

**Masterarbeit Nr. 91**

**Entwicklung eines automatisierten Auslegungs- und  
Optimierungsprogramms für anisotrope UAV-Propeller**

von

Lukas Johannes Hilbers, B.Sc.  
Im Studiengang: Luft- und Raumfahrttechnik M.Sc.  
Matr.-Nr.: 4334022

Erstprüfer: Prof. Dr.-Ing. Peter Horst  
Betreuer: Felix Nolte, M. Sc.

Braunschweig, April 2021

If you would like to cite this thesis, here's a BibTeX citation entry:

```
@Thesis{Hilbers2021,  
    author      = {Lukas Johannes Hilbers, B.Sc.},  
    year       = {2021},  
    institution = {Institut für Flugzeugbau und Leichtbau,  
                   Technische Universität Braunschweig},  
    title       = {Entwicklung eines automatisierten Auslegungs- und  
                   Optimierungsprogramms für anisotrope UAV-Propeller},  
    type        = {Master's Thesis},  
}
```



Masterarbeit für Herrn Lukas Hilbers B.Sc.

Matrikelnummer 4334022

ausgegeben am:

abgegeben am:

## Entwicklung eines automatisierten Auslegungs- und Optimierungsprogramms für anisotrope UAV-Propeller

### 1. Aufgabenstellung

Der Wirtschaftszweig der unbemannten Luftfahrtzeuge („unmanned aerial vehicles“, UAV) konnte in den letzten Jahren ein enormes Wachstum aufweisen. Insbesondere der Bereich der Multicopters ist dabei stark gewachsen. Eine möglichst leichte Bauweise von Multicoptern ist daher sowohl aus wirtschaftlichen als auch aus ökologischen Gesichtspunkten sinnvoll und sollte bereits bei der Strukturkonstruktion durch Optimierungen adressiert werden. Multicopterpropeller sind aufgrund ihrer im stationären Betriebszustand gut definierbaren Belastungen und ihrer verhältnismäßig simplen Geometrie prinzipiell gut für eine Strukturoptimierung geeignet. Andererseits werden sie oft aus Faserverbundwerkstoffen (FVW) konstruiert, die aufgrund ihres lagenweisen Aufbaus sowie ihrer anisotropen Eigenschaften eine Strukturoptimierung durch zahlreiche freie Materialparameter komplex werden lassen können. Beispielsweise können Änderungen von Lagenorientierung, -dicke und -material im Widerspruch zueinander stehen. Nichtsdestotrotz bietet eine detaillierte Strukturoptimierung signifikantes Potenzial bei der Propellergestaltung, z.B. bzgl. Massenreduktionen, Schwingungseigenschaften, Aeroelastic Tailoring etc. Abhilfe können hierfür automatisierte Optimierungsalgorithmen schaffen.

Im Rahmen dieser Masterarbeit soll ein rechnergestütztes, automatisiertes Auslegungs- und Optimierungsprogramm erarbeitet werden, welches auf Basis der Finite Elemente Methode (FEM) in der Lage ist, anhand vom Nutzer zu definierender Zielgrößen und Randbedingungen den optimalen anisotropen Materialaufbau für eine feste Propellergeometrie zu generieren. Hierfür soll ein geeigneter Open-Source-Optimierungs-Algorithmus mit dem FE-Löser ANSYS sowie dem Geometrie-Input gekoppelt werden und eine skriptbasierte Benutzerschnittstelle geschaffen werden. Nach einer theoretischen Betrachtung möglicher Optimierungsverfahren für anisotrope Werkstoffe soll eine Programmarchitektur erarbeitet werden und diese anschließend in der Programmiersprache Python implementiert werden. Nach einer Evaluation anhand analytisch gesicherter Referenzbeispiele soll das Programm zur Optimierung einer exemplarischen Propellergeometrie genutzt werden und mittels Parameterstudien das Programmverhalten studiert werden.

## 2. Arbeitsschritte

- Ausführliche Literaturrecherche und Knowhowtransfer des Stands der Technik
- Evaluation von Optimierungsverfahren für FVW und Auswahl eines geeigneten Algorithmus
- Analyse der strukturellen Belastung von UAV-Propellern und Ableitung repräsentativer Lastfälle
- Erarbeitung Programmarchitektur zur Kopplung von Geometrie, FEM und Optimierung
- Implementierung der Programmarchitektur in Python-Code
- Validierung des Optimierungsverfahrens anhand analytischer Standardprobleme
- Anwendung auf eine Referenz-Propellergeometrie mittels Parameterstudie und Ableitung eines optimierten Strukturaufbaus
- Auswertung und Diskussion aller Ergebnisse
- Ausführliche Dokumentation der Arbeit

## 3. Literatur

- [1] [www.pyopt.org](http://www.pyopt.org): Python-basierte Bibliothek mit Optimierungsalgorithmen
- [2] <https://akaszynski.github.io/pyansys/#>: Bibliothek zur Kopplung von Python und ANSYS
- [3] L. Harzheim. Strukturoptimierung – Grundlagen und Anwendungen. *Europa-Lehrmittel*, 2019.
- [4] S. Nikbakt, S. Kamarian, M. Shakeri. A Review on Optimization of Composite Structures Part I: Laminated Composites. *Composite Structures*, 2018.
- [5] P. W. Jansen, R. E. Perez. Constrained structural design optimization via a parallel augmented Lagrangian particle swarm optimization approach. *Computers and Structures*, 89:1352-1366, 2019.

## 4. Anlagen

Merkblatt für die Erstellung studentisch-wissenschaftlicher Arbeiten am IFL.

Die Arbeit muss während der Bearbeitungszeit Prof. Dr.-Ing. Horst oder dem Betreuer Felix Nolte M.Sc. mindestens dreimal vorgelegt werden. Die Bearbeitungszeit beträgt gemäß Prüfungsverordnung Master Luft- und Raumfahrttechnik 6 Monate.

Prof. Dr.-Ing. P. Horst

## **Schriftliche Erklärung**

Ich versichere, dass ich die vorliegende Masterarbeit „Entwicklung eines automatisierten Auslegungs- und Optimierungsprogramms für anisotrope UAV-Propeller“ selbstständig, ohne unerlaubte fremde Hilfe oder Beratung und nur unter Verwendung der angegebenen wissenschaftlichen Hilfsmittel und Literatur angefertigt habe.

Braunschweig, den 7. April 2021

---

# Abstract

**English** In this thesis, an automated program chain for the design and optimization of anisotropic UAV propellers made of fibre reinforced composites is developed. The augmented Lagrangian particle swarm method from the Python library PyOPT was used as the optimization strategy. The suitability and performance of the algorithm for the optimization of fibre composite components is demonstrated using two analytical reference problems.

Based on XFOIL and XROTOR, a program for load analysis of UAV propellers is developed and implemented in Python. This allows to calculate an envelope from several load cases and to derive a continuous lift and drag distribution for the entire propeller ground plan. A finite element model implemented in Python with ANSYS MECHANICAL APDL and PyMAPDL serves as the analysis model for the optimization. The pressure curves generated with the load calculation routine are used for the load definition.

With this program chain, a structural design is optimized for a reference propeller geometry (*T-Motor MF 3218*) under variation of fibre orientations, quantities and their distribution in the load-bearing cross-sections. Two different formulations are compared as the objective function: An optimization of the strength criteria towards a target value, and a direct minimization of the fibre quantities. The best results are achieved with the direct method. For the post-processing of the optimization results, a Python script with a graphical user interface is created in JUPYTERLAB, which facilitates the processing of the optimization results.

**Deutsch** In der vorliegenden Arbeit wird eine automatisierte Programmketten zur Auslegung und Optimierung anisotroper UAV-Propeller aus Faserverbundwerkstoffen entwickelt. Als Optimierungsstrategie wurde das Partikelschwarmverfahren mit der erweiterten Lagrange-Methode aus der Python-Bibliothek PyOPT verwendet. Die Eignung und Leistungsfähigkeit des Algorithmus für die Optimierung von Faserverbund-Bauteilen wird anhand von zwei analytischen Referenz-Problemen demonstriert.

Auf der Basis von XFOIL und XROTOR wird ein Programm zur Lastanalyse von UAV-Propellern entwickelt und in Python implementiert. Dieses erlaubt es, aus mehreren Lastfällen eine Envelope zu berechnen und eine kontinuierliche Auftriebs- und Widerstandsverteilung für den gesamten Propeller-Grundriss abzuleiten. Als Analysemodell für die Optimierung dient ein Finite-Elemente-Modell, welches in Python mit ANSYS MECHANICAL APDL und PyMAPDL implementiert wird. Zur Last-Definition werden die mit der Lastrechnungs-Routine generierten Druckverläufe verwendet.

Mit dieser Programmketten wird für eine Referenz-Propellergeometrie (*T-Motor MF 3218*) ein Strukturaufbau unter Variation von Faser-Orientierungen, -Mengen und ihre Verteilung in den tragenden Querschnitten optimiert. Als Zielfunktion werden zwei verschiedene Formulierungen verglichen: Eine Optimierung der Festigkeitskriterien auf einen Zielwert hin, sowie eine direkte Minimierung der Fasermengen. Die besten Ergebnisse werden mit der direkten Funktion erzielt. Für das Post-Processing der Optimierungsergebnisse wird zusätzlich ein Python-Skript mit graphischer Benutzeroberfläche in JUPYTERLAB erstellt, welches die Aufbereitung der Optimierungsergebnisse erleichtert.

# Inhaltsverzeichnis

<b>Aufgabenstellung</b>	<b>iii</b>
<b>Eidesstattliche Erklärung</b>	<b>v</b>
<b>Abstract</b>	<b>v</b>
<b>Inhaltsverzeichnis</b>	<b>vii</b>
<b>Abbildungsverzeichnis</b>	<b>viii</b>
<b>Abkürzungs- und Formelverzeichnis</b>	<b>ix</b>
<b>1. Einleitung</b>	<b>1</b>
<b>2. Stand der Technik</b>	<b>2</b>
2.1. Motivation der Strukturoptimierung . . . . .	2
2.2. Allgemeine Formulierung eines Optimierungsproblems . . . . .	3
2.3. Globale und lokale Minima . . . . .	4
2.4. Übersicht gängiger Optimierungsverfahren . . . . .	5
2.5. Strukturoptimierung von Faserverbundwerkstoffen . . . . .	8
<b>3. Auswahl und Validierung eines Optimierungsverfahrens</b>	<b>10</b>
3.1. Anforderungen an den Optimierungsalgorithmus . . . . .	10
3.2. Vorauswahl eines Verfahrens . . . . .	11
3.3. Partikelschwarmverfahren . . . . .	13
3.4. Validierung anhand analytischer Standardprobleme . . . . .	15
3.4.1. Optimierung einer Faserverbundplatte . . . . .	16
3.4.2. Optimierung eines Faserverbundbalkens . . . . .	19
<b>4. Lastanalyse von UAV-Propellern</b>	<b>22</b>
4.1. Verwendete Softwaretools . . . . .	22
4.2. Implementierung von XFOIL und XROTOR in Python . . . . .	24
4.3. Erstellung eines Lastrechnungsprogramms . . . . .	26
4.3.1. Architektur . . . . .	26
4.3.2. Loadcase-Klasse . . . . .	27
4.3.3. Airfoil-Klasse . . . . .	28
4.3.4. Propeller-Klasse . . . . .	29
4.3.5. Workflow bei der Lastenrechnung . . . . .	31
<b>5. Erstellung eines Optimierungstools</b>	<b>33</b>
5.1. Verwendete Software . . . . .	33
5.2. Programmarchitektur . . . . .	34
5.2.1. Femodel-Klasse . . . . .	35

5.2.2. Material-Klasse . . . . .	36
5.2.3. Propellermode-Klasse . . . . .	37
5.3. Aufbau des Analysemodells . . . . .	39
5.3.1. Geometrie und Vernetzung . . . . .	39
5.3.2. Lastaufprägung . . . . .	40
5.3.3. Post-Processing . . . . .	43
5.4. Aufbau der Optimierungsroutine . . . . .	44
5.4.1. Ansatz für die Zielfunktion . . . . .	44
5.4.2. Formulierung von Zielfunktion und Restriktionen . . . . .	45
5.4.3. Designvariablen . . . . .	46
<b>6. Anwendung auf Referenz-Geometrie</b>	<b>48</b>
6.1. Vermessung eines Propellers . . . . .	48
6.2. Lastrechnung . . . . .	50
6.3. Definition der Designvariablen . . . . .	52
6.4. Konfiguration des Analysemodells . . . . .	53
6.5. Einstellung der Optimierungs-Parameter . . . . .	56
6.6. Optimierung mit Beta-Zielfunktion . . . . .	58
6.6.1. Optimierungsproblem . . . . .	58
6.6.2. Prozessverlauf . . . . .	59
6.6.3. Ergebnisse . . . . .	61
6.7. Weiterführende Auswertung mit Python und JupyterLab . . . . .	63
6.7.1. Methodik . . . . .	63
6.7.2. Diskussion der Ergebnisse und Post-Processing . . . . .	65
6.8. Direkte Optimierung der Bauteilmasse . . . . .	67
6.8.1. Optimierungsproblem . . . . .	67
6.8.2. Ergebnisse . . . . .	68
<b>7. Zusammenfassung und Ausblick</b>	<b>70</b>
<b>Literatur</b>	<b>72</b>
<b>A. Verwendete Software</b>	<b>76</b>
<b>B. Inhalt Daten-CD</b>	<b>77</b>
<b>C. UML-Diagramme</b>	<b>78</b>

# Abbildungsverzeichnis

2.1.	Lösung eines eindimensionalen Optimierungsproblems . . . . .	4
2.2.	Vergleich zweier Zielfunktionen mit unterschiedlicher Anzahl an Minima . . . . .	5
2.3.	Veranschaulichung eines mehrdimensionalen, nicht-konvexen Designraums . . . . .	6
3.1.	Geschwindigkeitskomponenten beim Partikelschwarmverfahren . . . . .	14
3.2.	Modell einer Faserverbundplatte unter Schubbelastung . . . . .	16
3.3.	Hauptspannungszustand bei Schubbelastung nach SCHÜRMANN [39]. . . . .	18
3.4.	Schematische Skizze des Faserverbundbalkens . . . . .	19
3.5.	Pfadauswertung des Faserbruchkriteriums in den Holmgurten . . . . .	21
4.1.	Approximation der $c_a$ - $\alpha$ -Kurven in Python . . . . .	26
4.2.	UML-Klassendiagramm des Lastrechnungsprogramms . . . . .	27
4.3.	Übersicht über die Loadcase-Klasse . . . . .	28
4.4.	Übersicht über die Airfoil-Klasse . . . . .	29
4.5.	Übersicht über die Propeller-Klasse . . . . .	30
5.1.	UML-Klassendiagramm des Optimierungstools . . . . .	35
5.2.	Übersicht über die Femodel-Klasse . . . . .	36
5.3.	Übersicht über die Material-Klasse . . . . .	37
5.4.	Übersicht über die Propellermodel-Klasse . . . . .	38
5.5.	CAD-Modell der Mittenfläche eines Propellerblatts . . . . .	40
5.6.	FE-Netz der Mittenfläche eines Propellerblatts . . . . .	41
5.7.	FE-Netz der Propellerblatts mit Visualisierung der Elementdicke . . . . .	41
5.8.	FE-Netz mit aufgeprägten Druckkräften . . . . .	42
5.9.	FE-Netz mit aufgeprägten Reibungskräften . . . . .	43
5.10.	Veranschaulichung der Beta-Methode . . . . .	45
6.1.	Photogrammetrische Vermessung eines Querschnitts mit OPENCV. . . . .	49
6.2.	Ermittelte Verteilung von Schränkung und Profiltiefe des Propellers . . . . .	50
6.3.	Druckverteilung der Saugseite des Propellers für den definierten Lastfall. . . . .	51
6.4.	Veranschaulichung der Desingvariablen der Komponenten. . . . .	53
6.5.	Validierung der Lastaufprägung: Vergleich Reaktionskräfte-, Momente . . . . .	55
6.6.	Optimierungsproblems in Abhängigkeit verschiedener Prozessparameter . . . . .	57
6.7.	Verlauf von Zielfunktion und Restriktionen für verschiedene Iterationsschritte $k$ . . . . .	60
6.8.	Werte der Designvariablen $\rho_j$ und $\delta_j$ im Optimum $\vec{x}^* = \vec{x}^{80}$ . . . . .	62
6.9.	Interaktive Auswertung der Restriktionen mit JUPYTERLAB. . . . .	64
6.10.	Interaktive Auswertung der Versagenskriterien mit JUPYTERLAB. . . . .	64
6.11.	Vergleich Optimierungsergebnisse sowie Nachbearbeitung . . . . .	66
6.12.	Ergebnisse der direkten Optimierung der Bauteilmasse . . . . .	69
C.1.	UML-Klassendiagramm des Lastrechnungsprogramms . . . . .	78
C.2.	UML-Sequenzdiagramm des Ablaufs der Programmkette . . . . .	79

---

# Abkürzungs- und Symbolverzeichnis

## Abkürzungen

APDL	ANSYS Parametric Design Language
ALPSO	Augmented Lagrangian Particle Swarm Optimization
CAD	Computer Aided Design
FEM	Finite-Elemente-Methode
FE-Modell	Finite-Elemente-Modell
FVK	Faser-Kunststoff-Verbund
NACA	National Advisory Committee for Aeronautics
NSGA2	Non Sorting Genetic Algorithm II
PSO	Particle Swarm Optimization
SLSQP	Sequential Least Squares Quadratic Programming

## Formelzeichen

### Allgemein Physikalisch

$A$	$\text{m}^2$	Fläche
$B$	–	Anzahl der Blätter
$c/R$	–	relative Profiltiefe eines Propellerblatts
$c$	$\text{mm}$	Profiltiefe
$c_A$	–	Auftriebsbeiwert
$c_f$	–	Reibungsbeiwert
$c_p$	–	Druckbeiwert
$D$	$\text{mm}$	Durchmesser des Propellerkreises
$E$	$\text{GPa}$	Elastizitätsmodul
$F, P$	$\text{kN}$	Kraft
$G$	$\text{Mpa}$	Schubmodul
$I_{\text{fib}}$	–	Versagensindex auf Faserbruch nach PUCK
$I_{\text{mat}}$	–	Versagensindex auf Zwischenfaserbruch nach PUCK
$l$	$\text{m}$	Länge
$M$	$\text{kNm}$	Biegemoment
$n$	$1/\text{min}$	Drehzahl
$r/R$	–	relativer Radius eines Propellerblatts
$R$	$\text{mm}$	Radius des Propellerkreises
$Re$	–	Reynoldszahl
$T$	$\text{N}$	Propellerschub
$v$	$\text{m/s}$	Anströmgeschwindigkeit
$\alpha$	$^\circ$	aerodynamischer Anstellwinkel
$\beta$	$^\circ$	Schränkungswinkel
$\rho$	$\text{g/cm}^3$	Dichte
$\sigma_x$	$\text{MPa}$	Normalspannung
$\nu$	–	Querkontraktionszahl
$\omega$	$1/\text{s}$	Winkelgeschwindigkeit

**Optimierung Allgemein**

$f(\vec{x})$	Zielfunktion
$g_j(\vec{x})$	Ungleichheitsrestriktion $j$
$h_k(\vec{x})$	Gleichheitsrestriktion $k$
$k$	Iterationszahl
$\vec{p}^k$	Suchrichtung zum Iterationsschritt $k$
$\vec{x}$	Vektor der Designvariablen
$\vec{x}^k$	Vektor der Designvariablen zum Iterationsschritt $k$
$\vec{x}^*$	Vektor der Designvariablen im Minimum
$x_i^L \leq x_i \leq x_i^U$	Explizite Restriktion $i$
$\lambda$	Schrittweite

**ALPSO-Algorithmus**

$c_1$	Kognitiver Parameter
$c_2$	Sozialer Parameter
$\vec{g}_i^k$	Beste Position aus der Population zum Iterationsschritt $k$
$k_{max}$	Anzahl der inneren Iterationen
$\mathcal{L}(\vec{x}_i^k, \vec{\lambda}, \vec{r}_p)$	Erweiterte Lagrange-Funktion
$P_s$	Populationsgröße
$\vec{p}_i^k$	Beste Position des Partikels $i$ zum Iterationsschritt $k$
$r_1, r_2$	Zufallszahlen $0 \dots 1$
$\vec{r}_p$	Straffaktoren
$\Delta t$	Zeitschritt
$\vec{v}_i^k$	Geschwindigkeit des Partikels $i$ zum Iterationsschritt $k$
$w_1$	Anfangs-Trägheit
$w_2$	End-Trägheit
$\vec{x}_i^k$	Ort des Partikels $i$ zum Iterationsschritt $k$
$\vec{\lambda}$	Lagrange-Multiplikatoren

# 1. Einleitung

Die in der Populärwissenschaft oft zitierte MOORE'sche Gesetzmäßigkeit (englisch MOORE's law) besagt, dass sich die Anzahl an Transistoren, die in einen integrierten Schaltkreis festgelegter Größe passen, etwa alle zwei Jahre verdoppelt. Diese Aussage wird häufig wie folgt interpretiert [44]:

*Die Prozessor-Leistung für Computer verdoppelt sich alle 2 Jahre.*

Für die Ingenieurwissenschaften ergibt sich aus der zunehmenden Leistungsfähigkeit moderner Computer die Möglichkeit, immer aufwendigere, simulationsgestützte Entwicklungsmethoden zu verwenden.

Zudem werden fortlaufend innovative Berechnungsmethoden sowie Algorithmen entwickelt und oft als *Freie Software* [17] veröffentlicht. Ein Beispiel für diese Entwicklung ist die Programmiersprache *Python*, für die inzwischen freie Bibliotheken verschiedenster wissenschaftlicher und ingenieurwissenschaftlicher Disziplinen verfügbar sind:

Mit PyMAPDL [23], einem interaktiven Interface zu dem Finite-Elemente-Programm ANSYS MECHANICAL APDL, kann ein kompletter FEM-Simulationsprozess in ein Python-Programm eingebunden werden. In Python implementierte Optimierungsalgorithmen, z.B. aus der Bibliothek PyOpt [31], können dann mit geringem Programmierungs-Aufwand zur Durchführung automatisierter Parameterstudien verwendet werden.

Die Idee dieser Masterarbeit ist es, mit Vorgehensweisen der Strukturoptimierung und freien Toolboxen aus Python ein leistungsfähiges, automatisiertes Auslegungsprogramm für anisotrope UAV-Propeller zu entwickeln.

Der Markt für UAVs (englisch *unmanned aerial vehicles - unbemannte Luftfahrzeuge*), z.B. Multikopter, ist in den letzten Jahren stark gewachsen, da sie für eine Vielzahl unterschiedlichster Missionen verwendet werden können, für die mantragende Technologien ungeeignet oder zu teuer erscheinen.

Die Gestalt der Propeller, als Schub-erzeugende Komponenten, hat einen maßgeblichen Einfluss auf die Effizienz ihrer Antriebe und damit der gesamten Luftfahrzeuge. Angestrebt werden möglichst schlanke und leistungsfähige Geometrien bei gleichzeitig geringstmöglicher Masse.

Die vorliegende Arbeit soll die strukturmechanische Realisierung einer solchen Propeller-Gestalt ermöglichen, dazu ist sie wie folgt aufgebaut:

- Übersicht über die Strukturoptimierung und Stand der Technik
- Auswahl und Evaluierung eines Optimierungsverfahrens
- Erstellung eines Lastrechnungs-Tools für UAV-Propeller
- Erstellung eines Optimierungs-Tools
- Anwendung auf eine Propeller-Referenzgeometrie
- Zusammenfassung und Ausblick

## 2. Stand der Technik

### 2.1. Motivation der Strukturoptimierung

Die Strukturoptimierung ist ein Konzept für die Entwicklung mechanischer Strukturen. Sie kann überall dort eingesetzt werden wo Verbesserungspotential besteht, und versucht, dieses mit mathematischen Strategien möglichst komplett auszuschöpfen.

Typische Aufgabenstellungen in der Strukturoptimierung sind z.B. [18]:

- Minimiere die Masse einer Tragstruktur, sodass die zulässigen Festigkeits- und Stabilitätskriterien sowie Verformungen nicht überschritten werden.
- Verbessere die akustischen oder elastischen Eigenschaften eines Bauteils durch das Verschieben der Eigenfrequenzen in Bereiche, deren Moden kaum oder gar nicht angeregt werden.

Als Ausgangspunkt für die Optimierung wird dabei ein mathematisches Modell des mechanischen Verhaltens benötigt, das sogenannte *Analysemodell*. Die Komplexität des Modells ist abhängig von der Aufgabenstellung. In der Regel werden numerische Methoden, wie die *Methode der Finiten Elemente* eingesetzt. Für einfache Probleme können aber analytische Ansätze genügen.

Ebenso ist es möglich, eine aus vorhandenen Daten aufgebaute Approximation als Analysemodell zu verwenden.

Grundsätzlich hat die Strukturoptimierung zum Ziel, ein Produkt hinsichtlich der gegebenen Anforderungen zu verbessern. Deren Definition geschieht nach SCHUMACHER [38] anhand folgender Leitfragen:

- Was ist die Zielsetzung der Optimierung? Kann man für die Aufgabe Zielfunktionen definieren, die abhängig von den Einflussgrößen, den sog. *Designvariablen*, sind?
- Welche Nebenbedingungen (*Restriktionen*) können definiert werden? Sind diese Restriktionen ebenfalls von den Entwurfsvariablen abhängig?

Für die *Designvariablen*:

- Welche Größen im Analysemodell der Bauteilstruktur können verändert werden?
- Von welchen Größen kann welcher Einfluss auf das Bauteilverhalten erwartet werden?

Für die *Restriktionen*:

- Welchen Anforderungen soll die Bauteilstruktur genügen?
- In welcher Beziehung stehen die Anforderungen zu den Designvariablen?

## 2.2. Allgemeine Formulierung eines Optimierungsproblems

Ein Optimierungsproblem mit  $n$  Designvariablen,  $m$  Ungleichheitstrestrktionen und  $q$  Gleichheitsrestriktionen kann in der folgenden Form geschrieben werden [18]:

$$\min f(\vec{x}),$$

sodass

$$\begin{aligned} g_j(\vec{x}) &\leq 0; & j &= 1, m \\ h_k(\vec{x}) &= 0; & k &= 1, q \\ x_i^L \leq x_i &\leq x_i^U; & i &= 1, n \end{aligned}$$

Dabei wird  $f$  als Zielfunktion bezeichnet, die Ungleichheitsrestriktionen mit  $g_j(\vec{x})$  und die Gleichheitsrestriktionen mit  $h_k(\vec{x})$ . Die Beschränkungen der Designvariablen auf ein Intervall mit den unteren Grenzen  $x_i^L$  und oberen Grenzen  $x_i^U$  werden explizite Restriktionen genannt.

Die hier dargestellte Form, bei der die Zielfunktion minimiert wird, entspricht der allgemeinen Konvention. Probleme, bei denen die Zielfunktion maximiert werden soll, werden auf diese zurückgeführt, indem man die negative Zielfunktion minimiert:

$$\max f(\vec{x}) = \min (-f(\vec{x}))$$

Bei den Restriktionen herrschen in der Praxis meist die Ungleichheitsrestriktionen vor. Gleichheitsrestriktionen spielen eine eher untergeordnete Rolle, da für die meisten Anwendungen das Einhalten gewisser Grenzen und nur selten ein exakter Zielwert gefordert wird.

Bei Ungleichheitsrestriktionen kann man drei Fälle unterscheiden:

- $g_j < 0$  Restriktion erfüllt und nicht aktiv.
- $g_j = 0$  Restriktion erfüllt und aktiv.
- $g_j > 0$  Restriktion verletzt.

Der Bereich, in dem alle Restriktionen erfüllt sind, heißt zulässiger Bereich. Das Ziel der Optimierung ist es, innerhalb dieses Bereiches einen Vektor  $\vec{x}^*$  zu finden, der die Zielfunktion so minimiert, dass alle Restriktionen eingehalten werden.

Die Suche nach diesem Vektor erfolgt mittels eines iterativen Prozesses, bei dem der Algorithmus in jeder Iteration bestimmt, um welchen Wert  $\Delta\vec{x}^{k-1}$  die Variablen zu verändern sind.

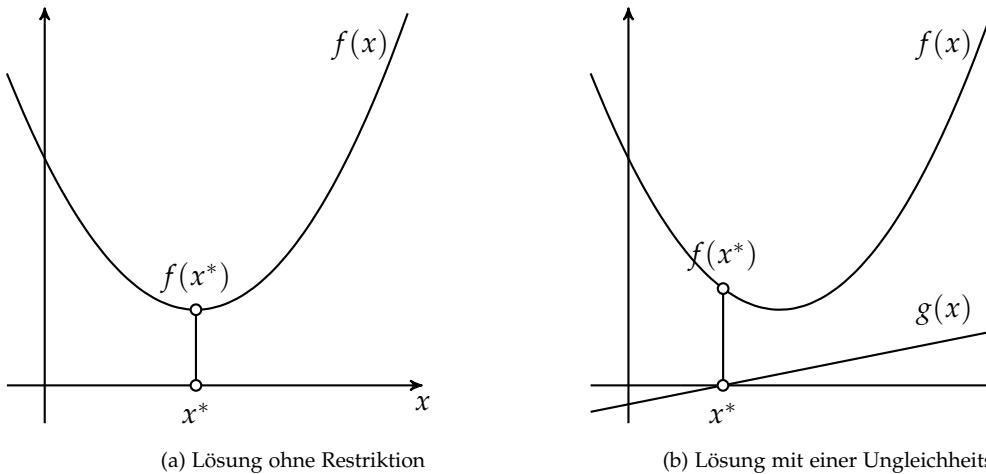


Abbildung 2.1.: Lösung eines eindimensionalen Optimierungsproblems mit und ohne Restriktion

$$\vec{x}^k = \Delta\vec{x}^{k-1} + \vec{x}^{k-1}$$

Zur Veranschaulichung sind in Abbildung 2.1 die Lösungen zweier Optimierungsprobleme dargestellt. Die Zielfunktion ist dabei eine einfache Parabel, und es existiert nur eine Designvariable  $x$ . Das Problem ist also eindimensional.

- Für das unbeschränkte Problem in Abbildung 2.1a entspricht die Lösung dem Minimum der Funktion.
  - In Abbildung 2.1b wurde eine Ungleichheitsrestriktion  $g(x)$  hinzugefügt. Für das Optimum muss also gelten:  $g(x) \leq 0$ .
- Das unbeschränkte Minimum liegt nun im unzulässigen Bereich. In dem beschränkten Minimum ist die Restriktion erfüllt und aktiv.

### 2.3. Globale und lokale Minima

In der Praxis gibt es Fälle, in denen die Zielfunktion mehrere Minima aufweist. Daraus folgt, dass das entsprechende Optimierungsproblem mehr als nur eine Lösung hat. In Abbildung 2.2 sind zwei Optimierungsprobleme dargestellt, für die nur explizite Restriktionen gelten. Während Abbildung 2.2a nur eine einzige Lösung hat, existieren in Abbildung 2.2b drei Minima. Das Minimum mit dem kleinsten Funktionswert wird als globales Minimum bezeichnet, die anderen als lokale Minima.

Für einen bestmöglichen Entwurf wird in der Praxis nach dem globalen Optimum gesucht. Bisher gibt es jedoch keine Optimierungsalgorithmen, die ausschließlich im globalen Minimum konvergieren [18]. Gerade Verfahren, die nur eine einzige Suchrichtung  $\Delta\vec{x}^{k-1}$  verwenden, bewegen sich von ihrem Startwert aus immer abwärts zum nächsten Minimum.

