1 Implementierung in Python mit FEniCS

1.1 ???

mache 2 große sections bib und main, dann bei bib untersections gitter, lösen von pdes und b
Nachdem wir in vorigen Kapiteln den theoretischen Hintergrund der Formoptimierung und gradientenbasierte Verfahren, wie dem L-BFGS-Verfahren, gelegt haben, möchten wir in diesem Abschnitt die Implementierung des Algorithmus cite in Python 3.5 Versicherung, dass es nur unter dieser Version läuft mit Hilfe des Moduls FEniCS vorstellen. Im Folgenden werden wir die in den Dateien enthaltenen Kommentare nicht oder nicht in voller Länge in den Codeausschnitten aufführen, da wir Redundanz bei den Erklärungen vermeiden möchten. Selbstverständlich sind in den Quellcodes selber ausführliche Kommentierungen vorhanden. Die Implementierung in Python besteht im wesentlichen aus den beiden Dateien

shape_main.py shape_bib.py.

Die Datei *shape_main.py* enthält hierbei den zusammenhängenden Hauptcode. Die Datei *shape_bib.py* ist eine Bibliothek, in welcher Funktionen zum Umgang mit Gittern, Berechnungen auf Formen, Löser für PDEs und der L-BFGS-Algorithmus mit den damit verbundenen Objekten gebündelt sind. Die Berechnungen auf den Formen und das Lösen der PDEs wird mittels FEniCS geschehen. FEniCS ist eine https://fenicsproject.org/ frei zugängliche Programmierung, welche ermöglicht, partielle Differentialgleichungen mit relativ geringem Aufwand zu lösen. Dabei bedient sich FEniCS der sogenannten *Unified Form Language* (UFL), was die Grundlage zur Implementierung der PDE's in schwacher Formulierung darstellt, mehr hierzu bei [5]. Diese nutzen wir, um Bevor wir uns der Lösung von PDE's und der Implementierung des L-BFGS-Algorithmus zuwenden, müssen wir zunächst klären, wie wir die notwendigen Gitter erzeugen und mit diesen umgehen.

subsection gitter?

Gitterdateien erzeugen wir mit Hilfe des offen zugänglichen Programms Gmsh 3.0.6 http://gmsh.info/. Hierbei muss man zunächst eine .geo Datei geschrieben werden. Wir zeigen dies am Beispiel eines kleinen Kreises. Zunächst setzen wir die für unser Gitter relevanten Punkte in ein 3-dimensionales Koordinatensystem

```
1 Point(1) = {0.0, 0.0, 0.0, 1.0};
2 Point(2) = {1.0, 0.0, 0.0, 1.0};
3 Point(3) = {0.0, 1.0, 0.0, 1.0};
```

```
Point(4) = {1.0, 1.0, 0.0, 1.0};
Point(5) = {0.5, 0.35, 0.0, 1.0};
Point(6) = {0.5, 0.5, 0.0, 1.0};
Point(7) = {0.5, 0.65, 0.0, 1.0};
```

Hierbei beschreiben die ersten 3 Einträge des Tupels die x-, y- und z-Koordinaten der Punkte, der vierte Eintrag gibt die sogenannte *characteristic length* der Punkte an, was lediglich die Elementgröße des Punktes ist. Punkte 1 bis 4 werden dazu dienen das Einheitsquadrat im $\mathbb{R}^2$ zu definieren, Punkte 5 bis 7 werden einen Kreis mit Mittelpunkt $(0.5, 0.5)$ definieren. Dies geschieht mittels der Befehle

```
Line(1) = {1, 2};
Line(2) = {2, 4};
Line(3) = {4, 3};
Line(4) = {3, 1};

Circle(5) = {5, 6, 7};
Circle(6) = {7, 6, 5};
```

Da diese Befehle lediglich Linien und Halbkreise aus den eingegebenen Punkten definieren, ist es nötig mittels eines Loop-Befehls diese zu einer gemeinsamen Form zu verbinden.

```
Line Loop(1) = {1, 2, 3, 4};
Line Loop(2) = {5, 6};
```

Um innere und äußere Gebiete, welche durch Abgrenzung mittels des Kreises definiert sind, zu markieren, setzen wir diese als Plane Line fest. Die erste Zahl gibt jeweils die Nummer des Line Loops des äußeren Randes, die zweite die des inneren Randes an.

```
Plane Surface(1) = {1, 2};
Plane Surface(2) = {2};
```

