- In Ubuntu: pip install git+https://github.com/dolfin-adjoint/pyadjoint.git@2019.1.0

Die Berechnung von $\hat{J}$ und $\partial_v \hat{J}$ lÃ¤uft in Dolfin-Adjoint automatisch ab. Die Theorie dahinter findet man im Kapital 1. Dabei benutzt Dolfin-Adjoint das tape- und annotate feature. Diese zeichnen alle Rechnungen von Fenics auf, wobei die Gleichungen mit ihren AbhÃ¤ngigkeiten nochmal seperat abegspeichert werden, d. h. dass man im Programmablauf die Nebenbedingung einmal fÃ¼r irgendein beliebiges v lÃ¶sen muss. Danach liegt es auf dem Tape vor und kann abgerufen werden. Nun muss das Ziel Funktional mit den unabhÃ¤ngigen Variablen explizit angegeben werden. Danach werden $\hat{J}$ und $\partial_v \hat{J}$ automatisch berechnet.
Dolfin-Adjoint bietet auÃerdem eine angepasste Version von scipy.minimize, welche Fenics Funktionen versteht.

1.2.2 Das GrundgerÃ¼st

```
from dolfin_adjoint import *

 mesh = ..
 V = ..
 u = Function(V)
 ...
 E = ..
 J = ..
 solve(..E..)
 (ZusammenfÃ¼gen der letzten beiden Ãbungen)

Jhat = ReducedFunctional(assemle(J), Control(v))
v = minimize(Jhat)
```

Die Funktion minimize(Jhat) hat einige Parameter. Darunter die Options, welche durch

```
options={'disp': True, 'ftol': 1e-8, 'gtol': 1e-8}
```

defniert werden. Die Option 'disp' besagt, ob nach jeder Iteration etwas zurÃ¼ck gegeben werden soll, 'ftol' ist die Toleranz in Bezug auf die Ãnderung des Funktionswertes und 'gtol' ist die Toleranz in Bezug auf die Norm des Gradienten (Abbruchkriterien). Desweiteren kann man durch method = '...' die Methode, die benutzt werden soll, angeben. Hier zum Beispiel:

- Nelder-Mead
- L-BFGS-B (Quasi-Newton-Verfahren kommt mit Bounds zurecht)

Als bounds werden auch Fenics Funktionen Ã¼bergeben. Zum Beispiel:

```
lb = project(Constant(-1000), V)
bc.apply(lb.vector())

ub = project(Constant(1000), V)
bc.apply(ub.vector())

v = minimize(Jhat, bounds=(lb, ub))
```

1.2.3 Dolfin-Adjoint Taylor Test

Zur Erinnerung:
Berechnung von $\hat{J}(v)$ die im Hintergrund passiert:

- u(v) als LÃ¶sung von $\partial_u E(u, v) = 0$ fÃ¼r gegebenes v
- $\hat{J}(v) = J(v, u(v))$

"$\partial_v \hat{J}(v)$":

- u(v) als LÃ¶sung von $\partial_u E(u, v) = 0$ fÃ¼r gegebenes v
- $\partial_u \partial_u E(u, v)(p) = \partial_u J(u, v)$ (p (Lagrange Multiplikator) wird berechnet, adjungierten Gleichung)
- $\partial_v \hat{J}(v) = \partial_u \partial_v E(u(v), v)(p) + \partial_v J(v, u(v))$

Da das ganze im Hintergrund passiert, sollte man checken, ob die Ableitungen richtig berechnet werden, denn es kann zu Fehlern kommen. Nicht alle Konstrukte werden unterstÃ¼tzt z. B. Expression(...). Eine andere Fehlerquelle ist die Falscherkennung von AbhÃ¤ngigkeiten z. B. wenn Randwerte auf Knoten doppelt definiert werden (auch wenn sie gleich sind). Zum testen bietet Dolfin-Adjoint auch eine Funktion, die als Basis die Taylor Entwicklung verwenden:

$$\hat{J}(v + h\phi) = \hat{J}(v) + \partial_v \hat{J}(v)(h\phi) + O(h^2)$$

```
taylor_test(Jhat, v, phi)
oder wenn man einen linearen Abfall haben mÃ¶chte:
taylor_test(Jhat, v, phi, dJdm = 0)
```

1.2.4 Beispiel Szenarien

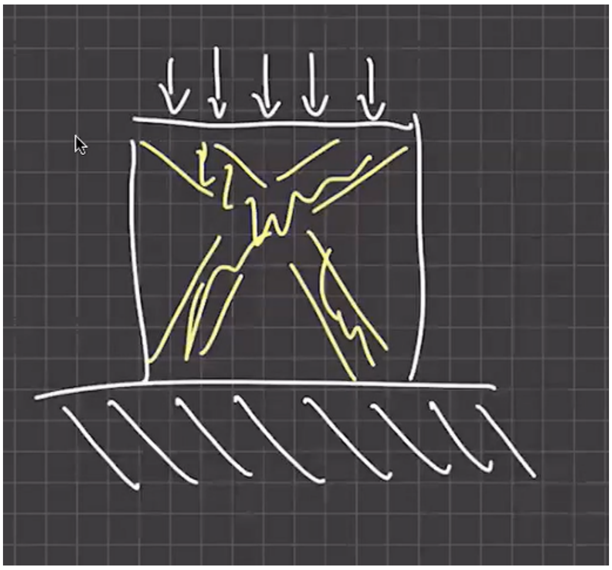

Abbildung 1: Carrier Plate

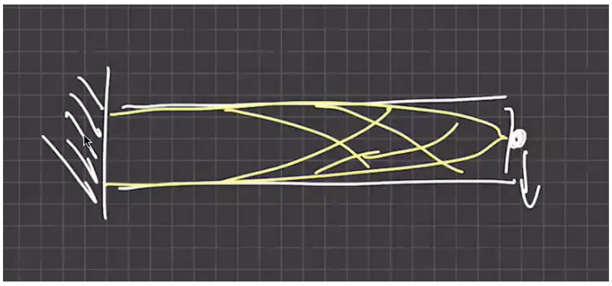

Abbildung 2: Cantilever beam

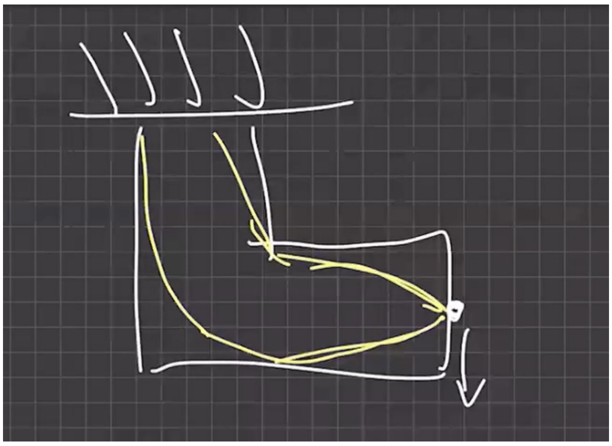

Abbildung 3: L-Shape