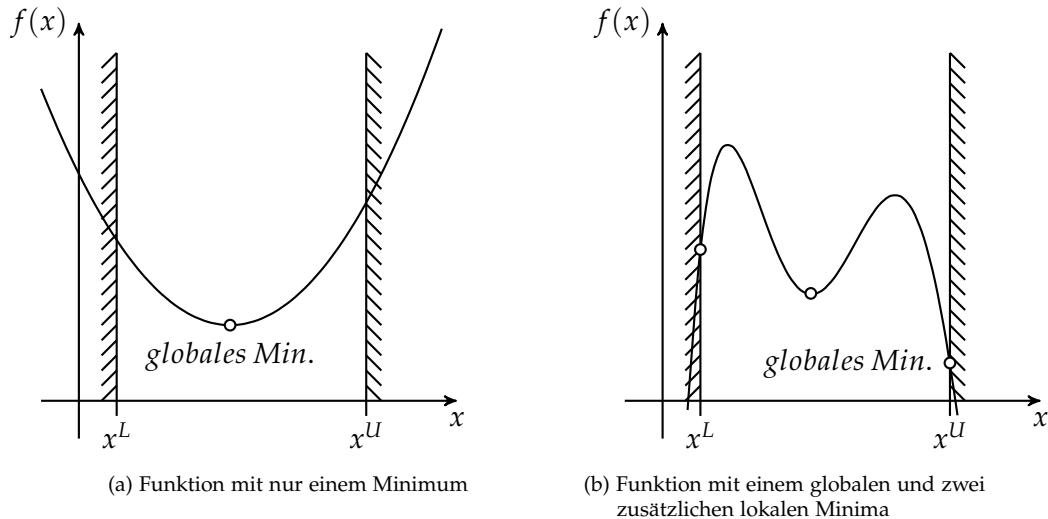


Abbildung 2.2.: Vergleich zweier Zielfunktionen mit unterschiedlicher Anzahl an Minima

Abhängig vom Startpunkt finden solche Optimierer also unterschiedliche (lokale) Minima. Im mehrdimensionalen Raum, wie in Abbildung 2.3 illustriert, führt dieses Verhalten sozusagen in Seitentäler der Zielfunktion und nicht zwangsläufig zum Ort mit dem tiefsten Funktionswert.

Es existiert zwar ein mathematisches Kriterium um zu überprüfen, dass es im Suchbereich des Verfahrens nur ein einziges Minimum gibt (sog. *Konvexität*), dieses kann jedoch nur für einfache Fälle ausgewertet werden [38]. Meistens ist also nicht bekannt, ob das zu lösende Problem mehrere Optima hat und ob das gefundene Minimum ein lokales ist.

Daraus folgen für die ingenieurmäßige Praxis einige Einsichten:

- Bei Verfahren, die Suchrichtungen verwenden, werden im Zweifel verschiedene Startpunkte ausprobiert oder mit Methoden der Versuchsplanung bestimmt.
- Vom pragmatischen Standort aus gesehen stellt auch das Erreichen eines lokalen Optimums eine Verbesserung dar, selbst wenn womöglich noch eine bessere Lösung existiert.

## 2.4. Übersicht gängiger Optimierungsverfahren

Aus den zuvor skizzierten Beweggründen haben sich eine Reihe unterschiedlicher Optimierungsverfahren entwickelt, die je nach Art der Optimierungsaufgabe Verwendung finden. Diese können nach NIKBAKT u. a. [28] in drei Klassen eingeteilt werden:

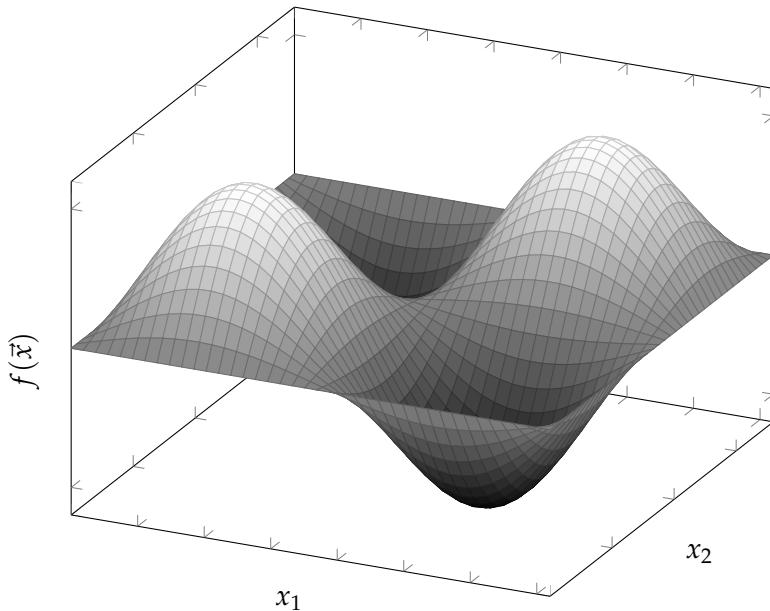


Abbildung 2.3.: Veranschaulichung eines mehrdimensionalen, nicht-konvexen Designraums

**Blinde Algorithmen** Diese Verfahren sind nur in der Lage, die Zielfunktion und ggf. Restriktionen auszuwerten. Sie benötigen einen großen Suchraum und haben eine geringe Konvergenzrate. Daher werden sie hier nicht ausführlicher betrachtet.

**Suchrichtungsmethoden** Die Strategie dieser Verfahren ist es, eine Suchrichtung festzulegen, und das Optimierungsproblem entlang dieser Richtung zu minimieren. Die Suchrichtung wird dabei von Iteration zu Iteration verändert.

Durch die Verwendung einer Suchrichtung wird das mehrdimensionale Problem auf ein eindimensionales zurückgeführt, für das sich die Extrema mit geringem Aufwand bestimmen lassen.

Der Algorithmus einer Suchrichtungsmethode lautet nach [18] wie folgt:

1. Nenne den Startpunkt  $\vec{x}^0$  und setze  $k = 0$ .
2. Bestimme eine Suchrichtung  $\vec{p}^{k+1}$  und löse das eindimensionale Optimierungsproblem bezüglich der Variablen für die Schrittweite  $\lambda$ .

$$\min f(\vec{x}^k + \lambda \cdot \vec{p}^{k+1}) = \min f(\lambda)$$

3. Bestimme den neuen Punkt  $\vec{x}^{k+1} = \vec{x}^k + \lambda^* \cdot \vec{p}^{k+1}$ , wobei  $\lambda^*$  die Lösung des Problems aus Schritt zwei ist.
4. Überprüfe Konvergenzkriterien. Wenn nicht erfüllt, setze  $k = k + 1$  und gehe nach 2.
5. Optimum ist  $\vec{x}^* = \vec{x}^{k+1}$ .

Das Knowhow dieser Verfahren liegt in der Art der Suchrichtungsbestimmung. Üblicherweise werden dazu Gradienten, also Ableitungen der Zielfunktion verwendet.

Das Spektrum reicht von sehr einfachen Verfahren wie der *Methode des steilsten Abstiegs*, bei der die Suchrichtung einfach dem negativen Gradienten der Zielfunktion  $\vec{p}^k = -\vec{\nabla}f(\vec{x}^{k-1})$  entspricht, bis hin zu komplexeren Algorithmen, welche neben dem Gradienten noch die Hesse-Matrix der Zielfunktion verwenden.

Vorteil dieser Klasse von Verfahren ist eine hohe Konvergenzrate, da durch die Verwendung von Gradienten sehr schnell ein Minimum gefunden werden kann. Da die Suche immer entlang des Abstiegs der Funktion führt, konvergiert der Algorithmus abhängig vom Startwert üblicherweise im nächstgelegenen lokalen Minimum. Der Wahl des Startwerts kommt also große Bedeutung zu.

Eine Voraussetzung für die Verwendung ist jedoch, dass Zielfunktion und Restriktionen stetig und differenzierbar sind, sodass Gradienten gebildet werden können.

Für die Anwendung in der Strukturoptimierung bedeutet dies, dass es nicht möglich ist, diskrete Designvariablen zu verwenden (etwa die Anzahl an Lagen eines Fasermaterials oder Blechstärken).

Wenn als Analysemodell ein FE-Modell verwendet wird, muss darüber hinaus eine Sensitivitätsanalyse durchgeführt werden, also eine numerische Bestimmung der Gradienten von Zielfunktion und Restriktionen. Dies kann einen hohen Implementierungs- und Rechenaufwand bedeuten und ist entscheidend dafür, wie schnell und effektiv die Optimierung verläuft [18].

**Globale Optimierungsverfahren** Globale Optimierungsverfahren verzichten auf die zuvor beschriebenen mathematischen Konzepte von Suchrichtungen, und weisen daher auch keine eindimensionale Optimierung auf. Daher werden diese oft auch als *gradientenfreie Verfahren* bezeichnet. Dies führt einerseits zu einer deutlichen Erhöhung des Rechenaufwandes, andererseits versetzt es den Algorithmus in die Lage, lokale Optima auf der Suche nach dem globalen Minimum zu überspringen. Die Beschränkung auf konvexe Probleme zum finden des Globalen Minimums besteht für diese Klasse von Verfahren also nicht.

Anstelle von Gradienten verwenden diese Algorithmen oft Konzepte, die an Prozesse in Biologie und Natur angelehnt sind, sowie zufällige Komponenten. SCHUMACHER [38] vergleicht sie daher im mathematischen Sinne eher mit Suchstrategien anstelle von Optimierungsalgorithmen.

Die enthaltene Zufallskomponente bewirkt einen nicht deterministischen Verlauf der Optimierung. Es gibt also keine Garantie, tatsächlich das globale Optimum zu identifizieren. Zwei Konzepte, die häufig Verwendung finden, werden im folgenden kurz vorgestellt.

- *Evolutionäre Algorithmen* bilden die biologische Evolution nach. Durch diese findet in der Natur ein Optimierungsprozess statt, bei dem nur die besten Individuen überleben und sich fortpflanzen können. Die Genetik, und damit die Eigenschaften der Nachfahren werden durch Rekombination und Mutation verändert. Durch den

sich wiederholenden Prozess von Fortpflanzung und Selektion verbessert sich die Anpassung der Generationen fortlaufend, bis schließlich ein Optimum erreicht ist.

- Die *Optimierung mit Teilchenschwärmern* ist ebenfalls ein natur-analoges Verfahren und simuliert die soziale Wechselwirkung von Schwärmen im Tierreich, beispielsweise von Vogelschwärmen auf der Suche nach Futterplätzen. Dabei teilen die einzelnen Individuen ihr Wissen und profitieren daraus so sehr, dass das kollektive Gedächtnis des Schwarms den Nachteil des Wettbewerbs um das Futter mehr als aufwiegt. Der bekannteste Vertreter dieser Gruppe ist das *Partikelschwarmverfahren (PSO)*.

## 2.5. Strukturoptimierung von Faserverbundwerkstoffen

Verglichen mit isotropen Werkstoffen wird das mechanische Verhalten der Faserverbundwerkstoffe von einer deutlich größeren Anzahl an Parametern beeinflusst. Neben den wichtigsten Größen wie der Lagenorientierung und -dicke können auch Faservolumengehalt, Anzahl der Lagen, Stapelreihenfolge, Material von Faser und Matrix etc. als Designvariablen in einem Optimierungsprozess verwendet werden.

Auch für die Zielsetzung der Optimierung bietet sich ein vielfältiges Spektrum: Als Zielfunktion können die Beulstabilität, Eigenfrequenzen, Tragfähigkeit, Verformungen oder Spannungen gewählt werden.

Ebenso hat die Art der zu optimierenden Tragstruktur einen Einfluss auf den Optimierungsprozess. Hier wird üblicherweise in Balkenstrukturen, Scheiben und Platten unterschieden. Eine Übersichtsarbeit über alle Themenbereiche der Faserverbund-Optimierung bieten NIKBAKT ET AL. in [28] und [29]:

**Balkenstrukturen** Balkenstrukturen stellen meist die hauptsächlich lasttragende Komponente in Flugzeug-Tragflügeln und Helikopter Rotorblättern dar. Zur Optimierung dieser Tragstrukturen existieren verschiedene Veröffentlichungen:

- Die Optimierung von Querschnittsabmessungen und Lagenorientierungen in Rotorblättern mit einem PSO-basierten Ansatz wurde von SURESH ET AL. in [40] untersucht. In einem Vergleich mit Evolutionären Algorithmen zeigte die PSO für diesen Anwendungsfall eine höhere Effizienz.
- Ein Vergleich von Suchrichtungsmethoden (GBM) und PSO bei der Optimierung von Hubschrauber-Rotorblättern wurde in [24] durchgeführt.

Als Analysemodell diente ein FE-Modell einer Balkenstruktur, welches auf das Tsai-Wu-Versagenskriterium ausgewertet wurde. Die global besten Ergebnisse zeigte das PSO-Verfahren. Die GBM-Methode lieferte bei gut gewählten Startpunkten brauchbare Ergebnisse mit einer hohen Effizienz.

**Faserverbund-Platten und -Schalen** Als Platten werden Bauteile bezeichnet, deren Abmessungen in einer Dimension im Vergleich zu den beiden anderen viel Geringer sind. Diese finden z.B. Anwendung als Schalenbauteile in Flugzeugräumen.

Zur Optimierung dieser Strukturen in Bezug auf ihr Gewicht wurden unter anderem folgende Arbeiten durchgeführt:

- Mehrere Veröffentlichungen handeln von der Optimierung der Stapelreihenfolge [36, 1]. Dabei wurden hauptsächlich genetische Algorithmen, also Vertreter der Evolutionsstrategien, verwendet. Es zeigte sich, dass die Effizienz je nach Art dieses Verfahrens stark variierte.
- Eine Kopplung von genetischen Algorithmen mit einem FE-Analysemodell wurde in [14] untersucht. Dabei wurden die Lagenorientierungen, Anzahl der Lagen, und Materialauswahl optimiert.
- Zur Optimierung von Stapelreihenfolgen, Lagendicken oder Beulsteifigkeiten mit Teilchenschwärmern existieren in [8, 26, 3] diverse Studien. Dabei zeigte sich eine hohe Effizienz und Zuverlässigkeit dieser Verfahren.

Die oben beschriebenen Studien bezogen sich in ihrer Zielsetzung zumeist auf die Verbesserung von Gewicht oder Versagenskriterien. Darüber hinaus existieren eine Vielzahl an Veröffentlichungen zu anderen Zielfunktionen wie Beulsteifigkeit, Vibrationsverhalten, Steifigkeit, Verformung oder smarten Mechanismen.

Ein Konsens der Forschung zu Faserverbundoptimierung ist die bevorzugte Verwendung globaler Optimierungsverfahren. Die Begründung dafür könnte einerseits der eingangs erwähnte große Parameterspielraum, andererseits die einfache Verwendbarkeit dieser Verfahren mit FE-Modellen sein.

### 3. Auswahl und Validierung eines Optimierungsverfahrens

Im Rahmen des Projekts *MesSBAR* [7] wird bei der Firma *Leichtwerk Research GmbH* ein Multicopter mit vier Propeller-Antrieben und einer maximalen Abflugmasse von 25 kg entwickelt. Für dieses Projekt sind Propeller mit einem Durchmesser von etwa 400 mm und ca. 175 N Standschub vorgesehen.

Im Rahmen einer vorangegangenen Arbeit [20] wurde die Möglichkeit evaluiert, Propeller dieser Größenordnung mit erneuerbaren Werkstoffen herzustellen. Es konnte gezeigt werden, dass mit Naturfaserverbundwerkstoffen, etwa aus Flachfasern, eine ausreichende Festigkeit erreicht werden kann. Um eine geringe Strukturmasse zu gewährleisten, wurde die Verwendung einer Sandwichbauweise empfohlen.

Innerhalb dieses Kapitels soll ein geeignetes Optimierungsverfahren für die automatisierte Auslegung solcher Propeller gefunden und validiert werden.

#### 3.1. Anforderungen an den Optimierungsalgorithmus

In diesem Abschnitt sollen die Anforderungen an das zu verwendende Optimierungsverfahren definiert werden.

**1. Eignung für Balkenstrukturen** In Abschnitt 2.5 wurde eine Einführung in die Grundzüge der Optimierung von Faserverbundwerkstoffen und verwendeter Algorithmen gegeben. Um für die Verwendung für Propeller geeignet zu sein, soll der Optimierer effizient mit den Eigenheiten von Balkenstrukturen arbeiten können.

**2. Python und freie Software** Aufgabenstellung dieser Arbeit ist die Implementierung eines Programmsystems in Python, dafür ist zumindest eine Programmierschnittstelle (API) zu Python oder zur Shell des Betriebssystems notwendig.

Besser ist eine komplette Implementierung in Python als freie Software. Dies würde erlauben, den Algorithmus zu forken und frei Änderungen zu implementieren, um sein Verhalten auf das Optimierungsproblem anzupassen.

**3. Verwendbarkeit von FE-Modellen** Es ist geplant, das Analysemodell der Optimierung mit der Finite-Elemente-Methode zu implementieren. Der verwendete Algorithmus muss dafür geeignet sein. Dies bedeutet, wie in Abschnitt 2.4 beschrieben, dass auf die Verwendung von Gradienten entweder verzichtet wird, oder deren numerische Bestimmung sehr effizient abläuft.

**4. Verarbeitung diskreter Designvariablen** Viele Optimierungsverfahren können ausschließlich kontinuierliche Designvariablen verarbeiten. Bei der Optimierung Faserverbundwerkstoffen existieren allerdings einige diskrete Größen, etwa Lagendicke oder Materialkennwerte. Dafür ist eine Funktionalität für diskrete Designvariablen notwendig.

**5. Umgang mit Restriktionen** Einige Optimierungsalgorithmen bieten keine Unterstützung für die Behandlung von Restriktionen. Um dennoch beschränkte Optimierungsprobleme lösen zu können, müssen Zielfunktion und Restriktionen in eine gemeinsame *Strafffunktion* (*penalty function*) überführt werden, sodass der Algorithmus auch die Verletzung der Restriktionen minimiert.

Da dieses zusätzlichen Implementierungsaufwand bedeutet und unter Umständen das Konvergenzverhalten negativ beeinträchtigt [18], ist ein nativer Umgang mit Restriktionen zu bevorzugen.

**6. Modularer Aufbau** Wünschenswert wäre ein objektorientierter Aufbau als Toolbox, die mehrere Algorithmen bereitstellt, und die Unabhängigkeit zwischen der Formulierung des Optimierungsproblems und seiner Lösung durch verschiedene Optimierer aufrechterhält. So können unterschiedliche Verfahren ausprobiert werden, falls Probleme auftreten. Weiterhin würde dies eine zweistufige Optimierungsstrategie ermöglichen, bei der zuerst mit einem globalen Verfahren die ungefähre Region des absoluten Minimums lokalisiert und anschließend mit Hilfe einer Suchrichtungsmethode genau bestimmt wird.

**7. Parallelisierung** Da die Verwendung gradientenfreier Optimierungsverfahren numerisch aufwendig ist, soll nach Möglichkeit eine parallelisierte Ausführung des Optimierungsalgorithmus unterstützt werden, um eine akzeptable Lösungsdauer zu gewährleisten.

## 3.2. Vorauswahl eines Verfahrens

Auf Grundlage der zuvor definierten Anforderungen und der in Abschnitt 2.5 dargestellten Erkenntnisse wurde eine Recherche nach möglichen Optimierungsalgorithmen durchgeführt. Da für die Optimierung von Balkenstrukturen in der Literatur hauptsächlich die Optimierung mit Teilchenschwärmern empfohlen wird [40, 24, 8, 26, 3], lag der Fokus der Recherche auf dieser Klasse von Verfahren.

Mit Abstrichen geeignet erscheinen die Evolutionsstrategien, sodass auch diese in Betracht gezogen werden.

Tabelle 3.1 zeigt eine Übersicht über die in Betracht kommenden, frei verfügbaren Optimierungswerkzeuge.

Die Bibliotheken *PySwarms* und *PSOPy* stellen jeweils einen Optimierer auf Basis von Teilchenschwärmern zur Verfügung (*PSO: Particle Swarm Optimization*). Sie unterstützen

	PySwarms [27]	PSOPy [41]	PyOpt [31]	PyGMO [2]
Verfahren	PSO	PSO	ALPSO NSGA <sub>2</sub> SLSQP ...	PSO NSGA <sub>2</sub> SLSQP ...
Eignung für Balkenmodelle	✓	✓	✓	✓
Freie Software	✓	✓	✓	✓
FE-Modelle verwendbar	✓	✓	✓	✓
Diskrete Designvariablen	✗	✗	✓	✗
Restriktionen unterstützt	✗	✗	✓	nur SLSQP
Modularer Aufbau	✗	✗	✓	✓
Parallelisierung	✗	✗	✓	✓

Tabelle 3.1.: Vergleich frei verfügbarer Optimierungsbibliotheken

allerdings kein diskreten Designvariablen und auch keine Restriktionen. Diese müssten also unter Verwendung einer Straffunktion implementiert werden.

Im Vergleich dazu bieten die Bibliotheken PyOpt und PyGMO eine umfangreiche Sammlung von Optimierungswerkzeugen (jeweils < 20) zur Lösung unterschiedlicher Probleme. Darunter befinden sich sowohl globale Optimierungsstrategien verschiedener Ansätze wie auch lokale Suchrichtungsmethoden:

- (AL-) PSO: *(Augmented Lagrangian) Particle Swarm Optimizer*  
Optimierung mit Teilchenschwärmern. In seiner Grundform kann dieses Verfahren keine beschränkten Probleme lösen. PyOpt bietet eine erweiterte Variante, welche Konzepte von Suchrichtungsmethoden verwendet um beschränkte Probleme lösen zu können.
- NSGA<sub>2</sub>: *Non Sorting Genetic Algorithm II*  
Dies ist ein Vertreter der Generischen Algorithmen (s. Abschnitt 2.4). Der in PyOpt implementierte Optimierer kann dabei auch restriktive Probleme lösen.
- SLSQP: *Sequential Least Squares Quadratic Programming*  
Die sequentielle quadratische Programmierung ist ein gradientenbasiertes Suchrichtungsverfahren. Dieses bietet die Möglichkeit für eine Mehrstufige Optimierungsstrategie oder für Detailoptimierungen ausgehend von einem händischen Vorentwurf.

Das Profil der PyOpt-Bibliothek, insbesondere mit dem ALPSO-Optimierer passt sehr gut zu den zuvor definierten Anforderungen. ALPSO unterstützt über kontinuierliche Designvariablen hinaus diskrete und ganzzahlige Variablen. Weiterhin bietet es strukturell gute Voraussetzungen für eine Parallelisierung des Optimierungsprozesses, dies wird im Folgenden noch ausführlicher beschrieben werden.

### 3.3. Partikelschwarmverfahren

**Grundlegender Algorithmus** Die Partikelschwarmoptimierung ist ein populationsbasiertes, gradientenfreies Optimierungverfahren, welches auf dem Verhalten basiert, das Schwärme im Tierreich bei der Nahrungssuche zeigen. Die einzelnen Individuen teilen dabei ihr Wissen und profitieren daraus so sehr, dass das kollektive Gedächtnis des Schwarms den Nachteil des Wettbewerbs um das Futter mehr als aufwiegt.

Der Anpassungsprozess ist von stochastischer Natur und hängt vom lokalen Gedächtnis jedes Individuums (Partikel) und dem globalen Gedächtnis der Population ab. Jede Partikelbewegung im Designraum wird durch Positions- und Geschwindigkeitsinformationen modelliert. Die Position  $\vec{x}_i^{k+1}$  und Geschwindigkeit  $\vec{v}_i^{k+1}$  eines Partikels  $i$  in der Iteration  $k + 1$  werden dabei wie folgt berechnet [22]. Eine graphische Illustration dieser Bewegungsgleichung ist in Abbildung 3.1 dargestellt.

$$\vec{x}_i^{k+1} = \vec{x}_i^k + \vec{v}_i^{k+1} \Delta t \quad (3.1)$$

$$\vec{v}_i^{k+1} = w \vec{v}_i^k + c_1 r_1 (\vec{p}_i^k - \vec{x}_i^k) + c_2 r_2 (\vec{g}_i^k - \vec{x}_i^k) \quad (3.2)$$

- $\vec{x}_i^k$  und  $\vec{v}_i^k$  entsprechen der Position und Geschwindigkeit des Partikels in Iteration  $k$ .
- $\Delta t$  ist der Zeitschritt.
- $r_1$  und  $r_2$  sind Zufallszahlen zwischen 0 und 1.
- $\vec{p}_i^k$  entspricht der besten Position, die Partikel  $i$  bis einschließlich Iteration  $k$  gefunden hat.
- $\vec{g}_i^k$  entspricht der besten Position, die in der gesamten Population bis einschließlich Iteration  $k$  gefunden wurde.
- Die restlichen Größen sind Prozessparameter, mit denen das Such- und Konvergenzverhalten des Algorithmus beeinflusst werden kann.  $c_1$  und  $c_2$  beeinflussen die Bewegung der Partikel in Richtung der eigenen besten Position, oder der des gesamten Schwarms.  $w$  stellt die Massenträgheit der Partikel dar, welche das Suchverhalten zwischen einem eher lokalen oder globalen Maßstab modifiziert.

Ausgehend von den oben erläuterten Positions- und Geschwindigkeitsupdates können die Schritte des Algorithmus wie folgt skizziert werden:

1. Erstelle einen initialen Satz von Partikelpositionen  $\vec{x}_i^0$  und -geschwindigkeiten  $\vec{v}_i^0$ , bei denen die Werte unter Einhaltung der expliziten Restriktionen und Höchstgeschwindigkeiten zufällig verteilt sind.
2. Werte die Zielfunktion  $f(\vec{x}_i^k)$  für alle Partikel basierend auf ihrer aktuellen Position im Designraum  $\vec{x}_i^k$  aus.
3. Falls notwendig, aktualisiere die optimale individuelle Position  $\vec{p}_i^k$  jedes Partikels sowie die kollektiv beste Position  $\vec{g}_i^k$  des Schwarms.
4. Aktualisiere Ort und Geschwindigkeit jedes Partikels anhand der in Gleichung 3.1 und 3.2 genannten Vorschrift.
5. Überprüfe Konvergenzkriterien. Wenn nicht erfüllt, setze  $k = k + 1$  und gehe nach 2.
6. Optimum ist  $\vec{x}^* = \vec{g}_i^k$ .

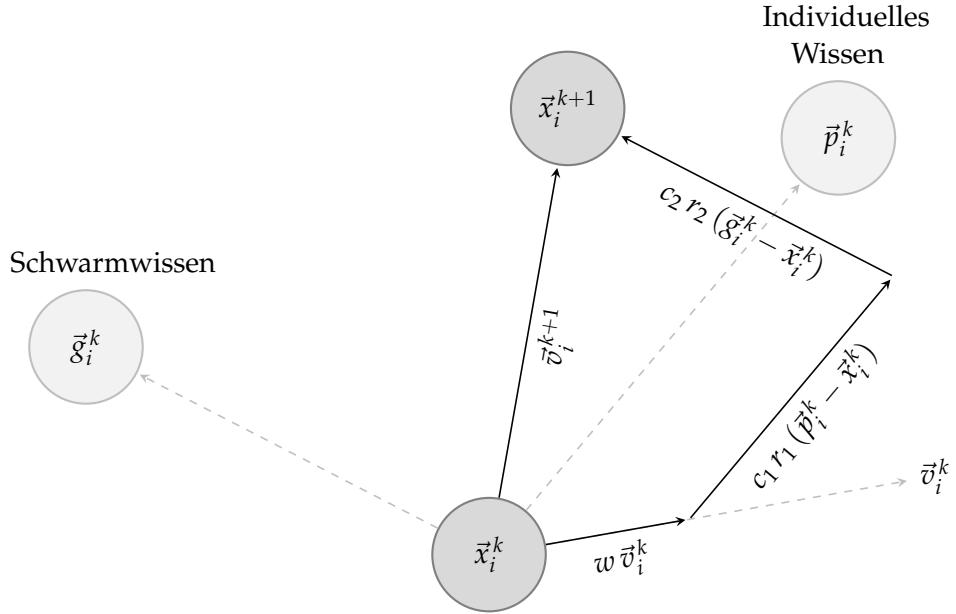


Abbildung 3.1.: Konstruktion der Geschwindigkeitskomponenten beim Partikelschwarmverfahren

**Die erweiterte Lagrange-Methode** Der zuvor dargestellte, grundlegende PSO-Algorithmus wurde als unbeschränkter Optimierungsalgorithmus entwickelt. Ein Ansatz, beschränkte Probleme mit der PSO zu lösen, stellt die Methode der erweiterten Lagrange-Multiplikatoren dar. Diese ist ein Konzept, welches ursprünglich bei der gradientenbasierten Optimierung verwendet wird.

Die Idee ist, das beschränkte Optimierungsproblem in ein unbeschränktes Problem zu überführen, welches der Optimierungsalgorithmus lösen kann. Dazu wird die um einen quadratischen Strafterm erweiterte Lagrange-Funktion als eine Art Pseudo-Zielfunktion verwendet [18, 30]:

$$\mathcal{L}(\vec{x}_i^k, \vec{\lambda}, \vec{r}_p) = f(\vec{x}_i^k) + \sum_{j=1}^m \lambda_j \theta_j(\vec{x}_i^k) + \sum_{j=1}^m r_{p,j} (\theta_j(\vec{x}_i^k))^2 \quad (3.3)$$

mit

$$\theta_j(\vec{x}_i^k) = \begin{cases} h_k(\vec{x}_i^k), & k = 1, \dots, q \\ \max[g_j(\vec{x}_i^k), \frac{-\lambda_j}{2r_{p,j}}], & j = q + 1, \dots, m \end{cases} \quad (3.4)$$

$f(\vec{x}_i^k)$  ist der Wert der Zielfunktion des Partikels  $i$ ,  $g_j(\vec{x}_i^k)$  und  $h_k(\vec{x}_i^k)$  sind die Werte der Restriktionen.  $\lambda_j$  sind die sog. Lagrange-Multiplikatoren und  $r_{p,j}$  Skalierungsfaktoren für den Strafterm.

Diese Funktion hat folgende Eigenschaften:

- Bei optimaler Wahl der Multiplikatoren  $\vec{\lambda} = \vec{\lambda}^*$  ergibt sich die Lösung des Optimierungsproblems  $\vec{x}_i^*$  als Minimum von  $\mathcal{L}(\vec{x}_i^k, \vec{\lambda}^*, \vec{r}_p)$ , unabhängig vom Wert des Skalierungsfaktors  $r_p$  [18].

- Aus der Lagrange-Funktion können Optimalitätskriterien abgeleitet werden, welche gewährleisten, dass sich die vom Algorithmus gefundene Lösung im zulässigen Bereich des Designraums befindet. Dieses sind die sogenannten *Karush-Kuhn-Tucker* oder *KKT*-Bedingungen [18]. Diese sind als Konvergenzkriterium im ALPSO-Verfahren implementiert [22].

Da der Vektor  $\vec{\lambda}^*$  abhängig vom Optimierungsproblem und zunächst nicht bekannt ist, wird dieser iterativ bestimmt, wobei in jeder Iteration die unbeschränkte Pseudo-Zielfunktion  $\mathcal{L}(\vec{x}_i^k, \vec{\lambda}, \vec{r}_p)$  bezüglich  $\vec{x}$  minimiert wird.

Die Iterationsvorschrift lautet:

1. Erstelle einen initialen Satz von Partikelpositionen  $\vec{x}_i^0$  und -geschwindigkeiten  $\vec{v}_i^0$  unter Einhaltung der angegebenen Begrenzungen.  
Initialisiere die Lagrange-Multiplikatoren mit  $\vec{\lambda}^0 = \vec{0}$  und Strafterme mit  $\vec{r}_p^0 = \vec{0}$ . Bestimme die Funktionswerte für alle Partikel anhand von Gleichung 3.3.
2. Löse das unbeschränkte Problem  $\mathcal{L}(\vec{x}_i^k, \vec{\lambda}, \vec{r}_p)$  für eine festgelegte Anzahl an Iterationen  $k_{max}$  mit dem zuvor beschriebenen PSO-Algorithmus.
3. Aktualisiere die Strafterme und Lagrange-Multiplikatoren anhand der Berechnungsvorschriften aus [22].
4. Überprüfe Konvergenzkriterien. Wenn nicht erfüllt, setze  $k = k + 1$  und gehe nach 2.
5. Optimum ist  $\vec{x}^* = \vec{g}_i^k$ .

Das Update der Straffaktoren und Lagrange-Multiplikatoren erfolgt dabei dergestalt, dass Partikelbewegungen in Richtung unzulässiger Funktionswerte bestraft, also mit ansteigenden Werten belegt werden. Die Berechnungsvorschrift dazu ist in [22] dokumentiert.

Letztlich wird mit dieser Methode das beschränkte Optimierungsproblem in eine Serie unbeschränkter Optimierungsprobleme aufgeteilt, welche vom PSO-Algorithmus gelöst werden können.