Weil wir später in der Implementierung auf die Ränder bzw. Formen zugreifen möchten, ist es abschließend noch nötig diese als sogenannte Physical Lines und Physical Surfaces zu definieren.

```
Physical Line(1) = {1};
Physical Line(2) = {2};
Physical Line(3) = {3};
Physical Line(4) = {4};
Physical Line(5) = {5};
Physical Line(6) = {6};

Physical Surface(1) = {1};
Physical Surface(2) = {2};
```

Die so geschriebene .geo-Datei wird nun mit Hilfe des Programms Gmsh mit dem Kommando

```
gmsh mesh_smallercircle.geo -2 -clscale 0.025
```

in eine .msh Datei konvertiert. Dabei ist mesh_smallercircle.geo der Name der gespeicherten .geo-Datei, -2 die Dimension des erzeugten Gitters und 0.025 die Feinheit des Gitters. Um diese Datei für FEniCS nutzbar zu machen, konvertieren wir diese mit Hilfe des Dolfin-Befehls

```
dolfin-convert mesh_smallercircle.msh mesh_smallercircle.xml
```

wobei der erste Eingabewert der Name der .msh-Datei ist, und der zweite der Name der erzeugten .xml-Datei. Hierbei werden außerdem neben dem bloßen Mesh auch eine facet_region.xml-Datei erstellt, mit welcher man die Ränder initialisieren kann, sowie eine physical_region.xml-Datei, welche zur Initialisierung der Gebiete des Inneren und Äußeren der Form dient.

neuer Abschnitt?

Nun besitzen wir die nötigen Gitterdateien, um auf diesen Formoptimierung zu betreiben. Wir stellen kurz vor, mit welchen Objekten und Funktionen wir mit diesen umgehen. Eines der beiden zentralen Objekte des Optimierungsprogramms ist die sogenannte *MeshData-Klasse*.

```
1  class MeshData:
2
3      # Objekt mit allen Daten des Gitters
4      def __init__(self, mesh, subdomains, boundaries, ind):
5
6          # FEniCS Mesh
7          self.mesh = mesh
8
9          # FEniCS Subdomains
10         self.subdomains = subdomains
11
12         # FEniCS Boundaries
13         self.boundaries = boundaries
14
15         # Indizes der Knotenpunkte mit Traeger nicht am inneren
       Rand
16         self.indNotIntBoundary = ind
```

In einem Objekt dieser Klasse werden sowohl das Gitter, als auch die Gebiete und Ränder bzw. Formen gespeichert. Weiterhin benötigen wir für spätere

folgende Berechnungen auch die Indizes der Knotenpunkte (engl. *Vertices*), welche keinen Träger am inneren Rand haben, gespeichert.Die Initialisierung erfolgt mit der von uns implementierten Funktion

load_mesh(Name),

wobei Name der Name der Mesh-Datei ohne .xml-Endung ist. Die subdomains und boundaries werden als sogenannte MeshFunction initialisiert. Dies sind Objekte einer in FEniCS implementierten Klasse, welche als Array im i-ten Eintrag die Nummer der subdomain bzw. der boundary zurückgibt, welche den Nummern der Physical Surface bzw. Physical Line in der .geo-Datei entsprechen. Diese Initialierungen geschehen über die Befehle

```
1    mesh       = Mesh(path_meshFile + ".xml")
2    subdomains = MeshFunction("size_t", mesh,
3                              path_meshFile +
4                              "_physical_region.xml")
5    boundaries = MeshFunction("size_t", mesh,
6                              path_meshFile +
7                              "_facet_region.xml")
8    ind = __get_index_not_interior_boundary(mesh, subdomains,
9                                            boundaries)
```

wobei Mesh als Eingabe den Pfad zur .xml-Datei enthält. Die Meshfunktionen erhalten neben dem mesh-Objekt den Typ der Funktion, in diesem Fall size_t, und die Pfade zu den jeweiligen Dateien _physical_region.xml und _facet_region.xml. Es bleibt noch, die Indexliste der Indizes mit Träger nicht am Inneren Rand zu initialisieren. Um diese Indexliste zu erzeugen haben wir die Funktion