**Parallelisierung** Ein Nachteil globaler Optimierungsalgorithmen ist der hohe numerische Berechnungsaufwand im Vergleich zu den gradientenbasierten Verfahren. Da bei der PSO die Funktionswerte der einzelnen Partikel voneinander unabhängig sind, kann die Zielfunktion für mehrere Partikel parallel ausgewertet werden.

Durch diese Eigenschaft ist die PSO gut für eine parallelisierte Ausführung des Rechenprozesses geeignet. Um zu vermeiden, dass innerhalb der inneren Iteration einzelne Kerne des Prozessors leer laufen, sollte die Populationsgröße ohne Rest durch die Anzahl der verwendeten Kerne teilbar sein [22].

### 3.4. Validierung anhand analytischer Standardprobleme

Die Eignung des ALPSO-Verfahrens zur Optimierung von Faserverunbauteilen soll in diesem Abschnitt anhand zweier, analytisch lösbarer, Testprobleme evaluiert werden.

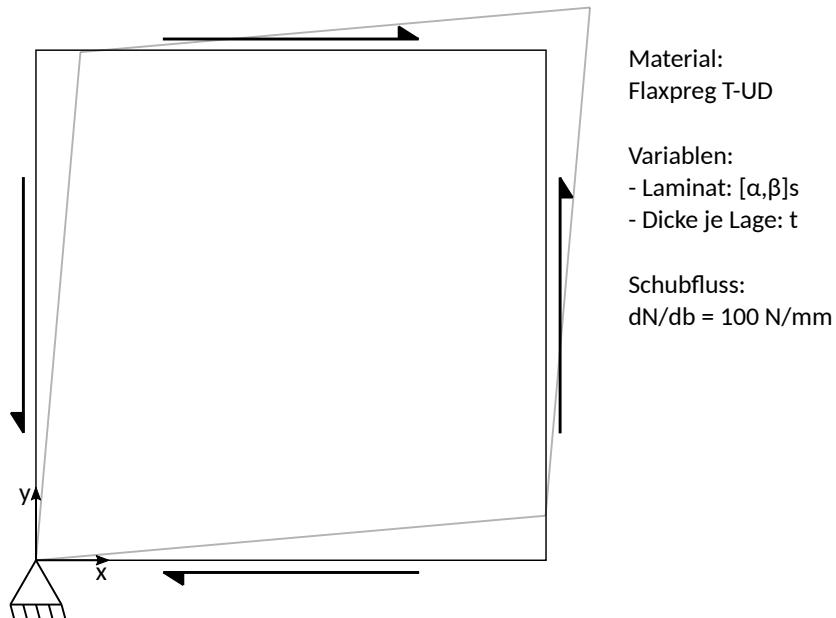


Abbildung 3.2.: Modell einer Faserverbundplatte unter Schubbelastung

Dazu werden zunächst eine Faserverbundplatte unter reinem Schub und anschließend ein Biegebalken unter einer Streckenlast optimiert. Als Analysemodell wird jeweils ein FE-Modell der entsprechenden Struktur verwendet.

Die Modellierung erfolgt mit *ANSYS Classic* und der Python-Bibliothek *PyMAPDL* [23], welche in Abschnitt Abschnitt 5.1 noch ausführlich beschrieben wird.

Der verwendete Optimierungsalgorithmus ist das zuvor beschriebene ALPSO-Verfahren aus der *PyOpt*-Bibliothek [31].

### 3.4.1. Optimierung einer Faserverbundplatte

Als zu optimierendes Bauteil wird eine Faserverbundplatte mit einem aus vier Schichten bestehenden, symmetrischen Lagenaufbau definiert. Als Werkstoff wird das Flachsfaserverprepreg *FLAXPREG T-UD* der Firma *EcoTechnilin* verwendet [16, 13].

Diese wird mit einem Schubfluss von  $n_{xy} = 100 \text{ N/mm}$  beaufschlagt.

Die Faserwinkel des Laminats  $[\alpha, \beta]_s$ , sowie die Dicke  $t$  der Einzellagen sollen so gewählt werden, dass die Masse der Platte unter Einhaltung des Zwischenfaserbruch-Kriteriums nach PUCK minimiert wird.

Eine schematische Skizze des Modells ist in Abbildung 3.2 dargestellt.

**Formulierung des Optimierungsproblems** Da die Masse des Bauteils proportional zur Schichtdicke der Einzellagen  $t$  ist, kann diese direkt als Zielfunktion verwendet werden:

$$\min f(t) = t \quad (3.5)$$

Als Restriktion wird festgelegt, dass im gesamten Bauteil der Versagensindex  $I_{mat,puck}$  nicht größer als 1 sein soll:

$$g(\alpha, \beta, t) = I_{mat,puck} - 1 \leq 0 \quad (3.6)$$

Als explizite Restriktionen werden gesetzt:

$$\begin{aligned} 0^\circ &\leq \alpha \leq 180^\circ \\ 0^\circ &\leq \beta \leq 180^\circ \\ 0 \text{ mm} &\leq t \leq 1 \text{ mm} \end{aligned}$$

Damit liegt ein Optimierungsproblem mit drei Designvariablen und einer Ungleichheitsrestriktion vor.

**Analytische Betrachtung** Im Bauteil liegt ein reiner Schubspannungszustand vor. Die dafür optimalen Faserwinkel werden klar, wenn man den äquivalenten Hauptspannungszustand in Abbildung 3.3 betrachtet. Man orientiert die Fasern in Richtung der Hauptspannungen. Im  $x, y$ -Koordinatensystem entspricht das einem  $\pm 45^\circ$ -Laminat [39].

Die Dicke der Einzellagen kann dann so dimensioniert werden, dass die Anstrengung auf Zwischenfaserbruch  $I_{mat,puck} = 1$  ist.

Eine Berechnung auf Basis der klassischen Laminattheorie mit Hilfe der Software *eLamX*<sup>2</sup> [19] ergibt eine Lagendicke  $t \approx 0,505 \text{ mm}$ .

**Aufbau des Analysemodells** Da die Abmessungen der Faserverbundplatte wesentlich größer als ihre Bauteildicke sind ( $l/t \gg 1$ ), kann das Bauteil als zweidimensionale Platte gemäß der Kirchhoffschen Plattentheorie abstrahiert werden. Die Modellierung erfolgte daher über Schalenelemente.

Zur Vernetzung wurde das quadratische Element *SHELL281* mit 8-Knoten und vollständiger Integration gewählt. Die Abmessungen Kanten betragen je 1000 mm und es wurden pro Kante 20 Elemente gleichmäßig angeordnet.

Die Aufprägung der Lasten erfolgte als Punktkräfte verteilt auf alle Knoten an den Bauteilkanten. Als Randbedingung wurde die gesamte Verformung in  $z$ -Richtung gesperrt sowie der Knoten im Koordinatenursprung fest eingespannt.

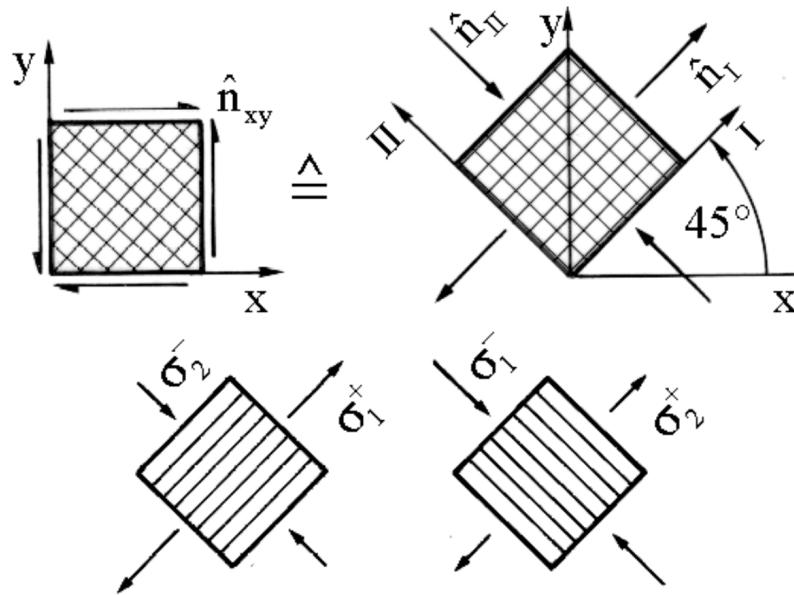


Abbildung 3.3.: Hauptspannungszustand bei Schubbelastung nach SCHÜRMANN [39].

		$\alpha^*$	$\beta^*$	$t^*$
Analytisch		45°	135°	0,505 mm
	Min	44,64°	134,85°	0,474 mm
Optimierung	$\varnothing$	44,94°	135,01°	0,475 mm
	Max	45,14°	135,15°	0,477 mm

Tabelle 3.2.: Optimierungsergebnis der Faserverbundplatte und Vergleich mit analytischer Rechnung

**Lösung des Optimierungsproblems** Als Prozessparameter für den ALPSO-Algorithmus wurden die Voreinstellungen der PyOpt-Toolbox verwendet (Population 40 Partikel,  $c_1 = 2$ ,  $c_2 = 1$ ). Da der Algorithmus nicht deterministisch ist, wurden insgesamt zehn Läufe der Optimierung durchgeführt, um Aussagen über die Stabilität des Verfahrens treffen zu können.

Das Verfahren konvergierte jeweils nach 3-4 äußereren Iterationen. Die Rechendauer betrug bei parallelisierter Rechnung mit vier Threads auf einem Notebook mit 2,7 GHz Dual-Core Prozessor zwischen 274 und 384 Sekunden.

Als Lösung wurde immer ein Punkt in der Nähe des analytisch bestimmten Optimums gefunden, eine Übersicht über die gefundenen Minima ist in Tabelle 3.2 dargestellt.

Zu sehen ist, dass der Algorithmus immer das selbe Optimum ermittelt hat, Ausreißer traten nicht auf. Die Größenordnung der Abweichung bei den gefundenen Optima voneinander resultiert aus dem Toleranzwert für die Lagrange-Funktion  $atol$ . Dieser ist ein Prozessparameter des Optimierungsverfahrens kann angepasst werden, um die Toleranz zu verfeinern oder die Konvergenzgeschwindigkeit zu erhöhen.

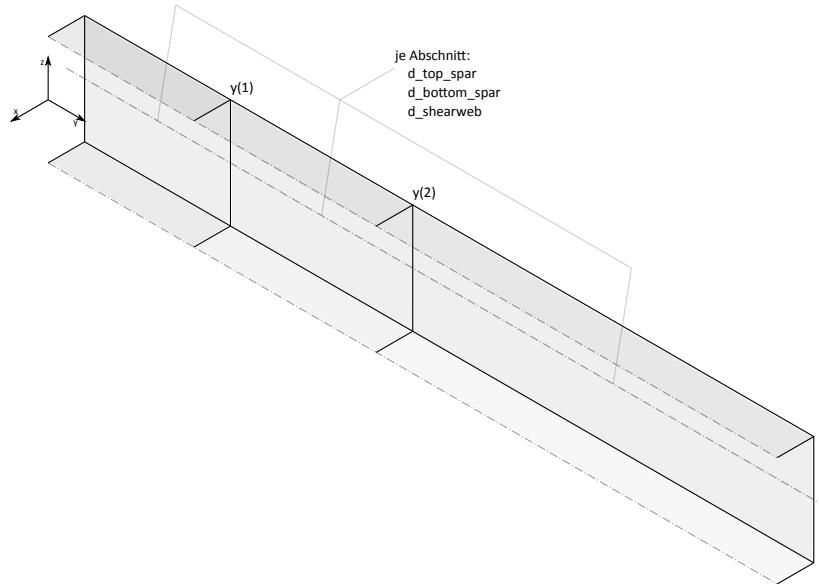


Abbildung 3.4.: Schematische Skizze des Faserverbundbalkens

### 3.4.2. Optimierung eines Faserverbundbalkens

Am vorangegangenen Beispiel der Faserverbundplatte konnte demonstriert werden, dass der ALPSO-Algorithmus für ein einfaches Problem in der Lage ist, die Ergebnisse der analytischen Auslegung zu reproduzieren.

Nun soll anhand eines weiteren generischen Modells mit einer höheren Komplexität geprüft werden, ob der Optimierer auch für komplexere Modelle mit vertretbarem Zeitaufwand sinnvolle Ergebnisse liefert. Dazu wird ein Modell eines Faserverbund-Rechteckbalkens definiert, welches in Abbildung 3.4 dargestellt ist.

Der gesamte Balken hat eine Länge  $l = 1000$  mm und ist in der  $xz$ -Ebene des Koordinatensystems fest eingespannt. Der Rechteckquerschnitt misst  $b = 20$  mm in der Breite und  $h = 30$  mm in der Höhe. Er wird durch eine konstante Streckenlast  $q_0 = 0,05 \text{ N/mm}$  belastet, so dass das Biegemoment  $M_x$  an der Einspannung 25 Nm beträgt.

Aufgrund der Symmetrie des Problems zur  $yz$ -Ebene wird nur eine Hälfte des Balkens modelliert und mit einer Symmetriebedingung versehen.

Die Wandstärken des Balkens im Steg und den Gurten sollen so gewählt werden, dass die Masse minimiert wird. Dabei sollen die Versagenskriterien nach Puck eingehalten werden. Zusätzlich wird dem Optimierer gestattet, die Lagen zwei mal abzustufen und den Ort der Abstufung in  $y$ -Richtung frei zu wählen.

Als Werkstoff wird wie zuvor das Flachsfaser-Prepreg *FLAXPREG T-UD* der Firma *EcoTech-nlin* verwendet. Die Fasern in den Holmgurten werden unidirektional in Längsrichtung des Balkens orientiert. Im Steg wird ein  $\pm 45^\circ$ -Schublaminat gesetzt.

**Formulierung des Optimierungsproblems** Als Zielfunktion wird die Bauteilmasse verwendet, diese kann beim Post-Processing des FE-Modells direkt aus *ANSYS* ausgelesen werden.

$$\min f(\vec{x}) = m(\vec{x}) \quad (3.7)$$

Als Restriktion wird festgelegt, dass in allen drei Abschnitten der Gurte und des Steges die Versagenskriterien nach Puck einzuhalten sind:

$$g_j(\vec{x}) = I_{\text{fib, top}, j} - 1 \leq 0; \quad j = 1, \dots, 3 \quad (3.8)$$

$$g_j(\vec{x}) = I_{\text{fib, bot}, j} - 1 \leq 0; \quad j = 4, \dots, 6 \quad (3.9)$$

$$g_j(\vec{x}) = I_{\text{fib, web}, j} - 1 \leq 0; \quad j = 7, \dots, 9 \quad (3.10)$$

$$g_j(\vec{x}) = I_{\text{mat, top}, j} - 1 \leq 0; \quad j = 10, \dots, 12 \quad (3.11)$$

$$g_j(\vec{x}) = I_{\text{mat, bot}, j} - 1 \leq 0; \quad j = 13, \dots, 15 \quad (3.12)$$

$$g_j(\vec{x}) = I_{\text{mat, web}, j} - 1 \leq 0; \quad j = 16, \dots, 18 \quad (3.13)$$

Die expliziten Restriktionen werden so festgelegt, dass die Orte der Trennstellen sich innerhalb des Bauteils befinden müssen, und sich alle Wandstärken im Bereich von 0,185 – 1,5 mm bewegen sollen.

Das vorliegende Optimierungsproblem verfügt insgesamt über 11 Designvariablen (2 für die Abstufungen und 9 für die Wandstärken) und 18 Restriktionen.

**Aufbau des Analysemodells** Da die Wandstärken des Bauteils im Vergleich zu den äußereren Abmessungen vernachlässigbar gering sind, kann die Theorie dünner Platten nach Kirchhoff verwendet werden. Das verwendete Element ist daher *SHELL281*.

Das Netz besteht aus je 400 rechteckigen Elementen pro Gurt und 1200 Elementen im Steg.

**Lösung des Optimierungsproblems** Wie bei der Optimierung der Faserverbundplatte zuvor wurden als Optimierungsparameter die Standardwerte des Algorithmus verwendet und das Problem insgesamt zehn mal gelöst.

Das Verfahren konvergierte in allen der zehn Fälle, die Strukturmasse im Optimum betrug 17,35 – 17,68 g und es wurden zwischen zwölf und 30 Iterationen benötigt.

Die Eigenschaften der gefundenen Minima werden am Beispiel eines der Läufe erläutert: In Abbildung 3.5 ist eine Pfadauswertung des Faserbruchkriteriums in den Holmgurten dargestellt. Es ist erkennbar, dass die Minimierung der Zielfunktion (Bauteilmasse) vom Optimierer so umgesetzt wird, dass er die Materialausnutzung maximiert.

Kritisch für das Bauteil ist die Belastung auf Faserbruch im druckbelasteten Holmgurt, dort strebt die Anstrengung gegen den höchstzulässigen Wert 1. Dieses Verhalten ist erwartbar, da bei Faserverbundwerkstoffen die Zugfestigkeit gegenüber der Druckfestigkeit generell überwiegt, im Falle des verwendeten Materials *FLAXPREG-T-UD* um etwa Faktor 3.

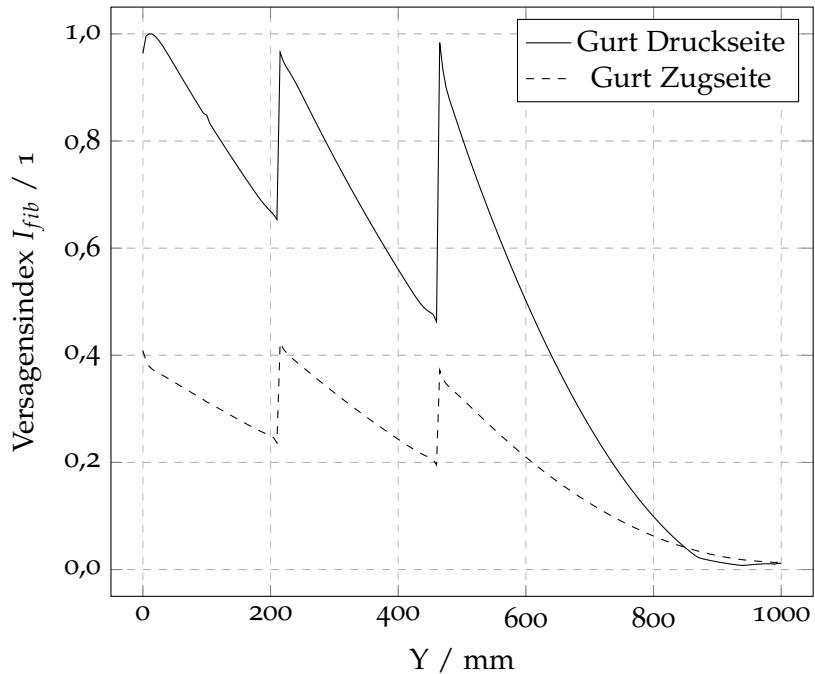


Abbildung 3.5.: Pfadauswertung des Faserbruchkriteriums in den Holmgurten

Bei der Betrachtung der Designvariablen im gefundenen Minimum in ?? wird deutlich, wie der Optimierer auf diese Anisotropie reagiert: Er verteilt die Wandstärken in den Gurten leicht asymmetrisch. Die Materialdicke des druckbelasteten Gurtes sind um etwa 10 % größer als im Zuggurt.

Anhand dieses einfachen, generischen Bauteils konnte also demonstriert werden, dass der Optimierungsalgorithmus in der Lage ist, die kritischen Belastungen des Bauteils zu erkennen, und die Designvariablen in Hinsicht darauf sinnvoll anzupassen.

		Wert
Trennstelle	$n_1$	215 mm
	$n_2$	465 mm
Zuggurt	$t_{top,1}$	0,72 mm
	$t_{top,2}$	0,39 mm
	$t_{top,3}$	0,19 mm
Druckgurt	$t_{bot,1}$	0,77 mm
	$t_{bot,2}$	0,50 mm
	$t_{bot,3}$	0,19 mm
Steg	$t_{web,1}$	0,19 mm
	$t_{web,2}$	0,22 mm
	$t_{web,3}$	0,19 mm

Tabelle 3.3.: Werte der Designvariablen im gefundenen Minimum

## 4. Lastanalyse von UAV-Propellern

Auf ein Propellerblatt wirken im Betrieb aerodynamische Lasten, resultierend aus Schub und Widerstand, sowie zentrifugale Lasten, welche durch die Massenträgheit des Blattes in Folge der Rotation entstehen.

Um das Ziel dieser Arbeit zu erreichen, ist es notwendig eine Lastrechnungsroutine zu implementieren, in die beliebige Propellergeometrien und Lastfälle eingegeben werden können. Dieser Abschnitt gibt einen Überblick über die Umsetzung und Funktionsweise des Lastrechnungstools.

Grundsätzlich existieren für die aerodynamische Analyse von Propellern, und damit auch die Lastrechnung, verschiedene Klassen von Berechnungsmethoden, welche sich hinsichtlich ihrer physikalischen Grundlage und des Berechnungsaufwandes unterscheiden:

- Ein sehr einfacher Ansatz ist die Verwendung sogenannter Handbuchmethoden. Diese Verfahren beruhen auf empirischen Beobachtungen von Propellern im Betrieb und haben einen eher geringen physikalischen Hintergrund. Die Genauigkeit dieser Verfahren ist daher eher beschränkt, allerdings benötigen sie nur wenige Eingabewerte und liefern sehr schnell Ergebnisse, weshalb sie oft im Vorentwurf angewendet werden. Handbuchverfahren für den Propellerentwurf sind z.B. in [4] zu finden.
- Genauere Ergebnisse lassen sich mit sogenannten (semi-)analytischen Verfahren erzielen. Diese nutzen mathematisch-physikalische Theorien wie die Traglinientheorie nach PRANDTL bzw. deren Übertragung auf Propeller durch BETZ [34]. Die Anwendung dieser Methoden wird durch verschiedene verfügbare Software-Pakete erleichtert, sodass die Algorithmen nicht selbst implementiert werden müssen. Aufgrund der analytischen Gestalt dieser Berechnungsmethoden ist der rechnerische Aufwand eher gering.
- Bei der numerischen Strömungsmechanik (englisch *Computational Fluid Dynamics, CFD*) werden strömungsmechanische Probleme näherungsweise mit numerischen Methoden gelöst. Dadurch können, analog zur FE-Methode in der Strukturmechanik, auch nichtlineare und geometrisch komplexe Probleme berechnet werden.  
Die Verwendung dieser Methoden erfordert einen deutlich größeren Modellierungs- und Berechnungsaufwand als bei den analytischen Verfahren.

Für die Verwendung im Rahmen dieser Programmkette bieten sich die analytischen Verfahren an, da diese für wenig komplexe Geometrien, wie die von Propellern, vermutlich ausreichend genaue Ergebnisse liefern. Der hohe Aufwand numerischer Methoden würde den zusätzlichen Nutzen also überwiegen [6, 21].

### 4.1. Verwendete Softwaretools

Im Rahmen einer Arbeit an der Technischen Universität Delft wurden von KLEIN drei verschiedene Softwaretools zur Propellerberechnung anhand von Versuchsdaten aus Wind-

kanalmessungen validiert [21]. Die Ergebnisse zeigten, dass für einen Großteil der Validierungsdaten das Tool XROTOR die besten Ergebnisse lieferte, während andere Programme in Teilbereichen nur wenig besser abschnitten.

**XROTOR** XROTOR ist ein interaktives Programm für den Entwurf von Propellern sowie Windkraft-Rotoren und wurde von MARK DRELA und HAL YOUNGREN entwickelt, von denen auch das weiter bekanntes Profilaerodynamik-Tool XFOIL stammt [11, 9].

XROTOR wurde in Fortran programmiert, ist Open-Source, und verfügt über eine Programmschnittstelle, mit der Batch-Jobs skriptbasiert ausgeführt werden können.

Um eine Propelleranalyse in XROTOR durchführen zu können, wird eine umfassende geometrische Beschreibung des Propellers und der Betriebsbedingungen benötigt. Diese umfasst:

- Geometrische Parameter wie Durchmesser von Nabe und Propellerkreis, Anzahl der Blätter.
- Die Geometrie des Propellers. Diese muss in zweidimensionale Abschnitte diskretisiert werden, von denen die Profiltiefe, die Steigung und die aerodynamischen Eigenschaften (Profilpolaren) bekannt sind.
- Physikalische Konstanten: Luftdichte, Kinematische Viskosität und Schallgeschwindigkeit.
- Betriebsbedingungen wie Anströmgeschwindigkeit und Drehzahl, Fortschrittsgrad oder Leistung.

Die Profilpolaren werden XROTOR als eine Sammlung charakteristischer Kenngrößen je Abschnitt übergeben. Diese sind abhängig von der geometrischen Form der Profile und müssen daher zusätzlich bestimmt werden, dazu gehören:

- Auftriebsanstieg
- Maximaler und minimaler Auftriebsbeiwert
- Nullauftriebssinkel
- Minimaler Widerstandsbeiwert und Auftriebsbeiwert bei minimalem Widerstand, Widerstandsanstieg
- Momentenbeiwerte
- Referenz-Reynolds- und Machzahl

Aus diesem Satz von Parametern können in XROTOR berechnet werden:

- Allgemeine Leistungsmerkmale wie Schub, Drehmoment, aufgenommene Leistung und Wirkungsgrad.
- Verteilte aerodynamische Größen: Auftriebs- und Widerstandsverläufe, induzierte Geschwindigkeiten etc.
- Schnittkraft und -Momentenverläufe.

**XFOIL** Wie zuvor beschrieben, werden als Eingabedaten für die XROTOR-Analyse die Profileigenschaften aller verwendeten Querschnitte benötigt, welche aus den Auftriebs-, Widerstands- und Momentenpolaren berechnet werden können. Dazu wird das Tool XFOIL [9] in die Programmkette integriert.

Die dort ermittelten Polaren sollen in Python eingelesen und in einer eigens entwickelten Routine durch analytische Funktionen angenähert und in das von XROTOR benötigte Format umgerechnet werden.

## 4.2. Implementierung von XFOIL und XROTOR in Python

**Technische Umsetzung** Die Programmierschnittstellen der beiden Tools sind weitgehend identisch, da sie von den gleichen Entwicklern stammen. Es wird jeweils ein Inputfile benötigt, welches die auszuführenden Befehle enthält. Dieses wird per Kommandozeile eingelesen, bearbeitet und die Ergebnisse werden in ein Outputfile geschrieben [10, 12]. Da im Netz kein stabiles und ausreichend dokumentiertes Python-Interface verfügbar war, wurde dieses selbst implementiert. Eine ausführliche Dokumentation ist auf der Daten-CD beigelegt.

Das Interface wurde objektorientiert als Python-Modul namens *util\_loads* implementiert, die grundlegende Funktionsweise ist in Code Listing 4.1 am Beispiel einer einfachen XFOIL-Analyse dargestellt.

Das Inputfile wird mit Hilfe eines Context-Managers innerhalb einer *with*-Umgebung erstellt<sup>1</sup>. Beim Eintreten in die Umgebung wird selbständig ein temporäres Input-File geöffnet. Innerhalb können z.B. mit der Methode *run* Befehle in das Inputfile geschrieben werden. Weiterhin ist es möglich alle Python-Objekte wie z.B. Schleifen oder Variablen zu verwenden. Beim Verlassen des Context-Managers wird der XFOIL/XROTOR-Prozess automatisch ausgeführt und das Inputfile anschließend gelöscht.

Nach dem Ausführen der Analyse können die Outputfiles mit statischen Methoden<sup>2</sup> eingelesen werden und stehen als *DataFrame* in Python zur Verfügung.

Code Listing 4.1: Funktionsweise des XFOIL-Interface

```
from util_loads import Xfoil, XRotor

polar_file = '_xfoil_polar.txt'

# Generierung des Input-Files mit Context-Manager
with Xfoil() as x:
    # Beispiel: Polare eines NACA-Profil ohne Reibung
    x.run('naca_0012')
```

<sup>1</sup>Ein Context-Manager ist ein Objekt, dass eine Laufzeitumgebung bereitstellt, und den Eintritt und das Verlassen der Umgebung verwaltet. Dadurch können z.B. Ressourcen gespeichert und wiederhergestellt oder Textdateien automatisiert geöffnet und geschlossen werden [35, S. 3.3.9].

<sup>2</sup>Statische Methoden sind Prozeduren, die im Namensraum einer Klasse enthalten sind, aber ohne die Instanziierung eines Objektes ausgeführt werden können [35].

```

x.run('oper')
x.run('pacc')
x.run(polar_file)
x.run('')
x.run('aseq_o_1o_1')
x.run('')
x.run('quit')

# Einlesen der Outputs mit statischen Methoden
polar = Xfoil.read_polar(polar_file)

```

**Berechnung der Profileigenschaften** XROTOR verwendet ein mathematisches Modell um die Polaren der Profile des Propellers zu beschreiben, sodass Auftrieb, Widerstand und Momente für die auftretenden Anstellwinkel abgeschätzt werden können. Die Parameter des Modells, die sogenannten *Profilcharakteristiken*, werden mit Hilfe einer dafür in Python implementierten Funktion aus den XFOIL-Outputs berechnet.

Zunächst wird die von XFOIL berechnete  $c_a/\alpha$ -Kurve durch eine mathematisch beschreibbare Funktion approximiert<sup>3</sup>, ein Beispiel dazu ist in Abbildung 4.1 zu sehen. Die Fitting-Funktion ist dreigeteilt: Der Stall-Bereich bei negativen Anstellwinkeln und der lineare Auftriebsanstieg werden durch zwei punktstetig verbundene Geraden angenähert. Daran schließt sich eine tangentenstetige Parabel für den positiven Stall-Bereich an. Aus der Approximation können durch Bestimmung der Extremwerte und Nullstellen der einzelnen Teilfunktionen direkt folgende XROTOR-Parameter entnommen werden:

- Auftriebsanstieg  $dc_a/d\alpha$
- Maximaler und minimaler Auftriebsbeiwert  $c_{a,max}$  und  $c_{a,min}$
- $c_a$ -Inkrement bis zum Strömungsabriss
- Nullauftriebswinkel  $\alpha_0$

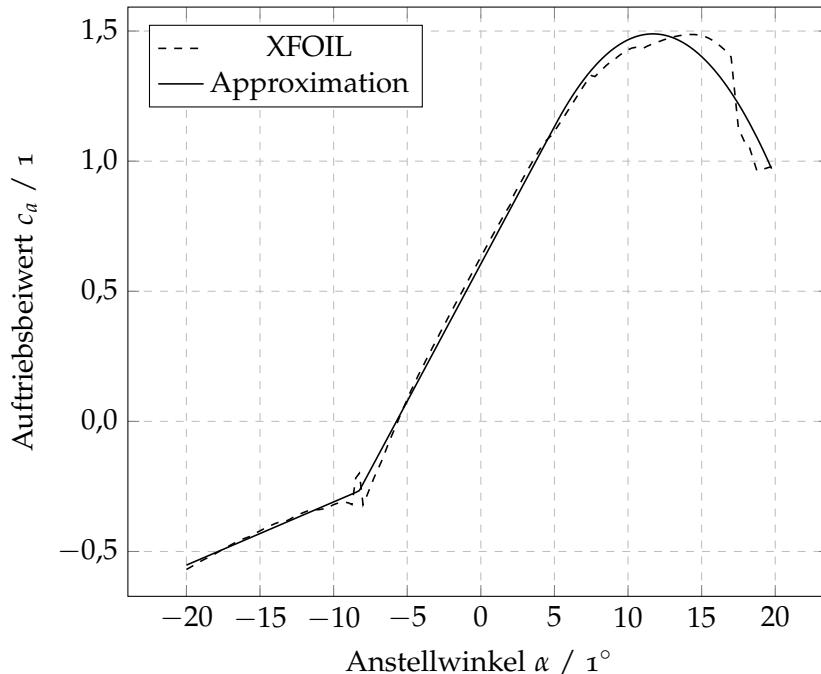
Die Widerstandspolare wird in XROTOR über drei Parameter beschrieben. Zwei davon, der minimale Widerstandsbeiwert  $c_{w,min}$  und sein entsprechender Auftriebsbeiwert  $c_a(c_{w,min})$  stehen direkt in den XFOIL-Outputfiles. Zusätzlich wird an diesem Betriebspunkt die zweite Ableitung des Widerstandsbeiwerts nach dem Auftriebsbeiwert  $dc_d/dc_a^2$  benötigt.

Umgesetzt wird dieses durch eine numerische Differentiation mit dem zentralen Differenzenquotienten. Da bei diesem Verfahren zunächst starkes numerisches Rauschen auftrat, wird die Widerstandspolare vor dem Ableiten mit Hilfe eines Filters aus der Signalverarbeitung<sup>4</sup> geglättet.

Der Drehmomentenbeiwert  $c_m$  kann direkt aus dem XFOIL-Output verwendet werden.

<sup>3</sup>Technisch erfolgt dies mit einer Optimierung der Abweichung der nichtlinearen kleinsten Quadrate durch die *curve\_fit*-Funktion aus dem SciPy-Paket [43].

<sup>4</sup>Verwendet wurde ein *Savitzky-Golay*-Filter aus der SciPy-Optimize Toolbox [43].

Abbildung 4.1.: Approximation der  $c_a$ - $\alpha$ -Kurven in Python

### 4.3. Erstellung eines Lastrechnungsprogramms

In diesem Abschnitt soll das Lastenrechnungsprogramm beschrieben werden, welches im Rahmen dieser Arbeit erstellt wurde. Es ist in Python implementiert und verwendet als Löser die Tools XFOIL und XROTOR, welche mit der zuvor beschriebenen Schnittstelle angesteuert werden.

Eine ausführliche technische Dokumentation ist auf der Daten-CD enthalten. In diesem Abschnitt werden die grundlegende Funktionsweise und die wichtigsten Methoden erläutert.

#### 4.3.1. Architektur

Als Programmierstil wurde die *Objektorientierte Programmierung* gewählt. Die Idee dieser Art der Programmierung besteht darin, die Softwarearchitektur an die Struktur des in der Realität zu behandelnden Problems anzulehnen. Dazu werden in der Anwendung vorkommende reale Komponenten durch sogenannte *Objekte* und *Klassen* abgebildet, und die Beziehung zwischen diesen modelliert.

Die Architektur des Lastrechnungsprogramms ist in Abbildung 4.2 dargestellt, es ist in die drei Klassen *Propeller*, *Airfoil* und *Loadcase* unterteilt. Die übergeordnete Klasse ist die Propeller-Klasse. In ihr können die geometrischen Daten des zu analysierenden Propellers gespeichert werden.

Da in einem Propeller üblicherweise mehrere verschiedene aerodynamische Profile verbaut

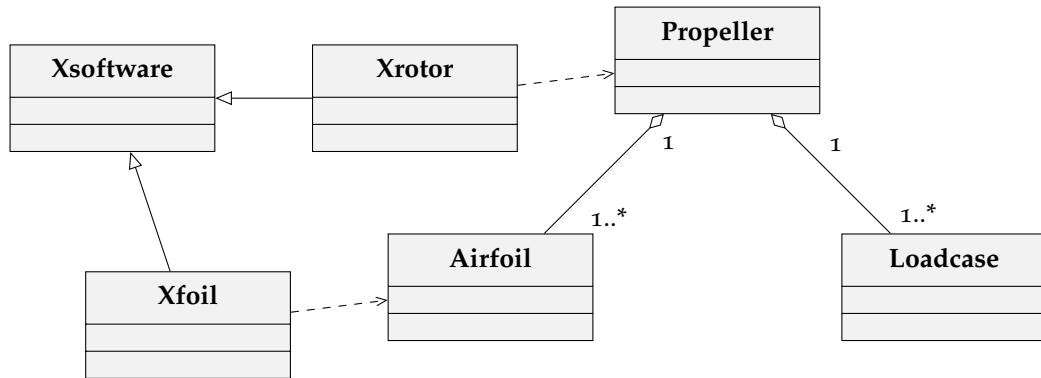


Abbildung 4.2.: UML-Klassendiagramm des Lastrechnungsprogramms

sind, ist als Aggregation (*besteht-aus*-Beziehung) die Klasse Airfoil angefügt. Ein Propeller-Objekt verfügt also immer über mindestens ein Airfoil-Objekt.

Eine weitere Aggregation besteht zu der Loadcase-Klasse. Einem Propeller können also mehrere Lastfälle hinzugefügt werden, was wichtig ist, um später einen einhüllenden Lastverlauf zu berechnen.

Als Aerodynamik-Löser verwendet die Airfoil-Klasse das XFOIL-Interface und die Propeller-Klasse das XROTOR-Interface. Beide erben gemeinsame Methoden von der Klasse Xsoftware, da die Schnittstellen einander stark ähneln.

#### 4.3.2. Loadcase-Klasse

Die Loadcase-Klasse dient zur Modellierung von Lastfällen. In Loadcase-Objekten werden entsprechend der Typ und die Betriebsdaten eines zu berechnenden Lastfalls hinterlegt. Eine Übersicht über die Loadcase-Klasse ist in Abbildung 4.3 zu sehen.

Bei der Instanziierung<sup>5</sup>, der Erstellung neuer Objekte, wird zu erstellenden Objekt jeweils ein Name sowie die Anströmgeschwindigkeit der Rotorebene (flight\_speed) zugewiesen.

Die Parameter des Lastfalles werden mit der Methode `set_data()` zugewiesen. Dazu stehen die verschiedenen Analysetypen zur Verfügung, welche von XROTOR angeboten werden [12]:

- **rpm:** Vorgabe der Drehzahl
- **adva:** Vorgabe des Fortschrittsgrades
- **thru:** Vorgabe des Schubs
- **torq:** Vorgabe des Drehmoments
- **powe:** Vorgabe der Leistung

Wird z.B. die Drehzahl vorgegeben, bestimmt XROTOR dazu die korrespondierenden Größen Fortschrittsgrad, Schub, Drehmoment und Leistung.

<sup>5</sup>Die Instanziierung wird in der Objektorientierten Programmierung über die Konstruktor-Methode geregelt. In Python heißt diese `__init__()`

Loadcase
<ul style="list-style-type: none"> <li>- <code>_data</code> = List</li> <li>- <code>_flight_speed</code> = Float</li> <li>- <code>_name</code> = Str</li> </ul>
<ul style="list-style-type: none"> <li>- <code>__init__(self, name, flight_speed)</code></li> </ul> <p>Getter:</p> <ul style="list-style-type: none"> <li>+ <code>data</code> : List</li> <li>+ <code>flight_speed</code> : Float</li> <li>+ <code>name</code> : Str</li> </ul> <p>Setter:</p> <ul style="list-style-type: none"> <li>+ <code>flight_speed(self, flight_speed)</code></li> <li>+ <code>name(self, name)</code></li> <li>+ <code>set_data(self, type, value, fix, value2)</code></li> </ul>

Abbildung 4.3.: Übersicht über die Loadcase-Klasse

#### 4.3.3. Airfoil-Klasse

Die Airfoil-Klasse repräsentiert die aerodynamischen Profile des Propellers. Um aerodynamische Berechnungen durchzuführen, ist sie an das XFOIL-Interface angebunden. Eine Übersicht ist in Abbildung 4.4 dargestellt.

Bei der Instanziierung eines Airfoil-Objekts wird diesem ein ASCII-File mit den Profilkordinaten, die Reynoldszahl und zwei XFOIL-eigene Parameter `ncrit`<sup>6</sup> und `iter`<sup>7</sup> übergeben. Im folgenden werden die wichtigsten Methoden der Klasse vorgestellt:

**set\_polar()** Die Profilpolaren werden mit der Methode `set_polar(alpha_start, alpha_stop, alpha_inc)` berechnet. Die Berechnung erfolgt dabei für alle Anstellwinkel von `alpha_start` bis `alpha_stop` in Schritten von `alpha_inc`. Da XFOIL sensitiv auf hohe Anstellwinkel reagiert, erfolgt die Berechnung zweigeteilt von der Mitte aus in Richtung der Grenzen. Nach der XFOIL-Analyse wird das Outputfile eingelesen und als DataFrame im Attribut `polar` hinterlegt. Zusätzlich wird die in Absatz 4.2 beschriebene Routine zur Bestimmung der Profileigenschaften durchgeführt. Diese werden als Dictionary in `xrotor_characteristics` gespeichert.

**cp\_vs\_x()** Die Methode `cp_vs_x(mode, value)` gibt die Druckverteilung auf dem Profil für einen gewünschten Betriebspunkt zurück. Dazu stehen als `mode` verschiedene Modi zur Verfügung:

- **alfa:** Vorgabe des Anstellwinkels  $\alpha$ .

<sup>6</sup>Ncrit ist ein Verstärkungsfaktor für Instabilitäten in der Anströmung und beeinflusst damit den laminar-turbulenten Übergang [10]. Alle Rechnungen in dieser Arbeit wurden mit `ncrit=9` durchgeführt.

<sup>7</sup>Das Iterationslimit für aerodynamische Berechnungen. Aus der Erfahrung des Autors empfiehlt es sich, mindestens 200 zu wählen.

- cl: Vorgabe des Auftriebsbeiwerts  $c_a$ .
- cli: Vorgabe des Auftriebsbeiwerts  $c_a$ , reibungslose Rechnung.

**interpolate()** Bei der Geometrieeingabe des Propellers werden die Profile nur an diskreten Stationen vorgegeben. Wenn mehrere, verschiedene Profile verbaut sind, gehen diese flächig in einander über. Zwischen den Stützstellen der Eingabe herrschen dann sogenannte Misch- oder Straakprofile vor. Die statische Methode `interpolate(airfoil1, airfoil2, fraction)` dient daher zur Interpolation zweier Profile.

Es müssen zwei Airfoil-Objekte und ein Mischungsverhältnis übergeben werden, die Ausgabe sind dann die Profilkoordinaten des interpolierten Profils.

Airfoil
- <code>__coordinates</code> = DataFrame
- <code>__parameters</code> = Dict
- <code>__polar</code> = DataFrame
- <code>__xrotor_characteristics</code> = Dict
- <code>__init__(self, airfoil_filename, re, ncrit, iter)</code>
- <code>__fit.cl_alpha_(self, x, xo, x1, a3, b1, b3, c2)</code>
+ <code>cp_vs_x(self, mode, value) : DataFrame</code>
+ <code>interpolate(Airfoil1, Airfoil2, fraction) : DataFrame</code>
Getter:
+ <code>coordinates : DataFrame</code>
+ <code>parameters : Dict</code>
+ <code>polar : DataFrame</code>
+ <code>xrotor_characteristics : Dict</code>
Setter:
+ <code>coordinates(self, coordinates)</code>
+ <code>set_polar(self, alpha_start, alpha_stop, alpha_inc)</code>
+ <code>xrotor_characteristics(self, characteristics)</code>

Abbildung 4.4.: Übersicht über die Airfoil-Klasse

#### 4.3.4. Propeller-Klasse

Der Propeller als Ganzes wird durch die Propeller-Klasse dargestellt, welche zur Lastrechnung über eine Anbindung ans Xrotor-Interface verfügt. Das UML-Diagramm der Klasse ist in Abbildung 4.5 zu sehen.

Propeller
<ul style="list-style-type: none"> <li>- __geometry = List</li> <li>- __loadcases = Dict</li> <li>- __load_envelope = Dict</li> <li>- __parameters = Dict</li> <li>- __sections = List</li>   <li>- __init__(self, number_of_blades, tip_radius, hub_radius)</li> <li>+ calc_loads(self)</li> <li>+ get_airfoil(self, rel_radius)</li> <li>+ pressure_distribution(self, loadcase) : np.array</li> <li>+ state(self, rel_chord, rel_radius, loadcase) : Dict</li>   <p>Getter:</p> <li>+ geometry : List</li> <li>+ loadcases : Dict</li> <li>+ load_envelope : Dict</li> <li>+ parameters : Dict</li> <li>+ sections : List</li>   <p>Setter:</p> <li>+ add_loadcase(self, loadcase)</li> <li>+ add_section(self, rR, airfoil)</li> <li>+ geometry(self, array)</li> <li>+ parameters(self, parameters)</li> <li>+ set_load_envelope(self)</li> </ul>

Abbildung 4.5.: Übersicht über die Propeller-Klasse

**Instanziierung und Geometrie** Bei der Instanziierung werden dem Propeller-Objekt die Parameter `number_of_blades`, `tip_radius` und `hub_radius` übergeben. Die Grundrissform des Propellers wird über das Attribut `geometry` definiert, welchem ein Array mit den Verteilungen von Profiltiefe und Schränkung übergeben wird.

Die Profilierung wird über das `sections`-Attribut als Array mit Airfoil-Objekten und ihren jeweiligen radialen Positionen definiert. Hier besteht die Aggregation zur Airfoil-Klasse.

**Lastfälle** Mit der Methode `add_loadcase` kann einem Propeller-Objekt eine beliebige Anzahl an Loadcase-Objekten übergeben werden. Diese werden in dem Attribut `loadcases` aggregiert.

Nach der Eingabe der Lastfälle wird die Lastrechnung mit `calc_loads()` gestartet. Diese führt für jeden Lastfall eine XROTOR-Rechnung aus und hängt die Ergebnisse (u.A. Verteilung von Auftrieb und Widerstand, Schnittkräfte und -momente) an die Loadcase-Objekte in `loadcases` an.

Mit der Methode `set_load_envelope()` kann daraus ein einhüllender Lastverlauf gebildet werden, der als `load_envelope` abgelegt wird.

**state()** `state(rel_chord, rel_radius, loadcase)` gibt für einen beliebigen Punkt auf dem Propeller (Eingabe als relativer Radius  $r/R$  und relative Profiltiefe  $x/C$ ) und Lastfall (oder Envelope) die lokalen Strömungsparameter als Dictionary zurück. Dieses enthält:

- **C<sub>l</sub>**: Auftriebsbeiwert  $c_a$
- **C<sub>d</sub>**: Widerstandsbeiwert  $c_w$
- **alpha**: Anstellwinkel  $\alpha$
- **Re**: Reynoldszahl  $Re$
- **C<sub>p\_suc</sub>**: Druckbeiwert  $c_p$  auf der Profiloberseite
- **C<sub>p\_pres</sub>**: Druckbeiwert  $c_p$  auf der Profilunterseite
- **C<sub>f</sub>**: Reibungsbeiwert  $c_f$

**get\_airfoil()** Die Straakprofile zwischen den Stützstellen der Geometrieeingabe können mit der Methode `get_airfoil(rel_radius)` bestimmt werden. Diese gibt mit der radialen Koordinate als Eingabe die Koordinaten des örtlichen Profils, seiner Dickenverteilung und der Skeletlinie zurück.

#### 4.3.5. Workflow bei der Lastenrechnung

Ein einfaches Beispiel für die Funktionsweise des Lastrechnungsprogramms ist in Code Listing 4.2 abgebildet. Der Ablauf ist so, dass zunächst die aerodynamischen Profile instanziert und deren Polaren berechnet werden. Anschließend wird ein Propeller-Objekt definiert, hier am Beispiel eines einfachen Zweiblattpropellers mit Doppeltrapez-Grundriss und zwei Profilen. Diesem können dann verschiedene Lastfälle zugewiesen werden, hier je ein Fall für maximalen Standschub und maximale Fluggeschwindigkeit.

Damit ist die Eingabe komplett. Mit den letzten beiden Befehlen wird die Berechnung ausgeführt und ein einhüllender Lastverlauf gebildet.

Die Ergebnisse sind als DataFrames im Propeller-Objekt hinterlegt und können mit allen üblichen Python-Werkzeugen visualisiert und verarbeitet werden.

Code Listing 4.2: Funktionsweise des Lastrechnungsprogramms

```

from util_loads import Propeller, Airfoil, Loadcase

# Definiere Profile
airfoil_inner = Airfoil('airfoil1.xfo', re=5e6, iter_limit=200)
airfoil_outer = Airfoil('airfoil2.xfo', re=5e6, iter_limit=200)

for foil in [airfoil_inner, airfoil_outer]:
    foil.set_polar(alpha_start=-20, alpha_stop=20, alpha_inc=0.25)

# Definiere Propeller
propeller = Propeller(number_of_blades=2,
                      tip_radius=0.5,
                      hub_radius=0.04,
                      )

# [r/R, c/R, beta]
propeller.geometry = np.array([[0.1, 0.1, 0],
                               [0.3, 0.2, 15],
                               [1., 0.1, 8],
                               ])

# [r/R, Airfoil]
propeller.sections = [[0.1, airfoil_inner],
                      [1., airfoil_outer],]

# Definiere Lastfaelle
lc_rpm = Loadcase(name='Max_Standschub', flight_speed=0.01)
lp_pow = Loadcase(name='Max_Speed', flight_speed=15.)
lc_rpm.set_data('rpm', 4000)
lc_pow.set_data('powe', 3000)

for case in [lc_rpm, lc_pow]:
    propeller.add_loadcase(case)

# Berechne Lasten
propeller.calc_loads()
propeller.set_load_envelope()

```

## 5. Erstellung eines Optimierungstools

Bei der Analyse mechanischer Strukturen ist die Methode der Finiten Elemente ein vielfach verwendetes Standardwerkzeug. Das Ziel dieser Arbeit ist es, ein Analysemodell auf Basis der FEM in einen automatisierten Optimierungsprozess einzubetten, sodass ein mächtiges und effizientes Auslegungswerkzeug für UAV-Propeller entsteht.

Das Analysemodell und Integration in einen Optimierungsprozess werden in diesem Kapitel vorgestellt.

### 5.1. Verwendete Software

**ANSYS APDL und PyMAPDL** Als FE-Programm wurde im Rahmen dieser Arbeit ANSYS 2010 R1 verwendet. Dieses bietet dem Nutzer verschiedene Arbeitsweisen zur Modellierung und Berechnung von Bauteilen an:

- Verwendung der graphischen Benutzeroberfläche ANSYS WORKBENCH
- Verwendung der ANSYS-eigenen Skriptsprache APDL (ANSYS Parametric Design Language)

Die erstgenannte Methode ist für einzelne, einmalige Rechnungen gut geeignet, da ein Modell durch graphische Menüführung schnell erstellt und analysiert werden kann. Die Arbeit mit Skripten funktioniert hingegen so, dass in einem externen Editor ein sogenanntes Inputfile geschrieben und dann in ANSYS eingelesen wird. ANSYS arbeitet alle darin enthaltenen Befehle nacheinander ab. Der Vorteil dabei ist, dass in dem Inputfile alle wichtigen Modellgrößen über Parameter definiert werden können, welche sich leicht verändern lassen, sodass Parameterstudien einfach durchgeführt werden können.

APDL ist von der Programmiersprache FORTRAN abgeleitet und hat im Vergleich zu modernen Sprachen einige Eigenarten, welche den Arbeitsablauf negativ beeinträchtigen:

- Für die Erstellung der Inputfiles existiert keine Entwicklungsumgebung, sodass herkömmliche Texteditor-Software verwendet werden muss. Dies bedingt, dass es keine graphische Visualisierung der Syntax oder Textvervollständigung gibt und die Dokumentation zu den verwendeten Befehlen händisch nachgeschlagen werden muss.
- Als Programmierstil ist nur ein prozeduraler Ansatz möglich, bei dem alle Zeilen des Inputfiles von oben nach unten abgearbeitet werden. Moderne Paradigmen wie die objektorientierte Programmierung können nicht verwendet werden.
- Es existiert kein Debugger. Fehler und Warnungen werden lediglich nach der Ausführung des Skripts in ein ASCII-File geschrieben, was die Fehlersuche ineffizient und zeitaufwändig gestaltet.

Von KASZYNSKI [23] wurde mit PyMAPDL ein Python-Interface zu ANSYS APDL entwickelt, mit dem der komplette Simulationsprozess interaktiv aus Python heraus gesteuert werden kann. Dies vermeidet die oben aufgeführten Probleme. Die Verwendung von Python erlaubt es, alle dafür verfügbaren Entwicklungsumgebungen und Debugger zu verwenden.

Zudem kann bei der Erstellung und Analyse des FE-Modells auf alle Bibliotheken und Objekte, die es für Python gibt, zugegriffen werden. So können z.B. Optimierer und Visualisierung- und Plotting-Tools direkt verwendet werden, ohne dass ein Zwischenschritt über das Ausgeben und Einlesen von ASCII-Files notwendig ist.

**PyOpt** Als Optimierungswerkszeug wird die Python-Bibliothek PyOPT [31] mit dem ALPSO-Optimierer verwendet, die in Kapitel 3 vorgestellt und evaluiert wurde.

**util.loads** Zur Berechnung der aerodynamischen Lasten wird das in Kapitel 4 vorgestellte Lastrechnungsprogramm verwendet.

## 5.2. Programmarchitektur

Für die Architektur des Auslegungs- und Optimierungstools wurde ein objektorientierter Ansatz gewählt. Eine Übersicht ist in Abbildung 5.1 dargestellt.

Bei der Konzeptionierung wurde darauf Wert gelegt, dass das Analysemodell eigenständig ohne eine Optimierungsroutine verwendet werden kann, um die Möglichkeit einer herkömmlichen Dimensionierung beizubehalten. Daher ist die Architektur in zwei Bereiche *Optimierung* und *Analysemodell* unterteilt.

Das Analysemodell hat die übergeordnete Klasse `Femodel`. In dieser sind alle Methoden implementiert, welche allgemein für die Simulation von FE-Modellen benötigt werden, also z.B. Solver-Einstellungen oder die Ausgabe von Reaktionskräften.

Sie ist über eine Aggregation mit der Klasse `MAPDL_Core` aus PyMAPDL verbunden, die das Interface zu Ansys herstellt. Zusätzlich können in einem `Femode1`-Objekt mehrere Instanzen der Material-Klasse aggregiert werden. In dieser ist eine Materialdatenbank hinterlegt, mit der die verwendeten Werkstoffe und ihre Ingenieurkonstanten verwaltet werden können. Die Klasse `Propellermodel` enthält alle für UAV-Propeller spezifischen Methoden, also z.B. für Geometrie- und Lastdefiniton und für das Post-Processing. Alle allgemeinen Methoden erbt sie von der `Femode1`-Klasse. Zur Lastaufprägung hat sie eine Aggregation zu der `Propeller`-Klasse aus Kapitel 4.

Für die Optimierungsroutine sind die Klassen `Optimization` und `ALPSO` aus PyOPT eingebunden. Das Optimierungsproblem wird als `Optimization`-Objekt definiert. Diesem werden Design-Variablen, Restriktionen und eine Zielfunktion zugewiesen. Die Zielfunktion nutzt zur Berechnung die Methoden des `Propellermodel`-Objekts aus dem Analysemodell.

Zur Lösung des Optimierungsproblems dient die `ALPSO`-Klasse mit dem Partikelschwarm-Algorithmus. Alternativ könnten bei Bedarf auch andere Optimierer aus der PyOPT-Bibliothek verwendet werden.

Im folgenden werden die einzelnen Klassen und ihre wichtigsten Methoden vorgestellt. Eine vollständige Dokumentation ist auf der Daten-CD beigelegt.

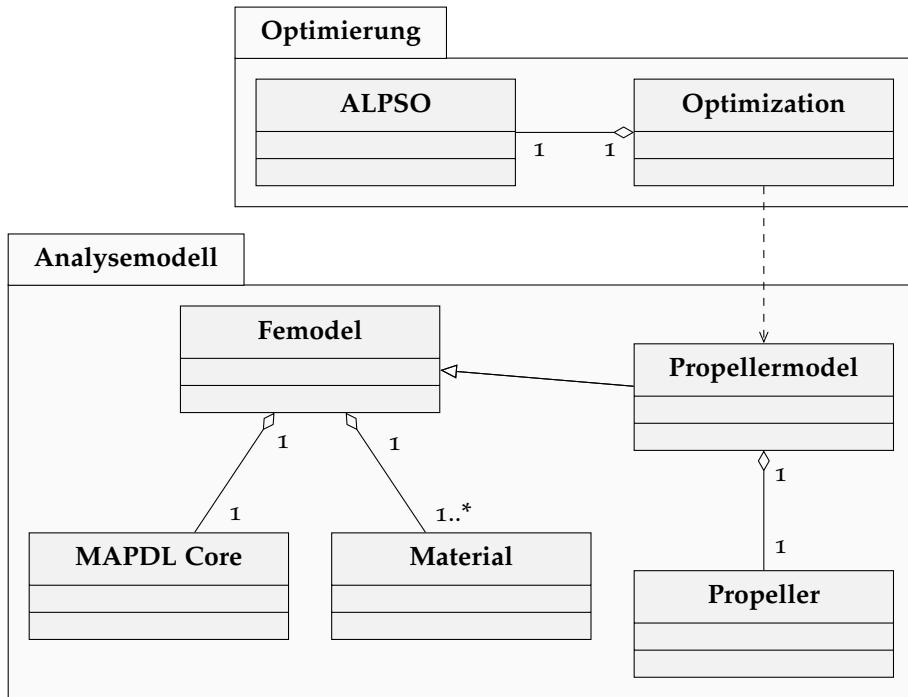


Abbildung 5.1.: UML-Klassendiagramm des Optimierungstools

### 5.2.1. Femodel-Klasse

Die `Femodel`-Klasse beinhaltet alle Methoden und Attribute, die allgemein bei der Simulation verschiedener Modelle und Bauteile benötigt werden. In Abbildung 5.2 ist ein UML-Diagramm der Klasse dargestellt. Bei der Instanziierung eines `Femodel`-Objekts wird über das Attribut `mapdl` eine Verbindung zu Ansys hergestellt.

Die Methoden enthalten Abfolgen von APDL-Befehlen, die wiederholt ausgeführt werden. Es ist jedoch auch möglich, einzelne Befehle direkt über die Python-Konsole auszuführen ohne eine `Femodel`-Instanz zu benutzen.

Da im Rahmen dieser Arbeit ausschließlich linear-statische Analysen durchgeführt werden, sind die entsprechend konfigurierten Solver-Befehle dafür in der `solve()`-Methode abgelegt. Die Methoden `cdwrite()`, `cdread()` und `clear()` dienen zum Verwalten der APDL-Session. Mit ihnen können Geometrie, Netz und Lastdefinitionen in ein ASCII-File geschrieben, bzw. daraus eingelesen werden. `clear()` setzt die gesamte APDL-Session zurück.

Für das Post-Processing wurde die Methode `fc_puck()` implementiert. Diese gibt für die in APDL ausgewählten Elemente<sup>1</sup> die maximalen Werte der PUCK'schen Versagenskriterien zurück.

<sup>1</sup>mit den Befehlen ESEL oder CMSEL

<b>Femodel</b>
<ul style="list-style-type: none"> <li>- <code>__ansys_input_filename</code> : Str</li> <li>+ <code>mapdl</code> : MapdlConsole</li> <li>- <code>__materials</code> : Dict</li> <li>- <code>__mesh_density_factor</code> : Int</li>   <li>- <code>__init__(self, mapdl, mesh_density_factor)</code></li> <li>+ <code>cdread(self)</code></li> <li>+ <code>cdwrite(self)</code></li> <li>+ <code>clear(self)</code></li> <li>+ <code>fc_puck(mapdl) : Float, Float</code></li> <li>+ <code>reaction_forces(self) : Dict</code></li> <li>+ <code>solve(self)</code></li> <li>Getter:</li> <li>+ <code>materials : Dict</code></li> <li>+ <code>mesh_density_factor : Int</code></li> <li>Setter:</li> <li>+ <code>materials(self, material)</code></li> <li>+ <code>mesh_density_factor(self, factor)</code></li> </ul>

Abbildung 5.2.: Übersicht über die Femodel-Klasse

### 5.2.2. Material-Klasse

Um die verwendeten Werkstoffe zu verwalten wurde die Material-Klasse erstellt, in der eine Materialdatenbank hinterlegt ist. Bei der Instanziierung eines Material-Objekts werden anhand eines ID-Strings die Kennwerte des gewünschten Materials aus der Datenbank geladen. Die Materialparameter und Festigkeitskennwerte sind anschließend als Dictionary in den Attributen `mp` und `fc` abgelegt.

Mit den Methoden `assign_mp` und `assign_fc` können die Kennwerte dann in der APDL-Sitzung zugewiesen werden.

Neue Werkstoffe können über die Methode `read_csv()` in der Datenbank abgelegt werden. Diese liest eine Textdatei aus, in der die Materialkennwerte mit der ADPL-Syntax und den Befehlen `MP` und `FC` aufgelistet sind. Mit der Methode `save_to_db()` können zudem Datensätze in der Datenbank überschrieben werden, um Veränderungen zu ermöglichen.

Material
<pre>+ fc : Dict + mapdl : MapdlConsole + mp : Dict - __name : Str - __number : Int - __path : Str  - __init__(self, mapdl, name, number) + assign_fc(self) + assign_mp(self) + load_from_db(self) + read_csv(self, csvfile) + save_to_db(self) Getter: + fc : Dict + mp : Dict + name : Str + number : Int Setter: + name(self, name) + number(self, number)</pre>

Abbildung 5.3.: Übersicht über die Material-Klasse

### 5.2.3. Propellermodel-Klasse

Die Modellierung des UAV-Propellers findet in der Propellermodel-Klasse statt. Dieser Abschnitt beschreibt die Methodik und Funktionsweise der Klasse. Die inhaltliche Dokumentation des FE-Modells ist Teil des folgenden Abschnittes 5.3.

Ein Propellermodel-Objekt empfängt bei der Instanziierung folgende Objekte:

- `mapdl`: Die APDL-Instanz, mit der die FE-Simulation durchgeführt wird.
- `propeller`: Ein Propeller-Objekt. Die Aggregation zu der Propeller-Klasse aus dem Lastrechnungstool dient zur Aufprägung der aerodynamischen Lasten.
- `n_sec`: Ein Integer-Wert. Dieser legt fest, in wie viele Komponenten (*Sections*) das Modell unterteilt wird. Für jede Komponente kann später mit einem Satz an Designvariablen ein eigener Lagenaufbau zugewiesen werden. Damit die Sections radial („spannweitig“) exakt gleichmäßig verteilt werden können, sollte die Anzahl der Elemente in radialer Richtung ganzzahlig durch die Anzahl der Komponenten teilbar sein.
- `mesh_density_factor`: Ein Integer-Wert zur Steuerung der Netzdichte.

**pre\_processing()** Das Pre-Processing des FE-Modells wurde der Übersichtlichkeit halber in mehrere Schritte unterteilt, welche von der `pre_processing()`-Methode in der richtigen Reihenfolge ausgeführt werden:

1. `__define_and_mesh_geometry__()`: Einlesen eines Geometrie-Inputfiles, welches zuvor mit CAD erstellt wurde. Die eingelesene Geometrie wird anschließend mit den APDL-eigenen Befehlen vernetzt.
2. `__calc_element_data__()`: Diese Methode iteriert über alle Elemente des zuvor erstellten Netzes. Für jedes Element werden mit Hilfe der Propeller-Klasse des Lastrechnungstools alle örtlichen Strömungs-Parameter bestimmt. Dazu gehören u. A.: Auftrieb und Auftriebsbeiwert, der Gesamtwiderstand mit Druck- und Reibungsanteil, lokale Geschwindigkeit. Die Daten werden als DataFrame im Attribut `element_data` abgelegt.
3. `__apply_loads__()`: Aufprägung der aerodynamischen und rotatiorischen Lasten sowie Randbedingungen.
4. Zuletzt wird mit `cdwrite()` ein ANSYS-Inputfile des Pre-Processings erzeugt und mit `clear()` die APDL-Session zurückgesetzt.