___get_index_not_interior_boundary(mesh, subdomains, boundaries, interior = True))

implementiert. Als Input erhält sie die oben gezeigten Objekte, falls interior = True eingestellt ist, so gibt die Funktion die Liste mit Indizes ohne Träger am Inneren Rand wieder. Wir möchten an dieser Stelle anmerken, dass Indizes auch mehrfach vorkommen, was für unser Programm kein Problem darstellt und bei Bedarf verbessert werden kann. Ist der Parameter interior = False, so gibt die Funktion eine Liste mit den Indizes der Knotenpunkte genau des inneren Randes wieder. Dies spart uns die Implementierung einer weiteren Funktion. Das erzeugen der Liste basiert auf Iterationen durch Facetten des Randes, deren Knoten und den benachbarten Knoten. Diese aufwändige Iteration ist nötig, da die Indizierung der Facetten in der Meshfunktion des Randes

in FEniCS nicht mit den Indizes der Mesh's übereinstimmen. Für die genaue Implementierung verweisen wir auf den von uns beigefügten Code.

neuer Abschnitt lösen von PDE und berechnungen, falls oben noch irgendwas dazugehören ka

Nun besitzen wir Gitterobjekte, auf welchen wir Berechnungen durchführen können. Der Hauptanteil der Berechnungen besteht in dem Lösen von partiellen Differentialgleichungen. Wir möchten exemplarisch an einer implementierten Funktion zeigen, wie dies in FEniCS praktisch passiert.

vergiss nicht die distance function

Literatur

[1] M. Genzen, A. Staab, Prof. E. Emmrich. Sobolew-Slobodeckij-Räume - die Theorie der gebrochenen Sobolew-Räume, Technische Universität Berlin. 2014.

[2] G. Geymonat. Trace Theorems for Sobolev Spaces on Lipschitz Domains. Necessary Conditions. 2007.

[3] J. M. Lee. *Introduction to Smooth Manifolds,, Second Edition*. Springer, Graduate Texts in Mathematics, 2013.

[4] H. P. Langtangen, A. Logg. *Solving PDEs in Python - The FEniCS Tutorial Volume I*. Springer, 2017.

[5] M. S. Alnæs, A. Logg. *UFL Specification and User Manual 0.3*. www.fenics.org, 2010.

[6] K. Burg, H. Haf, F. Wille, A. Meister. *Partielle Differentialgleichungen und funktionalanalytische Grundlagen, 5. Auflage*. Vieweg +Teubner Verlag, Springer Fachmedien, 2010.

[7] B. Zhong P.A. Sherar, C.P.Thompson, B. Xu. An optimization method based on b-spline shape functions & the knot insertion algorithm. *Proceedings of the World Congress on Engineering*, II, 2007.

[8] S. Schmidt. Weak and strong form shape hessians and their automatic generation. 2018, SIAM J. Sci. Comput., Vol. 40, No.2, pp. C210-C233.

[9] Volker Schulz. A riemannian view on shape optimization. *Foundations of computational Mathematics*, 14:483-501, 2014.

[10] B. Schweizer. *Partielle Differentialgleichungen - Eine anwendungsorientierte Einführung*. Springer Spektrum, 2013.

[11] Volker Schulz, Martin Siebenborn. Computational comparison of surface metrics for pde constrained shape optimization. *Comput. Methods Appl. Math 2016*, 2016.

[12] Kevin Sturm. *On shape optimization with non-linear partial differential equations*. PhD thesis, Technische Universität Berlin, 2015.

[13] W. Arendt, K. Urban. *Partielle Differenzialgleichungen - Eine Einführung in analytische und numerische Methoden.* Spektrum Akademischer Verlag Heidelberg, 2010.

[14] K. Welker. Suitable Spaces for Shape Optimization. 2017, arXiv: 1702.07579v2.

[15] Kathrin Welker. *Efficient PDE Constrained Shape Optimization in Shape Spaces.* PhD thesis, Universität Trier, 2016.

[16] Volker Schulz, Martin Siebenborn, Kathrin Welker. Towards a lagrange-newton approach for constrained shape optimization. *arXiv: 1405.3266v2*, 2014.

[17] Volker Schulz, Martin Siebenborn, Kathrin Welker. Pde constrained shape optimization as optimization on shape manifolds. *Geometric Science of Information, Lecture Notes in Computer Science*, 9389:pp. 499–508, 2015.

[18] Volker Schulz, Martin Siebenborn, Kathrin Welker. Efficient pde constrained shape optimization based on steklov-poincaré-type metrics. *SIAM J. OPTIM.*, Vol. 26, No. 4, pp. 2800-2819, 2016.

[19] Jorge Nocedal, Stephen J. Wright. *Numerical Optimization, Second Edition.* Springer, 2006.

[20] M. C. Delfour, J. P. Zolésio. *Shapes and Geometries: Metrics, Analysis, Differential Calculus, and Optimization, 2nd ed.* SIAM Advances in Design and Control, 2011.