Propellermodel	
- <code>__element_aoa_vector</code> : np.array	
- <code>__element_chord_vector</code> : np.array	
- <code>__element_data</code> : DataFrame	
- <code>__n_sec</code> : Int	
+ <code>propeller</code> : Propeller	
+ <code>mapdl</code> : MapdlConsole	
- <code>__materials</code> : Dict	
- <code>__mesh_density_factor</code> : Int	
- <code>__init__(self, mapdl, propeller, n_sec, mesh_density_factor)</code>	
- <code>__define_and_mesh_geometry__(self)</code>	
- <code>__get_edges__(self, y)</code> : List	
- <code>__apply_loads__(self)</code>	
+ <code>change_design_variables(self, global_vars, *args)</code>	
+ <code>convergence_study(self, mesh_density)</code> : List	
+ <code>evaluate(self, x)</code> : Float, List, List	
+ <code>pre_processing(self)</code>	
+ <code>post_processing(self)</code> : Float, List, List	
Getter:	
+ <code>element_aoa_vector</code> : np.array	
+ <code>element_chord_vector</code> : np.array	
+ <code>element_data(self)</code> : DataFrame	
Setter:	
- <code>__calc_element_data__(self)</code>	

Abbildung 5.4.: Übersicht über die Propellermodel-Klasse

**evaluate()** Eine Auswertung des FE-Modells geschieht mit der `evaluate()`-Methode. Diese empfängt eine Liste mit den Designvariablen für den Lagenaufbau des Propellers. Darauf basierend werden die folgenden Schritte ausgeführt:

1. `cdread()`: Das generierte Pre-Processing-Inputfile wird mit der `cdread()`-Methode eingelesen.
2. `change_design_variables(global_vars, *args)`: Definition des Lagenaufbaus anhand der Designvariablen.
3. `solve()`: Lösen des FE-Modells.
4. `post_processing()`: Auswertung von Verschiebungen, Versagenskriterien und Bauteilmasse.
5. `clear()`: Zurücksetzen der APDL-Sitzung.
6. Rückgabe der Ergebnisse des Post-Processings.

## 5.3. Aufbau des Analysemodells

### 5.3.1. Geometrie und Vernetzung

Da das FE-Modell als Analysemodell im Rahmen einer Strukturoptimierung verwendet wird, ist es notwendig, den Berechnungsaufwand bei der Lösung des Modells gering zu halten. Die erwartete Größenordnung der Anzahl an Funktionsauswertungen liegt im Bereich von  $\approx 10^4 - 10^5$ .

Da die Hauptabmessung, also die Blattlänge, wesentlich größer als die Dicke ist, wird das Bauteil gemäß der KIRCHHOFF'schen Plattentheorie als zweidimensionale Platte abstrahiert. Die Modellierung erfolgt dann über sogenannte Schalenelemente (*Shell Elements*), welche neben den 2D-Abmessungen Informationen über ihre Dicke und den Lagenaufbau enthalten. Zur Modellierung wird das Element *SHELL281* von ANSYS verwendet. Dieses ist ein quadratisches 8-Knoten-Element mit vollständiger Integration<sup>2</sup>. Der Berechnungsaufwand beim Lösen des Modells ist so geringer als bei der Verwendung von Volumenelementen.

Als Geometrieeingabe wird daher eine Fläche benötigt, welche das Propellerblatt in der Mitte seiner Dickenrichtung schneidet. Aerodynamisch gesehen ist dies eine Verbindungsfläche der Skelettlinien aller Profilquerschnitte des Propellers.

Diese Fläche kann mit den Freiform-Werkzeugen gängiger CAD-Software erstellt werden. Im Rahmen dieser Arbeit wurden zur Modellierung SIEMENS NX und CATIA V5 verwendet. Die Fläche wird aus dem CAD-Programm in das Austauschformat IGES abgeleitet und mit Hilfe des AUX15-Pre-Processors in Ansys eingelesen.

Es wurde die Erfahrung gemacht, dass bei der CAD-Modellierung auf eine glatte, krümmungsstetige Kontur zu achten ist, um Spannungs-Singularitäten beim Post-Processing des FE-Modells zu vermeiden.

In Abbildung 5.5 ist die Mittenfläche eines Propellerblatts beispielhaft abgebildet.

---

<sup>2</sup>Das Element wird mit KEYOPT,,8,1 verwendet, sodass Verschiebungen und Nachlaufgrößen für alle Lagen einzeln ausgewertet werden können.

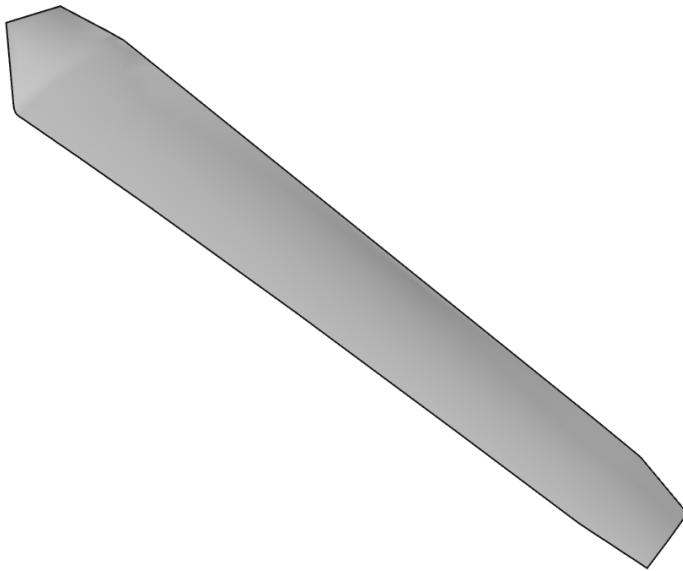


Abbildung 5.5.: CAD-Modell der Mittenfläche eines Propellerblatts

Zur Vernetzung werden die ANSYS-eigenen Methoden verwendet. Diese erfolgt als *Mapped Mesh*, bei dem die Form und Lage der Elemente so vorgegeben wird, dass ein regelmäßiges Netz entsteht.

Die Netzdichte ist über den Integer-Parameter `mesh_density_factor` veränderbar. Für `mesh_density_factor = 1` werden auf der Längsachse des Blatts 46 und entlang der Profilsehnen 15 Elemente verteilt. Dies ergibt im Mittel ein Seitenverhältnis der Elemente von ca 1:1. Ein solches Netz ist in Abbildung 5.6 dargestellt.

Die Zuweisung der Elementdicke erfolgt mit Hilfe der `get_airfoil()`-Methode der Propeller-Klasse, die für jeden Punkt auf dem Propeller die örtliche Profilgeometrie zurückgibt. Als Eingabewerte werden die Mittelpunkte aller Elemente verwendet. Abbildung 5.7 zeigt eine 3D-Visualisierung des Netzes mit Elementdicke.

Die Zuweisung des Lagenaufbaus der Elemente wird in der Methode `change_design_variables()` implementiert, diese weist jedem Element abhängig von der Elementdicke und den Designvariablen eine individuelle Belegung zu. Der Inhalt der Methode ist abhängig von dem verfolgten Strukturkonzept (Sandwich oder Vollmaterial, Stack ...) und kann je nach Zielsetzung ausgetauscht werden.

### 5.3.2. Lastaufprägung

**Luftkräfte** Zur Aufprägung der auf den Propeller wirkenden aerodynamischen Kräfte nutzt das FE-Modell die mit dem in Kapitel 4 beschriebenen Lastrechnungsprogramm berechneten Lastverläufe.

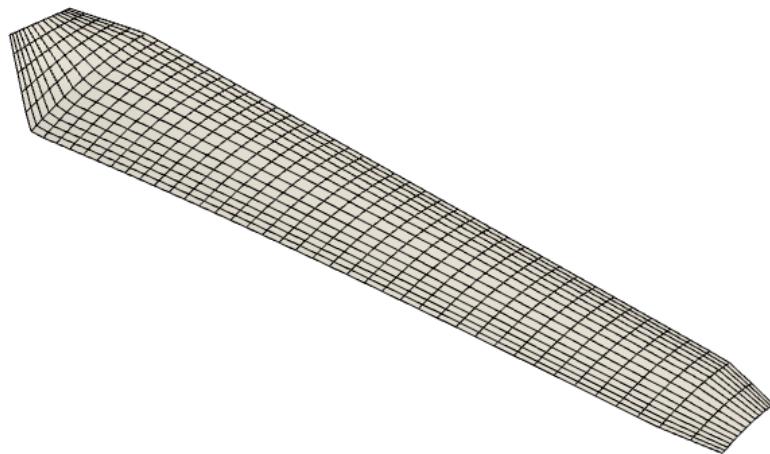


Abbildung 5.6.: FE-Netz der Mittenfläche eines Propellerblatts

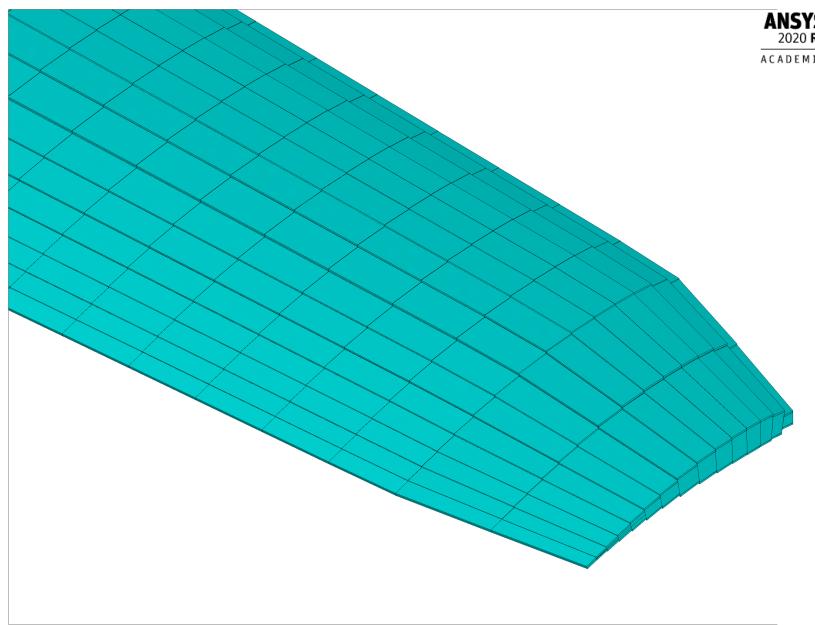


Abbildung 5.7.: FE-Netz der Propellerblatts mit Visualisierung der Elementdicke (APDL: ESHAPE,1).

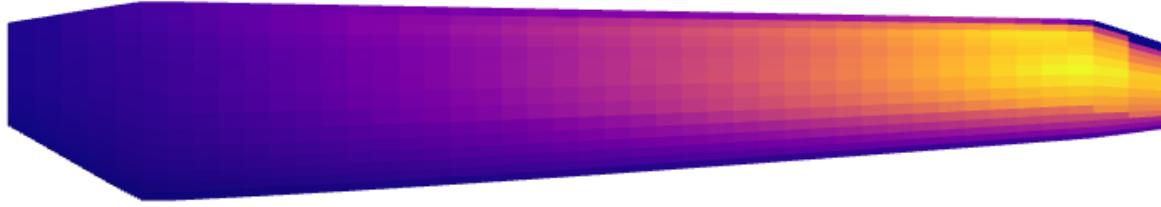


Abbildung 5.8.: FE-Netz mit aufgeprägten Druckkräften

Die wirkenden Kräfte bestehen nach SCHLICHTING und TRUCKENBRODT [37] aus:

- Druckkräften: Auftrieb und Druckwiderstand
- Schubspannungskräften: Reibungswiderstand

Diese werden für jedes Element einzeln berechnet und aufgeprägt, sodass eine flächig verteilte Lastaufprägung entsteht. Dazu wird die `state(rel_chord, rel_radius)`-Methode der Propeller-Klasse<sup>3</sup> verwendet. Mit den normierten Koordinaten des Elementmittelpunkts als Eingabewert liefert diese die Druckbeiwerte  $c_{p,suc}$  und  $c_{p,pres}$  auf der Profilober- und Unterseite, den Reibungsbeiwert  $c_f$  und Anstellwinkel  $\alpha$ .

Mit der Propellerdrehzahl  $f$  und dem radialen Abstand von der Propellernabe  $r$  kann dann der herrschende Druck bestimmt werden:

$$P = (c_{p,suc} + c_{p,pres}) \cdot \frac{\rho}{2} v^2 \quad (5.1)$$

$$= (c_{p,suc} + c_{p,pres}) \cdot \frac{\rho}{2} (2\pi r f)^2 \quad (5.2)$$

Dieser wird mit dem SFE-Befehl von APDL als Flächenlast (*Structural Pressure*) aufgeprägt. Eine beispielhafte Verteilung der aufgeprägten Lasten ist in Abbildung 5.8 zu sehen.

Der Reibungswiderstand kann über die Wandschubspannung  $\tau$  berechnet werden. Für diese gilt laut DRELA und YOUNGREN [10]:

$$\tau = c_f \cdot \frac{\rho}{2} v^2 \quad (5.3)$$

$$= c_f \cdot \frac{\rho}{2} (2\pi r f)^2 \quad (5.4)$$

Über die Multiplikation mit dem Flächeninhalt des Elements  $A_{elem}$  erhält man den Reibungswiderstand  $W_f$ .

$$W_f = A_{elem} \cdot \tau \quad (5.5)$$

Dieser wirkt per Definition parallel zur Anströmrichtung, die mit dem Anstellwinkel  $\alpha$  bestimmt werden kann. Die Aufprägung auf das FE-Modell erfolgt als Einzelkräfte auf die Elementknoten. Bei dem 8-Knoten-Element SHELL281 wird also jeder Knoten mit 1/8 der Widerstandskraft beaufschlagt. Die Kräfte an gemeinsamen Eck- und Kantenknoten angrenzender Elemente werden aufsummiert. Abbildung 5.9 zeigt eine Übersicht der aufgeprägten Kräfte.

<sup>3</sup>siehe Unterabschnitt 4.3.4.

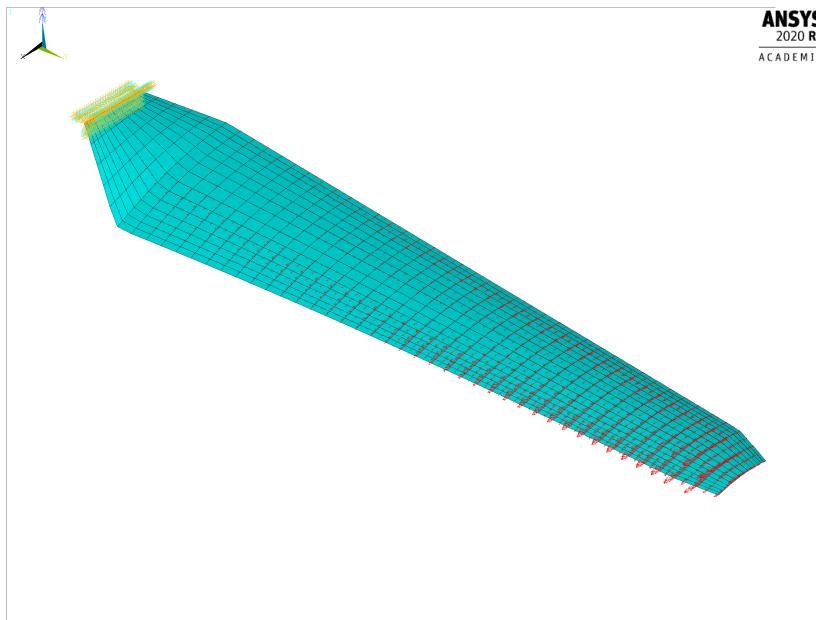


Abbildung 5.9.: FE-Netz mit aufgeprägten Reibungskräften

**Zentrifugalkraft und -beschleunigung** Mit den Materialparametern wird dem FE-Modell auch die Dichte der verwendeten Werkstoffe übergeben. Die Zentrifugalkräfte und Beschleunigungen können direkt über die APDL-Befehle OMEGA und DOMEWA aufgebracht werden, die eine Rotationsgeschwindigkeit bzw. -beschleunigung des Modells um den Koordinatenursprung definieren. ANSYS berechnet daraus zusammen mit den Element-Massenmatrizen Kraftvektoren auf die Elemente.

### 5.3.3. Post-Processing

Nach dem Lösen des FE-Modells wird ein automatisiertes Post-Processing durchgeführt. Der Quelltext dazu ist in der `post_processing()`-Methode implementiert. Ausgewertet werden:

- Die Bauteilmasse, für den Fall, dass diese als Zielfunktion in der Optimierung verwendet werden soll.
- Die Versagenskriterien nach PUCK auf Faserbruch und Zwischenfaserbruch. Diese werden jeweils als List-Objekt mit einem Eintrag für jede Komponente<sup>4</sup> zurückgegeben.

Da das Python-ANSYS-Interface interaktiv ist, kann ebenfalls ein händisches Post-Processing aus Python heraus durchgeführt werden. Dazu stehen zusätzlich zu den Befehlen des APDL-Post-Processors auch in PyMAPDL enthaltene Tools, wie etwa Interaktive Plotting-Objekte, zur Verfügung.

<sup>4</sup>Zuvor definiert mit dem `n_sec`-Eingabewert des *Propellermodel*-Konstruktors.

## 5.4. Aufbau der Optimierungsroutine

### 5.4.1. Ansatz für die Zielfunktion

Das Optimierungsproblem wird innerhalb des Programms durch ein Objekt der Optimization-Klasse von PyOPT abgebildet. Diesem muss bei der Instanziierung eine Zielfunktion zugewiesen werden. Dieser Abschnitt stellt den theoretischen Hintergrund der verwendeten Funktion vor.

Für die Optimierung der einfachen Testcases in Abschnitt 3.4 konnte als Ziel direkt die Bauteilmasse minimiert werden:

$$\min f(\vec{x}) = m(\vec{x}) \quad (5.6)$$

Optimiert man komplexere Bauteile, wie UAV-Propeller, mit der Masse als Zielfunktion stößt diese Formulierung jedoch an Grenzen: Bei Komponenten des Bauteils, an denen viel Masse konzentriert ist, reagiert die Zielfunktion deutlich sensitiver auf Veränderungen der Designvariablen als bei Komponenten mit geringer Masse.

Für Propeller bedeutet dies, dass im Wurzelbereich mit hoher Querschnittsfläche bei Veränderung der Wandstärke eine deutlich größere Rückkopplung auf die Zielfunktion besteht als im Blattspitzenbereich, wo der Querschnitt gering ist.

Numerisch gesehen führt die Masse als Zielfunktion in diesem Fall zu einer schlechten Konditionierung des Optimierungsproblems [18], also einem ungünstigen Verhältnis von Eingangs- zu Ausgangswerten.

Eine alternative Wahl der Zielfunktion vermeidet diese Problematik: Anstatt der Masse wird die Anstrengung der Bauteilkomponenten auf einen Zielwert hin optimiert. Damit wird überflüssiges Material abgebaut und indirekt eine Massenreduzierung herbeigeführt.

$$\min (\max |\{I_i, i \in B\} - I_{ref}|) \quad (5.7)$$

mit dem Versagensindex  $I$ , dem Zielwert  $I_{ref}$  und der Anzahl der Komponenten  $B$ . Mit der obigen Formulierung wird also immer die Bauteilkomponente mit der größten Differenz vom Zielwert optimiert. Das hat jedoch den Nachteil, dass unstetige Sprünge der Zielfunktion auftreten können, wenn sich während der Optimierung die maximale Differenz von einer zu einer anderen Komponente hin verlagert.

Dies kann laut HARZHEIM zu einer erhöhten Anzahl an Iterationen bis schlimmstenfalls zum Versagen des Algorithmus führen [18].

Ein glatterer Optimierungsverlauf wird durch die Verwendung der sog. *Beta-Methode* gewährleistet [18]. Diese führt für den Fall, dass von  $m$  Zielfunktionen  $f_j$  (1 je Komponente) die maximale Funktion minimiert werden soll, eine zusätzliche Designvariable  $\beta$  ein und löst folgendes Optimierungsproblem:

$$\min |\beta - I_{ref}| \quad (5.8)$$

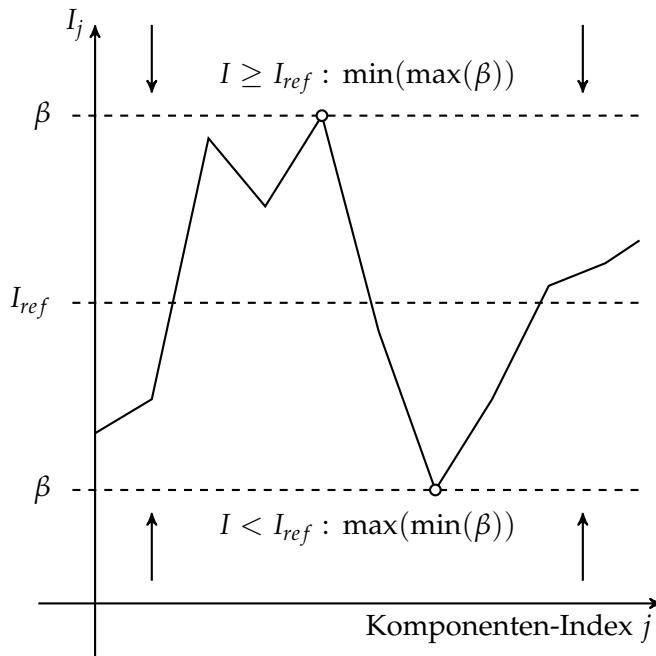


Abbildung 5.10.: Veranschaulichung der Beta-Methode

sodass

$$f_j \leq \beta; \quad \text{wenn } \beta \geq I_{ref} \quad (5.9)$$

$$f_j > \beta; \quad \text{wenn } \beta < I_{ref} \quad (5.10)$$

In Abbildung 5.10 ist die Funktionsweise der Beta-Methode graphisch dargestellt. Dort sind die Anstrengungen  $I_j$  für alle  $j$  Komponenten eines Bauteils aufgetragen. Die Restriktionen erzwingen, dass eine Minimierung der Zielfunktion ( $\beta$ ) nur möglich ist, wenn die maximale Differenz der Anstrengung  $I$  vom Zielwert  $I_{ref}$  abgesenkt wird. Für Komponenten mit Anstrengung  $I \geq I_{ref}$  entspricht dies dem äquivalenten Optimierungsproblem  $\min(\max(\beta))$ , bei  $I < I_{ref}$  dementsprechend  $\max(\min(\beta))$ .

Man kann sich die Zielfunktion also wie ein Korsett vorstellen, welches die Restriktionen zunehmend enger um den Zielwert zusammenschnürt.

#### 5.4.2. Formulierung von Zielfunktion und Restriktionen

Aufgrund der oben dargestellten Überlegungen soll das Optimierungsproblem zur Auslegung von UAV-Propellern mit der Beta-Methode formuliert werden. Um eine möglichst gute Materialausnutzung, und damit eine geringe Bauteilmasse, zu erhalten, wird die Anstren-

gung auf Faserbruch nach PUCK auf einen höchstzulässigen Wert von  $I_{ref} = 1$  hin optimiert. Die Zielfunktion ist daher:

$$\min f(\beta) \quad (5.11)$$

$$= \min |\beta - 1| \quad (5.12)$$

mit den Restriktionen der Beta-Methode

$$g_j(\vec{x}) = \begin{cases} I_{fib,j} - \beta; & \text{für } I_j \geq I_{ref} \\ \beta - I_{fib,j}; & \text{für } I_j < I_{ref} \end{cases} \quad (5.13)$$

Die weiteren Ungleichheitsrestriktionen betreffen die Versagenskriterien für Faserbruch und Zwischenfaserbruch. Durch sie soll sichergestellt werden, dass die Anstrengungen im gefundenen Minimum im zulässigen Bereich  $< 1$  liegen.

$$g_j(\vec{x}) = I_{fib,j} - 1 \quad (5.14)$$

$$g_j(\vec{x}) = I_{mat,j} - 1 \quad (5.15)$$

#### 5.4.3. Designvariablen

Die Auswahl der Designvariablen ist abhängig von dem verfolgten Strukturkonzept. Dazu sind verschiedenste Ansätze denkbar, folgende Auflistung zeigt einige Beispiele:

- Massiv- oder Sandwichbauweise.
- Schichtung des Lagenaufbaus und verwendete Faserorientierungen (Reines UD-Laminat oder z.B. zusätzliches Schublaminat zur Aufnahme von Torsion).
- Symmetrischer oder asymmetrischer Lagenaufbau.
- Art der Lasteinleitung im Nabenhochbereich.

Durch den objektorientierten Ansatz des Programms können verschiedenste Philosophien ausprobiert und verglichen werden, da dazu lediglich einzelne Objekte des Programms ausgetauscht oder verändert werden müssen. Dies sind die Designvariablen, welche in dem *Optimization*-Objekt definiert sind sowie deren Übersetzung in den Lagenaufbau mit der `change_design_variables()`-Methode des *Propellermodel*-Objekts.

Wie in Abschnitt 3.2 beschrieben, erlauben PyOPT und sein ALPSO-Algorithmus kontinuierliche, diskrete, und ganzzahlige Designvariablen [31]:

- *continuous - 'c'*: Mit kontinuierlichen Variablen können z.B. Faserorientierungen oder Faseranteile in einem Querschnitt implementiert werden.
- *integer - 'i'*: Integer-Designvariablen eignen sich z.B. zur Variation von Lagenanzahlen oder Abstufungen.
- *discrete - 'd'*: Diskrete Designvariablen können beispielsweise verwendet werden, um dem Optimierer eine Auswahl aus einer Liste verfügbarer Werkstoffe zu ermöglichen.

Für jede Variable ist es möglich, zusätzlich Grenzwerte, also explizite Restriktionen anzugeben.

Faserorientierungen müssen mit PyOPT also als kontinuierliche Variablen implementiert werden. Für den Fall, dass der Designraum möglicher Faserwinkel nicht eingeschränkt werden soll, werden als explizite Restriktionen  $x^L = 0^\circ$  und  $x^U = 180^\circ$  gesetzt.

Die Bewegungsgleichung der Partikel des ALPSO-Verfahrens (s. Abbildung 3.1) ist so implementiert, dass ein Partikel auf die Grenze des Designraums zurückgesetzt wird, falls es diese aufgrund seines aktuellen Geschwindigkeitsvektors überschreiten würde. Dieses Verhalten ist in Bezug auf polar verteilte Größen nicht sinnvoll, da diese keinen Grenzwert haben, sondern frei drehbar im Sinne eines Winkels sind.

Um Faserwinkel als Designvariablen mathematisch sinnvoll modellieren zu können, wurde im Rahmen dieser Arbeit ein weiterer Variablentyp in den ALPSO-Optimierer von PyOPT implementiert:

- *polar - 'p'*: Dieser Variablentyp dient zur Modellierung von Größen in Polarkoordinaten wie Faserwinkeln.

*polar*-Variablen sind von den kontinuierlichen Variablen abgeleitet. Sie funktionieren so, dass ein Partikel nach Überschreiten einer Grenze des Designraums um einen Betrag  $\delta x$ , um genau diesen Betrag entfernt von der jeweils anderen Grenze initialisiert wird.

Die entsprechenden Veränderungen im Quelltext der PyOPT-Bibliothek sind auf der Daten-CD enthalten.

## 6. Anwendung auf Referenz-Geometrie

Der Inhalt dieses Kapitels ist die Anwendung des erstellten Programms auf eine exemplarische Propellergeometrie. Die Programmkette erzeugt dabei basierend auf einer bestehenden aerodynamischen Hülle des Propellers einen lastgerechten Strukturentwurf. Da derzeit bei *Leichtwerk Research* noch kein aerodynamischer Entwurf besteht, wurde zunächst ein am Markt erhältlicher Propeller vermessen.

Ausgehend davon werden alle Schritte von der Lastrechnung über den Aufbau des Analysemodells bis zur Optimierung durchlaufen und die Ergebnisse dargestellt. Anschließend wird ein Beispiel gegeben, welche Möglichkeiten Python zur Auswertung und Interpretation der Optimierungsergebnisse bietet.

### 6.1. Vermessung eines Propellers

Die gewählte Propellergeometrie entspricht dem im Projekt *MesSBAR* [7] bisher vorgesehenen Propeller *MF3218* der Firma *T-Motor*.

Dieser hat einen Durchmesser von  $32,4'' \approx 824\text{ mm}$  und leistet bei der maximal zulässigen Drehzahl von  $4000\text{ 1/min}$  einen Schub von  $18\text{ kg} \approx 177\text{ N}$  [42].

**Methodik** Als Eingabe für die Lastrechnung werden Verteilungen von Profiltiefe und Steigung des Propellers über dem Radius sowie die Koordinaten der verbauten Profile benötigt. Zur Vermessung wurde ein Propeller in einem Quader aus Kunstharz vergossen, welcher in Scheiben regelmäßiger Dicke zersägt wurde.

Die so entstandenen Querschnitte des Propellers wurden photogrammetrisch vermessen und daraus die Profilkordinaten, Profiltiefe und Steigung jedes Schnittes errechnet. Als Software für die Vermessung wurde ein Python-Script mit Hilfe des Pakets *OpenCV (Open Computer Vision)* [5] erstellt. Dieses enthält unter anderem Funktionen zur Bildbinarisierung und zur Erkennung von Features wie Ecken und Kanten.

Die Arbeitsweise des Scripts ist in Abbildung 6.1 illustriert:

1. Zunächst wird ein Foto eines Querschnitts eingelesen und so rotiert, dass sie seitlichen Kanten des Blocks vertikal verlaufen.
2. Es wird ein Grauwert als Schwelle festgelegt und das Foto anhand dessen binarisiert. Dadurch wird alles außer dem schwarzen Propellerquerschnitt ausgeblendet.
3. Die Nasen- und Endkante werden detektiert und daraus Profiltiefe und Steigung berechnet.
4. Der Profilquerschnitt wird so rotiert, dass die Profilsehne horizontal verläuft.
5. Erkennung der Profilkontur.
6. Umrechnung und Normierung der Kontur in (x,y)-Koordinaten.
7. Glättung durch Filtern mit einem *Savitzky-Golay-Filter* [43].
8. Einlesen der Koordinaten in XFOIL. Reduktion auf 160 Punkte und neu verteilen anhand der Profil-Krümmung.

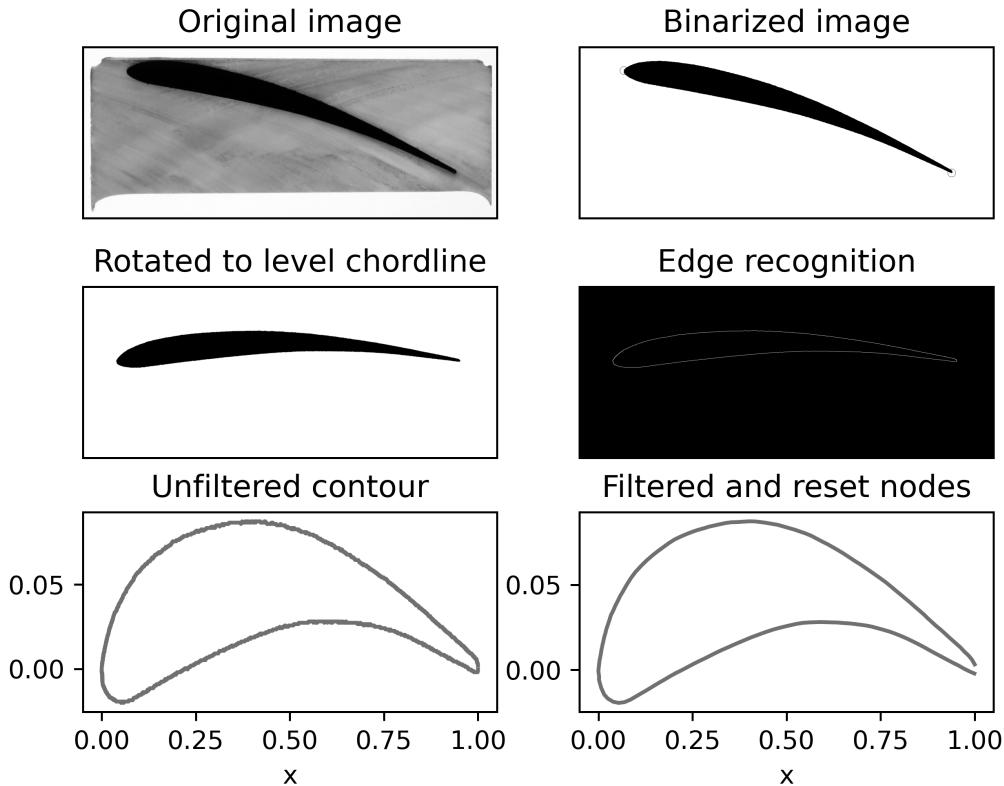


Abbildung 6.1.: Photogrammetrische Vermessung eines Querschnitts mit OPENCV.

Anschließend wurden mit XFOIL testweise einige Profilpolaren berechnet und festgestellt, dass mit den Profilkordinaten so keine sinnvollen Ergebnisse zu erzeugen waren: Durch noch vorhandene numerische Welligkeiten herrschte praktisch keine laminare Strömung vor, in den Druckverteilungs-Plots waren mehrere Ablöseblasen zu erkennen. Daher wurde von Extern eine Interpolation der Profile durch mathematisch stetige Funktionen durchgeführt<sup>1</sup>. Mit den interpolierten Profilkordinaten lieferte XFOIL plausible Druckverteilungen.

**Vermessungsergebnisse** Die ermittelten Daten von normierter Profiltiefe  $c/R$  und Steigungswinkel  $\beta$  sind in Abbildung 6.2 dargestellt. Der Propeller verfügt über einen einfachen Trapez-Grundriss mit zusätzlichen Schrägen an Nabe und Blattspitze.

Die Scans der Querschnitte ergaben, dass lediglich ein einziges aerodynamisches Profil verbaut ist, welches im Wurzelbereich auf einen Rechteckquerschnitt gestraakt wird. Der Straak beginnt kurz hinter der Einspannung bei  $r/R \approx 0,1$  und endet an der maximalen

<sup>1</sup>Zur Glättung von Profilkordinaten existieren mehrere wissenschaftliche Veröffentlichungen, anhand derer eine solche Funktion implementiert werden kann [25, 33]. Um Aufwand zu sparen, wurde die Glättung vom DLR-Institut für Aerodynamik und Strömungstechnik in Braunschweig mit einem dort vorhandenen Programm durchgeführt.

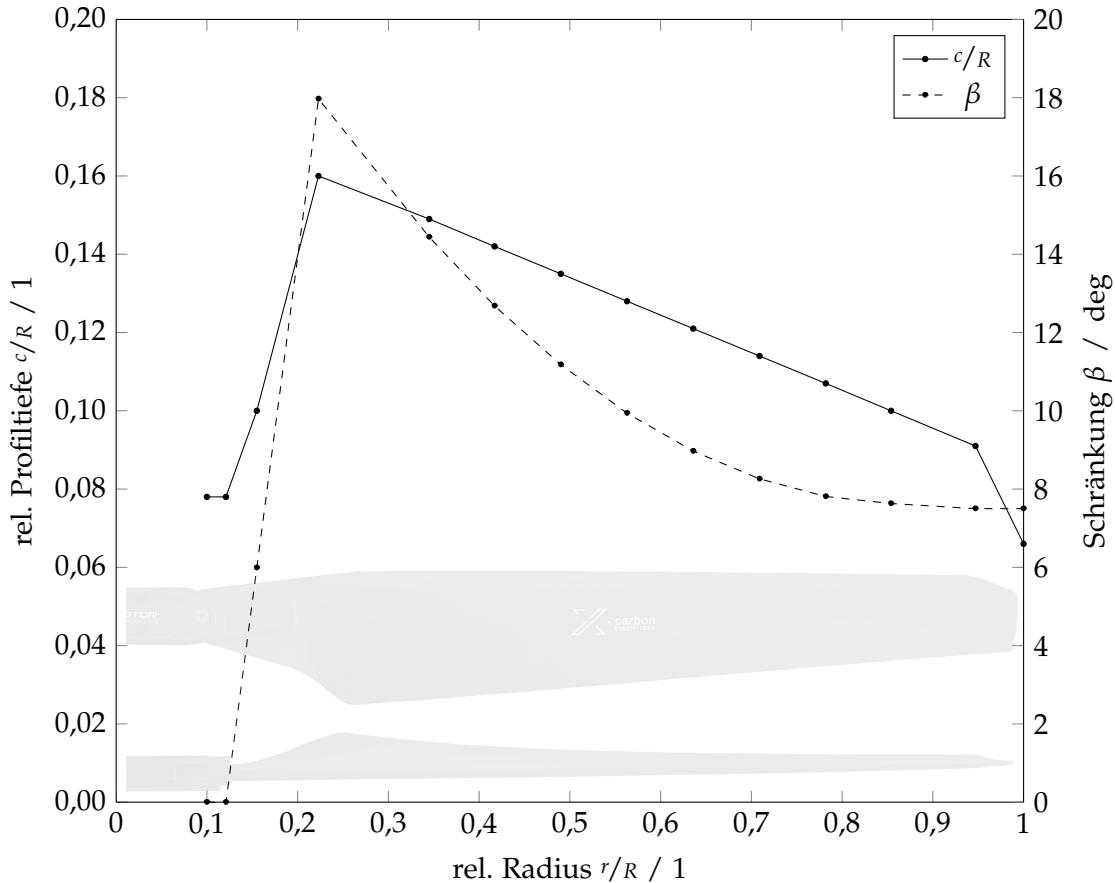


Abbildung 6.2.: Mittels Vermessung ermittelte Verteilung von Schränkung und Profiltiefe des Propellers MF3218 [42].

Profiltiefe bei  $r/R \approx 0,22$ . Das Profil (zu sehen in Abbildung 6.1) verfügt über eine relative Profildicke von  $d/c \approx 7,8\%$  und eine Dickenrücklage von  $x/c \approx 18\%$ .

## 6.2. Lastrechnung

Als dimensionierender Lastfall wurde der statische Startlastfall ausgewählt, dieser tritt bei Beginn des Startlaufs auf. Die Anströmgeschwindigkeit ist null und der Propeller wird mit maximaler Leistung und Drehzahl betrieben. BORST u. a. [4] gibt an, dass für diesen die größten statischen Lasten auftreten.

Der entwickelte Schub ist dabei der maximale Standschub, welcher im Datenblatt des Propellers angegeben ist. Anhand dessen können die Ergebnisse des Lastrechnungsprogramms auf Plausibilität überprüft werden.

Laut Datenblatt liegt die höchstzulässige Drehzahl des Propellers MF3218 bei  $4000 \text{ 1/m}$ . Mit einer Anströmgeschwindigkeit von  $0 \text{ m/s}$  und den atmosphärischen Daten der *Internationalen Standardatmosphäre* [15] liefert das Lastrechnungsprogramm folgende Daten:

- Schub: 175,5 N (Vgl. Datenblatt 177 N - Differenz  $\approx 1\%$ ).
- Querkraft  $F_z = 87,74\text{N}$  und Wurzelbiegemoment  $M_x = 25,09\text{Nm}$ .
- Querkraft  $F_x = 10,96\text{N}$  und Drehmoment  $M_z = 3,06\text{Nm}$ .

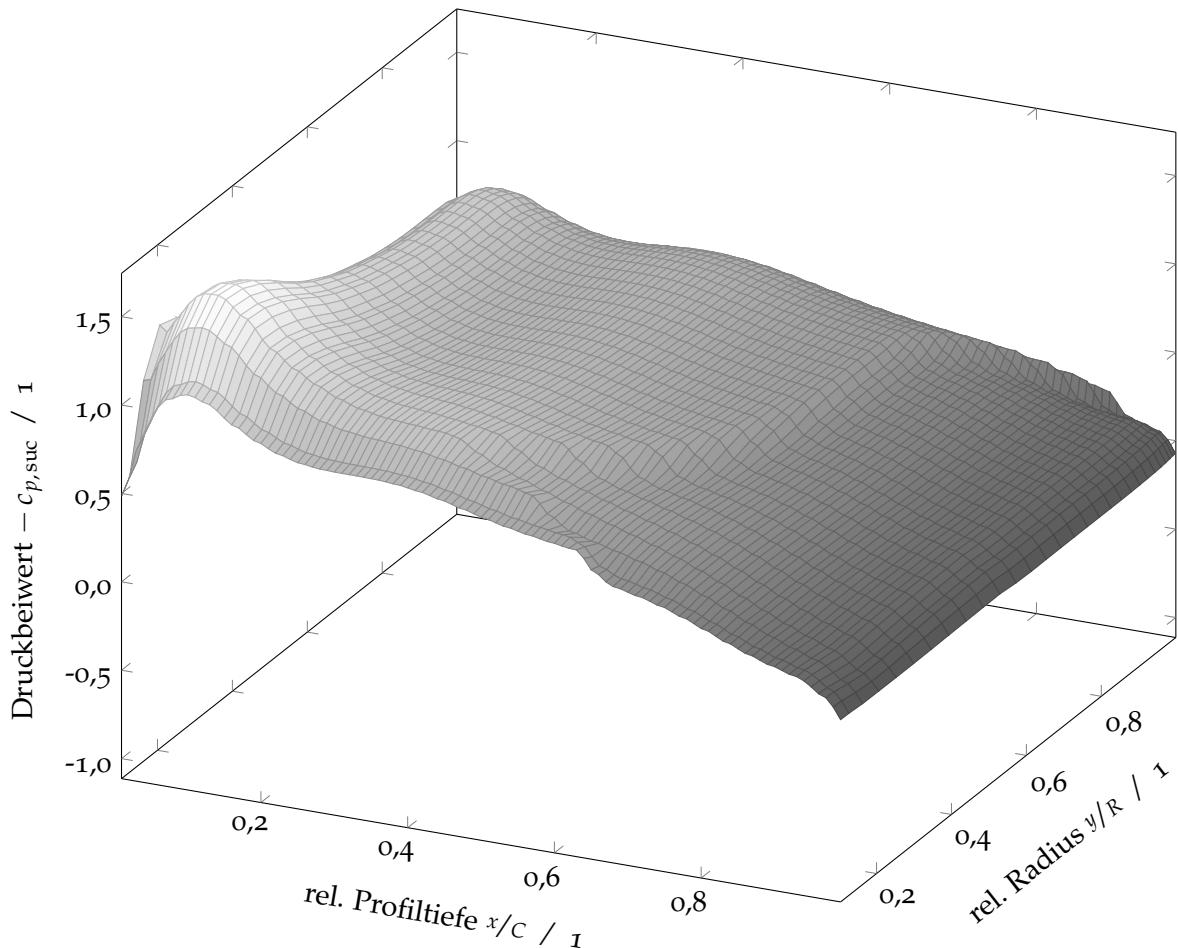


Abbildung 6.3.: Druckverteilung der Saugseite des Propellers für den definierten Lastfall.

Für die Eingabe in das FE-Modell werden mit dem Lastrechnungsprogramm die aerodynamischen Druckverteilungen des Propellers abgeleitet. In Abbildung 6.3 ist die Verteilung für die Saugseite (Profiloberseite) als Flächen-Plot dargestellt. Gut erkennbar sind die Saugspitze an der Profilnase sowie der laminar-turbulente Umschlag.

Diese Flächenplots können in Python interaktiv betrachtet werden und sollten vor der Übergabe der Lasten an das FE-Modell auf Plausibilität geprüft werden, da es möglich ist, dass Teile der XFOIL-Rechnung nicht konvergieren. Dies wäre an einer scharf ausgeprägten Saugspitze oder einer Delle im hinteren Staupunkt zu erkennen. Das Lastrechnungsprogramm gibt für Fälle möglicher Nicht-Konvergenz zusätzlich eine Warnung aus.

Auf dem hier dargestellten Plot sind die Saugspitzen sowie die Verbindungsline der hinteren Staupunkte ( $x/c = 1$ ) erkennbar glatt.

Die mechanischen Lasten durch die Rotation(-beschleunigung) werden direkt in ANSYS berechnet und aufgeprägt.

Das Eingabescript für die Lastrechnung ist auf der Daten-CD beigelegt.

### 6.3. Definition der Designvariablen

Im Rahmen einer vorangegangenen Arbeit von HILBERS [20] wurden Parameterstudien mit generischen Propellergeometrien und verschiedenen nachhaltigen Werkstoffen durchgeführt. Es wurde gezeigt, dass mit einer Kombination aus flachsfaserverstärkten Kunststoffen und einem Sandwichkern aus Balsaholz eine lastgerechte und Dimensionierung möglich ist. Daher soll im Rahmen dieses Abschnittes eine Sandwichstruktur für das Propellerblatt ausgelegt werden. Da das aus den aerodynamischen Lasten resultierende Torsionsmoment gering ist (dies wird im nächsten Abschnitt 6.4 gezeigt), sollen ausschließlich unidirektionale Lagen der selben Orientierung verwendet werden.

Es wird daher je eine Designvariable für die Orientierung des Faser- und des Stützstoffmaterials definiert.

- $\phi_0$ , Typ: 'p': Faserorientierung des Prepreg-Werkstoffs.
- $\phi_1$ , Typ: 'p': Faserorientierung des Stützstoffs.

Zusätzlich wird das Propellerblatt in  $j = 20$  Komponenten unterteilt, für diese soll die optimale Menge an Fasern und ihre Positionierung bestimmt werden. Dazu werden je Komponente  $j$  zwei Designvariablen definiert. Deren Konstruktion ist in Abbildung 6.4 veranschaulicht.

- $\rho_j$ , Typ: 'c',  $x^L : 0, x^U : 1$ : Anteil des FvK-Querschnitts an der Gesamt-Querschnittsfläche je Element. Der verbleibende Rest des Querschnitts wird mit Stützstoff aufgefüllt.
- $div_j$ , Typ: 'c',  $x^L : 0, x^U : 1$ : Positionierung des FvK-Querschnitts: Der Wert von  $div_j$  entspricht dem Anteil der Fasern auf der Oberseite des Elements, der verbleibende Rest wird entsprechend auf der Unterseite positioniert.

Durch diesen Variablensetz soll der Optimierer die Fasermenge so einstellen können, dass die Materialausnutzung maximiert wird. Zusätzlich gibt es über den Freiheitsgrad  $div$  die Möglichkeit, eine asymmetrische Verteilung des Faserquerschnitts vorzusehen. Da Faserverbundwerkstoffe weniger druckfest als zugfest sind, könnte durch Verschiebung des Faseranteils in Richtung der Druckseite eine Gewichtseinsparung ermöglicht werden.

Bei  $n\_sec = 20$  Komponenten verfügt das Modell so über 42 Designvariablen.

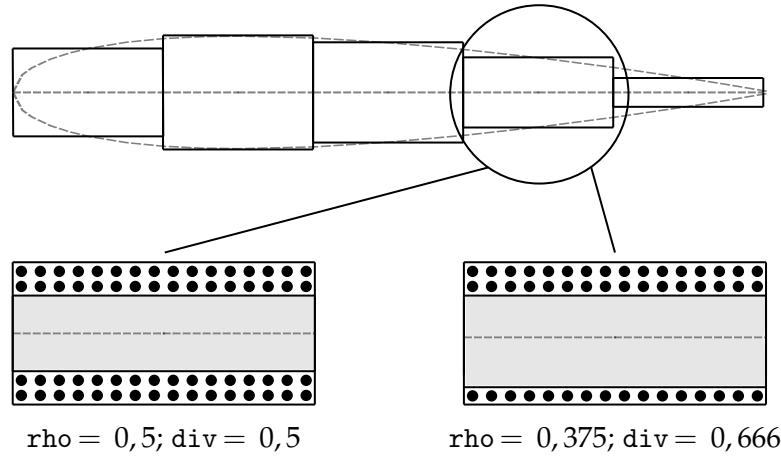


Abbildung 6.4.: Veranschaulichung der Designvariablen der Komponenten.

## 6.4. Konfiguration des Analysemodells

**Zuweisung des Lagenaufbaus** Die Zuweisung des Lagenaufbaus erfolgt mit der in Unterabschnitt 5.3.1 beschriebenen Methodik anhand der zuvor definierten Designvariablen. Dabei werden angrenzende, identische Lagen nicht einzeln definiert, sondern als „Ersatzlagenpaket“ mit einer aus den jeweiligen Designvariablen berechneten Gesamtdicke zusammengefasst. Dies hat den Vorteil, dass innerhalb der Optimierungsschleife bei der Variation der Designvariablen und beim Post-Processing Rechenaufwand eingespart wird, da dort über weniger Lagen iteriert werden muss.

Zusätzlich werden einige, konstruktive Randbedingungen implementiert:

- Für die Prepreg-Ersatzlagenpakete wird immer mindestens die Dicke einer Einzellage (für FLAXPREG T-ID: 0,185 mm) zugewiesen, selbst wenn die aus den Designvariablen berechnete Dicke geringer wäre.  
So wird sichergestellt, dass der Sandwichkern immer von mindestens einer Lage umhüllt ist, und dass eine ausreichende Stabilität der Endleiste durch mindestens zwei Faser-Lagen gewährleistet wird.
- Die Materialstärke des Sandwichkerns beträgt immer mindestens 1 mm. Wenn für ein Element aus den Werten der Designvariablen ein geringeres Maß resultiert, wird stattdessen ein Prepreg-Volllaminat zugewiesen.  
Dies beruht auf der Annahme, bei der Fertigung des Sandwichkerns nur Wandstärken mit einem gewissen Mindestmaß zu realisieren sind.

**Lastaufprägung und Validierung** Die Aufprägung der aerodynamischen Lasten erfolgt mit den in Unterabschnitt 5.3.2 beschriebenen Methoden. Um deren Gültigkeit zu prüfen, können die Reaktionskräfte des FE-Modells mit den durch das Lastrechnungsprogramm berechneten Verläufen verglichen werden.

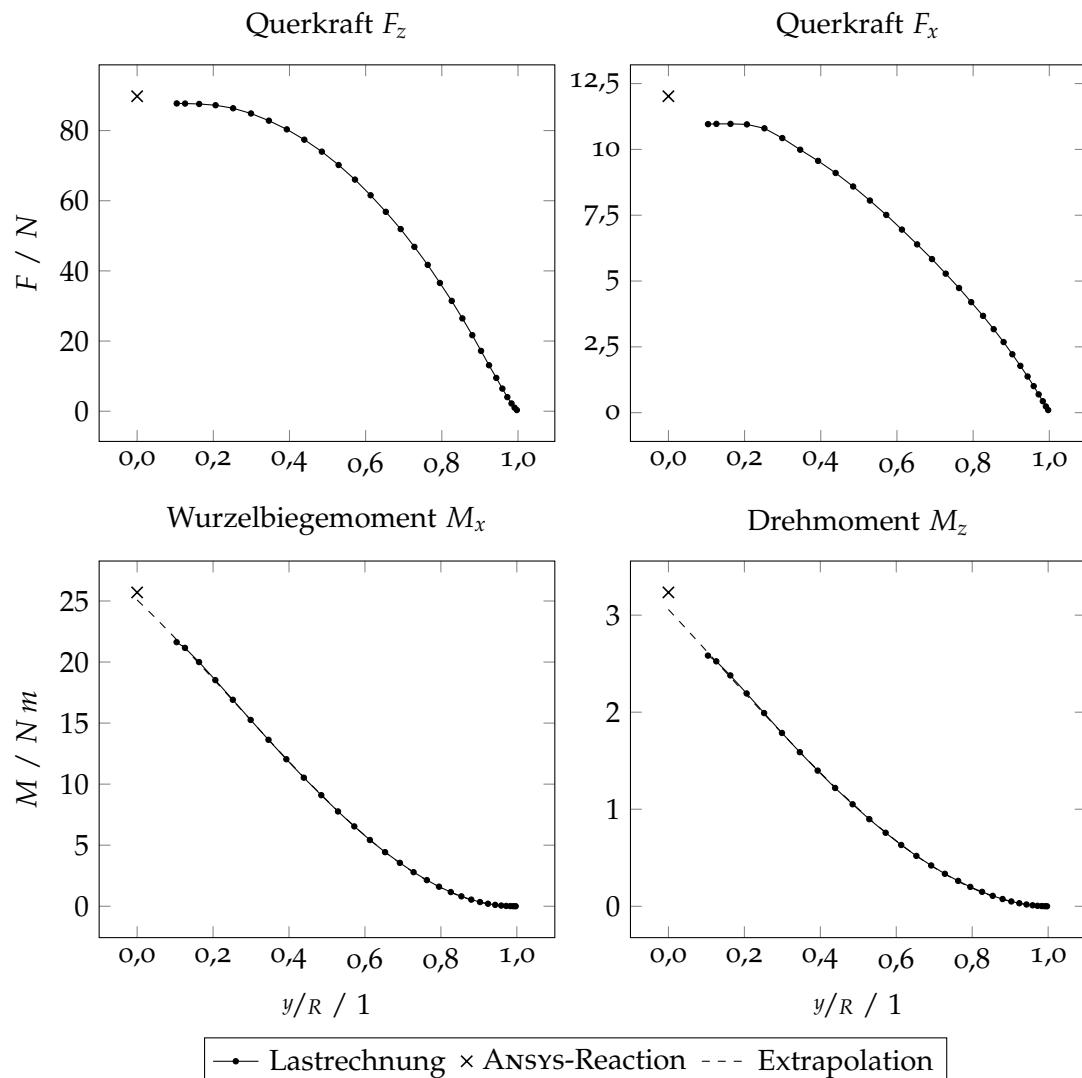
Abbildung 6.5 zeigt dafür die Querkraft- und Momentenverläufe aus der Lastrechnung sowie die auf den Koordinatenursprung bezogenen Auflagerkräfte des FE-Modells. Da die durch XROTOR berechneten Momentenverläufe erst bei  $r/R \approx 0,1$  beginnen, wurden diese bis  $r/R = 0$  extrapoliert.

- $F_z$  und  $M_x$  sind die Schnittreaktionen resultierend aus dem Propeller-Schub, sie sind in der linken Spalte der Abbildung dargestellt. Im FE-Modell entstehen an der Wurzel mit < 2,4 % geringfügig größere Schnittlasten, die Aufprägung ist also konservativ und wird daher als valide angesehen.
- $F_x$  und  $M_z$  sind die Schnittreaktionen aus dem Propeller-Drehmoment, sie sind in der rechten Spalte dargestellt. Auch hier sind die Werte des FE-Modells größer, also auf der konservativen Seite. Die Differenzen sind mit  $\Delta F_x \approx 9,7\%$  bzw.  $\Delta M_z \approx 5,6\%$  signifikant höher als in der Schubrichtung.  
Der größte Faktor dafür ist vermutlich die Idealisierung des Propellers durch Schalen-elemente: Die aerodynamischen Kräfte werden als *Surface Pressure* auf die Mittenfläche der Profilquerschnitte aufgebracht, und nicht, wie real, getrennt auf Ober- und Unterseite. Wegen der unterschiedlichen Krümmung resultiert daraus ein „Näherungsfehler“. Dass dieser Fehler für  $F_x$  größer ausfällt als für  $F_z$  ist ebenfalls plausibel und ergibt sich aus der Kleinwinkelnäherung ( $\cos(\alpha) \approx 1$  vs.  $\sin(\alpha) \approx \alpha$ ).
- Die Höhe des Torsionsmoments  $M_y$  hängt davon ab, wie die Profilquerschnitte relativ zur Blattlängsachse positioniert sind. Bei dem untersuchten Propeller *MF3218* liegt diese bei  $x/c \approx 0,2$ . XROTOR berechnet die Momentenverläufe hingegen mit  $x/c \approx 0,3$  [12], weshalb eine Validierung hier nicht möglich ist.
- Die aerodynamische Normalkraft  $F_y$  ist mit < 1 N im Vergleich zu den Zentrifugalkräften mit  $\approx 2$  kN gering.

Die Zentrifugalkräfte werden über mit dem Ansys-APDL-Befehl **OMEGA** definiert, der eine Rotation des Bauteils um den Koordinatenursprung simuliert. Entsprechend der Drehzahl von  $f = 4000\text{ 1/min}$  wird eine Winkelgeschwindigkeit von

$$\omega = 4000\text{ 1/min} \cdot 1/60\text{ s} \cdot 2\pi \approx 418,9\text{ rad/s}$$

zugewiesen.



	Lastrechnung	ANSYS	Differenz
$F_x$	10,96 N	12,02 N	9,7 %
$F_y$	0,67 N	0,61 N	9,8 %
$F_z$	87,74 N	89,79 N	2,3 %
$M_x$	25,09 Nm	25,70 Nm	2,4 %
$M_y$		0,95 Nm	
$M_z$	3,06 Nm	3,23 Nm	5,6 %

Abbildung 6.5.: Validierung der Lastaufprägung des FE-Modells: Vergleich der Reaktionskräfte- und Momente

## 6.5. Einstellung der Optimierungs-Parameter

Mit dem zuvor definierten Analysemodell und der in Unterabschnitt 5.4.2 beschriebenen Zielfunktion und Restriktionen ist das Optimierungsproblem vollständig definiert, sodass erste Testläufe der Optimierungsroutine durchgeführt werden können. Dazu wurde ein Desktop-PC mit *Intel Core i5-9400F* 6-Kern Prozessor ohne Hyperthreading verwendet, es konnten also sechs Funktionsauswertungen des Analysemodells parallel ausgeführt werden. Die Berechnungszeit pro Auswertung betrug etwa 0,65 s.

Bei der Optimierung der Testprobleme in Abschnitt 3.4 konvergierte die Schleife mit den *Default*-Prozessparametern

- Schwarmgröße  $P_s = 40$  Partikel
- Kognitiver Parameter  $c_1 = 1$  und sozialer Parameter  $c_2 = 2$
- Anfangs-Trägheit  $w_1 = 0,99$  und End-Trägheit  $w_2 = 0,55$

in hinreichend kurzer Zeit, sodass diese nicht verändert werden mussten. Die Optimierung des UAV-Propellers ist mit 42 Designvariablen jedoch komplexer.

Um die Geschwindigkeit des Verfahrens zu erhöhen, wird zunächst die Populationsgröße angepasst. PIOTROWSKI u. a. [32] haben für verschiedene PSO-Algorithmen den Einfluss der Population auf die Konvergenzgeschwindigkeit untersucht. Sie nennen als Optimum für Probleme im Bereich mit 10-100 Designvariablen einen Richtwert von 70-100 Partikeln. Zusätzlich gilt als Randbedingung für die parallele Funktionsauswertung, dass die Schwarmgröße ganzzahlig durch die Anzahl der Prozessor-Kerne teilbar sein sollte [22].

Aufgrund dieser Überlegungen wurde für die folgenden Berechnungen eine Populationsgröße von 84 Partikeln verwendet.

Pro äußerer Iteration des Algorithmus werden (mit  $k_{max} = 6$ )  $6 \cdot 84 = 504$  Funktionsauswertungen durchgeführt<sup>2</sup>. Dies entspricht einer Dauer von ca. 5,5 min. Mit einer maximalen Zahl bis zum Abbruch von 200 äußeren Iterationen beträgt die Lösungszeit bis zu ca. 18,5 h.

Um die restlichen Optimierungsparameter so einzustellen, dass die Optimierungsroutine innerhalb einer ausreichend geringen Zeitspanne konvergiert, wurden einige Testläufe durchgeführt und die Parameter auf Basis der vorigen Läufe angepasst.

Abbildung 6.6 zeigt für alle durchgeführten Läufe die besten gefundenen Funktionswerte  $f(\vec{g}^k)$  über der Iterationszahl  $k$ .

Zunächst wurden als kognitiver, sozialer Parameter und Trägheit die anfangs genannten Voreinstellungen verwendet. Den zugehörigen Optimierungsverlauf zeigt die erste Kurve. Es ist zu sehen, dass die Optimierung sehr schnell bei einem hohen Funktionswert konvergiert (innerhalb von 20 Iterationen), sich aber nach der anfänglichen Konvergenz nicht mehr verbessert. Dies deutet darauf hin, dass der Algorithmus ein lokales Suboptimum erreicht und nicht in der Lage ist, daraus zu entkommen. Laut JANSEN und PEREZ [22] tritt ein solcher Verlauf oft aufgrund eines zu starken Informationsaustauschs der Partikel auf.

---

<sup>2</sup> $k_{max}$ : Anzahl der inneren Iterationen bei der Lösung des unbeschränkten Ersatzproblems. Siehe Abschnitt 3.3.

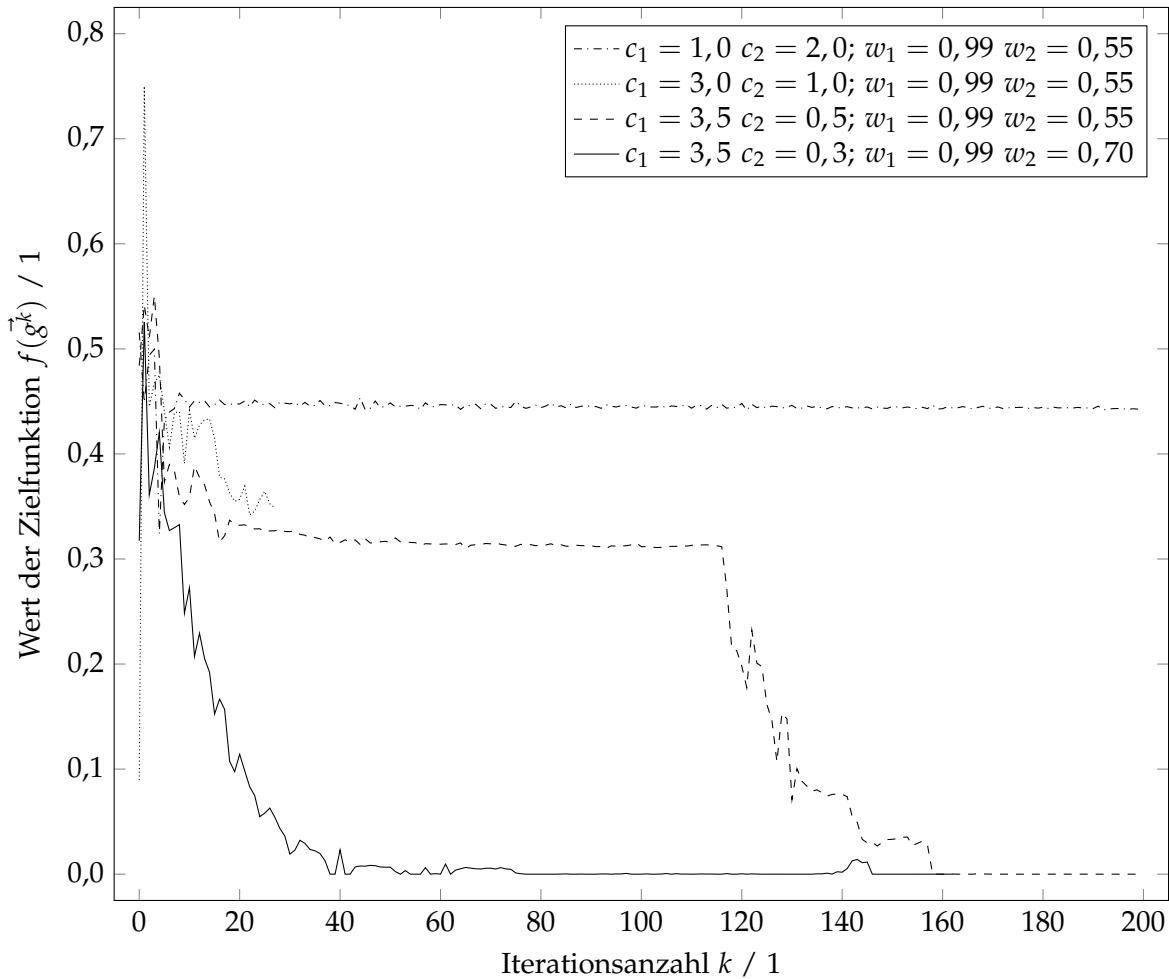


Abbildung 6.6.: Konvergenzverhalten des Optimierungsproblems in Abhangigkeit verschiedener Prozessparameter.

Daher wurde der nachste Lauf mit einem geringeren sozialen Parameter  $c_2 = 1$  und erhohtem kognitiven Wert  $c_1 = 3$  durchgefuhrt. Nach etwa 20 Iterationen begann sich wieder eine vorzeitige Konvergenz einzustellen, sodass die Rechnung nach der 27. Iteration abgebrochen wurde.

Fur den dritten Lauf wurde die Gewichtung der Parameter weiter in die kognitive Richtung verschoben:  $c_1 = 3,5$  und  $c_2 = 0,5$ . Wie anhand der Kurve zu sehen ist, stellt sich wieder eine vorzeitige Konvergenz ein. Der Algorithmus ist jedoch nun in der Lage, aus dem lokalen Optimum zu entkommen: Hier zeigt sich die Wirkung der stochastischen Komponente in der Partikel-Bewegungsgleichung 3.2, aufgrund einer zufalligen Variation in Richtung eines geringeren Funktionswerts kann der Algorithmus bei Iteration 117 das Suboptimum verlassen und erreicht schlielich nach etwa 160 Iterationen ein globales Minimum.

Um die Konvergenzgeschwindigkeit weiter zu erhohen wurde der kognitive Parameter auf  $c_2 = 0,3$  leicht verringert.

Zusätzlich wurde die Trägheit der Partikel angepasst. Der ALPSO-Algorithmus verfügt über eine adaptive Steuerung des Trägheitsparameters  $w$ . Dieser startet bei  $w_1 = 0,99$  und wird mit fortlaufender Iterationszahl schrittweise in Richtung des Werts  $w_2$  verringert. So soll erreicht werden, dass die Partikel zu Beginn des Optimierungsverlaufs durch eine hohe Trägheit nicht von lokalen Minima zum Stillstand gebracht werden, und gegen Ende der Optimierung dank einer geringeren Trägheit trotzdem feinfühlig auf Änderungen der Funktionswerte reagieren können.

Um die Sensitivität der Partikel auf lokale Optima im mittleren Optimierungsverlauf etwas zu verringern wurde die End-Trägheit  $w_2$  auf 0,7 leicht erhöht.

Als Optimierungsparameter ergeben sich so:

- Schwarmgröße  $P_s = 84$  Partikel
- Kognitiver Parameter  $c_1 = 3,5$  und sozialer Parameter  $c_2 = 0,3$
- Anfangs-Trägheit  $w_1 = 0,99$  und End-Trägheit  $w_2 = 0,70$

Der resultierende Optimierungsverlauf ist in der vierten Kurve von Abbildung 6.6 zu sehen. Sie zeigt eine ungestörte Konvergenz nach etwa 75 Iterationen, dies entspricht einer Rechendauer von ca. sieben Stunden. Da die Zielfunktion eine Betragsfunktion ist, muss das gefundene Minimum mit  $f(\vec{x}^*) = 0,0$  ein globales sein.

Die mit der hier angewendeten, heuristischen Vorgehensweise gefundenen Einstellungen liefern aller Wahrscheinlichkeit nach nicht die schnellstmögliche Konvergenz, sodass mit einer systematischen Parameterstudie eine weitere Verbesserung zu erwarten wäre.

Der dazu notwendige zeitliche Aufwand würde jedoch den zu erwartenden Nutzen deutlich überwiegen, sodass im Rahmen dieser Arbeit ausdrücklich darauf verzichtet wird<sup>3</sup>.

## 6.6. Optimierung mit Beta-Zielfunktion

### 6.6.1. Optimierungsproblem

Wie in Abschnitt 6.3 erläutert, wurde der UAV-Propeller in  $n\_sec = 20$  Komponenten unterteilt, deren Lagenaufbau optimiert werden soll. Damit verfügt das Analysemodell über 42 Designvariablen. Die Menge an Freiheitsgraden in der Optimierungsroutine kann jedoch weiter verringert werden: Alle Variablen, deren optimale Werte „trivial“, also sofort ersichtlich sind, werden bereits vor Beginn der Optimierung festgelegt.

Dies sind die Designvariablen der äußeren sechs Komponenten im Blattspitzenbereich. Für sie konnte durch eine Vorstudie ermittelt werden, dass mit der geringsten zugelassenen Fasermenge von je einer Lage pro Seite keine vollständige Materialauslastung erreicht werden kann. Daher werden folgende Werte zugewiesen:

$$\text{rho\_j} = 0,0; \quad j = 14, \dots, 19 \quad (6.1)$$

$$\text{div\_j} = 0,5; \quad j = 14, \dots, 19 \quad (6.2)$$

<sup>3</sup>Zum Beispiel müssten bei vier zu untersuchenden Parametern a drei Varianten  $4^3 = 64$  Rechnungen durchgeführt werden. Bei einer Dauer von 18 Stunden je Rechnung betrüge die Rechenzeit etwa zwei Monate.

Die Anzahl an Designvariablen beträgt so nunmehr 28.

Die Formulierung der Zielfunktion erfolgt, wie in Unterabschnitt 5.4.2 beschrieben, mit der Beta-Methode und einem Zielwert von  $I_{ref} = 1$ . Die zusätzlichen Restriktionen der Beta-Methode (Gleichung 5.13) werden für  $n\_sec = 1, \dots, 13$  gesetzt. Die Komponente  $n\_sec = 0$  wird nicht berücksichtigt, da für diese Randeffekte und Singularitäten durch die Einspannung zu erwarten sind, welche den Optimierungsprozess nicht beeinträchtigen sollen.

Das Optimierungsproblem lautet zusammengefasst:

$$\min f(\beta) \quad (6.3)$$

$$= \min |\beta - 1| \quad (6.4)$$

sodass:

$$g_{\text{beta}, j}(\vec{x}) = \begin{cases} I_{\text{fib}, j} - \beta; & \text{für } I_j \geq 1 \\ \beta - I_{\text{fib}, j}; & \text{für } I_j < 1 \end{cases} \leq 0; \quad j = 1, \dots, 13 \quad (6.5)$$

$$g_{\text{fib}, j}(\vec{x}) = I_{\text{fib}, j} - 1 \leq 0; \quad j = 0, \dots, 19 \quad (6.6)$$

$$g_{\text{mat}, j}(\vec{x}) = I_{\text{mat}, j} - 1 \leq 0; \quad j = 0, \dots, 19 \quad (6.7)$$

mit:

$$\vec{x} = (\text{phi\_0}, \text{phi\_1}, \text{rho\_0}, \text{div\_0}, \dots, \text{rho\_13}, \text{div\_13}) \quad (6.8)$$

$$0^\circ \leq \text{phi\_i} \leq 180^\circ; \quad i = 0, 1 \quad (6.9)$$

$$0 \leq \text{rho\_i} \leq 1; \quad i = 1, \dots, 13 \quad (6.10)$$

$$0,3 \leq \text{div\_i} \leq 0,7; \quad i = 1, \dots, 13 \quad (6.11)$$

## 6.6.2. Prozessverlauf

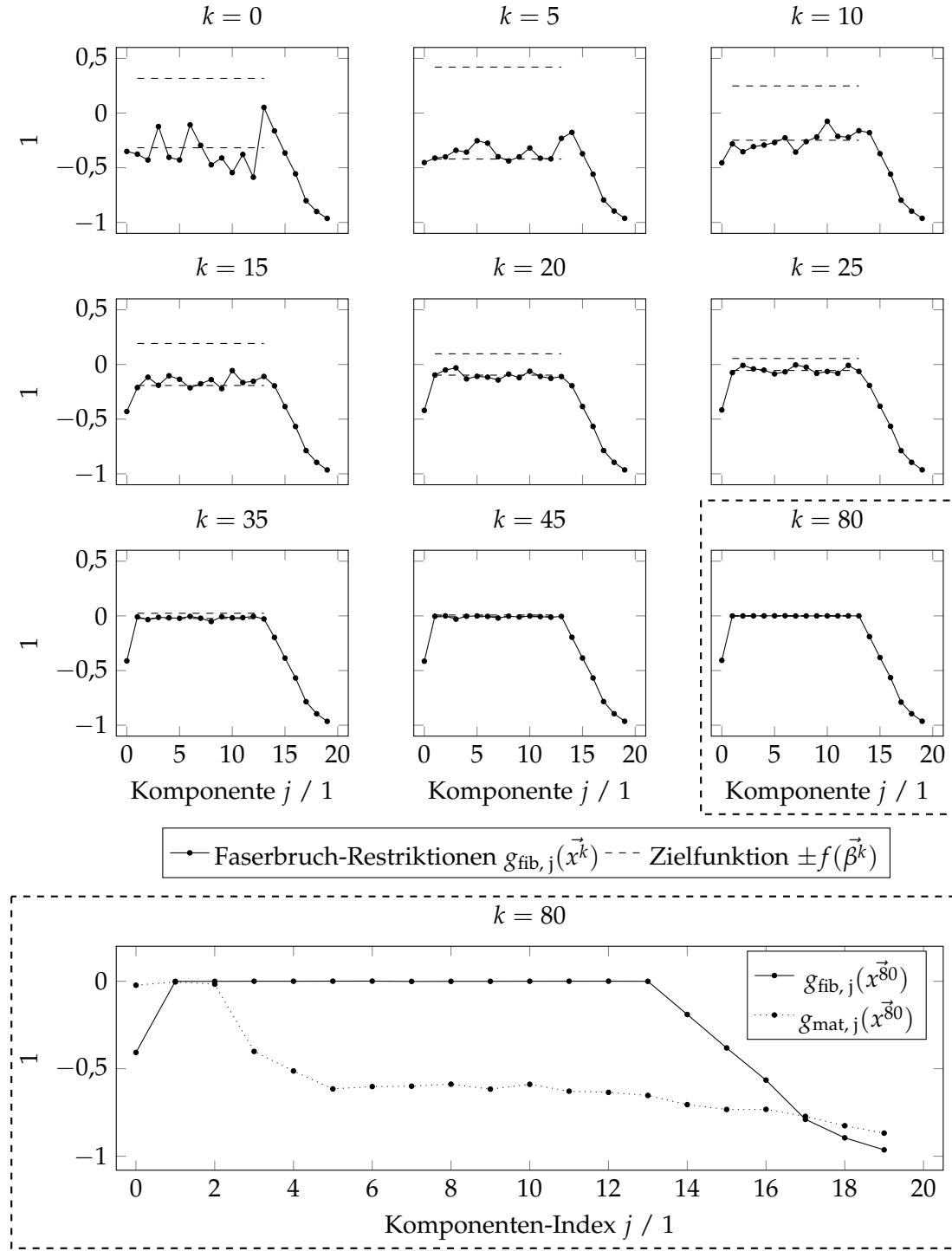
Der Verlauf des Optimierungsprozesses ist in Abbildung 6.7 anhand einiger, ausgewählter Iterationsschritte dargestellt.

Die Teilabbildungen zeigen jeweils für eine Iteration  $k$  die Faserbruch-Restriktionen  $g_{\text{fib}, n\_sec}(\vec{x}^k)$  aller Bauteilkomponenten  $n\_sec$  des besten Partikels<sup>4</sup>. Zusätzlich ist mit den gestrichelten Linien der Zielfunktionswert  $\pm f(\beta^k)$  aufgetragen, jeweils mit positivem und negativem Vorzeichen<sup>5</sup>. Das Intervall zwischen den beiden Linien ist damit der Bereich, in dem die Restriktion  $g_{\text{beta}, j}(\vec{x})$  aus Gleichung 6.5 kleiner oder gleich Null, also erfüllt und inaktiv ist.

---

<sup>4</sup>Also des Partikels  $\vec{x}_i^k$  aus Gleichung 3.2 und Abbildung 3.1

<sup>5</sup>Die Darstellung ist identisch wie bei der in Abbildung 5.10 veranschaulichten Herleitung der Beta-Methode.

Abbildung 6.7.: Verlauf von Zielfunktion und Restriktionen für verschiedene Iterationsschritte  $k$ .

Für die initiale Iteration  $k = 0$  liegen die Faserbruch-Versagenskriterien teilweise außerhalb dieses zulässigen Bereichs. Die entsprechenden Restriktionen sind also verletzt. Der Algorithmus erhöht daher für die folgenden Iterationen die Pseudo-Zielfunktion  $\mathcal{L}$  und den Wert von  $\beta$ , sodass sich die Partikel in Richtung des zulässigen Bereichs bewegen<sup>6</sup>.

Daraufhin wird der Zustand bei  $k = 5$  erreicht, indem nahezu alle Restriktionen erfüllt sind, und die Zielfunktion deshalb kaum bestraft werden muss. Aufgrund der Beta-Methode kann die Zielfunktion nun ausschließlich dann weiter minimiert werden, wenn gleichzeitig eine Erhöhung der Anstrengungen im Bauteil einhergeht.

Dies führt mit fortschreiten des Algorithmus ( $k = 10, 15 \dots$ ) dazu, dass die Faserbruch-Restriktionen durch die untere Beta-Linie „angehoben“ werden.

Ab einer Schwelle von etwa  $k = 80$  ist der Algorithmus konvergiert. Die Zielfunktion beträgt dort  $f(\vec{\beta}^{80}) \approx 4 \cdot 10^{-5} \approx 0$  und alle Restriktionen sind erfüllt (also  $\leq 0$ ), wie in der vergrößerten Ansicht zu sehen ist. Da die Zielfunktion nicht negativ sein kann, ist außerdem sichergestellt, dass das der gefundene Vektor  $\vec{x}^* = \vec{x}^{80}$  ein globales Minimum der Zielfunktion ist.

Bis zur Konvergenz bei  $k = 80$  wurden  $(k \cdot k_{max} + 1) \cdot P_s = 40404$  Funktionsauswertungen des Analysemodells durchgeführt. Bei einer mittleren Dauer von 0,65 s je Auswertung betrug die Berechnungszeit etwa 7,3 h.

### 6.6.3. Ergebnisse

Nachdem im vorigen Unterabschnitt 6.6.2 bereits die Werte von Zielfunktion Restriktionen im Optimum diskutiert wurden, soll nun der entsprechende Lösungsvektor  $\vec{x}^*$  dargestellt werden, welcher die Zielfunktion minimiert. Für die Faserwinkel hat der Algorithmus als optimale Werte ermittelt:

$$\begin{aligned} \text{phi\_0} &= 93,84^\circ; \quad \text{phi\_1} = 97,76^\circ \\ (90,0^\circ \text{ entspricht Orientierung parallel zu Blatt-Längsachse}) \end{aligned}$$

Der Algorithmus empfiehlt also eine leicht assymetrische Faserorientierung. Die Werte der weiteren Designvariablen  $\rho_j$  und  $d_j$  für alle Komponenten des Analysemodells sind in Abbildung 6.8 graphisch dargestellt.

---

<sup>6</sup>Das Anpassen der Pseudo-Zielfunktion geschieht über die Faktoren  $\theta_j$  und  $r_{p,j}$  aus Gleichung 3.3.

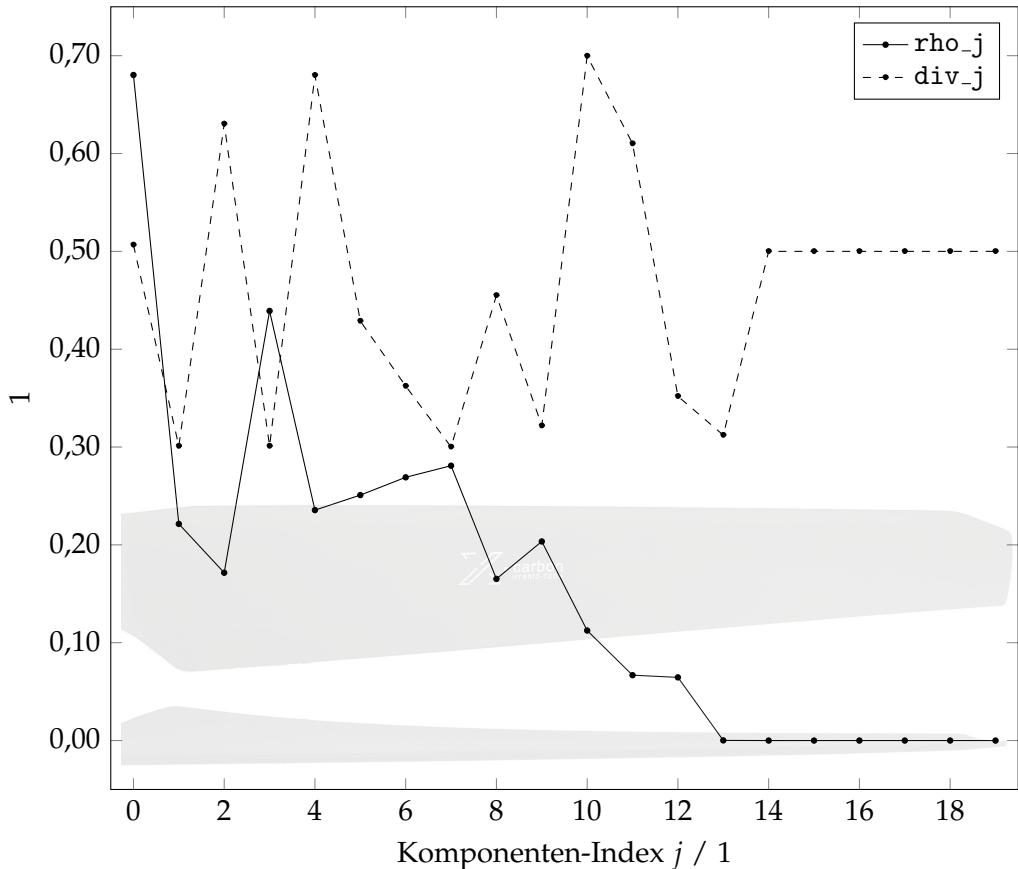


Abbildung 6.8.: Werte der Designvariablen  $\rho_j$  und  $\text{div}_j$  im Optimum  $\vec{x}^* = \vec{x}^{80}$ .

Insgesamt sind folgende Effekte zu erkennen:

- Die Orientierungen der Fasern ( $\phi_0$  und  $\phi_1$ ) weichen signifikant von der Blattlängsachse ab.
- Die gefundenen Werte für Faserdichte und Asymmetriefaktor  $\text{div}_j$  ergeben keinen glatten Verlauf, sondern unterliegen deutlichen Schwankungen.
- Die Schwankungen der Kurven scheinen zu koppeln: Ein erhöhter Wert für  $\text{div}_j$  geht mit niedrigem  $\rho_j$  einher und umgekehrt.

Bei einer analytischen Auslegung wäre vermutlich eine Faserorientierung parallel zur Blattlängsachse, ein konstanter Assymmetriefaktor  $\text{div}_j$  sowie ein glatter, von der Wurzel zur Blattspitze monoton abfallender Verlauf für  $\rho_j$  gewählt worden.

Die Optimierungsergebnisse wirken also zunächst kontraintuitiv. Es stellt sich daher die Frage, ob die aufgezählten Effekte physikalischen Ursprungs oder numerische Artefakte sind. Zur Beantwortung dieser Frage ist eine weiterführende Analyse des Optimierungsergebnisses notwendig. Die entsprechende Methodik wird im folgenden Abschnitt bereitgestellt.

## 6.7. Weiterführende Auswertung mit Python und JupyterLab

Durch die im vorigen Abschnitt vorgestellten Optimierungsergebnisse wird die grundlegende Notwendigkeit einer kritischen Prüfung deutlich. Eine Möglichkeit dazu ist es, einzelne Designvariablen zu verändern und den Effekt der Variation auf die Ausgabewerte des Analysemodells zu bewerten.

### 6.7.1. Methodik

Um diesen Nachlauf-Analyseprozess effizient und mit geringem Zeitaufwand durchführen zu können, wurde eine graphische Benutzeroberfläche für das Analysemodell implementiert. Dazu wurde die Software JUPYTERLAB verwendet. Diese stellt eine web-basierte, interaktive Umgebung zur Verfügung, die aus einer Liste von Ein- und Ausgabezellen besteht. Die Zellen können entweder Text oder Python-Code enthalten. Zellen mit Programmcode können wiederholt und in beliebiger Reihenfolge ausgeführt werden.

Zudem können über spezielle Python-Bibliotheken interaktive, graphische Elemente zur Ausführung von Codezellen hinzugefügt werden, wie Eingabefelder, Schieberegler, Buttons und viele mehr.

Das erstellte Programm liest die Output-Datei der Optimierungsroutine ein, welche für alle Iterationsschritte  $k$  die Werte der Designvariablen enthält.

Auf Basis dessen können für jede beliebige Iteration  $k$  alle Designvariablen mit Schieberegbern oder Textfeldern zur Eingabe neuer Werte verändert werden. Für den angepassten Variablensatz kann dann mit PyMAPDL und den in Kapitel 5 beschriebenen Objekten eine ANSYS-Rechnung und ein interaktives Post-Processing durchgeführt werden.

Abbildung 6.9 zeigt einen Screenshot der graphischen Benutzeroberfläche des Analysemodells. Die Schieberegler und Eingabefelder, teilweise im oberen Teil zu sehen, dienen zur Anpassung der Designvariablen des Modells. Über einen Button wird eine FE-Berechnung des Modells ausgelöst, die Restriktionen werden anschließend ausgewertet und zusammen mit den Designvariablen im aus den vorigen Abbildungen 6.7 und 6.8 bekannten Format dargestellt.

Aufgrund der geringen Berechnungsdauer von etwa einer Sekunde können so sehr effizient Parameterstudien ausgeführt werden.

Zusätzlich können mittels weniger Klicks Daten aus dem ANSYS-Postprocessor abgeleitet und auf eine interaktive Ansicht des Netzes projiziert werden. So können z.B. die Versagenskriterien für alle Elemente und Einzellagen graphisch ausgewertet werden, was in Abbildung 6.10 zu sehen ist.

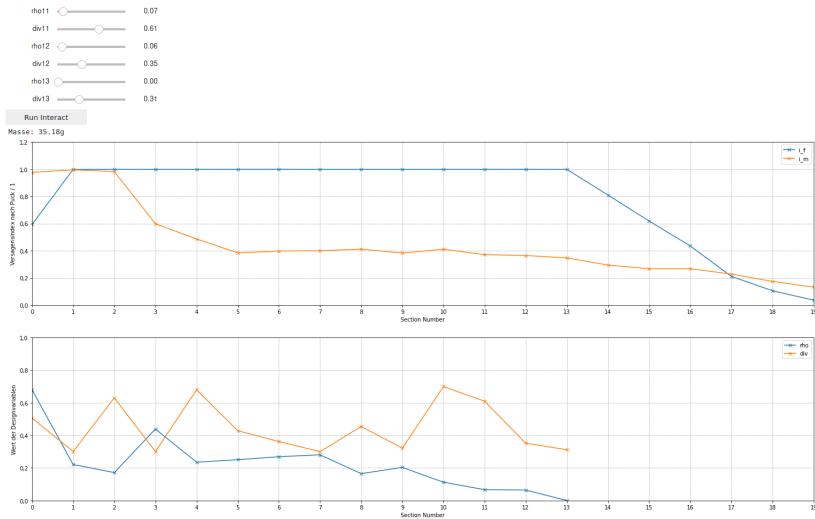


Abbildung 6.9.: Interaktive Auswertung der Restriktionen mit JUPYTERLAB.

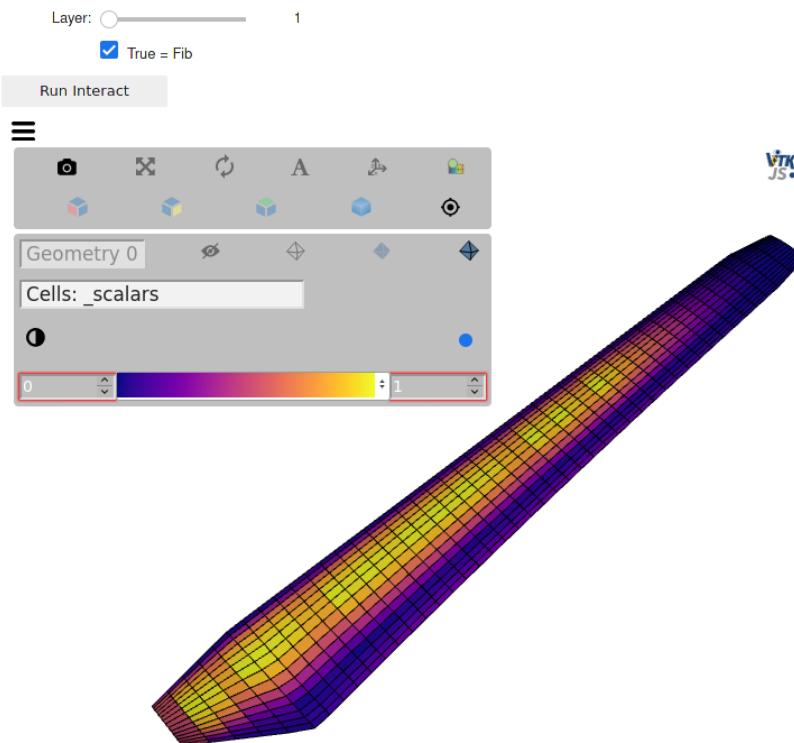


Abbildung 6.10.: Interaktive Auswertung der Versagenskriterien mit JUPYTERLAB.

### 6.7.2. Diskussion der Ergebnisse und Post-Processing

In Unterabschnitt 6.6.3 wurden die Optimierungsergebnisse kurz dargestellt. Diese sollen folgend mit der zuvor vorgestellten Methodik ausgewertet und diskutiert werden.

In Abbildung 6.11 sind dazu die Ergebnisse des Optimierers links, sowie eine mit Hilfe des JUPYTERLAB-Scripts angepasste Auslegung rechts zu sehen.

Um die Herkunft der unregelmäßigen Designvariablen-Verläufe zu untersuchen wurde zunächst der Wert aller Assymetrie-Variablen auf  $\text{div\_j} = 0,6$  gesetzt, also auf eine leichte Übergewichtung der druckbelasteten Blattoberseite. Dabei wurde beobachtet, dass die Belastung auf Faserbruch flächig um etwa 10 – 20 % abnahm.

Als Folge dessen konnten die Fasermengen aller Querschnitte über die  $\text{rho\_j}$ -Variablen etwas verringert werden, was eine Massenreduzierung von ursprünglich 35,18 g auf 31,3 g mit sich bringt.

Zudem konnte festgestellt werden, dass für die Dimensionierung auf Zwischenfaserbruch im Bereich des Blattanschlusses (Komponenten 0,1), die zugbelasteten Lagen kritisch sind. Daher wurde mit  $\text{div\_0}=\text{div\_1}=0,4$  dort etwas mehr Material angehäuft.

Das Ergebnis dieser Nachbearbeitung ist in Abbildung 6.11 rechts dargestellt. Es ist zu sehen, dass bei Einhaltung aller Restriktionen (alle Werte  $g \leq 0$ ) deutlich glattere Verläufe für die Designvariablen gefunden wurden.

Die unstetigen Verläufe des Optimierungs-Resultats sind in diesem Fall also nicht physikalisch begründet. Stattdessen ist davon auszugehen, dass durch die Formulierung des Optimierungsproblems mit der Beta-Methode und die Definition der Designvariablen, der Algorithmus die Variablentypen  $\text{div\_j}$  und  $\text{rho\_j}$  je Komponente beliebig so gegeneinander gewichten konnte, dass die lokale Beta-Restriktion  $g_{\beta, j}(\vec{x})$  minimiert wird.

Für die spätere Anwendung auf reale Propeller ergibt sich daraus folgende Erkenntnis: Vermutlich ist in solchen Fällen eine iterative Optimierungsstrategie sinnvoller, bei der zuerst mit  $\text{div\_j} = \text{const.}$  die Fasermengen optimiert werden. Anschließend könnten für diese per Optimierung die bestmöglichen Asymmetriefaktoren bestimmt werden um mit diesen erneut die Fasermengen anzupassen.

Alternativ könnten andere Formulierungen der Zielfunktion ausprobiert werden, welche direkt die Bauteilmasse beinhalten. Darauf wurde ursprünglich verzichtet, um die in Unterabschnitt 5.4.1 beschriebene Problematik zu umgehen. Deren Auswirkungen könnten jedoch möglicherweise durch eine geschickte Wahl der Optimierungsparameter (s. Abschnitt 6.5) reduziert werden.

Zudem wurden testweise die Variablen  $\text{phi\_0}=\text{phi\_1}=90^\circ$  gesetzt, wodurch die Anstrengungen in den Laminaten um etwa zwei Prozentpunkte leicht anstiegen, sodass die optimierten Werte aus Unterabschnitt 6.6.3 bestand haben.

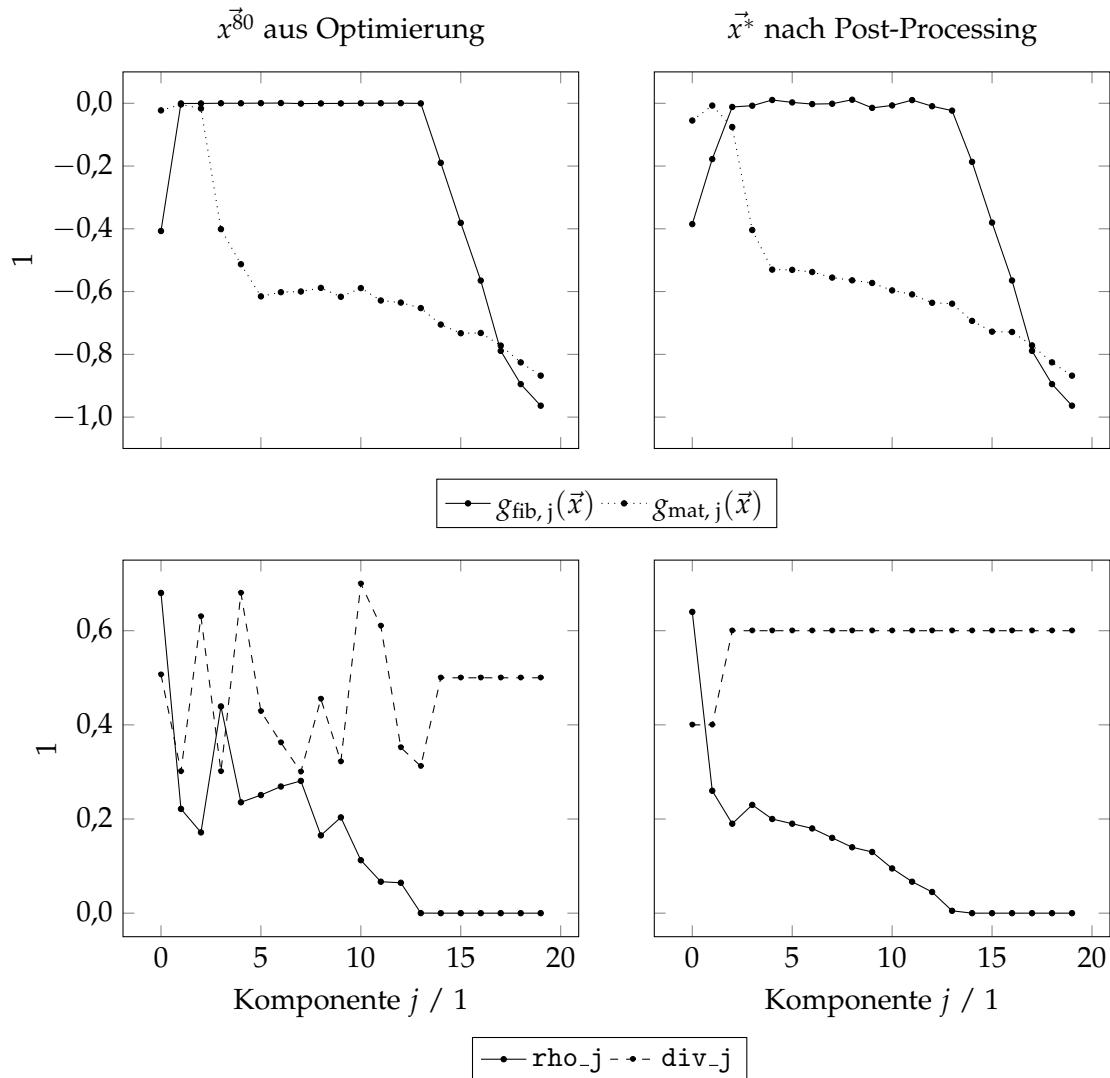


Abbildung 6.11.: Vergleich der ursprünglichen Optimierungsergebnisse sowie der Nachbearbeitung mit JUPYTERLAB.

## 6.8. Direkte Optimierung der Bauteilmasse

Da es, wie in Unterabschnitt 5.4.1 beschrieben, bei einer direkten Optimierung der Bauteilmasse wegen einer möglicherweise schlechten Konditionierung des Optimierungsproblems zu Problemen kommen kann, wurde zunächst eine mit der Beta-Methode formulierte Zielfunktion verwendet. Die entsprechenden Ergebnisse wurden in den vorigen Abschnitten dargestellt und diskutiert. Dort traten unstetige Verläufe der Designvariablen auf, die eine erhebliche Nachbearbeitung notwendig machten.

Daher soll nun ein weiterer Ansatz für die Zielfunktion vorgestellt und erprobt werden.

Die Designvariablen  $\rho_j$  steuern die Fasermenge im Bauteil. Daher stehen sie über die Materialdichte und Volumen in direkter Beziehung zur Bauteilmasse. Als Zielfunktion kann also die Summe über alle  $j$  Variablen  $\rho_j$  verwendet werden:

$$f(\vec{x}) = \sum_{j=0}^{n_{\text{sec}}} \rho_j \quad (6.12)$$

Da diese Formulierung, im Gegensatz zur Masse, unabhängig vom Volumen der einzelnen Komponenten  $j$  ist, entfällt der Effekt der schlechten Konditionierung aus Unterabschnitt 5.4.1, sodass ein verbessertes Konvergenzverhalten des Optimierungsproblems zu erwarten ist.

Dies soll im folgenden Erprobt werden.

### 6.8.1. Optimierungsproblem

Analog zu Unterabschnitt 6.6.1 wurden folgende Variablen vor Optimierungsbeginn festgelegt:

$$\rho_j = 0,0; \quad j = 14, \dots, 19 \quad (6.13)$$

$$\text{div}_j = 0,5; \quad j = 14, \dots, 19 \quad (6.14)$$

Das Optimierungsproblem lautet zusammengefasst:

$$\min f(\vec{x}) \quad (6.15)$$

$$= \min \sum_{j=0}^{13} \rho_j \quad (6.16)$$

sodass:

$$g_{\text{fib}, j}(\vec{x}) = I_{\text{fib}, j} - 1 \leq 0; \quad j = 0, \dots, 19 \quad (6.17)$$

$$g_{\text{mat}, j}(\vec{x}) = I_{\text{mat}, j} - 1 \leq 0; \quad j = 0, \dots, 19 \quad (6.18)$$

mit:

$$\vec{x} = (\phi_0, \phi_1, \rho_0, \text{div}_0, \dots, \rho_{13}, \text{div}_{13}) \quad (6.19)$$

$$0^\circ \leq \text{phi\_i} \leq 180^\circ; \quad i = 0, 1 \quad (6.20)$$

$$0 \leq \text{rho\_i} \leq 1; \quad i = 1, \dots, 13 \quad (6.21)$$

$$0,3 \leq \text{div\_i} \leq 0,7; \quad i = 1, \dots, 13 \quad (6.22)$$

### 6.8.2. Ergebnisse

Als Prozessparameter für den Optimierungsalgorithmus wurden die in Abschnitt 6.5 ermittelte Konfiguration verwendet. Die Optimierungsschleife konvergierte nach  $k = 164$  Iterationen, also mit etwa der doppelten Zeitspanne wie bei Verwendung der Beta-Methode. Die Ergebnisse sind in Abbildung 6.12 dargestellt.

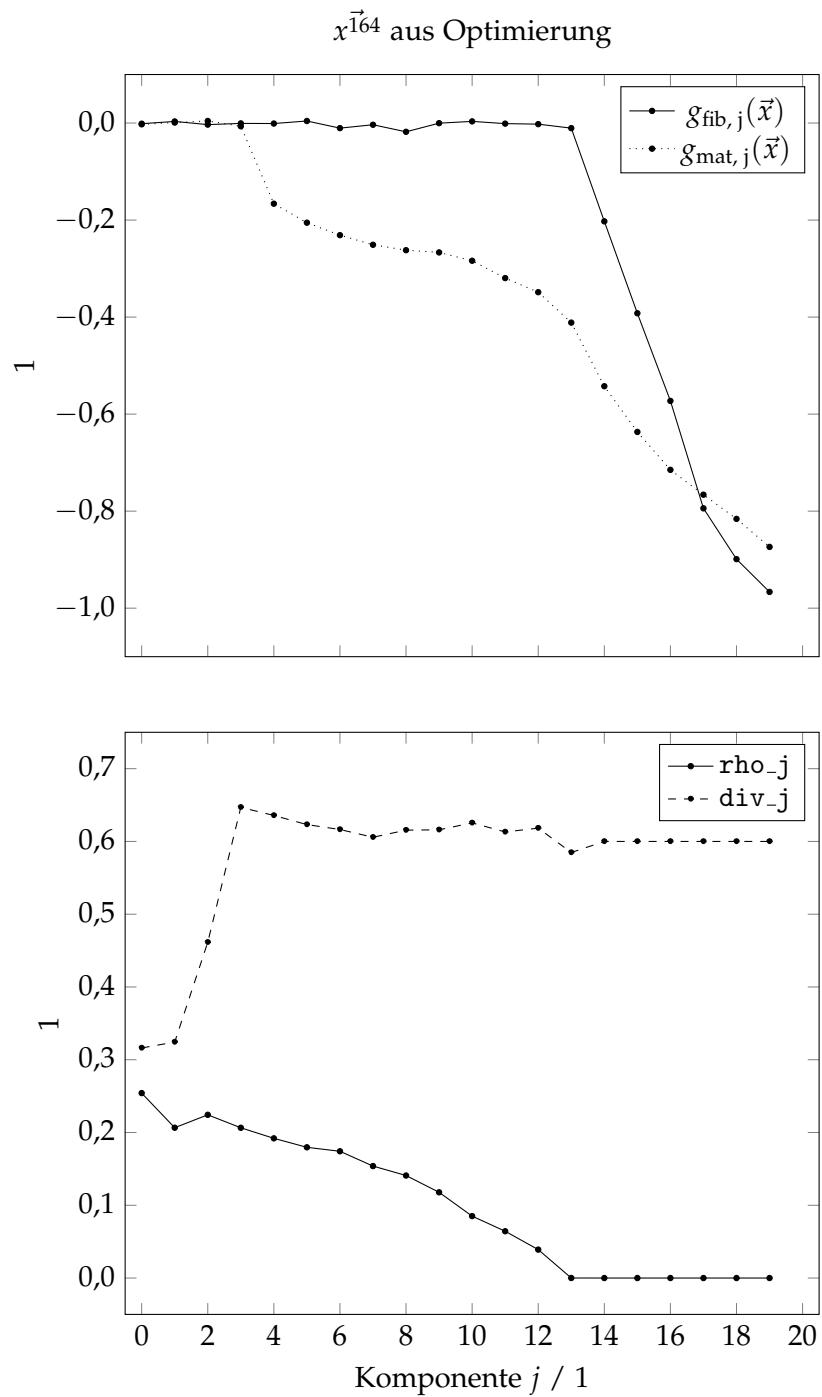
Zunächst fällt auf, dass die gefundenen Verläufe der Faserdichten und Asymmetriefaktoren im Vergleich zu den Ergebnissen aus Unterabschnitt 6.6.3 deutlich glatter ausfallen. Eine sichtbare Kopplung der Kurven ist nicht erkennbar.

Die erzielte Masse ist mit  $m = 30,18 \text{ g}$  um etwa  $5 \text{ g}$  geringer als in der vorigen Optimierung.

Die inneren vier Komponenten  $j = 0, \dots, 3$  zeigen imposant die Fähigkeiten der Optimierung. Hier findet der Algorithmus eine Variablenkonfiguration, die gleichzeitig die Faserbruch- als auch die Zwischenfaserbruch-Restriktionen maximal ausreizt.

Für den äußeren Teil des Blattes mit  $j \geq 4$ , zeigt sich, dass eine asymmetrische Verteilung der Fasern mit etwa  $\text{div}=0,6$  günstig ist, wie in Abschnitt 6.3 vermutet wurde.

Im Vergleich zu der Zielfunktion mit Beta-Methode lieferte der Optimierer hier also insgesamt bessere Ergebnisse. Aufgrund der erkennbar glatteren Designvariablen-Verläufe ist der Post-Processing-Aufwand deutlich geringer. Ebenfalls wird eine geringere Strukturmasse erzielt und die Versagenskriterien vollständiger ausgenutzt.



$$\begin{aligned} \text{phi\_0} &= 97,25^\circ, \quad \text{phi\_1} = 97,95^\circ \\ \text{Masse } m &= 30,18 \text{ g} \end{aligned}$$

Abbildung 6.12.: Ergebnisse der direkten Optimierung der Bauteilmasse

## 7. Zusammenfassung und Ausblick

**Zusammenfassung** In dieser Arbeit wird eine automatisierte Programmkette zur Auslegung und Optimierung von UAV-Propellern aus Faserverbundwerkstoffen entwickelt. Dazu werden zunächst die grundlegenden Konzepte der Strukturoptimierung vorgestellt und ein Überblick über deren Anwendung zur Auslegung von Faserverbund-Bauteilen gegeben.

Auf Basis der Recherche wird als Strategie die *Optimierung mit Teilchenschwärm*en ausgewählt und verschiedene Implementierungen in Python evaluiert, wobei sich der *ALPSO*-Algorithmus aus PyOPT [31] als am günstigsten für die Anwendung erweist.

Die Eignung und Leistungsfähigkeit des Algorithmus für die Optimierung von Faserverbund-Bauteilen wird anhand von zwei analytischen Referenz-Problemen demonstriert.

Anschließend wird auf Basis der Aerodynamik-Tools XFOIL [9] und XROTOR [11] ein Programm zur Lastanalyse von UAV-Propellern entwickelt und in Python implementiert. Dieses erlaubt es, für eine Summe von Lastfällen eine einhüllende Last-Envelope zu berechnen und daraus eine kontinuierliche Auftriebs- und Widerstandsverteilung für den gesamten Propeller-Grundriss abzuleiten.

Das Analysemodell für die Optimierung wird mit der Finite-Elemente-Methode erstellt. Die Implementierung erfolgt mit einem objektorientierten Ansatz in Python, wobei zur Lösung ANSYS MECHANICAL APDL und das Python-Ansys-Interface PyMAPDL [23] verwendet werden.

Dank der Implementierung in Python können zur Last-Definition direkt die mit dem zuvor erstellten Lastrechnungs-Tool generierten Druckverläufe verwendet werden.

Die so erstellte Programmkette besteht aus dem Lastrechnungsprogramm, dem in Python implementierten FE-Modell und ALPSO-Optimierer aus PyOPT. Zur Erklärung und Demonstration ihrer Funktionsweise wird für eine Referenz-Propellergeometrie (*T-Motor MF 3218* [42]) ein optimierter Strukturaufbau berechnet. Dabei werden die Faser-Orientierungen, -Mengen und ihre Verteilung in den tragenden Querschnitten variiert. Als Zielfunktion werden zwei verschiedene Formulierungen vorgestellt: Zunächst wird die Bauteilmasse mittelbar über die Materialanstrengungen im Bauteil minimiert. Die Formulierung erfolgt mit der Beta-Methode nach HARZHEIM [18]. Anschließend wird die Bauteilmasse direkt minimiert, indem die gesamte Fasermenge als Zielfunktion vorgegeben wird. Es stellt sich heraus, dass letztere Formulierung besser geeignet ist, da sie glattere Designvariablen-Verläufe und eine geringere Masse liefert. Für das Post-Processing der Optimierungsergebnisse wird zusätzlich ein Python-Skript mit graphischer Benutzeroberfläche in JUPYTERLAB erstellt.

**Ausblick** Die in dieser Arbeit erstellte Programmkette ist als Werkzeugkasten für die Strukturauslegung und Optimierung von UAV-Propellern zu betrachten. Aufgrund ihres objektorientierten Ansatzes können leicht einzelne Bestandteile verändert oder ersetzt werden, sodass sich eine Vielzahl unterschiedlicher Strukturkonzepte und Optimierungsprobleme untersuchen lassen.

Bei Bedarf ist auch eine Erweiterung für eine inter-disziplinäre Optimierung möglich, bei der zusätzlich zur Strukturmechanik gleichzeitig eine aerodynamische bzw. aeroelastische Optimierung durchgeführt wird. Lohnenswerte Fragestellungen und Probleme für die Anwendung des Programmes lauten beispielsweise:

- Wie groß sind die geringsten Profildicken, für die eine lastgerechte Auslegung realisierbar ist?
- Kann über eine Biege-Torsions-Kopplung die aerodynamische Verdrillung des Propellers neutralisiert werden, sodass der Propeller im Betrieb effizienter wird (*aeroelastic tailoring*)?
- Welche Massenersparnis lässt sich über komplexere Lagenaufbauten erzielen?
- Multidisziplinäre Optimierung: z.B. Einbindung von XROTOR in die Optimierungsschleife und direkte Optimierung der aerodynamischen Effizienz.
- Einbindung des Blatt-Nabe-Anschlussbereichs in die Optimierungsroutine.

Zudem könnte das Programm hin zu einer kompletten Design-Umgebung für UAV-Propeller weiterentwickelt werden: Mittlerweile haben einige CAD-Programme Programmierschnittstellen zu Python, wie z.B. das freie FREECAD. So könnte einerseits die Eingabegeometrie komplett skriptbasiert erstellt werden. Andererseits wäre es möglich, aus den Optimierungsergebnissen automatisiert Lagenaufbauten und Fertigungsunterlagen abzuleiten.

Losgelöst von der konkreten Anwendung auf UAV-Propeller bieten die verwendeten Werkzeuge ein erhebliches Verbesserungs-Potential für numerische Simulationen: Über die Schnittstelle PyMAPDL können innerhalb der FEM-Prozesskette alle Software-Pakete des Python-Universums verwendet werden. Die hier verwendeten Optimierungs-Bibliotheken sind nur ein Beispiel dafür.

Genauso denkbar ist eine Verwendung von Tools für maschinelles Lernen bzw. Data Science, Computer Vision, Symbolisches Rechnen und vielen mehr.

## Literatur

- [1] H. AN u. a. »Laminate stacking sequence optimization with strength constraints using two-level approximations and adaptive genetic algorithm«. In: *Structural and Multidisciplinary Optimization* 51.4 (2014), S. 903–918 (siehe S. 9).
- [2] F. BISCANI und D. IZZO. »A parallel global multiobjective framework for optimization: pagmo«. In: *Journal of Open Source Software* 5.53 (2020), S. 2338 (siehe S. 12).
- [3] M. W. BLOOMFIELD u. a. »Enhanced two-level optimization of anisotropic laminated composite plates with strength and buckling constraints«. In: *Thin-Walled Structures* 47.11 (2009), S. 1161–1167 (siehe S. 9, 11).
- [4] H. BORST u. a. *Summary of Propeller Design Procedures and Data. Volume 2. Structural Analysis and Blade Design.* 1973 (siehe S. 22, 50).
- [5] G. BRADSKI. »The OpenCV Library«. In: *Dr. Dobb's Journal of Software Tools* (2000) (siehe S. 48).
- [6] W. J. BRÄUNLING. *Flugzeugtriebwerke: Grundlagen, Aero-Thermodynamik, ideale und reale Kreisprozesse, Thermische Turbomaschinen, Komponenten, Emissionen und Systeme (VDI-Buch) (German Edition)*. Springer Vieweg, 2015 (siehe S. 22).
- [7] BUNDESMINISTERIUM FÜR VERKEHR UND DIGITALE INFRASTRUKTUR. *Automatisierte luftgestützte Messung der Schadstoffbelastung in der erdnahen Atmosphäre in urbanen Räumen - MesSBAR.* 2020. URL: <https://www.bmvi.de/SharedDocs/DE/Artikel/DG/mfund-projekte/messbar.html> (siehe S. 10, 48).
- [8] J. CHEN u. a. »Probabilistic optimal design of laminates using improved particle swarm optimization«. In: *Engineering Optimization* 40.8 (2008), S. 695–708 (siehe S. 9, 11).
- [9] M. DRELA und H. YOUNGREN. XFOIL. 2013. URL: <https://web.mit.edu/drela/Public/web/xfoil/> (besucht am 11.02.2021) (siehe S. 23, 24, 70).
- [10] M. DRELA und H. YOUNGREN. XFOIL 6.9 User Primer (siehe S. 24, 28, 42).
- [11] M. DRELA und H. YOUNGREN. XROTOR Download Page. 2011. URL: <https://web.mit.edu/drela/Public/web/xrotor/> (besucht am 11.02.2021) (siehe S. 23, 70).
- [12] M. DRELA und H. YOUNGREN. XROTOR User Guide. 2003. URL: [https://web.mit.edu/drela/Public/web/xrotor/xrotor\\_doc.txt](https://web.mit.edu/drela/Public/web/xrotor/xrotor_doc.txt) (besucht am 11.02.2021) (siehe S. 24, 27, 54).
- [13] A. DURAISAMY. »Understanding the acoustic behaviour of natural fiber composites and the effects of temperature and humidity«. Magisterarb. Structures und Composite Materials Laboratory, McGill University, Montreal, 2017 (siehe S. 16).
- [14] T. VO-DUY u. a. »Multi-objective optimization of laminated composite beam structures using NSGA-II algorithm«. In: *Composite Structures* 168 (2017), S. 498–509 (siehe S. 9).
- [15] DWD - DEUTSCHER WETTERDIENST. ICAO-Standardatmosphäre (ISA). 2021. URL: [https://www.dwd.de/DE/service/lexikon/begriffe/S/Standardatmosphaere\\_pdf.pdf?\\_\\_blob=publicationFile&v=3](https://www.dwd.de/DE/service/lexikon/begriffe/S/Standardatmosphaere_pdf.pdf?__blob=publicationFile&v=3) (besucht am 10.03.2021) (siehe S. 50).

- [16] ECO TECHNILIN. *TECHNICAL DATA SHEET-FLAXPREG T-UD*. 2020. URL: <https://eco-technilin.com/img/cms/2019%20-%20TDS%20FlaxPreg%20T-UD.pdf> (siehe S. 16).
- [17] GNU PROJECT. *What is free software? - The Free Software Definition*. 2021. URL: <https://www.gnu.org/philosophy/free-sw> (besucht am 25.03.2021) (siehe S. 1).
- [18] L. HARZHEIM. *Strukturoptimierung: Grundlagen und Anwendungen*. 2019 (siehe S. 2–4, 6, 7, 11, 14, 15, 44, 70).
- [19] A. HAUFFE. *ELAMX2 - expandable Laminate eXplorer*. Hrsg. von CHAIR OF AIRCRAFT ENGINEERING, TECHNISCHE UNIVERSITÄT DRESDEN. 2021. URL: <https://tu-dresden.de/ing/maschinenwesen/ilr/lft/elamx2/elamx> (siehe S. 17).
- [20] L. HILBERS. »Evaluierung der Eignung von nachwachsenden Rohstoffen für die Entwicklung von Multicopterpropellern«. Studienarbeit. Institut für Flugzeugbau und Leichtbau, Technische Universität Braunschweig, 2020 (siehe S. 10, 52).
- [21] ING. PETER C. KLEIN. »Parametric Modeling and Optimization of Advanced Propellers for Next-Generation Aircraft«. Magisterarb. Delft University of Technology, 2017 (siehe S. 22, 23).
- [22] P. JANSEN und R. PEREZ. »Constrained structural design optimization via a parallel augmented Lagrangian particle swarm optimization approach«. In: *Computers & Structures* 89.13–14 (2011), S. 1352–1366 (siehe S. 13, 15, 56).
- [23] A. KASZYNSKI. *pyansys: Python Interface to MAPDL and Associated Binary and ASCII Files*. en. 2020 (siehe S. 1, 16, 33, 70).
- [24] R. KATHIRAVAN und R. GANGULI. »Strength design of composite beam using gradient and particle swarm optimization«. In: *Composite Structures* 81.4 (2007), S. 471–479 (siehe S. 8, 11).
- [25] W. LI und S. KRIST. »Spline-Based Airfoil Curvature Smoothing and Its Applications«. In: *Journal of Aircraft* 42.4 (2005), S. 1065–1074 (siehe S. 49).
- [26] R. H. LOPEZ u. a. »An approach for the reliability based design optimization of laminated composites«. In: *Engineering Optimization* 43.10 (2011), S. 1079–1094 (siehe S. 9, 11).
- [27] L. J. V. MIRANDA. »PySwarms: a research toolkit for Particle Swarm Optimization in Python«. In: *The Journal of Open Source Software* 3.21 (2018), S. 433 (siehe S. 12).
- [28] S. NIKBAKT u. a. »A review on optimization of composite structures Part I: Laminated composites«. In: *Composite Structures* 195 (2018), S. 158–185 (siehe S. 5, 8).
- [29] S. NIKBAKT u. a. »A review on optimization of composite structures Part II: Functionally graded materials«. In: *Composite Structures* 214 (2019), S. 83–102 (siehe S. 8).
- [30] R. PEREZ und K. BEHDINAN. »Particle swarm approach for structural design optimization«. In: *Computers & Structures* 85.19–20 (2007), S. 1579–1588 (siehe S. 14).
- [31] R. E. PEREZ u. a. »pyOpt: a Python-based object-oriented framework for nonlinear constrained optimization«. In: *Structural and Multidisciplinary Optimization* 45.1 (2011), S. 101–118 (siehe S. 1, 12, 16, 34, 46, 70).

- [32] A. P. PIOTROWSKI u. a. »Population size in Particle Swarm Optimization«. In: *Swarm and Evolutionary Computation* 58 (2020), S. 100718 (siehe S. 56).
- [33] R. PONZA und E. BENINI. »Airfoil Data Fitting Using Multivariate Smoothing Thin Plate Splines«. In: *AIAA Journal* 49.2 (2011), S. 349–364 (siehe S. 49).
- [34] L. PRANDTL. *Vier Abhandlungen zur Hydrodynamik und Aerodynamik*. Göttingen: Universitätsverlag Göttingen c/o SUB Göttingen, 2010 (siehe S. 22).
- [35] PYTHON SOFTWARE FOUNDATION. *Python 3.9.1 documentation*. 2021. URL: <https://docs.python.org/3/> (besucht am 12.02.2021) (siehe S. 24).
- [36] R. SATHEESH u. a. »Conservative Design Optimization of Laminated Composite Structures Using Genetic Algorithms and Multiple Failure Criteria«. In: *Journal of Composite Materials* 44.3 (2009), S. 369–387 (siehe S. 9).
- [37] H. SCHLICHTING und E. TRUCKENBRODT. *Aerodynamik des Flugzeuges*. Springer Berlin Heidelberg, 2001 (siehe S. 42).
- [38] A. SCHUMACHER. *Optimierung mechanischer Strukturen*. Springer Berlin Heidelberg, 2013 (siehe S. 2, 5, 7).
- [39] H. SCHÜRMANN. *Konstruieren mit Faser-Kunststoff-Verbunden*. Springer Berlin Heidelberg, 2007 (siehe S. 17, 18).
- [40] S. SURESH u. a. »Particle swarm optimization approach for multi-objective composite box-beam design«. In: *Composite Structures* 81.4 (2007), S. 598–605 (siehe S. 8, 11).
- [41] A. THEOPHILUS. *PSOPy: A python implementation of Particle Swarm Optimization*. 2021. URL: <https://github.com/jerrytheo/psopy> (siehe S. 12).
- [42] TMOTOR. *MF3218 Information and Specifications*. 2021. URL: <http://store-en.tmotor.com/goods.php?id=747> (siehe S. 48, 50, 70).
- [43] P. VIRTANEN u. a. »SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python«. In: *Nature Methods* 17 (2020), S. 261–272 (siehe S. 25, 48).
- [44] ZEIT ONLINE. *50 Jahre Moore's Law - Ein Grundsatz, der die IT-Branche seit 50 Jahren antreibt*. 2015. URL: <https://www.zeit.de/digital/2015-04/moorsches-gesetz-computerchip-50-jahre> (besucht am 25.03.2021) (siehe S. 1).

# **Appendix**

## A. Verwendete Software

Typ	Notebook	Desktop-PC
Betriebssystem	Arch Linux x86_64	CentOS Linux 7
Pakete	xorg-server-xvfb 1.20.10-3 openmpi 4.0.5-2  xfoil 6.99-2 xrotor 7.55-1 ANSYS Mechanical APDL 2020 R1	xorg-x11-server-Xvfb-x86_64 1.20.4-13.el7-9 openmpi3-x86_64 3.1.3-2.el7 openmpi3-devel.x86_64 3.1.3-2.el7 xfoil.x86_64 6.99-1.el7 xrotor.x86_64 7.55-4.el7 ANSYS Mechanical APDL 2019 R1
Python	Python 3.8.5 mpi4py 3.0.3 numpy 1.19.5 opencv-python 4.4.0.46 pandas 1.1.3 pyansys 0.44.17 pyopt 1.2.0 scipy 1.5.3	Python 3.8.3 mpi4py 3.0.3 numpy 1.19.5 opencv-python 4.4.0.46 pandas 1.1.3 pyansys 0.44.17 pyopt 1.2.0 scipy 1.6.0

Tabelle A.1.: Verwendete Software mit Versionen

## B. Inhalt Daten-CD

1. **1\_Text:** Git-Repository des Textes, aller Abbildungen und PDF-Datei
2. **2\_Software-Reposiotys**

- a) **helics\_opt:** Repository mit Lastrechnungs- und Optimierungstool samt html-Dokumentation und UML-Diagrammen
- b) **photogrammetrie:** OPENCV-SCRIPT zur photogrammetrischen Propeller-Vermessung
- c) **pyopt:** Repository der modifizierten PyOPT Bibliothek.

## C. UML-Diagramme

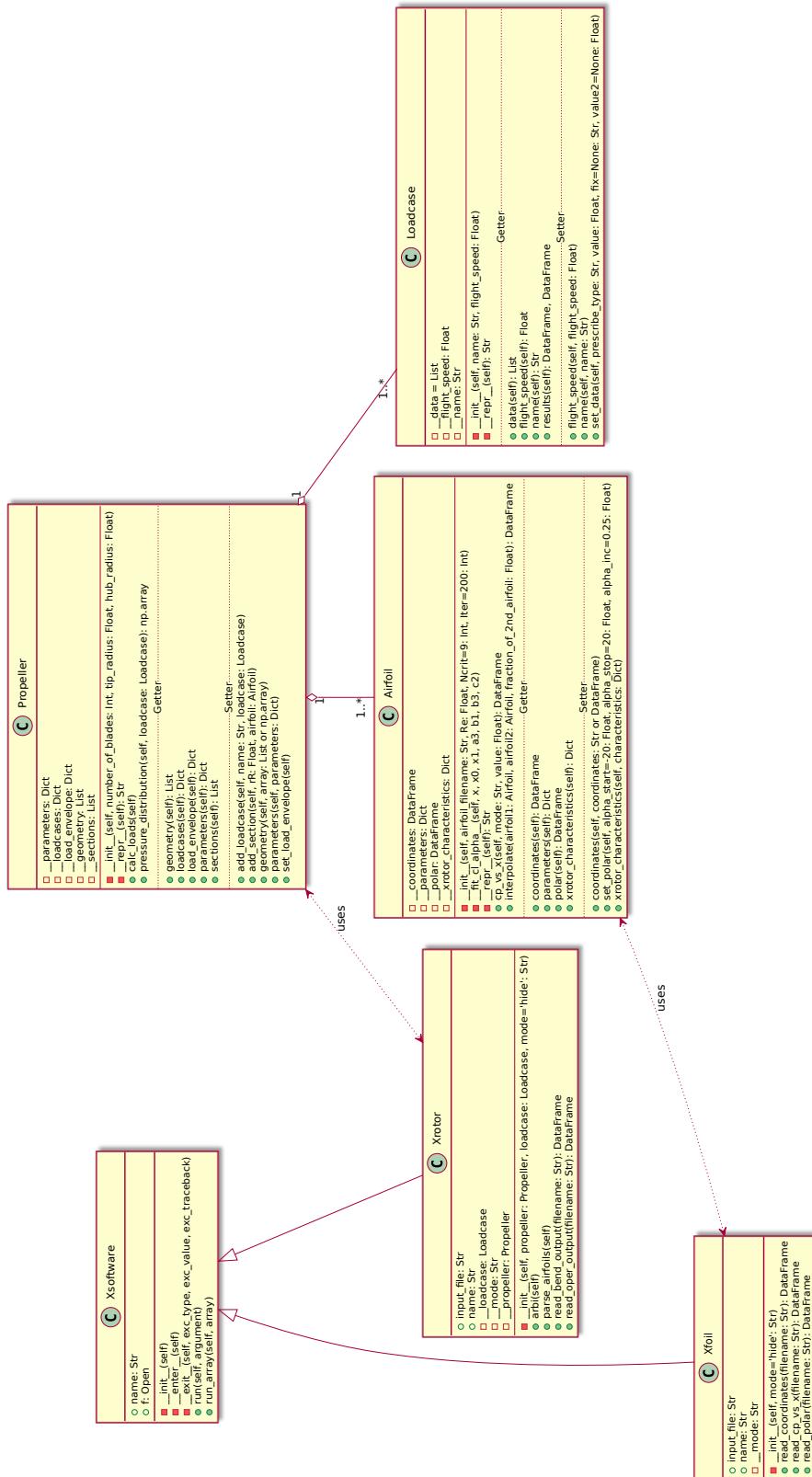


Abbildung C.1.: UML-Klassendiagramm des Lastrechnungsprogramms

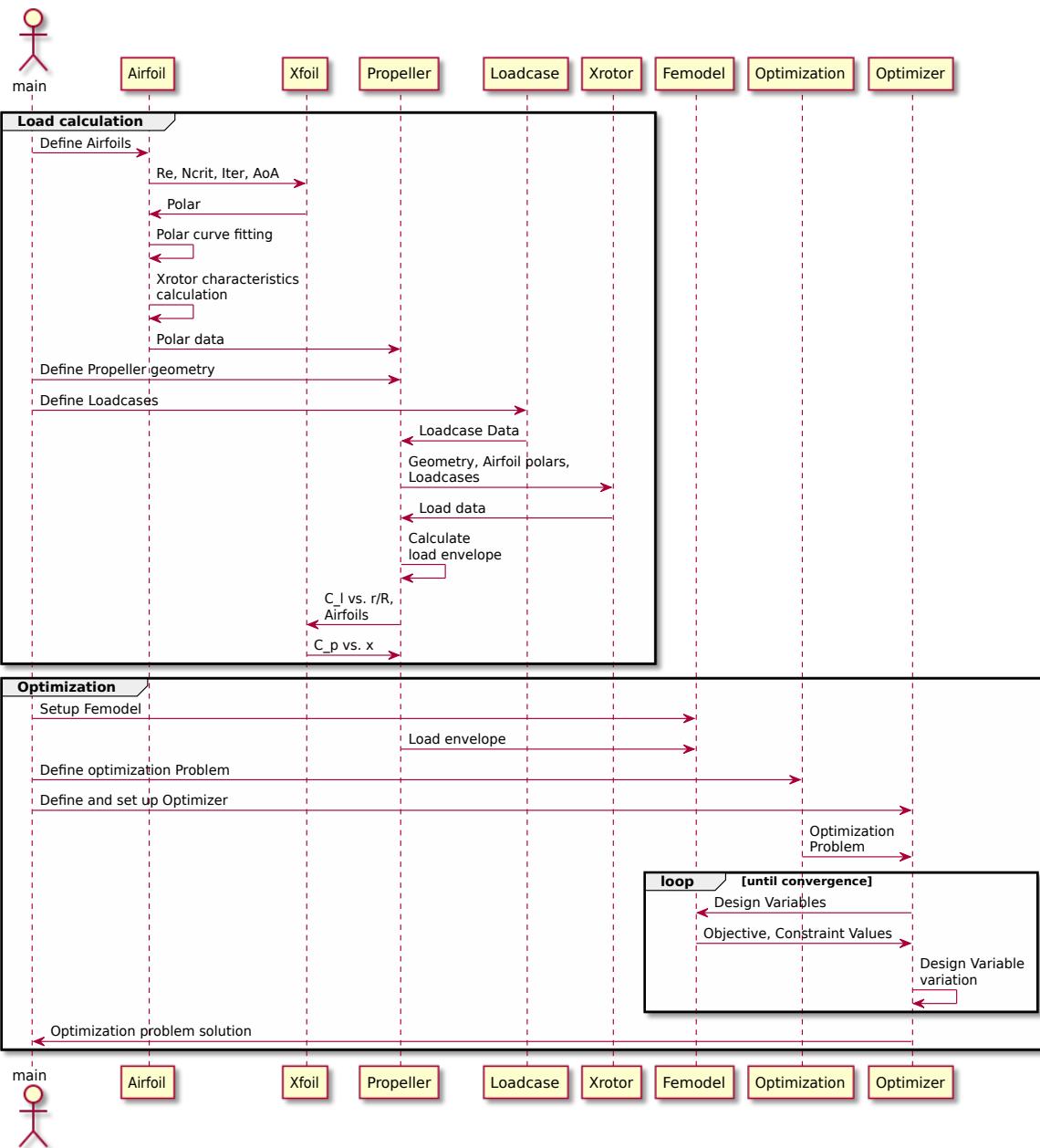


Abbildung C.2.: UML-Sequenzdiagramm des Ablaufs der Programmketten